Diethelm Würtz
Yohan Chalabi
Andrew Ellis

A Discussion of Time Series Objects for R in Finance

September 24, 2009

Rmetrics Association & Finance Online

Zurich

Series Editors:
PD Dr. Diethelm Würtz
Institute of Theoretical Physics and
Curriculum for Computational Science
Swiss Federal Institute of Technology
Hönggerberg, HIT K 32.2
8093 Zurich

Dr. Martin Hanf
Finance Online GmbH
Weinbergstrasse 41
8006 Zurich

Contact Address:
Rmetrics Association
Weinbergstrasse 41
8006 Zurich
info@rmetrics.org

Publisher:
Finance Online GmbH
Swiss Information Technologies
Weinbergstrasse 41
8006 Zurich

Authors:
Yohan Chalabi, ETH Zurich
Andrew Ellis, Finance Online GmbH Zurich
Diethelm Würtz, ETH Zurich

ISBN:
eISBN:
DOI:

Revision History

May 2009: 1st edition

September 2009: 1st edition

Several small fixes in the packages zoo, xts and timeSeries have been considered in this 2nd edition.

zoo has recently added a print `style` argument to its print function, the discussion of this will go to the 3rd edition.

Preface

About this Book

Are you working with R and Rmetrics in the field of finance? Then you will usually use either xts/zoo or timeSeries/timeDate as your preferred time series package of functions and methods to create and manage financial time series objects.

Often you may have asked yourself how certain functions in one time series package compare to their counterparts in the other package. Does a given function from one package have an equivalent one in the other packages and if yes, what are the specific differences. In which sense do functions behave differently when you work with the xts/zoo package or with the timeSeries/timeDate package.

This ebook tries to answer any questions you might have about these issues. For each question you will find an answer together with a generic example. In some cases, the answers are still incomplete, or even missing. In other cases, we not yet found a typical example. We are counting on your support to keep this FAQ up-to-date. We would also very much appreciate it if users and developers of the packages under discussion would send us further FAQs together with the answer, for inclusion in this ebook.

The ebook does not recommend or give any preferences to one of the time series classes. The answers are intended to be unbiased and if you are unsatisfied with any answer, please let us know and give us a suggestion how to improve it. We hope that the ebook can also serve as a source to generate (common) efforts of the developers to align the design and concepts used in zoo, xts and timeSeries.

This FAQ is a Sweave document and we will keep it up-to-date with the most recent packages available on R. The FAQs are published under the GNU GPL License[1] and can be downloaded from the www.rmetrics.org website.

Audience Background

It is assumed that the reader has a basic familiarity with the R statistical environment and knows the packages `xts/zoo` and `timeSeries/timeDate`. A background in finance will be helpful but not necessary. Most importantly, the authors assume that the reader is interested in analyzing and modelling financial data sets with R and Rmetrics.

Note that the book is not only intended as a FAQ. The goal is also that you learn to interpret and to better understand the output of the R date and time series management functions.

Getting Started

When this ebook was written, the most recent copy of R was:

- R version 2.9.1 (2009-06-26), `i386-pc-mingw32`
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils

It can be downloaded from the CRAN[2] (Comprehensive R Archive Network) web site. The contributed R packages `zoo`, `xts`, `timeDate` and `timeSeries` are also downloadable from this site, we do not use development versions from the r-forge repository. Alternatively, packages can be installed directly in

[1] http://www.gnu.org/licenses/licenses.html

[2] http://cran-r-project.org

the R environment. The R packages used in this ebook are described in the appendix where you can also find a copy of the description file.

Acknowledgements

This book would not be possible without the R environment developed by the R Development Core Team (2009).

We are also grateful to many people who have read and commented on draft material and on previous manuscripts. Thanks also to those who contribute to the R-sig-finance mailing list, helping us to test the software.

We cannot name all who have helped us but we would to thank Enrique Bengoechea, Peter Carl, William Chen, Dirk Eddelbuettel, Alexios Ghalanos, Francisco Gochez, Oliver Greshake, Martin Hanf, Elmar Heeb, Kurt Hornik, Stephan Joeri, Dominik Locher, David Lüthi, Dominik Lutz, Martin Mächler, Mahendra Mehta, Brian Petersen, Bernhard Pfaff, Jeff Ryan, David Scott, Stefan Theussl, Achim Zeileis, the Institute for Theoretical Physics at ETH Zurich, Finance Online GmbH in Zurich, and the participants of the R/R-metrics Meielisalp Workshops 2007, 2008 and 2009, and the participants of the RinFinance Workshop in Chicago April 2009.

This book is the second in a series of Rmetrics ebooks. These ebooks will cover the whole spectrum of Rmetrics and related packages, from managing chronological objects to dealing with risk and managing portfolios.

Enjoy it!

Diethelm Würtz
Zurich, May 2009, 1st Edition
Zurich, September 2009, 2nd Edition

Contents

Chapter 1
Time Series Basics

1.1 Creating Time Series Objects

Required R package(s):

```
> library(zoo)
> library(xts)
> library(timeSeries)
```

Which are the preferred time series classes in R for financial applications?

These are `zoo` from the R package zoo, its extension `xts` from R's `xts` package, and `timeSeries` from the R/Rmetrics package `timeSeries`, which uses the `timeDate` package for its time stamps.

What are the major differences concerning the time stamps of these three time series classes ?

`zoo` and `xts` are time series classes with time stamp indices depending on the time stamp index class which we used for their creation. `timeSeries` objects are independent of the type of time stamps used for its creation. Time stamps in timeSeries objects are always of class `numeric`.

What time stamp index classes are usually used to create timeSeries objects?

For `zoo` and `xts` these are `Date` for the use with daily data records where we don't care about time zone, TZ, information and POSIXct when work with intraday data records and when time zones and daylight saving times, DST, become relevant. The time stamps used for `timeSeries` objects are `timeDate` objects with internal Olsen time zone data base information.

Do time series depend on the way they were created?

This may be the case for `zoo` and `xts` time series which were created in different time zones under different operating systems. For this cases the result depends on the internal implementation of time zone and daylight saving time rules which may be different for different operating systems. Using `timeSeries` objects you get always the same result on any computer. The reason is that `timeSeries` objects are handled internally with `timeDate` objects which always operate on POSIXct objects in "GMT" and save the TZ and DST information separate as delivered from Olsens time zone database made available by Rmetrics.

For `zoo` and `xts` these are `Date` for the use with daily data records where we don't care about time zone, TZ, information and `POSIXct` when working with intraday data records and when time zones and daylight saving times, DST, are relevant. The time stamps used for `timeSeries` objects are `timeDate` objects which rely on Olsen's time zone data base information.

What influence does your current time zone environment have on the creation of time series when you don't care about time zone settings of your environment?

Using for `zoo` and `xts` the function `as.POSIXct` with default settings as index class creates time stamps with respect to the setting of the time zone in your local environment.

```
> args(zoo)

function (x = NULL, order.by = index(x), frequency = NULL)
NULL

> args(xts)

function (x = NULL, order.by = index(x), frequency = NULL, unique = TRUE,
    ...)
NULL

> args(as.POSIXct)

function (x, tz = "", ...)
NULL
```

Thus, users in London, New York or Tokyo will get different results. The same holds if we create the index from the function `ISOdatetime`.

```
> args(ISOdatetime)

function (year, month, day, hour, min, sec, tz = "")
NULL
```

Be careful in using the function `ISOdate`.

```
> args(ISOdate)

function (year, month, day, hour = 12, min = 0, sec = 0, tz = "GMT")
NULL
```

By default the function generates an index in GMT and an additional offset of 12 hours. Note that all three functions return an index of class `POSIXct`. For `timeSeries` objects this considerations are obsolete when you create `timeSeries` objects from character strings and the appropriate time zone or in our notation financial centre information.

```
> args(timeSeries)

function (data, charvec, units = NULL, format = NULL, zone = "",
    FinCenter = "", recordIDs = data.frame(), title = NULL, documentation =
NULL,
    ...)
NULL

> args(timeDate)

function (charvec, format = NULL, zone = "", FinCenter = "")
NULL
```

What is my current time zone environment?

Use the function `Sys.timezone()` to find out your current time zone settings.

```
> Sys.timezone()

[1] "GMT"
```

How should I create time series objects with daily time stamps when they are given as character formatted time stamps?

For `zoo` and `xts` time series object the best index class to create a series with daily time stamps is to use `Date`.

```
> args(as.Date)

function (x, ...)
NULL
```

`Date` objects don't care about time zone information, they are handled like objects with TZ "GMT". For timeSeries object the preferred option is to use ans input argument for the time stamps the character vector.

Common Data:

```
> set.seed(1953)
> data <- rnorm(6)
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> zoo(data, as.Date(charvec))

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
  0.021925  -0.904325   0.413238   0.186621   0.230818   0.235680
```

xts:

```
> xts(data, as.Date(charvec))

                [,1]
2009-01-01  0.021925
2009-02-01 -0.904325
2009-03-01  0.413238
2009-04-01  0.186621
2009-05-01  0.230818
2009-06-01  0.235680
```

timeSeries:

```
> timeSeries(data, charvec)

GMT
                TS.1
2009-01-01  0.021925
2009-02-01 -0.904325
2009-03-01  0.413238
2009-04-01  0.186621
2009-05-01  0.230818
2009-06-01  0.235680
```

How can I create time series objects with daily time stamps for the previous 50 days?

Common Data:

```
> set.seed(1953)
> data <- matrix(rnorm(22), ncol = 2)
> now <- "2009-01-05"
```

zoo:

```
> zoo(data, as.Date(now)-0:10)

2008-12-26 -0.1326078  0.473622
2008-12-27 -1.4211978  0.201590
2008-12-28 -0.0046855 -0.259976
2008-12-29  1.0994627  2.504959
2008-12-30  1.4837389  0.570239
2008-12-31  0.2356802  0.791782
2009-01-01  0.2308177 -1.517750
2009-01-02  0.1866212 -0.523212
2009-01-03  0.4132378 -2.330700
2009-01-04 -0.9043255 -0.075514
2009-01-05  0.0219246  0.164796
```

xts:

```
> xts(data, as.Date(now)-0:10)
```

```
                 [,1]      [,2]
2008-12-26 -0.1326078  0.473622
2008-12-27 -1.4211978  0.201590
2008-12-28 -0.0046855 -0.259976
2008-12-29  1.0994627  2.504959
2008-12-30  1.4837389  0.570239
2008-12-31  0.2356802  0.791782
2009-01-01  0.2308177 -1.517750
2009-01-02  0.1866212 -0.523212
2009-01-03  0.4132378 -2.330700
2009-01-04 -0.9043255 -0.075514
2009-01-05  0.0219246  0.164796
```

timeSeries:

```
> timeSeries(data, as.Date(now)-0:10)
```

```
GMT
                 TS.1      TS.2
2009-01-05  0.0219246  0.164796
2009-01-04 -0.9043255 -0.075514
2009-01-03  0.4132378 -2.330700
2009-01-02  0.1866212 -0.523212
2009-01-01  0.2308177 -1.517750
2008-12-31  0.2356802  0.791782
2008-12-30  1.4837389  0.570239
2008-12-29  1.0994627  2.504959
2008-12-28 -0.0046855 -0.259976
2008-12-27 -1.4211978  0.201590
2008-12-26 -0.1326078  0.473622
```

The timeSeries function can also handle R's Date objects.

What can go wrong when I create time series objects from POSIXct indices?

Common Data:

```
> set.seed(1953)
> data <- rnorm(6)
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec
```

```
[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> z1 <- zoo(data, as.POSIXct(charvec))
> z2 <- zoo(data, ISOdatetime(2009, 1:6, 1, 0, 0, 0))
> z3 <- zoo(data, ISOdate(2009, 1:6, 1, 0))
> z1; z2; z3

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
  0.021925  -0.904325   0.413238   0.186621   0.230818   0.235680

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
  0.021925  -0.904325   0.413238   0.186621   0.230818   0.235680

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
  0.021925  -0.904325   0.413238   0.186621   0.230818   0.235680
```

Note that all three examples for zoo time series objects deliver the same output, nothing can bee seen from the output about the conditions under which the the time stamps were created.

xts:

```
> x1 <- xts(data, as.POSIXct(charvec))
> x2 <- xts(data, ISOdatetime(2009, 1:6, 1, 0, 0, 0))
> x3 <- xts(data, ISOdate(2009, 1:6, 1, 0))
> x1; x2; x3

                [,1]
2009-01-01  0.021925
2009-02-01 -0.904325
2009-03-01  0.413238
2009-04-01  0.186621
2009-05-01  0.230818
2009-06-01  0.235680

                [,1]
2009-01-01  0.021925
2009-02-01 -0.904325
2009-03-01  0.413238
2009-04-01  0.186621
```

```
2009-05-01  0.230818
2009-06-01  0.235680

                [,1]
2009-01-01  0.021925
2009-02-01 -0.904325
2009-03-01  0.413238
2009-04-01  0.186621
2009-05-01  0.230818
2009-06-01  0.235680
```

timeSeries:

```
> s1 <- timeSeries(data, charvec)
> s2 <- timeSeries(data, ISOdatetime(2009, 1:6, 1, 0, 0, 0))
> s3 <- timeSeries(data, ISOdate(2009, 1:6, 1, 0))
> s1; s2; s3

GMT
                TS.1
2009-01-01  0.021925
2009-02-01 -0.904325
2009-03-01  0.413238
2009-04-01  0.186621
2009-05-01  0.230818
2009-06-01  0.235680

GMT
                TS.1
2009-01-01  0.021925
2009-02-01 -0.904325
2009-03-01  0.413238
2009-04-01  0.186621
2009-05-01  0.230818
2009-06-01  0.235680

GMT
                TS.1
2009-01-01  0.021925
2009-02-01 -0.904325
2009-03-01  0.413238
2009-04-01  0.186621
2009-05-01  0.230818
2009-06-01  0.235680
```

Does the index/time of a time series object depend on how the object was created?

Common Data:

We take the time series from the previous FAQ.

zoo:

```
> class(index(z1))

[1] "POSIXt"  "POSIXct"

> class(index(z2))

[1] "POSIXt"  "POSIXct"

> class(index(z3))

[1] "POSIXt"  "POSIXct"
```

xts:

```
> class(index(x1))

[1] "POSIXt"  "POSIXct"

> class(index(x2))

[1] "POSIXt"  "POSIXct"

> class(index(x3))

[1] "POSIXt"  "POSIXct"
```

timeSeries:

```
> class(time(s1))

[1] "timeDate"
attr(,"package")
[1] "timeDate"

> class(time(s2))
```

```
[1] "timeDate"
attr(,"package")
[1] "timeDate"
```

```
> class(time(s3))
```

```
[1] "timeDate"
attr(,"package")
[1] "timeDate"
```

1.2 Regular Time Series Objects

Required R package(s):

```
> library(zoo)
> library(xts)
> library(timeSeries)
```

First find out what is a regular time series? The oracle data base management manual tells us:

A time series can be regular or irregular, depending on how predictably data points arrive or occur:

- In a regular time series, the data arrives predictably at predefined intervals. For example, daily summaries of stock market data form a regular time series, and one such time series might be the set of trade volumes and opening, high, low, and closing prices for stock XYZ for the year 1997.
- In an irregular time series, unpredictable bursts of data arrive at unspecified points in time, or most timestamps cannot be characterized by a repeating pattern. For example, account deposits and withdrawals from a bank automated teller machine (ATM) form an irregular time series. An irregular time series may have long periods with no data or short periods with bursts of data.

So the definition here is quite different to that what a R user or Programmer has in mind. In R we understand a regular time series as a time series with fixed equidistant units in a calendarical book keeping define a Δ for the distances between two time stamps or frequency of incoming data points. This is very restrictive. Here are some examples:

The most widely used regular time series include monthly and quarterly data points neglecting day and time stamps. The calendarical counter are then the years (and months), their frequency (per year) is then 12 for monthly data, and 4 for quarterly data.

```
> args(ts)

function (data = NA, start = 1, end = numeric(0), frequency = 1,
    deltat = 1, ts.eps = getOption("ts.eps"), class = if (nseries >
        1) c("mts", "ts") else "ts", names = if (!is.null(dimnames(data)))
colnames(data) else paste("Series",
        seq(nseries)))
NULL

> data <- round(rnorm(24), 4)
> ts(data, start = c(2008, 3), frequency = 12)

         Jan     Feb     Mar     Apr     May     Jun     Jul     Aug
2008                  0.8979  1.0271  0.0648 -1.9318 -0.7276  0.3644
2009  0.6240 -1.2801  0.1193  0.1597 -0.4968 -1.0906  2.0209 -0.1219
2010 -0.8061 -1.0101
         Sep     Oct     Nov     Dec
2008 -1.0174  0.0841  0.8969  0.8801
2009 -0.0517 -0.7233 -0.2446  1.0659
2010

> ts(data, start = c(2008, 3), frequency = 4)

        Qtr1    Qtr2    Qtr3    Qtr4
2008                  0.8979  1.0271
2009  0.0648 -1.9318 -0.7276  0.3644
2010 -1.0174  0.0841  0.8969  0.8801
2011  0.6240 -1.2801  0.1193  0.1597
2012 -0.4968 -1.0906  2.0209 -0.1219
2013 -0.0517 -0.7233 -0.2446  1.0659
2014 -0.8061 -1.0101
```

How can I create a regular monthly time series object?

Common Data:

```
> data <- round(rnorm(24), 4)
```

ts:

```
> tm <- ts(data, start = c(2008, 3), frequency = 12)
> tm

         Jan     Feb     Mar     Apr     May     Jun     Jul     Aug
2008                  1.4640 -1.7286 -0.0438  0.1268 -0.1307  0.2274
2009  0.9402 -1.3392  0.4471 -0.2742  0.4798 -0.7621 -0.4900  0.3214
2010 -0.0621 -0.1795
         Sep     Oct     Nov     Dec
2008 -1.7091 -0.1085  0.4938 -0.5464
```

```
2009 -1.7714 -0.9545 -0.3695  0.0146
2010
```

zoo:

```
> zm <- zooreg(data, start = c(2008, 3), frequency = 12)
> zm

 2008(3)  2008(4)  2008(5)  2008(6)  2008(7)  2008(8)  2008(9) 2008(10)
  1.4640  -1.7286  -0.0438   0.1268  -0.1307   0.2274  -1.7091  -0.1085
2008(11) 2008(12)  2009(1)  2009(2)  2009(3)  2009(4)  2009(5)  2009(6)
  0.4938  -0.5464   0.9402  -1.3392   0.4471  -0.2742   0.4798  -0.7621
 2009(7)  2009(8)  2009(9) 2009(10) 2009(11) 2009(12)  2010(1)  2010(2)
 -0.4900   0.3214  -1.7714  -0.9545  -0.3695   0.0146  -0.0621  -0.1795
```

xts:

```
> xm <- as.xts(tm)
> xm

            [,1]
Mar 2008  1.4640
Apr 2008 -1.7286
May 2008 -0.0438
Jun 2008  0.1268
Jul 2008 -0.1307
Aug 2008  0.2274
Sep 2008 -1.7091
Oct 2008 -0.1085
Nov 2008  0.4938
Dec 2008 -0.5464
Jan 2009  0.9402
Feb 2009 -1.3392
Mar 2009  0.4471
Apr 2009 -0.2742
May 2009  0.4798
Jun 2009 -0.7621
Jul 2009 -0.4900
Aug 2009  0.3214
Sep 2009 -1.7714
Oct 2009 -0.9545
Nov 2009 -0.3695
```

```
Dec 2009  0.0146
Jan 2010 -0.0621
Feb 2010 -0.1795
```

timeSeries:

```
> sm <- as.timeSeries(tm)
> sm

GMT
              TS.1
2008-03-31  1.4640
2008-04-30 -1.7286
2008-05-31 -0.0438
2008-06-30  0.1268
2008-07-31 -0.1307
2008-08-31  0.2274
2008-09-30 -1.7091
2008-10-31 -0.1085
2008-11-30  0.4938
2008-12-31 -0.5464
2009-01-31  0.9402
2009-02-28 -1.3392
2009-03-31  0.4471
2009-04-30 -0.2742
2009-05-31  0.4798
2009-06-30 -0.7621
2009-07-31 -0.4900
2009-08-31  0.3214
2009-09-30 -1.7714
2009-10-31 -0.9545
2009-11-30 -0.3695
2009-12-31  0.0146
2010-01-31 -0.0621
2010-02-28 -0.1795
```

Can `timeSeries` objects be printed in a regular time series style? Yes.

```
> print(sm, style = "h")

2008-03-31 2008-04-30 2008-05-31 2008-06-30 2008-07-31 2008-08-31
    1.4640    -1.7286    -0.0438     0.1268    -0.1307     0.2274
2008-09-30 2008-10-31 2008-11-30 2008-12-31 2009-01-31 2009-02-28
   -1.7091    -0.1085     0.4938    -0.5464     0.9402    -1.3392
```

```
2009-03-31 2009-04-30 2009-05-31 2009-06-30 2009-07-31 2009-08-31
    0.4471    -0.2742     0.4798    -0.7621    -0.4900     0.3214
2009-09-30 2009-10-31 2009-11-30 2009-12-31 2010-01-31 2010-02-28
   -1.7714    -0.9545    -0.3695     0.0146    -0.0621    -0.1795
```

Can `timeSeries` objects be printed in customized format? Yes.

```
> print(sm, style = "h", format = "%Y %b")
```

```
2008 Mar 2008 Apr 2008 May 2008 Jun 2008 Jul 2008 Aug 2008 Sep 2008 Oct
  1.4640  -1.7286  -0.0438   0.1268  -0.1307   0.2274  -1.7091  -0.1085
2008 Nov 2008 Dec 2009 Jan 2009 Feb 2009 Mar 2009 Apr 2009 May 2009 Jun
  0.4938  -0.5464   0.9402  -1.3392   0.4471  -0.2742   0.4798  -0.7621
2009 Jul 2009 Aug 2009 Sep 2009 Oct 2009 Nov 2009 Dec 2010 Jan 2010 Feb
 -0.4900   0.3214  -1.7714  -0.9545  -0.3695   0.0146  -0.0621  -0.1795
```

```
> print(sm, style = "h", format = "%Y(%m)")
```

```
2008(03) 2008(04) 2008(05) 2008(06) 2008(07) 2008(08) 2008(09) 2008(10)
  1.4640  -1.7286  -0.0438   0.1268  -0.1307   0.2274  -1.7091  -0.1085
2008(11) 2008(12) 2009(01) 2009(02) 2009(03) 2009(04) 2009(05) 2009(06)
  0.4938  -0.5464   0.9402  -1.3392   0.4471  -0.2742   0.4798  -0.7621
2009(07) 2009(08) 2009(09) 2009(10) 2009(11) 2009(12) 2010(01) 2010(02)
 -0.4900   0.3214  -1.7714  -0.9545  -0.3695   0.0146  -0.0621  -0.1795
```

How can I create a regular quarterly time series object?

Common Data:

```
> data <- round(rnorm(24), 4)
```

ts:

```
> tq <- ts(data, start = c(2008, 3), frequency = 4)
> tq
```

```
        Qtr1    Qtr2    Qtr3    Qtr4
2008                 -0.0169  0.1863
2009 -0.0093  0.5507 -0.8574 -0.4968
2010  0.2355  1.4373 -2.1298  0.3721
2011  0.1971 -1.5040 -0.8292 -0.2629
2012  0.3991  1.3544  0.5100  0.5955
2013  0.4053 -1.3288  2.5491  1.8821
2014 -0.7443  0.9128
```

zoo:

```
> zq <- zooreg(data, start = c(2008, 3), frequency = 4)
> zq
```

```
2008(3) 2008(4) 2009(1) 2009(2) 2009(3) 2009(4) 2010(1) 2010(2) 2010(3)
-0.0169  0.1863 -0.0093  0.5507 -0.8574 -0.4968  0.2355  1.4373 -2.1298
2010(4) 2011(1) 2011(2) 2011(3) 2011(4) 2012(1) 2012(2) 2012(3) 2012(4)
 0.3721  0.1971 -1.5040 -0.8292 -0.2629  0.3991  1.3544  0.5100  0.5955
2013(1) 2013(2) 2013(3) 2013(4) 2014(1) 2014(2)
 0.4053 -1.3288  2.5491  1.8821 -0.7443  0.9128
```

xts:

```
> xq <- as.xts(tq)
> head(xq)
```

```
           [,1]
2008 Q3 -0.0169
2008 Q4  0.1863
2009 Q1 -0.0093
2009 Q2  0.5507
2009 Q3 -0.8574
2009 Q4 -0.4968
```

timeSeries:

```
> sq <- as.timeSeries(tq)
> head(sq)
```

```
GMT
              TS.1
2008-09-30 -0.0169
2008-12-31  0.1863
2009-03-31 -0.0093
2009-06-30  0.5507
2009-09-30 -0.8574
2009-12-31 -0.4968
```

Can `timeSeries` objects be printed in a regular time series style? Yes.

```
> print(sq, style = "h")
```

```
2008-09-30 2008-12-31 2009-03-31 2009-06-30 2009-09-30 2009-12-31
   -0.0169     0.1863    -0.0093     0.5507    -0.8574    -0.4968
2010-03-31 2010-06-30 2010-09-30 2010-12-31 2011-03-31 2011-06-30
    0.2355     1.4373    -2.1298     0.3721     0.1971    -1.5040
2011-09-30 2011-12-31 2012-03-31 2012-06-30 2012-09-30 2012-12-31
   -0.8292    -0.2629     0.3991     1.3544     0.5100     0.5955
2013-03-31 2013-06-30 2013-09-30 2013-12-31 2014-03-31 2014-06-30
    0.4053    -1.3288     2.5491     1.8821    -0.7443     0.9128
```

Can `timeSeries` objects be printed in customized format? Yes.

```
> print(sq, style = "h", format = "%Y %b")
```

```
2008 Sep 2008 Dec 2009 Mar 2009 Jun 2009 Sep 2009 Dec 2010 Mar 2010 Jun
 -0.0169   0.1863  -0.0093   0.5507  -0.8574  -0.4968   0.2355   1.4373
2010 Sep 2010 Dec 2011 Mar 2011 Jun 2011 Sep 2011 Dec 2012 Mar 2012 Jun
 -2.1298   0.3721   0.1971  -1.5040  -0.8292  -0.2629   0.3991   1.3544
2012 Sep 2012 Dec 2013 Mar 2013 Jun 2013 Sep 2013 Dec 2014 Mar 2014 Jun
  0.5100   0.5955   0.4053  -1.3288   2.5491   1.8821  -0.7443   0.9128
```

```
> print(sq, style = "h", format = "%Q")
```

```
2008 Q3 2008 Q4 2009 Q1 2009 Q2 2009 Q3 2009 Q4 2010 Q1 2010 Q2 2010 Q3
-0.0169  0.1863 -0.0093  0.5507 -0.8574 -0.4968  0.2355  1.4373 -2.1298
2010 Q4 2011 Q1 2011 Q2 2011 Q3 2011 Q4 2012 Q1 2012 Q2 2012 Q3 2012 Q4
 0.3721  0.1971 -1.5040 -0.8292 -0.2629  0.3991  1.3544  0.5100  0.5955
2013 Q1 2013 Q2 2013 Q3 2013 Q4 2014 Q1 2014 Q2
 0.4053 -1.3288  2.5491  1.8821 -0.7443  0.9128
```

How I can find out if a time series is a regular time series?

zoo:

```
> z <- zooreg(data, start = c(2008, 3), frequency = 4)
> is.regular(z)
```

```
[1] TRUE
```

1.3 Time Zone and Daylight Saving Time

Required R package(s):

```
> library(zoo)
> library(xts)
> library(timeSeries)
```

How can I create a time series object which takes care of time zone settings?

Let us create a time series in Zurich which belongs to the "Central European Time" zone, CET.

Common Data:

```
> data <- 1:6
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> z1.zrh <- zoo(data, as.POSIXct(charvec, tz = "CET"))
> z1.zrh

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
         1          2          3          4          5          6
```

xts:

```
> x1.zrh <- xts(data, as.POSIXct(charvec, tz = "CET"))
> x1.zrh
```

```
                    [,1]
2008-12-31 23:00:00    1
2009-01-31 23:00:00    2
2009-02-28 23:00:00    3
2009-03-31 22:00:00    4
2009-04-30 22:00:00    5
2009-05-31 22:00:00    6
```

With the newest version of `xts` this cannot any longer be done. To our understanding `xts` now forces always the time stamps to `GMT` time zone.

timeSeries:

```
> s1.zrh <- timeSeries(data, charvec, zone = "Zurich", FinCenter = "Zurich")
> s1.zrh

Zurich
           TS.1
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
```

Note that the `timeSeries` function has two time zone relevant inputs. The argument `zone` holds the information to which time zone or to be more specific to which financial centre the `charvec` of time stamps belongs, and the second argument `FinCenter` tells us in which time zone or at which financial centre we want to display and use the time series data.

```
> args(timeSeries)

function (data, charvec, units = NULL, format = NULL, zone = "",
    FinCenter = "", recordIDs = data.frame(), title = NULL, documentation =
NULL,
    ...)
NULL
```

Have a look at the output of the possible four options.

```
> timeSeries(data, charvec, zone = "Zurich", FinCenter = "Zurich", units = "s1
.zrh.zrh")
```

```
Zurich
           s1.zrh.zrh
2009-01-01          1
2009-02-01          2
2009-03-01          3
2009-04-01          4
2009-05-01          5
2009-06-01          6

> timeSeries(data, charvec, zone = "GMT", FinCenter = "Zurich", units = "s1.
gmt.zrh")

Zurich
                    s1.gmt.zrh
2009-01-01 01:00:00          1
2009-02-01 01:00:00          2
2009-03-01 01:00:00          3
2009-04-01 02:00:00          4
2009-05-01 02:00:00          5
2009-06-01 02:00:00          6

> timeSeries(data, charvec, zone = "Zurich", FinCenter = "GMT", units = "s1.
zrh.gmt")

GMT
                    s1.zrh.gmt
2008-12-31 23:00:00          1
2009-01-31 23:00:00          2
2009-02-28 23:00:00          3
2009-03-31 22:00:00          4
2009-04-30 22:00:00          5
2009-05-31 22:00:00          6

> timeSeries(data, charvec, zone = "GMT", FinCenter = "GMT", units = "s1.gmt.
gmt")

GMT
           s1.gmt.gmt
2009-01-01          1
2009-02-01          2
2009-03-01          3
2009-04-01          4
2009-05-01          5
2009-06-01          6
```

When I print a time series can I see what time zone the series belongs to?

Common Data:

```
> data <- 1:6
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

Note, the zoo object must be indexed by an index which supports time zones, e.g. objects of class `POSIX`.

```
> z1.zrh <- zoo(data, as.POSIXct(charvec, tz = "CET"))
> z1.zrh

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
         1          2          3          4          5          6
```

For `zoo` time series objects the time zone cannot be seen from printing the series.

xts:

Also `xts` objects must be indexed by an index which supports time zones.

```
> x1.zrh <- xts(data, as.POSIXct(charvec, tz = "CET"))
> x1.zrh

                    [,1]
2008-12-31 23:00:00    1
2009-01-31 23:00:00    2
2009-02-28 23:00:00    3
2009-03-31 22:00:00    4
2009-04-30 22:00:00    5
2009-05-31 22:00:00    6
```

timeSeries:

```
> s1.zrh <- timeSeries(data, charvec, zone = "Zurich", FinCenter = "Zurich")
> s1.zrh

Zurich
           TS.1
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
```

Note that `timeSeries` objects show on top to which time zone (or more specific financial centre) they belong. The information is taken from the time stamps which are objects of class `timeDate`.

How can I find out to what time zone a time series belongs?

To find out the time zone information we can use the functions `index` for `zoo` and `xts` objects and the function `time` for `timeSeries` objects to display the zone information

Common Data:

```
> data <- 1:6
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> data <- 1:6
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> z1.zrh <- zoo(data, as.POSIXct(charvec, tz = "CET"))
> index(z1.zrh)
```

```
[1] "2009-01-01 CET"  "2009-02-01 CET"  "2009-03-01 CET"  "2009-04-01 CEST"
[5] "2009-05-01 CEST" "2009-06-01 CEST"
```

xts:

```
> x1.zrh <- xts(data, as.POSIXct(charvec, tz = "CET"))
> index(x1.zrh)
```

```
[1] "2008-12-31 23:00:00 GMT" "2009-01-31 23:00:00 GMT"
[3] "2009-02-28 23:00:00 GMT" "2009-03-31 22:00:00 GMT"
[5] "2009-04-30 22:00:00 GMT" "2009-05-31 22:00:00 GMT"
```

timeSeries:

```
> s1.zrh <- timeSeries(data, charvec, zone = "Zurich", FinCenter = "Zurich")
> time(s1.zrh)
```

```
Zurich
[1] [2009-01-01] [2009-02-01] [2009-03-01] [2009-04-01] [2009-05-01]
[6] [2009-06-01]
```

Note that for `timeSeries` objects we can also retrieve and save zone information using the accessor function `finCenter()`.

```
> currentCenter <- finCenter(s1.zrh)
> currentCenter
```

```
[1] "Zurich"
```

How can I display the DST rules used for the creation of time series objects?

zoo and xts:

There is no obvious way to do it.

timeSeries:

For `timeSeries` objects you can look in the `data.frame` of DST rules. To shorten the output we subset it here to some records

```
> Zurich()[54:64, ]
```

```
                Zurich offSet isdst TimeZone    numeric
54 2005-03-27 01:00:00   7200     1     CEST 1111885200
55 2005-10-30 01:00:00   3600     0      CET 1130634000
56 2006-03-26 01:00:00   7200     1     CEST 1143334800
57 2006-10-29 01:00:00   3600     0      CET 1162083600
58 2007-03-25 01:00:00   7200     1     CEST 1174784400
59 2007-10-28 01:00:00   3600     0      CET 1193533200
60 2008-03-30 01:00:00   7200     1     CEST 1206838800
61 2008-10-26 01:00:00   3600     0      CET 1224982800
62 2009-03-29 01:00:00   7200     1     CEST 1238288400
63 2009-10-25 01:00:00   3600     0      CET 1256432400
64 2010-03-28 01:00:00   7200     1     CEST 1269738000
```

The table shows when the clock was changed in Zurich, the offset in seconds with respect to GMT, a flag which tells us if DST is in effect or not, the time zone abbreviation, and a integer value in seconds belonging to the time stamp when the clock was changed.

Which time zones are supported for the creation of time series objects?

zoo and xts:

We do not know how to create a list of time zones available in R's POSIXt implementation and thus which time zones are available to use to create `zoo` and `xts` time series objects. Can you please quickly tell us to which time zone the stock exchange in Mumbai belongs?
This is operating-system dependent; on Unix systems the list is in /usr/share/tzone

timeSeries:

For `timeSeries` objects we can print the list of supported financial centres using the functions `listFinCenter`

```
> length(listFinCenter())
```

```
[1] 397
```

These are too many to get printed here. To display a limited number you can select them by `greped` pattern. Here are some examples, financial centres in the Pacific region, and cities starting with "L"

```
> listFinCenter("Pacific")

 [1] "Pacific/Apia"        "Pacific/Auckland"     "Pacific/Chatham"
 [4] "Pacific/Efate"       "Pacific/Enderbury"    "Pacific/Fakaofo"
 [7] "Pacific/Fiji"        "Pacific/Funafuti"     "Pacific/Galapagos"
[10] "Pacific/Gambier"     "Pacific/Guadalcanal"  "Pacific/Guam"
[13] "Pacific/Honolulu"    "Pacific/Johnston"     "Pacific/Kiritimati"
[16] "Pacific/Kosrae"      "Pacific/Kwajalein"    "Pacific/Majuro"
[19] "Pacific/Marquesas"   "Pacific/Midway"       "Pacific/Nauru"
[22] "Pacific/Niue"        "Pacific/Norfolk"      "Pacific/Noumea"
[25] "Pacific/Pago_Pago"   "Pacific/Palau"        "Pacific/Pitcairn"
[28] "Pacific/Ponape"      "Pacific/Port_Moresby" "Pacific/Rarotonga"
[31] "Pacific/Saipan"      "Pacific/Tahiti"       "Pacific/Tarawa"
[34] "Pacific/Tongatapu"   "Pacific/Truk"         "Pacific/Wake"
[37] "Pacific/Wallis"

> listFinCenter(".*/L")

 [1] "Africa/Lagos"                "Africa/Libreville"
 [3] "Africa/Lome"                 "Africa/Luanda"
 [5] "Africa/Lubumbashi"           "Africa/Lusaka"
 [7] "America/Argentina/La_Rioja"  "America/Kentucky/Louisville"
 [9] "America/La_Paz"              "America/Lima"
[11] "America/Los_Angeles"         "Arctic/Longyearbyen"
[13] "Australia/Lindeman"          "Australia/Lord_Howe"
[15] "Europe/Lisbon"               "Europe/Ljubljana"
[17] "Europe/London"               "Europe/Luxembourg"
```

Yo can even create your own financial centres. For example the DST rules for Germany are in "Germany/Berlin", as a banker you may prefer Frankfurt, then just create your own table.

```
> Frankfurt <- Berlin
> timeSeries(runif(1:12), timeCalendar(), zone = "Frankfurt", FinCenter = "
Frankfurt")

Frankfurt
                TS.1
2009-01-01 0.7571371
2009-02-01 0.0056799
2009-03-01 0.9546555
2009-04-01 0.1146348
```

```
2009-05-01 0.7857066
2009-06-01 0.9515243
2009-07-01 0.2741616
2009-08-01 0.3301645
2009-09-01 0.5268424
2009-10-01 0.8832456
2009-11-01 0.3436831
2009-12-01 0.6653792
```

Note that timeCalendar is a function from the package timeDate and creates monthly time stamps for the current year.

How can I change the time zone representation for an existing time series?

zoo and xts:

We don't know how to change time zone in a direct way. As a workaround we recommend to extract the index from the time series, then to add the time shift between the new and old timezone, and finally to reassign the new index to the `zoo` or `xts` object.

timeSeries:

In `timeSeries` we use the assignment function `finCenter<-` to assign a new financial centre to an already existing `timeSeries` object. For example to express a time series recorded in London for use in Zurich in time stamps of local time in New York proceed as follows

```
> ZRH <- timeSeries(rnorm(6), timeCalendar(2009)[7:12], zone = "London",
FinCenter = "Zurich")
> ZRH

Zurich
                       TS.1
2009-07-01 01:00:00 1.51711
2009-08-01 01:00:00 0.91393
2009-09-01 01:00:00 0.37900
2009-10-01 01:00:00 0.32346
2009-11-01 01:00:00 0.51925
2009-12-01 01:00:00 3.06632
```

```
> finCenter(ZRH) <- "New_York"
> ZRH

New_York
                       TS.1
2009-06-30 19:00:00 1.51711
2009-07-31 19:00:00 0.91393
2009-08-31 19:00:00 0.37900
2009-09-30 19:00:00 0.32346
2009-10-31 20:00:00 0.51925
2009-11-30 19:00:00 3.06632
```

This example reflects also the fact that Europe and USA changed from summer time to winter time in different months, i.e October and November respectively! Have a look in the DST tables to confirm this

```
> Zurich()[63,]

                Zurich offSet isdst TimeZone    numeric
63 2009-10-25 01:00:00   3600     0      CET 1256432400

> New_York()[179,]

               New_York offSet isdst TimeZone    numeric
179 2009-11-01 06:00:00 -18000     0      EST 1257055200
```

How can I handle data recorded in two cities which belong to the same time zone but have different DST rules?

zoo and xts:

Not supported by `zoo` and `xts` objects.

timeSeries:

This happened for example in Germany and Switzerland several times in the years between 1940 and 1985. Have a look on the table with the DST rules.

```
> Berlin()[8:38,]
```

```
                 Berlin offSet isdst TimeZone    numeric
8  1940-04-01 01:00:00   7200     1     CEST -938905200
9  1942-11-02 01:00:00   3600     0      CET -857257200
10 1943-03-29 01:00:00   7200     1     CEST -844556400
11 1943-10-04 01:00:00   3600     0      CET -828226800
12 1944-04-03 01:00:00   7200     1     CEST -812502000
13 1944-10-02 01:00:00   3600     0      CET -796777200
14 1945-04-02 01:00:00   7200     1     CEST -781052400
15 1945-05-24 00:00:00  10800     1     CEMT -776563200
16 1945-09-24 00:00:00   7200     1     CEST -765936000
17 1945-11-18 01:00:00   3600     0      CET -761180400
18 1946-04-14 01:00:00   7200     1     CEST -748479600
19 1946-10-07 01:00:00   3600     0      CET -733273200
20 1947-04-06 02:00:00   7200     1     CEST -717631200
21 1947-05-11 01:00:00  10800     1     CEMT -714610800
22 1947-06-29 00:00:00   7200     1     CEST -710380800
23 1947-10-05 01:00:00   3600     0      CET -701910000
24 1948-04-18 01:00:00   7200     1     CEST -684975600
25 1948-10-03 01:00:00   3600     0      CET -670460400
26 1949-04-10 01:00:00   7200     1     CEST -654130800
27 1949-10-02 01:00:00   3600     0      CET -639010800
28 1980-04-06 01:00:00   7200     1     CEST  323830800
29 1980-09-28 01:00:00   3600     0      CET  338950800
30 1981-03-29 01:00:00   7200     1     CEST  354675600
31 1981-09-27 01:00:00   3600     0      CET  370400400
32 1982-03-28 01:00:00   7200     1     CEST  386125200
33 1982-09-26 01:00:00   3600     0      CET  401850000
34 1983-03-27 01:00:00   7200     1     CEST  417574800
35 1983-09-25 01:00:00   3600     0      CET  433299600
36 1984-03-25 01:00:00   7200     1     CEST  449024400
37 1984-09-30 01:00:00   3600     0      CET  465354000
38 1985-03-31 01:00:00   7200     1     CEST  481078800
```

```
> Zurich()[2:15,]
```

```
                 Zurich offSet isdst TimeZone    numeric
2  1941-05-05 00:00:00   7200     1     CEST -904435200
3  1941-10-06 00:00:00   3600     0      CET -891129600
4  1942-05-04 00:00:00   7200     1     CEST -872985600
5  1942-10-05 00:00:00   3600     0      CET -859680000
6  1981-03-29 01:00:00   7200     1     CEST  354675600
7  1981-09-27 01:00:00   3600     0      CET  370400400
8  1982-03-28 01:00:00   7200     1     CEST  386125200
9  1982-09-26 01:00:00   3600     0      CET  401850000
10 1983-03-27 01:00:00   7200     1     CEST  417574800
11 1983-09-25 01:00:00   3600     0      CET  433299600
12 1984-03-25 01:00:00   7200     1     CEST  449024400
```

```
13 1984-09-30 01:00:00   3600     0      CET  465354000
14 1985-03-31 01:00:00   7200     1     CEST  481078800
15 1985-09-29 01:00:00   3600     0      CET  496803600
```

We have end-of-day data 16:00 recorded in Zurich in local time, but now we want to use them in Berlin. The question now is: what is the earliest time we can start with the investigation of the data at local Berlin time.

```
> charvec <- paste("1980-0", 2:5, "-15 16:00:00", sep = "")
> charvec

[1] "1980-02-15 16:00:00" "1980-03-15 16:00:00" "1980-04-15 16:00:00"
[4] "1980-05-15 16:00:00"

> timeSeries(runif(4), charvec, zone = "Zurich", FinCenter = "Berlin", units =
 "fromZurich")

Berlin
                    fromZurich
1980-02-15 16:00:00    0.74680
1980-03-15 16:00:00    0.90749
1980-04-15 17:00:00    0.99422
1980-05-15 17:00:00    0.79543
```

So in February and March we can start our investigation in Berlin at the same time as in Zurich 16:00, but in April and May we can start with our investigation one hour later at 17:00 due to the time difference to Zurich.

1.4 Ordering and Overlapping of Time Series

Required R package(s):

```
> library(zoo)
> library(xts)
> library(timeSeries)
```

How can I create a time series with time stamps in reverse order?

Common Data:

```
> set.seed <- 1953
> data <- rnorm(6)
> charvec <- rev(paste("2009-0", 1:6, "-01", sep = ""))
> charvec

[1] "2009-06-01" "2009-05-01" "2009-04-01" "2009-03-01" "2009-02-01"
[6] "2009-01-01"
```

Note the character vector `charvec` is in reverse order.

zoo and xts:

```
> zoo(data, as.Date(charvec))

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
  1.640329  -0.024625   0.716483  -0.535918  -0.049799  -0.128710

> xts(data, as.Date(charvec))

                [,1]
2009-01-01  1.640329
2009-02-01 -0.024625
2009-03-01  0.716483
2009-04-01 -0.535918
2009-05-01 -0.049799
2009-06-01 -0.128710
```

Time series objects of class `zoo` are always forced to be ordered in time. The same holds for `xts` time series objects. It is not possible to create a `zoo` or `xts` object in reverse time order.

timeSeries:

`timeSeries` objects can be created in increasing and decreasing order, and they can even be sampled arbitrarily.

```
> tS <- timeSeries(data, charvec)
> tS

GMT
                TS.1
2009-06-01 -0.128710
2009-05-01 -0.049799
2009-04-01 -0.535918
2009-03-01  0.716483
2009-02-01 -0.024625
2009-01-01  1.640329
```

Reverse the time order

```
> rev(tS)

GMT
                TS.1
2009-01-01  1.640329
2009-02-01 -0.024625
2009-03-01  0.716483
2009-04-01 -0.535918
2009-05-01 -0.049799
2009-06-01 -0.128710
```

Sample the time series in arbitray time order

```
> sample(tS)

GMT
                TS.1
2009-06-01 -0.128710
2009-04-01 -0.535918
2009-03-01  0.716483
2009-05-01 -0.049799
2009-01-01  1.640329
2009-02-01 -0.024625
```

How can I create a time series with overlapping time stamps?

You proceed in the same way as with any other unique time series.

Common Data:

```
> data1 <- c(1:6, 0)
> charvec1 <- c(paste("2009-0", 1:6, "-01", sep = ""), "2009-04-01")
> charvec1

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01" "2009-04-01"
```

```
> data2 <- 0:6
> charvec2 <- c("2009-04-01", paste("2009-0", 1:6, "-01", sep = ""))
> charvec2

[1] "2009-04-01" "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01"
[6] "2009-05-01" "2009-06-01"
```

zoo:

```
> zoo(data1, as.Date(charvec1))

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-04-01 2009-05-01
         1          2          3          4          0          5
2009-06-01
         6

> zoo(data2, as.Date(charvec2))

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-04-01 2009-05-01
         1          2          3          0          4          5
2009-06-01
         6
```

`zoo()` will return a warning message, stating that possibly not all `zoo()` methods will work.

xts:

`xts()` has extended this point in `zoo()`, and overlapping time stamps can be handled.

```
> xts(data1, as.Date(charvec1))

           [,1]
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-04-01    0
2009-05-01    5
2009-06-01    6

> xts(data2, as.Date(charvec2))

           [,1]
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    0
2009-04-01    4
2009-05-01    5
2009-06-01    6
```

timeSeries:

`timeSeries` objects allow for overlapping time stamps

```
> timeSeries(data1, charvec1)

GMT
           TS.1
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
2009-04-01    0

> timeSeries(data2, charvec2)
```

```
GMT
           TS.1
2009-04-01    0
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
```

but the order is the same as that provided by the `charvec` time stamps. This is a useful feature since it can reflect the order in which you received the data records. To sort the series use the function `sort()`

```
> sort(timeSeries(data1, charvec1))

GMT
           TS.1
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-04-01    0
2009-05-01    5
2009-06-01    6

> sort(timeSeries(data2, charvec2))

GMT
           TS.1
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    0
2009-04-01    4
2009-05-01    5
2009-06-01    6
```

Note that you can also sort a `timeSeries` object in decreasing order.

```
> sort(timeSeries(data1, charvec1), decreasing = TRUE)

GMT
           TS.1
2009-06-01    6
2009-05-01    5
```

```
2009-04-01    4
2009-04-01    0
2009-03-01    3
2009-02-01    2
2009-01-01    1
```

If you want to keep the information of the original order, you can save it in the `@recordIDs` slot of the `timeSeries` object. The following example shows you how to keep and retrieve this information.

```
> args(timeSeries)

function (data, charvec, units = NULL, format = NULL, zone = "",
    FinCenter = "", recordIDs = data.frame(), title = NULL, documentation =
NULL,
    ...)
NULL

> data3 <- round(rnorm(7), 2)
> charvec3 <- sample(charvec1)
> tS <- sort(timeSeries(data3, charvec3, recordIDs = data.frame(1:7)))
> tS

GMT
            TS.1 X1.7*
2009-01-01  1.01     5
2009-02-01  0.22     2
2009-03-01  0.27     4
2009-04-01  1.16     1
2009-04-01  0.13     7
2009-05-01 -0.18     3
2009-06-01 -2.29     6
```

Now retrieve the order in the same way as for the information

```
> tS@recordIDs

  X1.7
5    5
2    2
4    4
1    1
7    7
3    3
6    6
```

or you can print it in the form of a direct comparison report.

```
> cbind(series(tS), as.matrix(tS@recordIDs))

            TS.1 X1.7
2009-01-01  1.01    5
2009-02-01  0.22    2
2009-03-01  0.27    4
2009-04-01  1.16    1
2009-04-01  0.13    7
2009-05-01 -0.18    3
2009-06-01 -2.29    6

> data3

[1]  1.16  0.22 -0.18  0.27  1.01 -2.29  0.13
```

How can I handle additional attributes of a time series object?

Let us consider the following (slightly more complex) time series example. We have at given dates, `dateOfOffer`, price offers, `offeredPrice`, provided by different companies, `providerCompany`, which are rated, `ratingOfOffer`. The information about the providers and ratings is saved in a `data.frame` named `priceInfo`.

```
> offeredPrice <- 100 * c(3.4, 3.2, 4, 4, 4.1, 3.5, 2.9)
> offeredPrice

[1] 340 320 400 400 410 350 290

> dateOfOffer <- paste("2009-0", c(1:3, 3, 4:6), "-01", sep = "")
> dateOfOffer

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-03-01" "2009-04-01"
[6] "2009-05-01" "2009-06-01"

> providerCompany <- c(rep("UISA Ltd", times = 3), "HK Company", rep("UISA Ltd
", times = 3))
> providerCompany

[1] "UISA Ltd"   "UISA Ltd"   "UISA Ltd"   "HK Company" "UISA Ltd"
[6] "UISA Ltd"   "UISA Ltd"

> ratingOfOffer <- c(rep("AAA", times = 3), "BBB", rep("AAB", times = 3))
> ratingOfOffer
```

```
[1] "AAA" "AAA" "AAA" "BBB" "AAB" "AAB" "AAB"

> priceInfo <- data.frame(providerCompany, ratingOfOffer)
> priceInfo

  providerCompany ratingOfOffer
1        UISA Ltd           AAA
2        UISA Ltd           AAA
3        UISA Ltd           AAA
4      HK Company           BBB
5        UISA Ltd           AAB
6        UISA Ltd           AAB
7        UISA Ltd           AAB
```

zoo:

We create a `zoo` time series object from dates and prices. The additional information on providers and the ratings can be added by an attribute named `"info"`.

```
> zp <- zoo(offeredPrice, as.Date(dateOfOffer))
> attr(zp, "info") = priceInfo
> zp

2009-01-01 2009-02-01 2009-03-01 2009-03-01 2009-04-01 2009-05-01
       340        320        400        400        410        350
2009-06-01
       290

> zp

2009-01-01 2009-02-01 2009-03-01 2009-03-01 2009-04-01 2009-05-01
       340        320        400        400        410        350
2009-06-01
       290

> attr(zp, "info")

  providerCompany ratingOfOffer
1        UISA Ltd           AAA
2        UISA Ltd           AAA
3        UISA Ltd           AAA
4      HK Company           BBB
5        UISA Ltd           AAB
6        UISA Ltd           AAB
7        UISA Ltd           AAB
```

xts:

For xts we proceed it in the same way.

```
> xp <- xts(offeredPrice, as.Date(dateOfOffer))
> attr(xp, "info") = priceInfo
> xp

           [,1]
2009-01-01  340
2009-02-01  320
2009-03-01  400
2009-03-01  400
2009-04-01  410
2009-05-01  350
2009-06-01  290

> attr(xp, "info")

  providerCompany ratingOfOffer
1       UISA Ltd           AAA
2       UISA Ltd           AAA
3       UISA Ltd           AAA
4     HK Company           BBB
5       UISA Ltd           AAB
6       UISA Ltd           AAB
7       UISA Ltd           AAB
```

timeSeries:

```
> tS <- timeSeries(offeredPrice, dateOfOffer, recordIDs = priceInfo)
> tS

GMT
           TS.1
2009-01-01  340
2009-02-01  320
2009-03-01  400
2009-03-01  400
2009-04-01  410
2009-05-01  350
2009-06-01  290

> tS@recordIDs
```

```
    providerCompany ratingOfOffer
  1       UISA Ltd           AAA
  2       UISA Ltd           AAA
  3       UISA Ltd           AAA
  4     HK Company           BBB
  5       UISA Ltd           AAB
  6       UISA Ltd           AAB
  7       UISA Ltd           AAB
```

For `timeSeries` objects most attributes can be handled as a data frame through the `@recordsIDs` slot.

What happens with attributes when I modify the time series?

Let us consider the example from the previous FAQ. We want to remove offer No. 4 from the time series.

zoo:

```
> zx <- zp[-4]
> zx

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
       340        320        400        410        350        290

> attr(zx, "info")

NULL
```

The attribute has been lost from the `zoo` time series object. A workaround is to check and re-add the subsetted attribute.

```
> zx

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
       340        320        400        410        350        290

> attr(zx, "info") <- priceInfo[-4,]
> zx

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
       340        320        400        410        350        290
```

Now let us inspect what happens with `xts`. We delete the fourth offer by subsetting and print the result again.

```
> xx <- xp[-4]
> xx

           [,1]
2009-01-01  340
2009-02-01  320
2009-03-01  400
2009-04-01  410
2009-05-01  350
2009-06-01  290
```

In contrast to `zoo`, `xts` time series objects keep attributes. However, they are not subsetted.

A workaround is to delete the record in the attribute.

```
> attr(xx, "info") <- attr(xx, "info")[-4, ]
> xx

           [,1]
2009-01-01  340
2009-02-01  320
2009-03-01  400
2009-04-01  410
2009-05-01  350
2009-06-01  290
```

timeSeries:

```
> tX <- tS[-4, ]
> tX

GMT
           TS.1
2009-01-01  340
2009-02-01  320
2009-03-01  400
2009-04-01  410
2009-05-01  350
2009-06-01  290

> tX@recordIDs
```

```
  providerCompany ratingOfOffer
1        UISA Ltd           AAA
2        UISA Ltd           AAA
3        UISA Ltd           AAA
5        UISA Ltd           AAB
6        UISA Ltd           AAB
7        UISA Ltd           AAB
```

Note that `timeSeries` functions also handle attributes that are saved in the `@recordIDs` slot. This should work without further workarounds.

1.5 Binding and Merging of Time Series

Required R package(s):

```
> library(zoo)
> library(xts)
> library(timeSeries)
```

How can I bind two time series objects by row?

To bind a time series row by row use the function `rbind`.

Common Data:

```
> data <- c(1:6)
> charvec1 <- paste("2009-0", 1:6, "-01", sep = "")
> charvec1

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"

> charvec2 <- c(paste("2009-0", 7:9, "-01", sep = ""),
                paste("2009-", 10:12, "-01", sep = ""))
> charvec2

[1] "2009-07-01" "2009-08-01" "2009-09-01" "2009-10-01" "2009-11-01"
[6] "2009-12-01"
```

zoo:

```
> z1 <- zoo(data, as.Date(charvec1))
> z2 <- zoo(data+6, as.Date(charvec2))

> rbind(z1, z2)

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
         1          2          3          4          5          6
2009-07-01 2009-08-01 2009-09-01 2009-10-01 2009-11-01 2009-12-01
         7          8          9         10         11         12
```

```
> rbind(z2, z1)

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
         1          2          3          4          5          6
2009-07-01 2009-08-01 2009-09-01 2009-10-01 2009-11-01 2009-12-01
         7          8          9         10         11         12
```

Note that the bound series do not depend on the order of the arguments in the function `rbind`.

xts:

```
> x1 <- xts(data, as.Date(charvec1))
> x2 <- xts(data+6, as.Date(charvec2))

> rbind(x1, x2)

           [,1]
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
2009-07-01    7
2009-08-01    8
2009-09-01    9
2009-10-01   10
2009-11-01   11
2009-12-01   12

> rbind(x2, x1)

           [,1]
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
2009-07-01    7
2009-08-01    8
2009-09-01    9
2009-10-01   10
2009-11-01   11
2009-12-01   12
```

The result is again an ordered (bound) time series.

timeSeries:

```
> s1 <- timeSeries(data, charvec1)
> s2 <- timeSeries(data+6, charvec2)

> rbind(s1, s2)

GMT
           TS.1_TS.1
2009-01-01         1
2009-02-01         2
2009-03-01         3
2009-04-01         4
2009-05-01         5
2009-06-01         6
2009-07-01         7
2009-08-01         8
2009-09-01         9
2009-10-01        10
2009-11-01        11
2009-12-01        12

> rbind(s2, s1)

GMT
           TS.1_TS.1
2009-07-01         7
2009-08-01         8
2009-09-01         9
2009-10-01        10
2009-11-01        11
2009-12-01        12
2009-01-01         1
2009-02-01         2
2009-03-01         3
2009-04-01         4
2009-05-01         5
2009-06-01         6
```

Note that the result depends on the ordering of the two arguments in the function `rbind()`. It is important to note that this is not a bug but a feature. If you want to obtain the same result, then just sort the series

```
> sort(rbind(s2, s1))

GMT
           TS.1_TS.1
2009-01-01         1
2009-02-01         2
2009-03-01         3
2009-04-01         4
2009-05-01         5
2009-06-01         6
2009-07-01         7
2009-08-01         8
2009-09-01         9
2009-10-01        10
2009-11-01        11
2009-12-01        12
```

Can overlapping time series be bound by row?

Common Data:

```
> data1 <- 1:6
> data2 <- 3:9
> charvec1 <- paste("2009-0", 1:6, "-01", sep = "")
> charvec1

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"

> charvec2 <- paste("2009-0", 3:9, "-01", sep = "")
> charvec2

[1] "2009-03-01" "2009-04-01" "2009-05-01" "2009-06-01" "2009-07-01"
[6] "2009-08-01" "2009-09-01"
```

zoo:

```
> z1 <- zoo(data1, as.Date(charvec1))
> z2 <- zoo(data2, as.Date(charvec2))

> print(try(rbind(z1, z2)))

[1] "Error in rbind(deparse.level, ...) : indexes overlap\n"
attr(,"class")
[1] "try-error"
```

```
> print(try(rbind(z2, z1)))

[1] "Error in rbind(deparse.level, ...) : indexes overlap\n"
attr(,"class")
[1] "try-error"
```

Time series with overlapping indices cannot be bound in zoo.

xts:

```
> x1 <- xts(data1, as.Date(charvec1))
> x2 <- xts(data2, as.Date(charvec2))

> rbind(x1, x2)

           [,1]
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-03-01    3
2009-04-01    4
2009-04-01    4
2009-05-01    5
2009-05-01    5
2009-06-01    6
2009-06-01    6
2009-07-01    7
2009-08-01    8
2009-09-01    9

> rbind(x2, x1)

           [,1]
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-03-01    3
2009-04-01    4
2009-04-01    4
2009-05-01    5
2009-05-01    5
2009-06-01    6
2009-06-01    6
2009-07-01    7
2009-08-01    8
2009-09-01    9
```

Time series with overlapping indices cannot be bound row by row in `xts`, and the original ordering will not be preserved, i.e. the series are sorted.

timeSeries:

```
> s1 <- xts(data1, as.Date(charvec1))
> s2 <- xts(data2, as.Date(charvec2))

> rbind(s1, s2)

           [,1]
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-03-01    3
2009-04-01    4
2009-04-01    4
2009-05-01    5
2009-05-01    5
2009-06-01    6
2009-06-01    6
2009-07-01    7
2009-08-01    8
2009-09-01    9

> rbind(s2, s1)

           [,1]
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-03-01    3
2009-04-01    4
2009-04-01    4
2009-05-01    5
2009-05-01    5
2009-06-01    6
2009-06-01    6
2009-07-01    7
2009-08-01    8
2009-09-01    9
```

Binding of overlapping `timeSeries` objects is fully supported by the `timeSeries` class. The time order of the records is fully preserved.

How can I bind two time series objects by column?

Common Data:

```
> data1 <- 1:6
> data2 <- data1 + 6
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec
```

```
[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> z1 <- zoo(data1, as.Date(charvec))
> z2 <- zoo(data2, as.Date(charvec))

> cbind(z1, z2)
```

```
           z1 z2
2009-01-01  1  7
2009-02-01  2  8
2009-03-01  3  9
2009-04-01  4 10
2009-05-01  5 11
2009-06-01  6 12
```

```
> cbind(z2, z1)
```

```
           z2 z1
2009-01-01  7  1
2009-02-01  8  2
2009-03-01  9  3
2009-04-01 10  4
2009-05-01 11  5
2009-06-01 12  6
```

Note that the ordering of the columns depends on the order in which the two arguments are to the function `cbind`.

xts:

```
> x1 <- xts(data1, as.Date(charvec))
> x2 <- xts(data2, as.Date(charvec))
> cbind(x1, x2)

           ..1 ..2
2009-01-01   1   7
2009-02-01   2   8
2009-03-01   3   9
2009-04-01   4  10
2009-05-01   5  11
2009-06-01   6  12

> cbind(x2, x1)

           ..1 ..2
2009-01-01   7   1
2009-02-01   8   2
2009-03-01   9   3
2009-04-01  10   4
2009-05-01  11   5
2009-06-01  12   6
```

timeSeries:

```
> s1 <- timeSeries(data1, as.Date(charvec))
> s2 <- timeSeries(data2, as.Date(charvec))

> cbind(s1, s2)

GMT
           TS.1.1 TS.1.2
2009-01-01      1      7
2009-02-01      2      8
2009-03-01      3      9
2009-04-01      4     10
2009-05-01      5     11
2009-06-01      6     12

> cbind(s2, s1)

GMT
           TS.1.1 TS.1.2
2009-01-01      7      1
2009-02-01      8      2
2009-03-01      9      3
2009-04-01     10      4
2009-05-01     11      5
2009-06-01     12      6
```

As in the case of `zoo` and `xts` time series objects, the ordering of the columns depends on which of the two arguments is first passed to the function `cbind()`.

Can overlapping time series be bound by column?

Common Data:

```
> data1 <- 1:6
> data2 <- 4:8
> charvec1 <- paste("2009-0", 1:6, "-01", sep = "")
> charvec1

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"

> charvec2 <- paste("2009-0", 4:8, "-01", sep = "")
> charvec2

[1] "2009-04-01" "2009-05-01" "2009-06-01" "2009-07-01" "2009-08-01"
```

zoo:

```
> z1 <- zoo(data1, as.Date(charvec1))
> z2 <- zoo(data2, as.Date(charvec2))

> cbind(z1, z2)

           z1 z2
2009-01-01  1 NA
2009-02-01  2 NA
2009-03-01  3 NA
2009-04-01  4  4
2009-05-01  5  5
2009-06-01  6  6
2009-07-01 NA  7
2009-08-01 NA  8
```

`zoo()` can bind overlapping time series, missing values are substituted by `NA`s.

xts:

```
> x1 <- xts(data1, as.Date(charvec1))
> x2 <- xts(data2, as.Date(charvec2))

> cbind(x1, x2)

           ..1 ..2
2009-01-01   1  NA
2009-02-01   2  NA
2009-03-01   3  NA
2009-04-01   4   4
2009-05-01   5   5
2009-06-01   6   6
2009-07-01  NA   7
2009-08-01  NA   8
```

`xts()` can bind overlapping time series, missing values are substituted by `NA`s. The behaviour is the same as with the column names. Note that the column names of the series are lost.

timeSeries:

```
> s1 <- timeSeries(data1, as.Date(charvec1), units = "s1")
> s2 <- timeSeries(data2, as.Date(charvec2), units = "s2")

> cbind(s1, s2)

GMT
           s1 s2
2009-01-01  1 NA
2009-02-01  2 NA
2009-03-01  3 NA
2009-04-01  4  4
2009-05-01  5  5
2009-06-01  6  6
2009-07-01 NA  7
2009-08-01 NA  8
```

Binding of overlapping `timeSeries` objects is fully supported by the `timeSeries` class. `timeSeries` also substitutes missing values by `NA`s, as is the case for `zoo` and `xts` time series objects.

How can I merge two time series objects and what is the difference to binding time series objects?

The base package of R has a function `merge` which can merge two data frames.

```
> args(merge.data.frame)

function (x, y, by = intersect(names(x), names(y)), by.x = by,
    by.y = by, all = FALSE, all.x = all, all.y = all, sort = TRUE,
    suffixes = c(".x", ".y"), incomparables = NULL, ...)
NULL
```

In the description subsection of the help page we can read: *Merge two data frames by common columns or row names, or do other versions of database "join" operations.* In our sense the merge of two time series object would work in the same way as for data frames.

Note that a natural implementation for time series would mean that merging behaves in a similar manner as for data frames.

What happens when I merge two identical univariate time series objects?

Common Data:

```
> data <- 1:6
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> z <- zoo(data, as.Date(charvec))


> merge(z, z)

           z z.1
2009-01-01 1   1
2009-02-01 2   2
```

```
2009-03-01 3   3
2009-04-01 4   4
2009-05-01 5   5
2009-06-01 6   6
```

`zoo()` returns a bivariate time series object.

xts:

```
> x <- xts(data, as.Date(charvec))

> merge(x, x)

           x x.1
2009-01-01 1   1
2009-02-01 2   2
2009-03-01 3   3
2009-04-01 4   4
2009-05-01 5   5
2009-06-01 6   6
```

`xts()` also returns a bivariate time series object.

timeSeries:

```
> s <- timeSeries(data, charvec)

> merge(s, s)

GMT
           TS.1
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6

> merge(as.data.frame(s), as.data.frame(s))
```

```
  TS.1
1    1
2    2
3    3
4    4
5    5
6    6
```

`timeSeries` objects operate differently, in that they show the same behaviour as we would expect from `data.frame` objects.

How are two different univariate time series merged?

Common Data:

```
> data1 <- 1:6
> data2 <- data1 + 3
> charvec1 <- paste("2009-0", 1:6, "-01", sep = "")
> charvec1

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"

> charvec2 <- paste("2009-0", 4:9, "-01", sep = "")
> charvec2

[1] "2009-04-01" "2009-05-01" "2009-06-01" "2009-07-01" "2009-08-01"
[6] "2009-09-01"
```

zoo:

```
> z1 <- zoo(data1, as.Date(charvec1))
> z2 <- zoo(data2, as.Date(charvec2))

> merge(z1, z2)

           z1 z2
2009-01-01  1 NA
2009-02-01  2 NA
2009-03-01  3 NA
2009-04-01  4  4
2009-05-01  5  5
2009-06-01  6  6
```

```
2009-07-01 NA  7
2009-08-01 NA  8
2009-09-01 NA  9
```

xts:

```
> x1 <- xts(data1, as.Date(charvec1))
> x2 <- xts(data2, as.Date(charvec2))

> merge(x1, x2)

           x1 x2
2009-01-01  1 NA
2009-02-01  2 NA
2009-03-01  3 NA
2009-04-01  4  4
2009-05-01  5  5
2009-06-01  6  6
2009-07-01 NA  7
2009-08-01 NA  8
2009-09-01 NA  9
```

timeSeries:

```
> s1 <- timeSeries(data1, as.Date(charvec1), units = "s1")
> s2 <- timeSeries(data2, as.Date(charvec2), units = "s2")

> merge(s1, s2)

GMT
           s1 s2
2009-01-01  1 NA
2009-02-01  2 NA
2009-03-01  3 NA
2009-04-01  4  4
2009-05-01  5  5
2009-06-01  6  6
2009-07-01 NA  7
2009-08-01 NA  8
2009-09-01 NA  9
```

All three time series classes work in the same way.

What happens if I merge two univariate time series with the same underlying information set?

Common Data:

```
> data1 <- 1:6
> data2 <- data1 + 3
> charvec1 <- paste("2009-0", 1:6, "-01", sep = "")
> charvec1

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"

> charvec2 <- paste("2009-0", 4:9, "-01", sep = "")
> charvec2

[1] "2009-04-01" "2009-05-01" "2009-06-01" "2009-07-01" "2009-08-01"
[6] "2009-09-01"
```

zoo:

We don't know how to do this with univariate zoo time series objects.

xts:

For `xts` time series objects we can set identical column names and then merge the two series:

```
> x1 <- xts(data1, as.Date(charvec1))
> colnames(x1) <- "x"
> z2 <- xts(data2, as.Date(charvec2))
> colnames(x2) <- "x"

> merge(x1, x2)

            x x.1
2009-01-01  1  NA
2009-02-01  2  NA
2009-03-01  3  NA
2009-04-01  4   4
2009-05-01  5   5
2009-06-01  6   6
2009-07-01 NA   7
```

```
2009-08-01 NA   8
2009-09-01 NA   9
```

`xts()` returns a bivariate time series with column names `x` and `x.1`

timeSeries:

```
> s1 <- timeSeries(data1, charvec1, units = "s")
> s2 <- timeSeries(data2, charvec2, units = "s")

> merge(s1, s2)

GMT
           s
2009-01-01 1
2009-02-01 2
2009-03-01 3
2009-04-01 4
2009-05-01 5
2009-06-01 6
2009-07-01 7
2009-08-01 8
2009-09-01 9
```

`timeSeries()` returns a different result. We obtain a univariate series, since both series are from the same information set “s".

Can I merge a time series object with a numeric value?

Common Data:

```
> data <- 1:6
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"

> const <- 3.4
```

zoo:

```
> z <- zoo(data, as.Date(charvec))

> merge(z, const)

           z const
2009-01-01 1   3.4
2009-02-01 2   3.4
2009-03-01 3   3.4
2009-04-01 4   3.4
2009-05-01 5   3.4
2009-06-01 6   3.4
```

xts:

```
> x <- xts(data, as.Date(charvec))

> merge(x, const)

           x const
2009-01-01 1   3.4
2009-02-01 2   3.4
2009-03-01 3   3.4
2009-04-01 4   3.4
2009-05-01 5   3.4
2009-06-01 6   3.4
```

timeSeries:

```
> s <- timeSeries(data, charvec)

> merge(s, const)

GMT
           TS.1   s
2009-01-01    1 3.4
2009-02-01    2 3.4
2009-03-01    3 3.4
2009-04-01    4 3.4
2009-05-01    5 3.4
2009-06-01    6 3.4
```

Can I merge a time series object with a numeric vector?

Common Data:

```
> data <- 1:6
> data

[1] 1 2 3 4 5 6

> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"

> vec <- 3.4 - 1:6
> vec

[1]  2.4  1.4  0.4 -0.6 -1.6 -2.6
```

zoo:

```
> z <- zoo(data, as.Date(charvec))

> merge(z, vec)

           z  vec
2009-01-01 1  2.4
2009-02-01 2  1.4
2009-03-01 3  0.4
2009-04-01 4 -0.6
2009-05-01 5 -1.6
2009-06-01 6 -2.6
```

xts:

```
> x <- xts(data, as.Date(charvec))

> merge(x, vec)

           x  vec
2009-01-01 1  2.4
2009-02-01 2  1.4
2009-03-01 3  0.4
```

```
2009-04-01 4 -0.6
2009-05-01 5 -1.6
2009-06-01 6 -2.6
```

timeSeries:

```
> s <- timeSeries(data, charvec)

> merge(s, vec)

GMT
           TS.1    s
2009-01-01    1  2.4
2009-02-01    2  1.4
2009-03-01    3  0.4
2009-04-01    4 -0.6
2009-05-01    5 -1.6
2009-06-01    6 -2.6
```

Can I merge a time series object with a numeric matrix?

Common Data:

```
> data <- 1:6
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"

> mat <- matrix((1:12)-6, ncol = 2) - 3.4
> mat

     [,1] [,2]
[1,] -8.4 -2.4
[2,] -7.4 -1.4
[3,] -6.4 -0.4
[4,] -5.4  0.6
[5,] -4.4  1.6
[6,] -3.4  2.6
```

zoo:

```
> z <- zoo(data, as.Date(charvec))
```

```
> merge(z, mat)
```

```
           z mat.1 mat.2
2009-01-01 1  -8.4  -2.4
2009-02-01 2  -7.4  -1.4
2009-03-01 3  -6.4  -0.4
2009-04-01 4  -5.4   0.6
2009-05-01 5  -4.4   1.6
2009-06-01 6  -3.4   2.6
```

xts:

```
> x <- xts(data, as.Date(charvec))
```

```
> merge(x, mat)
```

```
           x  mat mat.1
2009-01-01 1 -8.4  -2.4
2009-02-01 2 -7.4  -1.4
2009-03-01 3 -6.4  -0.4
2009-04-01 4 -5.4   0.6
2009-05-01 5 -4.4   1.6
2009-06-01 6 -3.4   2.6
```

timeSeries:

```
> s <- timeSeries(data, charvec)
```

```
> merge(s, mat)
```

```
GMT
           TS.1 mat.1 mat.2
2009-01-01    1  -8.4  -2.4
2009-02-01    2  -7.4  -1.4
2009-03-01    3  -6.4  -0.4
2009-04-01    4  -5.4   0.6
2009-05-01    5  -4.4   1.6
2009-06-01    6  -3.4   2.6
```

1.6 Subsetting Time Series Objects

Required R package(s):

```
> library(zoo)
> library(xts)
> library(timeSeries)
```

How can I subset a vector and a matrix?

A vector is a linear "object", thus we need only one index to subset it.

```
> vec <- rnorm(6)
> vec

[1]  0.18090 -0.19046  0.75169  2.59615  0.13801 -0.57736

> vec[3:4]

[1] 0.75169 2.59615
```

A vector has a dimension NULL, and its length is the number of its elements.

```
> dim(vec)

NULL

> length(vec)

[1] 6
```

A matrix is a rectangular object which requires a pair of indices to subset it, one to choose the columns and another one to choose the rows.

```
> mat <- matrix(rnorm(18), ncol = 3)
> mat

          [,1]       [,2]     [,3]
[1,] -0.344935  0.5432789  1.27887
[2,]  0.020904 -0.0019142  0.29875
[3,]  1.705969  0.1627861  0.35675
[4,]  0.774526 -0.0763961  1.10594
[5,] -1.496030  0.3881734 -0.82036
[6,]  0.071777  0.2315970 -0.90531
```

```
> mat[3:4,]

        [,1]      [,2]    [,3]
[1,] 1.70597  0.162786 0.35675
[2,] 0.77453 -0.076396 1.10594

> mat[, 2:3]

           [,1]     [,2]
[1,]  0.5432789  1.27887
[2,] -0.0019142  0.29875
[3,]  0.1627861  0.35675
[4,] -0.0763961  1.10594
[5,]  0.3881734 -0.82036
[6,]  0.2315970 -0.90531

> mat[3:4, 2:3]

          [,1]    [,2]
[1,]  0.162786 0.35675
[2,] -0.076396 1.10594
```

For a matrix we have

```
> dim(mat)

[1] 6 3

> length(mat)

[1] 18
```

Thus the function `dim` returns the number of rows and the number of columns of the matrix and the function `length` the total number of elements of the matrix.

What happens when we subset a single column or a single row of matrix?

```
> mat[3,]

[1] 1.70597 0.16279 0.35675

> mat[, 2]

[1]  0.5432789 -0.0019142  0.1627861 -0.0763961  0.3881734  0.2315970
```

Then the rectangular object becomes linear, the result is a vector. To prevent this we have to take care that we do not drop the redundant extent information.

```
> rowmat <- mat[3, ,drop = FALSE]
> colmat <- mat[, 2, drop = FALSE]
```

When I am subsetting a time series, does it behave like subsetting a vector and a matrix?

yes and no, it depends what time series class we use.

Common Data:

```
> data1 <- rnorm(1:6)
> data2 <- matrix(rnorm(18), ncol = 3)
> colnames(data2) <- LETTERS[1:3]
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec
```

```
[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

A univariate time series

```
> z <- zoo(data1, as.Date(charvec))
> z[3:4]
```

```
2009-03-01 2009-04-01
  -0.76079   -0.20898
```

A multivariate time series

```
> Z <- zoo(data2, as.Date(charvec))
> Z[3:4,]
```

```
                  A       B       C
2009-03-01 -0.52654 -1.4912 0.57747
2009-04-01 -0.31823  2.6105 0.25730
```

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

```
                  B         C
2009-01-01  0.40434 -0.688973
2009-02-01  0.34507 -1.114796
2009-03-01 -1.49123  0.577469
2009-04-01  2.61052  0.257301
2009-05-01  2.67869 -0.045377
2009-06-01 -0.99734  1.164478
```

```
> Z[3:4, 2:3]

                B       C
2009-03-01 -1.4912 0.57747
2009-04-01  2.6105 0.25730
```

```
> Z[, 2:3]

                 B         C
2009-01-01  0.40434 -0.688973
2009-02-01  0.34507 -1.114796
2009-03-01 -1.49123  0.577469
2009-04-01  2.61052  0.257301
2009-05-01  2.67869 -0.045377
2009-06-01 -0.99734  1.164478
```

```
> Z[3:4, 2:3]

                B       C
2009-03-01 -1.4912 0.57747
2009-04-01  2.6105 0.25730
```

We keep in mind, a univariate `zoo` time series object behaves like a vector, and a multivariate `zoo` time series object behaves like a matrix. Extracting single rows or columns drops an index.

xts:

A univariate time series

```
> x <- xts(data1, as.Date(charvec))
> x[3:4]

              [,1]
2009-03-01 -0.76079
2009-04-01 -0.20898
```

A multivariate time series

```
> X <- xts(data2, as.Date(charvec))
> X[3:4,]

                 A       B       C
2009-03-01 -0.52654 -1.4912 0.57747
2009-04-01 -0.31823  2.6105 0.25730
```

```
> X[, 2:3]

                  B         C
2009-01-01  0.40434 -0.688973
2009-02-01  0.34507 -1.114796
2009-03-01 -1.49123  0.577469
2009-04-01  2.61052  0.257301
2009-05-01  2.67869 -0.045377
2009-06-01 -0.99734  1.164478

> X[3:4, 2:3]

                 B       C
2009-03-01 -1.4912 0.57747
2009-04-01  2.6105 0.25730
```

```
> X[, 2:3]

                  B         C
2009-01-01  0.40434 -0.688973
2009-02-01  0.34507 -1.114796
2009-03-01 -1.49123  0.577469
2009-04-01  2.61052  0.257301
2009-05-01  2.67869 -0.045377
2009-06-01 -0.99734  1.164478

> X[3:4, 2:3]

                 B       C
2009-03-01 -1.4912 0.57747
2009-04-01  2.6105 0.25730
```

Time series objects in `xts` are always consider as rectangular objects even in the univariate case. This uniqueness is introduced to make life easier for subsetting financial time series

timeSeries:

A univariate time series

```
> s <- timeSeries(data1, charvec)
> s[3:4]

[1] -0.76079 -0.20898

> s[3:4, ]
```

```
GMT
               TS.1
2009-03-01 -0.76079
2009-04-01 -0.20898
```

A multivariate time series

```
> S <- timeSeries(data2, charvec)
> S[3:4,]

GMT
                  A       B       C
2009-03-01 -0.52654 -1.4912 0.57747
2009-04-01 -0.31823  2.6105 0.25730

> S[, 2:3]

GMT
                  B         C
2009-01-01  0.40434 -0.688973
2009-02-01  0.34507 -1.114796
2009-03-01 -1.49123  0.577469
2009-04-01  2.61052  0.257301
2009-05-01  2.67869 -0.045377
2009-06-01 -0.99734  1.164478

> S[3:4, 2:3]

GMT
                 B       C
2009-03-01 -1.4912 0.57747
2009-04-01  2.6105 0.25730
```

Note that a `timeSeries` object has always to be subsetted by a pair of indices. This a `timeSeries` feature not a bug.)

```
> S[, 2:3]

GMT
                  B         C
2009-01-01  0.40434 -0.688973
2009-02-01  0.34507 -1.114796
2009-03-01 -1.49123  0.577469
2009-04-01  2.61052  0.257301
2009-05-01  2.67869 -0.045377
2009-06-01 -0.99734  1.164478

> S[3:4, 2:3]
```

```
GMT
                 B       C
2009-03-01 -1.4912 0.57747
2009-04-01  2.6105 0.25730
```

`timeSeries` objects are like `xts` objects rectangular objects. The behave in the same way.

Can I subset a multivariate time series by column names?

Common Data:

```
> data <- matrix(rnorm(18), ncol = 3)
> colnames(data) <- LETTERS[1:3]
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> Z <- zoo(data, as.Date(charvec))
> Z[, "A"]

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
   1.32385   -1.58880    1.22988    0.22244   -1.07909    0.40826
```

xts:

```
> X <- xts(data, as.Date(charvec))
> X[, "A"]

                  A
2009-01-01  1.32385
2009-02-01 -1.58880
2009-03-01  1.22988
2009-04-01  0.22244
2009-05-01 -1.07909
2009-06-01  0.40826
```

timeSeries:

```
> S <- timeSeries(data, charvec)
> S[, "A"]
```

```
GMT
                  A
2009-01-01  1.32385
2009-02-01 -1.58880
2009-03-01  1.22988
2009-04-01  0.22244
2009-05-01 -1.07909
2009-06-01  0.40826
```

Can I subset a multivariate time series by column using the $operator?

Common Data:

```
> data <- matrix(rnorm(18), ncol = 3)
> colnames(data) <- LETTERS[1:3]
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec
```

```
[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> Z <- zoo(data, as.Date(charvec))
> Z$B
```

```
2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
   1.84425   -1.08238   -1.37841   -1.13910    0.13289    0.49833
```

xts:

```
> X <- xts(data, as.Date(charvec))
> X$B
```

```
                  B
2009-01-01  1.84425
2009-02-01 -1.08238
2009-03-01 -1.37841
2009-04-01 -1.13910
2009-05-01  0.13289
2009-06-01  0.49833
```

timeSeries:

```
> S <- timeSeries(data, charvec)
> S$B
```

```
[1]  1.84425 -1.08238 -1.37841 -1.13910  0.13289  0.49833
```

Can I subset a multivariate time series by character time stamps?

Common Data:

```
> data <- matrix(rnorm(18), ncol = 3)
> colnames(data) <- LETTERS[1:3]
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec
```

```
[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> Z <- zoo(data, as.Date(charvec))
> charvec[3:4]
```

```
[1] "2009-03-01" "2009-04-01"
```

```
> Z[charvec[3:4], ]
```

```
     A B C
```

xts:

```
> X <- xts(data, as.Date(charvec))
> charvec[3:4]

[1] "2009-03-01" "2009-04-01"

> X[charvec[3:4], ]

                  A       B       C
2009-03-01 1.280866 0.79083 1.01366
2009-04-01 0.079842 0.93683 0.41269
```

timeSeries:

```
> S <- timeSeries(data, charvec)
> charvec[3:4]

[1] "2009-03-01" "2009-04-01"

> S[charvec[3:4], ]

GMT
                  A       B       C
2009-03-01 1.280866 0.79083 1.01366
2009-04-01 0.079842 0.93683 0.41269
```

Can I subset a multivariate time series by its intrinsic time objects?

Common Data:

```
> data <- matrix(rnorm(18), ncol = 3)
> colnames(data) <- LETTERS[1:3]
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> Z <- zoo(data, as.Date(charvec))
> timeStamp <- index(Z)[3:4]
> timeStamp

[1] "2009-03-01" "2009-04-01"

> Z[timeStamp, ]

                 A       B        C
2009-03-01 2.51329 0.30254 -2.21588
2009-04-01 0.14760 0.24991  0.39797
```

xts:

```
> X <- xts(data, as.Date(charvec))
> timeStamp <- index(X)[3:4]
> timeStamp

[1] "2009-03-01" "2009-04-01"

> X[timeStamp, ]

                 A       B        C
2009-03-01 2.51329 0.30254 -2.21588
2009-04-01 0.14760 0.24991  0.39797
```

timeSeries:

```
> S <- timeSeries(data, charvec)
> timeStamp <- time(S)[3:4]
> timeStamp

GMT
[1] [2009-03-01] [2009-04-01]

> S[timeStamp, ]

GMT
                 A       B        C
2009-03-01 2.51329 0.30254 -2.21588
2009-04-01 0.14760 0.24991  0.39797
```

Can I subset a multivariate time series by logical predicates?

Common Data:

```
> data <- matrix(rnorm(18), ncol = 3)
> colnames(data) <- LETTERS[1:3]
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec
```

```
[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> Z <- zoo(data, as.Date(charvec))
> timeStamp <- index(Z) > index(Z)[3]
> timeStamp
```

```
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

```
> Z[timeStamp, ]
```

```
                   A        B       C
2009-04-01  0.065068 -0.11104 -1.2661
2009-05-01  0.069340 -2.12725  1.0568
2009-06-01 -0.489019 -0.54277  3.5996
```

xts:

```
> X <- xts(data, as.Date(charvec))
> timeStamp <- index(X) > index(X)[3]
> timeStamp
```

```
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

```
> X[timeStamp, ]
```

```
                   A        B       C
2009-04-01  0.065068 -0.11104 -1.2661
2009-05-01  0.069340 -2.12725  1.0568
2009-06-01 -0.489019 -0.54277  3.5996
```

timeSeries:

```
> S <- timeSeries(data, charvec)
> timeStamp <- time(S) > time(S)[3]
> timeStamp

[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE

> S[timeStamp, ]

GMT
                   A        B       C
2009-04-01  0.065068 -0.11104 -1.2661
2009-05-01  0.069340 -2.12725  1.0568
2009-06-01 -0.489019 -0.54277  3.5996
```

How to extract the start and end date of a series?

Common Data:

```
> data <- 1:6
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> z <- zoo(data, as.Date(charvec))
> c(start(x), end(x))

[1] "2009-01-01" "2009-06-01"
```

xts:

```
> x <- xts(data, as.Date(charvec))
> c(start(x), end(x))

[1] "2009-01-01" "2009-06-01"
```

timeSeries:

```
> s <- timeSeries(data, charvec)
> c(start(s), end(s))

GMT
[1] [2009-01-01] [2009-06-01]
```

Since `timeSeries` also support non-sorted time stamps keep in mind, that the first and last entry of the series ar not necessarily the start and end values, see

```
> s <- timeSeries(data, sample(charvec))
> c(start(s), end(s))

GMT
[1] [2009-01-01] [2009-06-01]

> c(time(s[1, ]), time(s[6, ]))

GMT
[1] [2009-03-01] [2009-05-01]
```

1.7 Group Generics for Time Series Objects

Required R package(s):

```
> library(zoo)
> library(xts)
> library(timeSeries)
```

First inspect the help pages for `groupGeneric()` in R's `base` package and the help page for `groupGeneric()` in R's `methods` package

S3 Group Generic Functions:

Group generic methods can be defined for four pre-specified groups of functions, `Math`, `Ops`, `Summary` and `Complex`.

- `Math(x, ...)`
- `Ops(e1, e2)`
- `Complex(z)`
- `Summary(..., na.rm = FALSE)`

S4 Group Generic Functions:

Group generic methods can be defined for four pre-specified groups of functions, texttttArith, texttttCompare, `Ops`, `Logic`, `Math`, `Math2`, `Sumary`, and `Complex`.

- `Arith(e1, e2)`
- `Compare(e1, e2)`
- `Ops(e1, e2)`
- `Logic(e1, e2)`
- `Math(x)`
- `Math2(x, digits)`
- `Summary(x, ..., na.rm = FALSE)`
- `Complex(z)`

In the following we test some selected functions which are relevant for use in financial time series analysis. We test them on multivariate time series objects

Common Data:

```
> data <- matrix(runif(18), ncol = 3)
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec
```

```
[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> Z <- zoo(data, as.Date(charvec))
> colnames(Z) <- paste("Z", 1:3, sep = ".")
> Z
```

```
               Z.1      Z.2      Z.3
2009-01-01 0.99161 0.020039 0.661693
2009-02-01 0.42011 0.319320 0.326017
2009-03-01 0.37923 0.337101 0.060902
2009-04-01 0.67986 0.956925 0.799286
2009-05-01 0.41569 0.688525 0.188548
2009-06-01 0.58708 0.780628 0.250198
```

xts:

```
> X <- xts(data, as.Date(charvec))
> colnames(X) <- paste("X", 1:3, sep = ".")
> X
```

```
               X.1      X.2      X.3
2009-01-01 0.99161 0.020039 0.661693
2009-02-01 0.42011 0.319320 0.326017
2009-03-01 0.37923 0.337101 0.060902
2009-04-01 0.67986 0.956925 0.799286
2009-05-01 0.41569 0.688525 0.188548
2009-06-01 0.58708 0.780628 0.250198
```

timeSeries:

```
> S <- timeSeries(data, charvec)
> colnames(S) <- paste("S", 1:3, sep = ".")
> S
```

```
GMT
                S.1      S.2      S.3
2009-01-01 0.99161 0.020039 0.661693
2009-02-01 0.42011 0.319320 0.326017
2009-03-01 0.37923 0.337101 0.060902
2009-04-01 0.67986 0.956925 0.799286
2009-05-01 0.41569 0.688525 0.188548
2009-06-01 0.58708 0.780628 0.250198
```

And we test them on univariate time series objects

zoo:

```
> z <- Z[, 1]
> z

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
   0.99161    0.42011    0.37923    0.67986    0.41569    0.58708
```

xts:

```
> x <- X[, 1]
> x

                X.1
2009-01-01 0.99161
2009-02-01 0.42011
2009-03-01 0.37923
2009-04-01 0.67986
2009-05-01 0.41569
2009-06-01 0.58708
```

timeSeries:

```
> s <- S[, 1]
> s

GMT
                S.1
2009-01-01 0.99161
```

```
2009-02-01 0.42011
2009-03-01 0.37923
2009-04-01 0.67986
2009-05-01 0.41569
2009-06-01 0.58708
```

Which "Arithmetic" operations are supported by time series objects?

Arith:

```
"+", "-", "*", "^", "%%", "%/%", "/"
```

e.g.

zoo:

```
> Z[, 1] + Z[, 2]
```

```
2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
   1.01165    0.73943    0.71633    1.63678    1.10422    1.36771
```

xts:

```
> X[, 1] + X[, 2]
```

```
               X.1
2009-01-01 1.01165
2009-02-01 0.73943
2009-03-01 0.71633
2009-04-01 1.63678
2009-05-01 1.10422
2009-06-01 1.36771
```

timeSeries:

```
> S[, 1] + S[, 2]
```

```
GMT
                S.1
2009-01-01 1.01165
2009-02-01 0.73943
2009-03-01 0.71633
2009-04-01 1.63678
2009-05-01 1.10422
2009-06-01 1.36771
```

Which "Compare" operations are supported by time series objects?

Compare:

```
"==", ">", "<", "!=", "<=", ">="
```

e.g.

zoo:

```
> Z[, 1] > Z[, 2]

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
      TRUE       TRUE       TRUE      FALSE      FALSE      FALSE
```

xts:

```
> X[, 1] > X[, 2]

             X.1
2009-01-01  TRUE
2009-02-01  TRUE
2009-03-01  TRUE
2009-04-01 FALSE
2009-05-01 FALSE
2009-06-01 FALSE
```

timeSeries:

```
> S[, 1] > S[, 2]

GMT
             S.1
2009-01-01  TRUE
2009-02-01  TRUE
2009-03-01  TRUE
2009-04-01 FALSE
2009-05-01 FALSE
2009-06-01 FALSE
```

Which "Logic" operations are supported by time series objects?

Logic:

```
"&", "|"
```

e.g.

zoo:

```
> (Z[, 1] > Z[, 2]) & (Z[, 2] < Z[, 3])

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
      TRUE       TRUE      FALSE      FALSE      FALSE      FALSE
```

xts:

```
> (X[, 1] > X[, 2]) & (X[, 2] < X[, 3])

             X.1
2009-01-01  TRUE
2009-02-01  TRUE
2009-03-01 FALSE
2009-04-01 FALSE
2009-05-01 FALSE
2009-06-01 FALSE
```

timeSeries:

```
> (S[, 1] > S[, 2]) & (S[, 2] < S[, 3])

GMT
             S.1
2009-01-01  TRUE
2009-02-01  TRUE
2009-03-01 FALSE
2009-04-01 FALSE
2009-05-01 FALSE
2009-06-01 FALSE
```

Which "Ops" operations are supported by time series objects?

Ops:

```
"Arith", "Compare", "Logic"
```

Which "Math" operations are supported by time series objects?

Math:

```
"abs", "sign", "sqrt", "ceiling", "floor", "trunc", "cummax", "cummin", "cumprod",
"cumsum", "log", "log10", "log2", "log1p", "acos", "acosh", "asin", "asinh",
"atan", "atanh", "exp", "expm1", "cos", "cosh", "sin", "sinh", "tan", "tanh",
"gamma", "lgamma", "digamma", "trigamma"
```

e.g.

zoo:

```
> log(abs(Z))

                 Z.1      Z.2      Z.3
2009-01-01 -0.0084213 -3.91009 -0.41295
2009-02-01 -0.8672350 -1.14156 -1.12080
2009-03-01 -0.9696104 -1.08737 -2.79850
2009-04-01 -0.3858696 -0.04403 -0.22404
```

```
2009-05-01 -0.8778063 -0.37320 -1.66840
2009-06-01 -0.5325981 -0.24766 -1.38550
```

xts:

```
> log(abs(X))

                  X.1      X.2      X.3
2009-01-01 -0.0084213 -3.91009 -0.41295
2009-02-01 -0.8672350 -1.14156 -1.12080
2009-03-01 -0.9696104 -1.08737 -2.79850
2009-04-01 -0.3858696 -0.04403 -0.22404
2009-05-01 -0.8778063 -0.37320 -1.66840
2009-06-01 -0.5325981 -0.24766 -1.38550
```

timeSeries:

```
> log(abs(S))

GMT
                  S.1      S.2      S.3
2009-01-01 -0.0084213 -3.91009 -0.41295
2009-02-01 -0.8672350 -1.14156 -1.12080
2009-03-01 -0.9696104 -1.08737 -2.79850
2009-04-01 -0.3858696 -0.04403 -0.22404
2009-05-01 -0.8778063 -0.37320 -1.66840
2009-06-01 -0.5325981 -0.24766 -1.38550
```

Which "Math2" operations are supported by time series objects?

Math2:

```
"round", "signif"
```

e.g.

zoo:

```
> round(Z, 2)

            Z.1  Z.2  Z.3
2009-01-01 0.99 0.02 0.66
2009-02-01 0.42 0.32 0.33
2009-03-01 0.38 0.34 0.06
2009-04-01 0.68 0.96 0.80
2009-05-01 0.42 0.69 0.19
2009-06-01 0.59 0.78 0.25

> signif(Z, 2)

            Z.1  Z.2   Z.3
2009-01-01 0.99 0.02 0.660
2009-02-01 0.42 0.32 0.330
2009-03-01 0.38 0.34 0.061
2009-04-01 0.68 0.96 0.800
2009-05-01 0.42 0.69 0.190
2009-06-01 0.59 0.78 0.250
```

xts:

```
> round(X, 2)

            X.1  X.2  X.3
2009-01-01 0.99 0.02 0.66
2009-02-01 0.42 0.32 0.33
2009-03-01 0.38 0.34 0.06
2009-04-01 0.68 0.96 0.80
2009-05-01 0.42 0.69 0.19
2009-06-01 0.59 0.78 0.25

> signif(X, 2)

            X.1  X.2   X.3
2009-01-01 0.99 0.02 0.660
2009-02-01 0.42 0.32 0.330
2009-03-01 0.38 0.34 0.061
2009-04-01 0.68 0.96 0.800
2009-05-01 0.42 0.69 0.190
2009-06-01 0.59 0.78 0.250
```

timeSeries:

```
> round(S, 2)

GMT
            S.1  S.2  S.3
2009-01-01 0.99 0.02 0.66
2009-02-01 0.42 0.32 0.33
2009-03-01 0.38 0.34 0.06
2009-04-01 0.68 0.96 0.80
2009-05-01 0.42 0.69 0.19
2009-06-01 0.59 0.78 0.25

> signif(S, 2)

GMT
            S.1  S.2   S.3
2009-01-01 0.99 0.02 0.660
2009-02-01 0.42 0.32 0.330
2009-03-01 0.38 0.34 0.061
2009-04-01 0.68 0.96 0.800
2009-05-01 0.42 0.69 0.190
2009-06-01 0.59 0.78 0.250
```

Which "Summary" operations are supported by time series objects?

Summary:

```
"max", "min", "range", "prod", "sum", "any", "all"
```

e.g.

zoo:

```
> max(z)

[1] 0.99161

> min(z)

[1] 0.37923

> range(z)
```

```
[1] 0.37923 0.99161

> prod(z)

[1] 0.026212

> sum(z)

[1] 3.4736
```

xts:

```
> max(x)

[1] 0.99161

> min(x)

[1] 0.37923

> range(x)

[1] 0.37923 0.99161

> prod(x)

[1] 0.026212

> sum(x)

[1] 3.4736
```

timeSeries:

```
> max(s)

[1] 0.99161

> min(s)

[1] 0.37923

> range(s)

[1] 0.37923 0.99161

> prod(s)

[1] 0.026212

> sum(s)

[1] 3.4736
```

Which "Complex" operations are supported by time series objects?

Complex:

```
"Arg", "Conj", "Im", "Mod", "Re"
```

e.g. with

```
> u <- c(0, 2i)
> u

[1] 0+0i 0+2i
```

zoo:

```
> Arg(z + u)

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
    0.0000     1.3638     0.0000     1.2431     0.0000     1.2853

> Conj(z + u)

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
0.99161+0i 0.42011-2i 0.37923+0i 0.67986-2i 0.41569+0i 0.58708-2i

> Im(z + u)

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
         0          2          0          2          0          2

> Mod(z + u)

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
   0.99161    2.04365    0.37923    2.11239    0.41569    2.08438

> Re(z + u)

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
   0.99161    0.42011    0.37923    0.67986    0.41569    0.58708
```

xts:

```
> Arg(x + u)
```

```
              X.1
2009-01-01 0.0000
2009-02-01 1.3638
2009-03-01 0.0000
2009-04-01 1.2431
2009-05-01 0.0000
2009-06-01 1.2853
```

```
> Conj(x + u)
```

```
                  X.1
2009-01-01 0.99161+0i
2009-02-01 0.42011-2i
2009-03-01 0.37923+0i
2009-04-01 0.67986-2i
2009-05-01 0.41569+0i
2009-06-01 0.58708-2i
```

```
> Im(x + u)
```

```
           X.1
2009-01-01   0
2009-02-01   2
2009-03-01   0
2009-04-01   2
2009-05-01   0
2009-06-01   2
```

```
> Mod(x + u)
```

```
               X.1
2009-01-01 0.99161
2009-02-01 2.04365
2009-03-01 0.37923
2009-04-01 2.11239
2009-05-01 0.41569
2009-06-01 2.08438
```

```
> Re(x + u)
```

```
               X.1
2009-01-01 0.99161
2009-02-01 0.42011
2009-03-01 0.37923
2009-04-01 0.67986
2009-05-01 0.41569
2009-06-01 0.58708
```

timeSeries:

```
> Arg(s + u)

GMT
                  S.1
2009-01-01 0.0000
2009-02-01 1.3638
2009-03-01 0.0000
2009-04-01 1.2431
2009-05-01 0.0000
2009-06-01 1.2853

> Conj(s + u)

GMT
                       S.1
2009-01-01 0.99161+0i
2009-02-01 0.42011-2i
2009-03-01 0.37923+0i
2009-04-01 0.67986-2i
2009-05-01 0.41569+0i
2009-06-01 0.58708-2i

> Im(s + u)

GMT
              S.1
2009-01-01   0
2009-02-01   2
2009-03-01   0
2009-04-01   2
2009-05-01   0
2009-06-01   2

> Mod(s + u)

GMT
                   S.1
2009-01-01 0.99161
2009-02-01 2.04365
2009-03-01 0.37923
2009-04-01 2.11239
2009-05-01 0.41569
2009-06-01 2.08438

> Re(s + u)

GMT
                   S.1
```

```
2009-01-01 0.99161
2009-02-01 0.42011
2009-03-01 0.37923
2009-04-01 0.67986
2009-05-01 0.41569
2009-06-01 0.58708
```

1.8 Missing Data Handling

Required R package(s):

```
> library(zoo)
> library(xts)
> library(timeSeries)
```

How can I omit missing values in a univariate time series?

Note that *omit* can interpreted in several ways. By default we mean that we remove the record wit `NA`s from the data set. Later we will extend this meaning by replacing and/or interpolating missing values in a time series object. Now let us remove `NA`s from a time series.

Common Data:

```
> data <- rnorm(9)
> data[c(1, 7)] <- NA
> data

[1]       NA  0.90058  0.32398  0.96776  0.86024  0.41014       NA -0.14042
[9]  2.58655

> charvec <- paste("2009-0", 1:9, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01" "2009-07-01" "2009-08-01" "2009-09-01"
```

zoo:

```
> z <- zoo(data, as.Date(charvec))
> na.omit(z)

2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01 2009-08-01
   0.90058    0.32398    0.96776    0.86024    0.41014   -0.14042
2009-09-01
   2.58655
```

xts:

```
> x <- xts(data, as.Date(charvec))
> na.omit(x)
```

```
               [,1]
2009-02-01  0.90058
2009-03-01  0.32398
2009-04-01  0.96776
2009-05-01  0.86024
2009-06-01  0.41014
2009-08-01 -0.14042
2009-09-01  2.58655
```

timeSeries:

```
> s <- timeSeries(data, charvec)
> na.omit(s)
```

```
GMT
               TS.1
2009-02-01  0.90058
2009-03-01  0.32398
2009-04-01  0.96776
2009-05-01  0.86024
2009-06-01  0.41014
2009-08-01 -0.14042
2009-09-01  2.58655
```

How can I omit missing values in a multivariate time series?

Common Data:

```
> data <- matrix(rnorm(7*3), ncol = 3)
> data[1,1] <- NA
> data[3,1:2] <- NA
> data[4,2] <- NA
> data
```

```
         [,1]     [,2]      [,3]
[1,]       NA -0.96967 -0.588760
[2,] -0.09214 -1.08348  0.298535
[3,]       NA       NA -0.047918
```

```
[4,]  1.03234       NA -1.100902
[5,] -0.77068 -0.30157  0.525291
[6,] -0.93308 -0.65321 -0.911978
[7,]  0.27576  0.82323  1.422525
```

```
> charvec <- paste("2009-0", 1:7, "-01", sep = "")
> charvec
```

```
[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01" "2009-07-01"
```

zoo:

```
> Z <- zoo(data, as.Date(charvec))
> na.omit(Z)
```

```
2009-02-01 -0.09214 -1.08348  0.29853
2009-05-01 -0.77068 -0.30157  0.52529
2009-06-01 -0.93308 -0.65321 -0.91198
2009-07-01  0.27576  0.82323  1.42253
attr(,"na.action")
[1] 1 3 4
attr(,"class")
[1] "omit"
```

xts:

```
> X <- xts(data, as.Date(charvec))
> na.omit(X)
```

```
               [,1]     [,2]     [,3]
2009-02-01 -0.09214 -1.08348  0.29853
2009-05-01 -0.77068 -0.30157  0.52529
2009-06-01 -0.93308 -0.65321 -0.91198
2009-07-01  0.27576  0.82323  1.42253
```

timeSeries:

```
> s <- timeSeries(data, charvec)
> na.omit(s)
```

```
GMT
                TS.1     TS.2     TS.3
2009-02-01 -0.09214 -1.08348  0.29853
2009-05-01 -0.77068 -0.30157  0.52529
2009-06-01 -0.93308 -0.65321 -0.91198
2009-07-01  0.27576  0.82323  1.42253
```

How can I replace missing values in a univariate time series for example by zeros, the mean or the median object?

Common Data:

```
> data <- rnorm(9)
> data[c(1, 7)] <- NA
> charvec <- paste("2009-0", 1:9, "-01", sep = "")
> charvec
```

```
[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01" "2009-07-01" "2009-08-01" "2009-09-01"
```

It may be of interest to replace missing values in a financial return series by a constant value, e.g. zero, the mean, or the median.

zoo:

Replace for example missing values in a series of financial returns by zero values.

```
> z <- zoo(data, as.Date(charvec))
> z[is.na(z)] <- 0
> z
```

```
2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
   0.00000   -0.23273    0.29271    1.09026   -0.74019   -1.23671
2009-07-01 2009-08-01 2009-09-01
   0.00000    0.44108   -1.05783
```

Or Replace for example missing values in a series of financial returns by their sample mean.

```
> z <- zoo(data, as.Date(charvec))
> z[is.na(z)] <- mean(z, na.rm = TRUE)
> z
```

```
2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
  -0.20620   -0.23273    0.29271    1.09026   -0.74019   -1.23671
2009-07-01 2009-08-01 2009-09-01
  -0.20620    0.44108   -1.05783
```

Or Replace for example missing values in a series of financial returns by a robust estimate of the mean, using the median.

```
> z <- zoo(data, as.Date(charvec))
> z[is.na(z)] <- median(z, na.rm = TRUE)
> z
```

```
2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
  -0.74019   -0.23273    0.29271    1.09026   -0.74019   -1.23671
2009-07-01 2009-08-01 2009-09-01
  -0.74019    0.44108   -1.05783
```

xts:

```
> x <- xts(data, as.Date(charvec))
> x[is.na(x)] <- 0
> x
```

```
               [,1]
2009-01-01  0.00000
2009-02-01 -0.23273
2009-03-01  0.29271
2009-04-01  1.09026
2009-05-01 -0.74019
2009-06-01 -1.23671
2009-07-01  0.00000
2009-08-01  0.44108
2009-09-01 -1.05783
```

```
> x <- xts(data, as.Date(charvec))
> x[is.na(x)] <- mean(x, na.rm = TRUE)
> x
```

```
               [,1]
2009-01-01 -0.20620
2009-02-01 -0.23273
2009-03-01  0.29271
2009-04-01  1.09026
2009-05-01 -0.74019
```

```
2009-06-01 -1.23671
2009-07-01 -0.20620
2009-08-01  0.44108
2009-09-01 -1.05783

> x <- xts(data, as.Date(charvec))
> x[is.na(x)] <- median(x, na.rm = TRUE)
> x

               [,1]
2009-01-01 -0.74019
2009-02-01 -0.23273
2009-03-01  0.29271
2009-04-01  1.09026
2009-05-01 -0.74019
2009-06-01 -1.23671
2009-07-01 -0.74019
2009-08-01  0.44108
2009-09-01 -1.05783
```

timeSeries:

```
> s <- timeSeries(data, charvec)
> s[is.na(s)] <- 0
> s

GMT
               TS.1
2009-01-01  0.00000
2009-02-01 -0.23273
2009-03-01  0.29271
2009-04-01  1.09026
2009-05-01 -0.74019
2009-06-01 -1.23671
2009-07-01  0.00000
2009-08-01  0.44108
2009-09-01 -1.05783

> s <- timeSeries(data, charvec)
> s[is.na(s)] <- mean(s, na.rm = TRUE)
> s

GMT
               TS.1
2009-01-01 -0.20620
```

```
2009-02-01 -0.23273
2009-03-01  0.29271
2009-04-01  1.09026
2009-05-01 -0.74019
2009-06-01 -1.23671
2009-07-01 -0.20620
2009-08-01  0.44108
2009-09-01 -1.05783
```

```
> s <- timeSeries(data, charvec)
> s[is.na(s)] <- median(s, na.rm = TRUE)
> s
```

```
GMT
               TS.1
2009-01-01 -0.23273
2009-02-01 -0.23273
2009-03-01  0.29271
2009-04-01  1.09026
2009-05-01 -0.74019
2009-06-01 -1.23671
2009-07-01 -0.23273
2009-08-01  0.44108
2009-09-01 -1.05783
```

How can I replace missing values in a multivariate time series object?

It may be of interest to replace missing values in a series of financial returns by a constant value, e.g. zero, the (column) mean, or the robust (column) median.

Common Data:

```
> data <- matrix(rnorm(7*3), ncol = 3)
> data[1,1] <- NA
> data[3,1:2] <- NA
> data[4,2] <- NA
> data
```

```
          [,1]    [,2]     [,3]
[1,]        NA 0.96519  0.70410
```

```
[2,] -0.629991 1.44003  0.55432
[3,]        NA      NA -0.35576
[4,] -1.349331      NA  0.79447
[5,] -0.359353 0.79603  1.03604
[6,] -0.760570 0.89683 -0.71239
[7,] -0.029310 0.22897 -0.74984
```

```
> charvec <- paste("2009-0", 1:7, "-01", sep = "")
> charvec
```

```
[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01" "2009-07-01"
```

zoo:

```
> Z <- zoo(data, as.Date(charvec))
> Z
```

```
2009-01-01        NA 0.96519  0.70410
2009-02-01 -0.629991 1.44003  0.55432
2009-03-01        NA      NA -0.35576
2009-04-01 -1.349331      NA  0.79447
2009-05-01 -0.359353 0.79603  1.03604
2009-06-01 -0.760570 0.89683 -0.71239
2009-07-01 -0.029310 0.22897 -0.74984
```

```
> Z[is.na(Z)] <- 0
> Z
```

```
2009-01-01  0.000000 0.96519  0.70410
2009-02-01 -0.629991 1.44003  0.55432
2009-03-01  0.000000 0.00000 -0.35576
2009-04-01 -1.349331 0.00000  0.79447
2009-05-01 -0.359353 0.79603  1.03604
2009-06-01 -0.760570 0.89683 -0.71239
2009-07-01 -0.029310 0.22897 -0.74984
```

xts:

```
> X <- xts(data, as.Date(charvec))
> X
```

```
                [,1]    [,2]     [,3]
2009-01-01        NA 0.96519  0.70410
```

```
2009-02-01 -0.629991 1.44003  0.55432
2009-03-01        NA      NA -0.35576
2009-04-01 -1.349331      NA  0.79447
2009-05-01 -0.359353 0.79603  1.03604
2009-06-01 -0.760570 0.89683 -0.71239
2009-07-01 -0.029310 0.22897 -0.74984
```

```
> X[is.na(X)] <- 0
> X
```

```
                [,1]    [,2]     [,3]
2009-01-01  0.000000 0.96519  0.70410
2009-02-01 -0.629991 1.44003  0.55432
2009-03-01  0.000000 0.00000 -0.35576
2009-04-01 -1.349331 0.00000  0.79447
2009-05-01 -0.359353 0.79603  1.03604
2009-06-01 -0.760570 0.89683 -0.71239
2009-07-01 -0.029310 0.22897 -0.74984
```

timeSeries:

```
> S <- timeSeries(data, as.Date(charvec))
> S
```

```
GMT
                TS.1    TS.2     TS.3
2009-01-01        NA 0.96519  0.70410
2009-02-01 -0.629991 1.44003  0.55432
2009-03-01        NA      NA -0.35576
2009-04-01 -1.349331      NA  0.79447
2009-05-01 -0.359353 0.79603  1.03604
2009-06-01 -0.760570 0.89683 -0.71239
2009-07-01 -0.029310 0.22897 -0.74984
```

```
> S[is.na(S)] <- 0
> S
```

```
GMT
                TS.1    TS.2     TS.3
2009-01-01  0.000000 0.96519  0.70410
2009-02-01 -0.629991 1.44003  0.55432
2009-03-01  0.000000 0.00000 -0.35576
2009-04-01 -1.349331 0.00000  0.79447
2009-05-01 -0.359353 0.79603  1.03604
2009-06-01 -0.760570 0.89683 -0.71239
2009-07-01 -0.029310 0.22897 -0.74984
```

How can I interpolate missing values in a univariate time series object?

For this `zoo` and `xts` have the functions `na.approx` and `na.spline`, `timeSeries` has the function `na.omit` with the argument `method` for selecting interpolation.

Common Data:

```
> data <- 1:9
> data[c(1, 7)] <- NA
> data

[1] NA  2  3  4  5  6 NA  8  9

> charvec <- paste("2009-0", 1:9, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01" "2009-07-01" "2009-08-01" "2009-09-01"
```

zoo:

Linear Interpolation using the function `approx()`

```
> z <- zoo(data, as.Date(charvec))
> z

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
        NA          2          3          4          5          6
2009-07-01 2009-08-01 2009-09-01
        NA          8          9

> na.approx(z)

2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01 2009-07-01
    2.0000     3.0000     4.0000     5.0000     6.0000     6.9836
2009-08-01 2009-09-01
    8.0000     9.0000
```

Spline Interpolation using the function `spline()`

```
> z
```

```
2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
        NA          2          3          4          5          6
2009-07-01 2009-08-01 2009-09-01
        NA          8          9
```

```
> na.spline(z)
```

```
2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
   0.63057    2.00000    3.00000    4.00000    5.00000    6.00000
2009-07-01 2009-08-01 2009-09-01
   6.97966    8.00000    9.00000
```

Last Observation Carried Forward

```
> z
```

```
2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
        NA          2          3          4          5          6
2009-07-01 2009-08-01 2009-09-01
        NA          8          9
```

```
> na.locf(z)
```

```
2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01 2009-07-01
         2          3          4          5          6          6
2009-08-01 2009-09-01
         8          9
```

Trim Leading/Trailing Missing Observations

```
> z
```

```
2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
        NA          2          3          4          5          6
2009-07-01 2009-08-01 2009-09-01
        NA          8          9
```

```
> na.trim(z, sides = "left")
```

```
2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01 2009-07-01
         2          3          4          5          6         NA
2009-08-01 2009-09-01
         8          9
```

xts:

```
> xts <- xts(data, as.Date(charvec))
> xts
```

```
           [,1]
2009-01-01   NA
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
2009-07-01   NA
2009-08-01    8
2009-09-01    9
```

```
> na.approx(xts)
```

```
             [,1]
2009-02-01 2.0000
2009-03-01 3.0000
2009-04-01 4.0000
2009-05-01 5.0000
2009-06-01 6.0000
2009-07-01 6.9836
2009-08-01 8.0000
2009-09-01 9.0000
```

```
> x
```

```
               [,1]
2009-01-01 -0.74019
2009-02-01 -0.23273
2009-03-01  0.29271
2009-04-01  1.09026
2009-05-01 -0.74019
2009-06-01 -1.23671
2009-07-01 -0.74019
2009-08-01  0.44108
2009-09-01 -1.05783
```

```
> na.spline(x)
```

```
               [,1]
2009-01-01 -0.74019
2009-02-01 -0.23273
2009-03-01  0.29271
2009-04-01  1.09026
2009-05-01 -0.74019
2009-06-01 -1.23671
2009-07-01 -0.74019
2009-08-01  0.44108
2009-09-01 -1.05783
```

Last Observation Carried Forward

```
> x

                [,1]
2009-01-01 -0.74019
2009-02-01 -0.23273
2009-03-01  0.29271
2009-04-01  1.09026
2009-05-01 -0.74019
2009-06-01 -1.23671
2009-07-01 -0.74019
2009-08-01  0.44108
2009-09-01 -1.05783

> na.locf(x)

                [,1]
2009-01-01 -0.74019
2009-02-01 -0.23273
2009-03-01  0.29271
2009-04-01  1.09026
2009-05-01 -0.74019
2009-06-01 -1.23671
2009-07-01 -0.74019
2009-08-01  0.44108
2009-09-01 -1.05783
```

Trim Leading/Trailing Missing Observations

```
> x

                [,1]
2009-01-01 -0.74019
2009-02-01 -0.23273
2009-03-01  0.29271
2009-04-01  1.09026
2009-05-01 -0.74019
2009-06-01 -1.23671
2009-07-01 -0.74019
2009-08-01  0.44108
2009-09-01 -1.05783

> na.trim(x, sides = "left")

                [,1]
2009-01-01 -0.74019
2009-02-01 -0.23273
```

```
2009-03-01  0.29271
2009-04-01  1.09026
2009-05-01 -0.74019
2009-06-01 -1.23671
2009-07-01 -0.74019
2009-08-01  0.44108
2009-09-01 -1.05783
```

timeSeries:

Interpolation of time series is done by calling internally the function `approx` whoch provides linear interpolation.

Interpolate and remove (trim) `NAs` at the beginning and end of the series

```
> s <- timeSeries(data, charvec)
> na.omit(s, "ir")

GMT
           TS.1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
2009-07-01    6
2009-08-01    8
2009-09-01    9
```

Interpolate and replace `NAs` at the beginning and end of the series with zeros

```
> s <- timeSeries(data, charvec)
> na.omit(s, "iz")

GMT
           TS.1
2009-01-01    0
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
2009-07-01    6
```

```
2009-08-01    8
2009-09-01    9
```

Interpolate and extrapolate `NA`s at the beginning and end of the series

```
> s <- timeSeries(data, charvec)
> na.omit(s, "ie")
```

```
GMT
           TS.1
2009-01-01    2
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
2009-07-01    6
2009-08-01    8
2009-09-01    9
```

Through the argument `interp=c("before", "linear", "after")` we can select how to interpolate. In the case of `"before"` we carry the last observation forward (see zoo's na.locf), and in the case of `"after"` we carry the next observation backward.

```
> sn <- na.omit(s, method = "iz", interp = "before")
> sn[is.na(s)]
```

```
[1] 0 6
```

```
> sn <- na.omit(s, method = "iz", interp = "linear")
> sn[is.na(s)]
```

```
[1] 0 7
```

```
> sn <- na.omit(s, method = "iz", interp = "after")
> sn[is.na(s)]
```

```
[1] 0 8
```

Note that the way how to perform the linear interpolation can be modified by changing the defaults settings of the arguments in the function approx.

```
> args(approx)
```

```
function (x, y = NULL, xout, method = "linear", n = 50, yleft,
    yright, rule = 1, f = 0, ties = mean)
NULL
```

Does the function `na.contiguous()` work for a univariate time series objects?

The function `na.contiguous()` finds the longest consecutive stretch of non-missing values in a time series object. Note that in the event of a tie, this will be the first such stretch.

Common Data:

```
> data <- rnorm(12)
> data[c(3, 8)] = NA
> data

 [1] -0.13887 -0.75352       NA -0.42813  1.15183 -1.67703  0.33651
 [8]       NA  0.65252 -0.62797 -0.78623 -0.24610

> charvec <- as.character(timeCalendar(2009))
> charvec

 [1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
 [6] "2009-06-01" "2009-07-01" "2009-08-01" "2009-09-01" "2009-10-01"
[11] "2009-11-01" "2009-12-01"
```

ts:

```
> t <- ts(data, start = 2009, frequency = 12)
> t

          Jan      Feb      Mar      Apr      May      Jun      Jul
2009 -0.13887 -0.75352       NA -0.42813  1.15183 -1.67703  0.33651
          Aug      Sep      Oct      Nov      Dec
2009       NA  0.65252 -0.62797 -0.78623 -0.24610

> na.contiguous(t)

          Apr      May      Jun      Jul
2009 -0.42813  1.15183 -1.67703  0.33651
```

zoo:

```
> z <- zoo(data, as.Date(charvec))
> z
```

```
2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
  -0.13887   -0.75352         NA   -0.42813    1.15183   -1.67703
2009-07-01 2009-08-01 2009-09-01 2009-10-01 2009-11-01 2009-12-01
   0.33651         NA    0.65252   -0.62797   -0.78623   -0.24610
```

```
> na.contiguous(z)
```

```
2009-04-01 2009-05-01 2009-06-01 2009-07-01
  -0.42813    1.15183   -1.67703    0.33651
```

xts:

```
> x <- xts(data, as.Date(charvec))
> x
```

```
               [,1]
2009-01-01 -0.13887
2009-02-01 -0.75352
2009-03-01       NA
2009-04-01 -0.42813
2009-05-01  1.15183
2009-06-01 -1.67703
2009-07-01  0.33651
2009-08-01       NA
2009-09-01  0.65252
2009-10-01 -0.62797
2009-11-01 -0.78623
2009-12-01 -0.24610
```

```
> na.contiguous(x)
```

```
               [,1]
2009-04-01 -0.42813
2009-05-01  1.15183
2009-06-01 -1.67703
2009-07-01  0.33651
```

timeSeries:

```
> s <- timeSeries(data, charvec)
> s
```

```
GMT
               TS.1
```

```
2009-01-01 -0.13887
2009-02-01 -0.75352
2009-03-01       NA
2009-04-01 -0.42813
2009-05-01  1.15183
2009-06-01 -1.67703
2009-07-01  0.33651
2009-08-01       NA
2009-09-01  0.65252
2009-10-01 -0.62797
2009-11-01 -0.78623
2009-12-01 -0.24610
```

```
> na.contiguous(s)
```

```
GMT
               TS.1
2009-04-01 -0.42813
2009-05-01  1.15183
2009-06-01 -1.67703
2009-07-01  0.33651
```

Chapter 2
Reporting

2.1 Printing Time Series Objects

Required R package(s):

```
> library(zoo)
> library(xts)
> library(timeSeries)
```

How can I print a time series object?

Time series objects are displayed in the usual way as other data objects in R

```
> data <- 1:6
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

xts:

```
> zoo(data, as.Date(charvec))

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
         1          2          3          4          5          6
```

xts:

```
> xts(data, as.Date(charvec))

           [,1]
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
```

timeSeries:

```
> timeSeries(data, charvec)

GMT
           TS.1
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
```

Alternatively we can use explicitely the generic print function `print()`

```
> data <- 1:6
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

xts:

```
> print(zoo(data, as.Date(charvec)))

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
         1          2          3          4          5          6
```

xts:

```
> print(xts(data, as.Date(charvec)))

           [,1]
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
```

timeSeries:

```
> print(timeSeries(data, charvec))

GMT
           TS.1
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
```

Since a timeSeries object is represented by a S4 class we can also use

```
> show(timeSeries(data, charvec))

GMT
           TS.1
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
```

How can I select the style for printing an univariate times series?

Common Data:

```
> data <- 1:6
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

Using the `print` function univariate `zoo` time series are always displayed in an horizontal style.

```
> print(zoo(data, as.Date(charvec)))

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
         1          2          3          4          5          6
```

xts:

Using the `print` function univariate `xts` time series are always displayed in an vertical style.

```
> print(xts(data, as.Date(charvec)))

           [,1]
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
```

If you want to have printed zoo objects vertically and xts horizontically type

```
> print(as.zoo(xts(data, as.Date(charvec))))

2009-01-01 1
2009-02-01 2
2009-03-01 3
2009-04-01 4
2009-05-01 5
2009-06-01 6

> print(as.xts(xts(data, as.Date(charvec))))

           [,1]
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
```

timeSeries:

The `timesSeries` class comes with a generic print function which allows to customize the printout in several ways. For example an univariate `timeSeries` object is printed by default in vertical style

```
> s <- timeSeries(data, charvec)
> print(s)

GMT
           TS.1
2009-01-01    1
2009-02-01    2
2009-03-01    3
2009-04-01    4
2009-05-01    5
2009-06-01    6
```

but it can also be printed in horizontal style

```
> print(s, style = "h")

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
         1          2          3          4          5          6
```

Is it possible to display beside the ISO date/time format other formats when we print a a time series ?

Common Data:

```
> data <- 1:6
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

To our knowledge, no.

xts:

To our knowledge, no.

timeSeries:

The generic `print` function for the `timeSeries` class allows to customize the printing format. Here are some examples

```
> s <- timeSeries(pi, "2009-05-08 19:26:22", units = "s")
> print(s)

GMT
                         s
2009-05-08 19:26:22 3.1416

> print(s, format = "%m/%d/%y %H:%M")

GMT
                    s
05/08/09 19:26 3.1416

> print(s, format = "%m/%d/%y  %A")

GMT
                      s
05/08/09  Friday 3.1416

> print(s, format = "DayNo %j  Hour: %H")

GMT
                         s
DayNo 128  Hour: 19 3.1416
```

The first example prints in default ISO format, the second in month-day-year format with the full name of the weekday, and the last example prints the day of the year together the truncated hour of the day.

Can time series Objects be printed in the style of R's ts objects?

zoo:

In the case of a regular zoo time series yes, otherwise not.

xts:

To our knowledge, no.

timeSeries:

The print method for `timeSeries` objects has a `style="ts"` option. For example if you have monthly records

```
> data <- 1:6
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> s <- timeSeries(data, charvec)
> print(s, style = "ts")

     Jan Feb Mar Apr May Jun
2009   1   2   3   4   5   6
```

and in the case of quarterly records

```
> data <- 1:4
> charvec <- paste("2009-", c("03", "06", "09", "12"), "-01", sep = "")
> s <- timeSeries(data, charvec)
> print(s, style = "ts", by = "quarter")

     Qtr1 Qtr2 Qtr3 Qtr4
2009                1    2
2010    3    4    1    2
2011    3    4    1    2
```

How can we print a time series with respect to another time zone?

zoo:

There is no direct way to do it.

xts:

There is no direct way to do it.

timeSeries:

The print method for `timeSeries` objects has a `FinCenter=NULL` option. For example if you have monthly records

```
> data <- rnorm(6)
> charvec <- paste("2009-0", 1:6, "-01 16:00:00", sep = "")
> s <- timeSeries(data, charvec, zone = "Chicago", FinCenter = "Zurich")
> print(s, FinCenter = "Chicago")

Chicago
                        TS.1
2009-01-01 16:00:00  0.61180
2009-02-01 16:00:00 -0.62115
2009-03-01 16:00:00 -0.25072
2009-04-01 16:00:00 -0.45696
2009-05-01 16:00:00 -0.49877
2009-06-01 16:00:00 -0.47390

> print(s, FinCenter = "Zurich")

Zurich
                        TS.1
2009-01-01 23:00:00  0.61180
2009-02-01 23:00:00 -0.62115
2009-03-01 23:00:00 -0.25072
2009-04-01 23:00:00 -0.45696
2009-05-01 23:00:00 -0.49877
2009-06-01 23:00:00 -0.47390

> print(s, FinCenter = "Tokyo")

Tokyo
                        TS.1
2009-01-02 07:00:00  0.61180
2009-02-02 07:00:00 -0.62115
2009-03-02 07:00:00 -0.25072
2009-04-02 06:00:00 -0.45696
2009-05-02 06:00:00 -0.49877
2009-06-02 06:00:00 -0.47390
```

2.2 Plotting Time Series Objects

Required R package(s):

```
> library(zoo)
> library(xts)
> library(timeSeries)
```

How can I plot a univariate time series using the generic plot function?

Common Data:

```
> set.seed(1953)
> data <- runif(6)
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> args(plot.zoo)

function (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, ...)
NULL

> z <- zoo(data, as.POSIXct(charvec, tz = "GMT"))
> plot(z)
```

Note for this example the labels on the x axis have no year numbers.

xts:

```
> args(plot.xts)
```

```
function (x, y = NULL, type = "l", auto.grid = TRUE, major.ticks = "auto",
    minor.ticks = TRUE, major.format = TRUE, bar.col = "grey",
    candle.col = "white", ann = TRUE, axes = TRUE, ...)
NULL
```

```
> x <- xts(data, as.POSIXct(charvec, tz = "GMT"))
> plot(x)
```

timeSeries:

`timeSeries` has an S4 generic method for plotting.

```
> s <- timeSeries(data, charvec)
> plot(s)
```

How can I plot a multivariate time series using the generic plot function?

Common Data:

```
> set.seed(1953)
> data <- matrix(runif(12), ncol = 2)
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec
```

```
[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> Z <- zoo(data, as.POSIXct(charvec, tz = "GMT"))
> plot(Z)
```

xts:

```
> X <- xts(data, as.POSIXct(charvec, tz = "GMT"))
> plot(X)
```

`xts` does not support multivariate time series plots in in multiple charts.

timeSeries:

```
> S <- timeSeries(data, charvec)
> plot(S)
```

How can I plot a multivariate time series in a single chart using the generic plot function?

Common Data:

```
> set.seed(1953)
> data <- matrix(runif(18), ncol = 3)
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec
```

```
[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> Z <- zoo(data, as.POSIXct(charvec, tz = "GMT"))
> plot(Z, plot.type = "single")
```

xts:

`xts` does not support multivariate time series plots in in single charts.

timeSeries:

```
> S <- timeSeries(data, as.POSIXct(charvec, FinCenter = "GMT"))
> plot(S, plot.type = "single")
```

How can I add a line to an existing plot?

Common Data:

```
> set.seed(1953)
> data1 <- rnorm(9)
> data2 <- rnorm(9, sd = 0.2)
> charvec <- paste("2009-0", 1:9, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01" "2009-07-01" "2009-08-01" "2009-09-01"
```

zoo:

```
> z1 <- zoo(data1, as.POSIXct(charvec, tz = "GMT"))
> z2 <- zoo(data2, as.POSIXct(charvec, tz = "GMT"))
> plot(z1)
> lines(z2, col = 2)
```

timeSeries:

```
> s1 <- timeSeries(data1, charvec, FinCenter = "GMT")
> s2 <- timeSeries(data2, charvec, FinCenter = "GMT")
> plot(s1)
> lines(s2, col = 2)
```

How can I add points to an existing plot?

Common Data:

```
> set.seed(1953)
> data <- rnorm(9)
> charvec <- paste("2009-0", 1:9, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01" "2009-07-01" "2009-08-01" "2009-09-01"
```

zoo:

```
> z <- zoo(data, as.POSIXct(charvec, tz = "GMT"))
> plot(z)
> points(z, col = 2, pch = 19)
```

timeSeries:

```
> s <- timeSeries(data, charvec, FinCenter = "GMT")
> plot(s)
> points(s, col = 2, pch = 19)
```

Can I use abline for time series plots?

Common Data:

```
> set.seed(1953)
> data <- rnorm(9)
> charvec <- paste("2009-0", 1:9, "-01", sep = "")
> charvec
```

```
[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01" "2009-07-01" "2009-08-01" "2009-09-01"
```

zoo:

```
> z <- zoo(data, as.POSIXct(charvec, tz = "GMT"))
> plot(z)
> abline(v = index(z), col = "darkgrey", lty = 3)
```

timeSeries:

```
> s <- timeSeries(data, charvec, FinCenter = "GMT")
> plot(s)
> abline(v = time(s), col = "darkgrey", lty = 3)
```

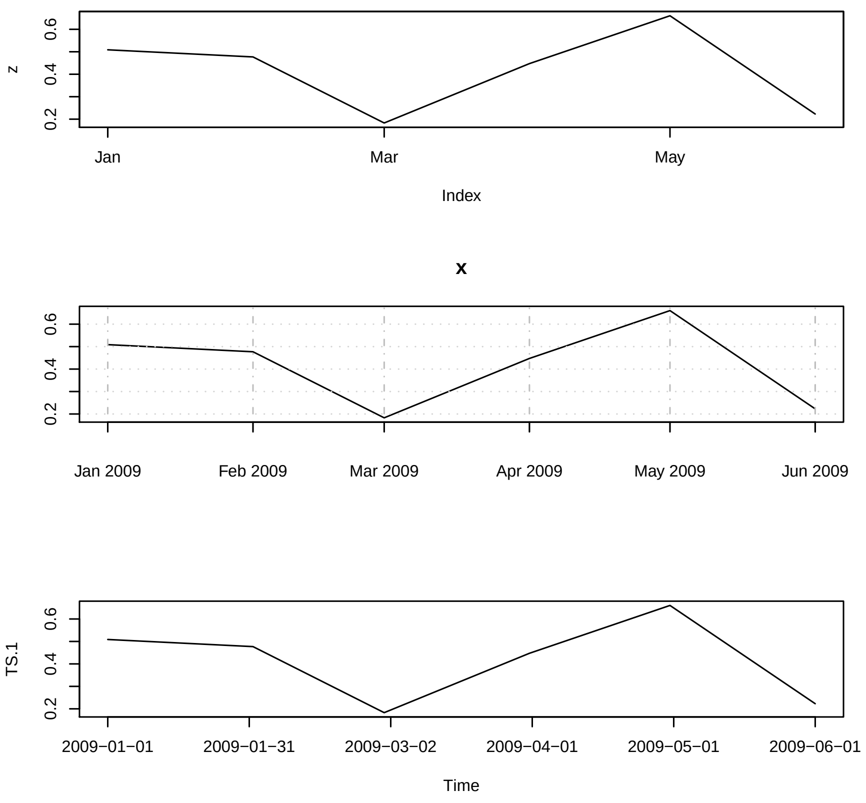

Figure 2.1: Plot 1 - Example of a single panel plot from zoo, xts and timeSeries

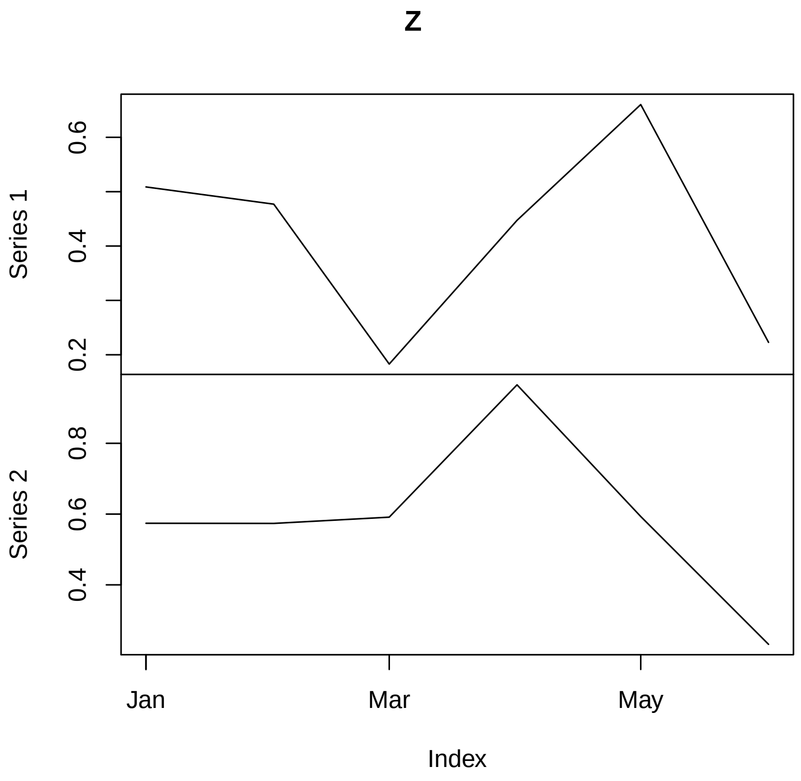

Figure 2.2: Plot 2a - Multivariate Time Series plots in multiple graphs

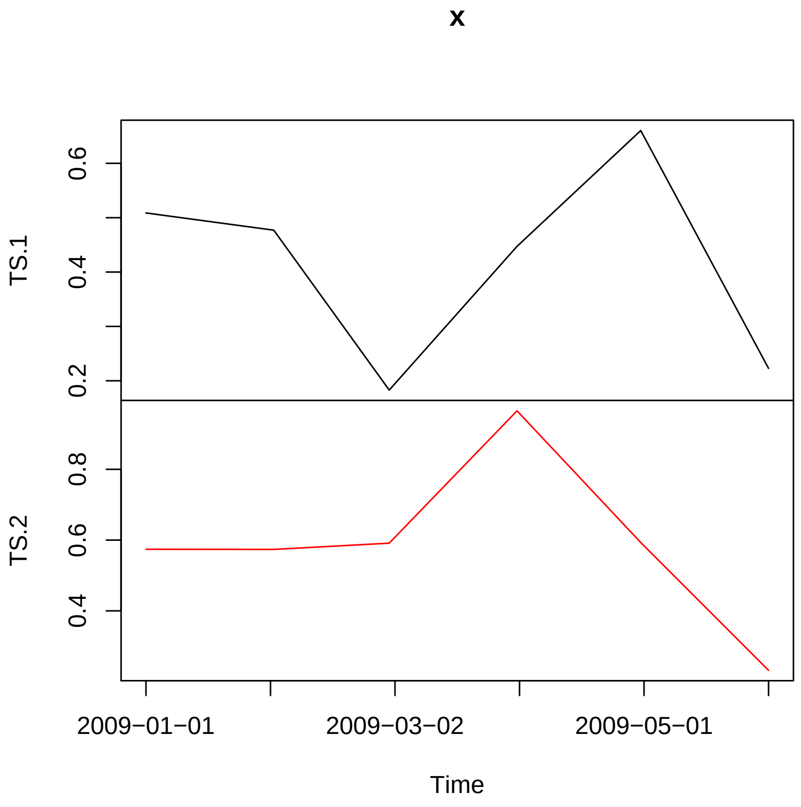

Figure 2.3: Plot 2b - Multivariate Time Series plots in multiple graphs

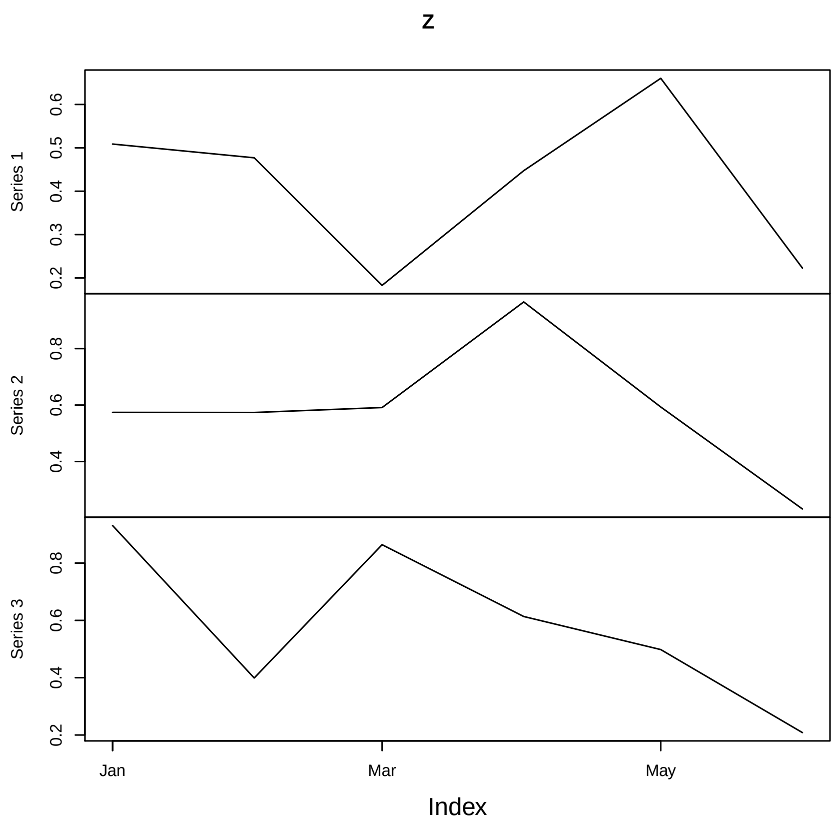

Figure 2.4: Plot3a - Example of a single panel plot from zoo

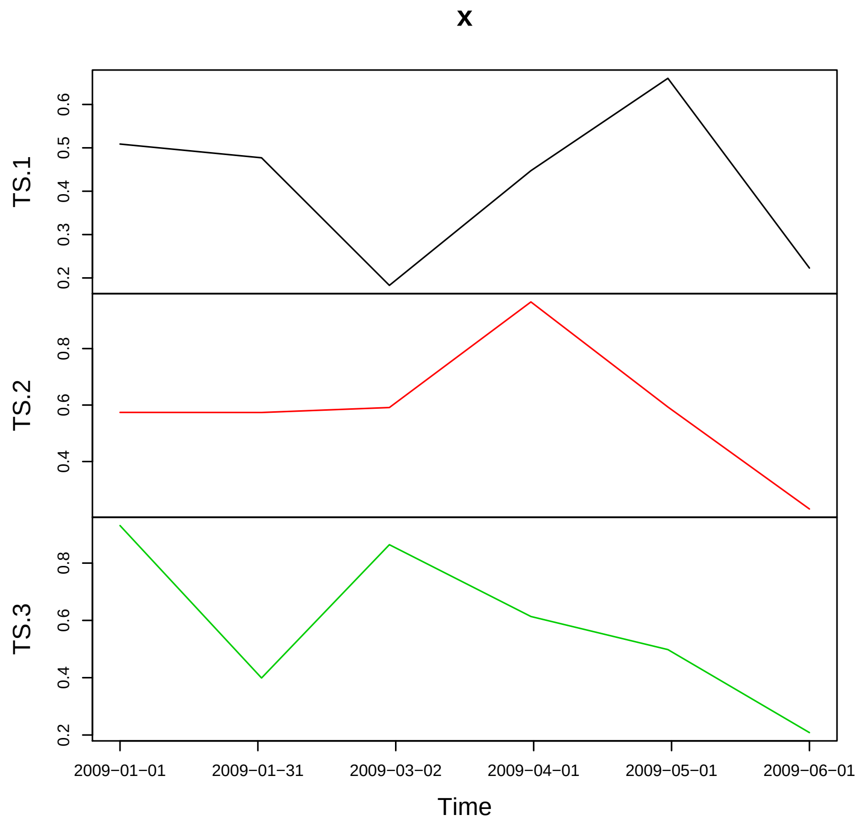

Figure 2.5: Plot 3b - Example of a single panel plot from timeSeries

Chapter 3
Using R Functions

3.1 Functions and Methods from base R

Required R package(s):

```
> library(zoo)
> library(xts)
> library(timeSeries)
```

In the following we test some selected functions that are relevant for use in financial time series analysis. We test them on multivariate time series objects

```
> set.seed(1953)
> data <- matrix(runif(18), ncol = 3)
> charvec <- rev(paste("2009-0", 1:6, "-01", sep = ""))
> charvec

[1] "2009-06-01" "2009-05-01" "2009-04-01" "2009-03-01" "2009-02-01"
[6] "2009-01-01"
```

zoo:

```
> Z <- zoo(data, as.Date(charvec))
> colnames(Z) = paste("Z", 1:3, sep = ".")
```

xts:

```
> X <- xts(data, as.Date(charvec))
> colnames(X) = paste("X", 1:3, sep = ".")
```

timeSeries:

```
> S <- timeSeries(data, charvec)
> colnames(S) = paste("S", 1:3, sep = ".")
```

and/or univariate time series objects.

zoo:

```
> z <- Z[, 1]
```

xts:

```
> x <- X[, 1]
```

timeSeries:

```
> s <- S[, 1]
```

Can I use apply like in R's base package?

The function `apply()` returns a vector or array or list of values obtained by applying a function to margins of an array. How does the function works together with time Series objects?

zoo:

```
> apply(Z, 2, mean)

    Z.1     Z.2     Z.3
0.41652 0.58821 0.58572
```

xts:

```
> apply(X, 2, mean)

    X.1     X.2     X.3
0.41652 0.58821 0.58572
```

timeSeries:

```
> apply(S, 2, mean)
```

```
    S.1     S.2     S.3
0.41652 0.58821 0.58572
```

Can I use attach like in R's base package?

The database is attached to the R search path. This means that the database is searched by R when evaluating a variable, so objects in the database can be accessed by simply giving their names.

zoo:

```
> print(try(attach(Z)))

[1] "Error in attach(Z) : \n  'attach' only works for lists, data frames and
environments\n"
attr(,"class")
[1] "try-error"
```

xts:

```
> print(try(attach(X)))

[1] "Error in attach(X) : \n  'attach' only works for lists, data frames and
environments\n"
attr(,"class")
[1] "try-error"
```

timeSeries:

```
> attach(S)
> colnames(S)

[1] "S.1" "S.2" "S.3"

> S.2

[1] 0.57402 0.57365 0.59127 0.96524 0.59316 0.23193
```

Can I use diff like in R's base package?

The function `diff()` returns suitably lagged and iterated differences.

zoo:

```
> diff(Z)

                Z.1         Z.2      Z.3
2009-02-01  0.437452  0.36122800  0.28980
2009-03-01 -0.212986  0.37207930  0.11545
2009-04-01 -0.264387 -0.37396709  0.25063
2009-05-01  0.294113 -0.01761714 -0.46521
2009-06-01  0.031721  0.00036655  0.53205

> diff(z)

2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
  0.437452  -0.212986  -0.264387   0.294113   0.031721
```

xts:

```
> diff(X)

                X.1         X.2      X.3
2009-01-01        NA          NA       NA
2009-02-01  0.437452  0.36122800  0.28980
2009-03-01 -0.212986  0.37207930  0.11545
2009-04-01 -0.264387 -0.37396709  0.25063
2009-05-01  0.294113 -0.01761714 -0.46521
2009-06-01  0.031721  0.00036655  0.53205

> diff(x)

                X.1
2009-01-01        NA
2009-02-01  0.437452
2009-03-01 -0.212986
2009-04-01 -0.264387
2009-05-01  0.294113
2009-06-01  0.031721
```

timeSeries:

```
> diff(S)

GMT
                  S.1         S.2      S.3
2009-06-01         NA          NA       NA
2009-05-01 -0.031721 -0.00036655 -0.53205
2009-04-01 -0.294113  0.01761714  0.46521
2009-03-01  0.264387  0.37396709 -0.25063
2009-02-01  0.212986 -0.37207930 -0.11545
2009-01-01 -0.437452 -0.36122800 -0.28980
```

```
> diff(s)

GMT
                  S.1
2009-06-01         NA
2009-05-01 -0.031721
2009-04-01 -0.294113
2009-03-01  0.264387
2009-02-01  0.212986
2009-01-01 -0.437452
```

Can I use dim like in R's base package?

The function `dim()` retrieves or sets the dimension of an object.

zoo:

```
> dim(Z)

[1] 6 3

> dim(z)

NULL
```

xts:

```
> dim(X)

[1] 6 3
```

```
> dim(x)

[1] 6 1
```

timeSeries:

```
> dim(S)

[1] 6 3

> dim(s)

[1] 6 1
```

Can I use rank like in R's base package?

The function `rank()` returns the sample ranks of the values in a vector. Ties (i.e., equal values) and missing values can be handled in several ways.

zoo:

```
> print(try(rank(Z)))

[1] "Error in names(y) <- nm[!nas] : \n  'names' attribute [18] must be the
same length as the vector [0]\n"
attr(,"class")
[1] "try-error"

> print(try(rank(z)))

[1] "Error in if (xi == xj) 0L else if (xi > xj) 1L else -1L : \n  argument is
 of length zero\n"
attr(,"class")
[1] "try-error"
```

xts:

```
> print(try(rank(X)))

[1] "Error in `[.xts`(x, !nas) : i is out of range\n\n"
attr(,"class")
[1] "try-error"
```

```
> print(try(rank(x)))

[1] "Error in if (xi == xj) 0L else if (xi > xj) 1L else -1L : \n  argument is
 of length zero\n"
attr(,"class")
[1] "try-error"
```

timeSeries:

```
> rank(S)

GMT
           S.1 S.2 S.3
2009-06-01   5   3   6
2009-05-01   4   2   2
2009-04-01   1   4   5
2009-03-01   3   6   4
2009-02-01   6   5   3
2009-01-01   2   1   1

> rank(s)

GMT
           S.1
2009-06-01   5
2009-05-01   4
2009-04-01   1
2009-03-01   3
2009-02-01   6
2009-01-01   2
```

Can I use rev like in R's base package?

The function `rev()` provides a reversed version of its argument.

zoo:

```
> print(try(rev(Z)))

[1] "Error in `[.zoo`(x, length(x):1L) : subscript out of bounds\n"
attr(,"class")
[1] "try-error"
```

```
> rev(z)

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
   0.22283    0.66028    0.44730    0.18291    0.47702    0.50875
```

xts:

```
> print(try(rev(X)))

[1] "Error in `[.xts`(x, length(x):1L) : subscript out of bounds\n"
attr(,"class")
[1] "try-error"

> rev(x)

               X.1
2009-01-01 0.22283
2009-02-01 0.66028
2009-03-01 0.44730
2009-04-01 0.18291
2009-05-01 0.47702
2009-06-01 0.50875
```

timeSeries:

```
> rev(S)

GMT
               S.1     S.2     S.3
2009-01-01 0.22283 0.23193 0.20833
2009-02-01 0.66028 0.59316 0.49813
2009-03-01 0.44730 0.96524 0.61359
2009-04-01 0.18291 0.59127 0.86422
2009-05-01 0.47702 0.57365 0.39901
2009-06-01 0.50875 0.57402 0.93106

> rev(s)

GMT
               S.1
2009-01-01 0.22283
2009-02-01 0.66028
2009-03-01 0.44730
2009-04-01 0.18291
```

```
2009-05-01 0.47702
2009-06-01 0.50875
```

Can I use sample like in R's base package?

The function `sample()` takes a sample of the specified size from the elements of `x` using either with or without replacement.

zoo:

```
> print(try(sample(Z)))

[1] "Error in `[.zoo`(x, .Internal(sample(length(x), size, replace, prob))) :
\n  subscript out of bounds\n"
attr(,"class")
[1] "try-error"

> sample(z)

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
   0.22283    0.66028    0.44730    0.18291    0.47702    0.50875
```

xts:

```
> print(try(sample(X)))

[1] "Error in `[.xts`(x, .Internal(sample(length(x), size, replace, prob))) :
\n  subscript out of bounds\n"
attr(,"class")
[1] "try-error"

> sample(x)

               X.1
2009-01-01 0.22283
2009-02-01 0.66028
2009-03-01 0.44730
2009-04-01 0.18291
2009-05-01 0.47702
2009-06-01 0.50875
```

timeSeries:

```
> sample(S)

GMT
                S.1     S.2     S.3
2009-03-01 0.44730 0.96524 0.61359
2009-02-01 0.66028 0.59316 0.49813
2009-04-01 0.18291 0.59127 0.86422
2009-05-01 0.47702 0.57365 0.39901
2009-06-01 0.50875 0.57402 0.93106
2009-01-01 0.22283 0.23193 0.20833

> sample(s)

GMT
                S.1
2009-04-01 0.18291
2009-05-01 0.47702
2009-06-01 0.50875
2009-02-01 0.66028
2009-01-01 0.22283
2009-03-01 0.44730
```

Can I use scale like in R's base package?

The function `scale()` is generic function whose default method centers and/or scales the columns of a numeric matrix. How does it work together with time serie sobjects?

zoo:

```
> scale(Z)

                 Z.1       Z.2      Z.3
2009-01-01 -1.06742 -1.534521 -1.36437
2009-02-01  1.34343  0.021305 -0.31667
2009-03-01  0.16964  1.623869  0.10073
2009-04-01 -1.28742  0.013175  1.00683
2009-05-01  0.33347 -0.062703 -0.67501
2009-06-01  0.50829 -0.061125  1.24849
attr(,"scaled:center")
    Z.1     Z.2     Z.3
```

```
0.41652 0.58821 0.58572
attr(,"scaled:scale")
    Z.1     Z.2     Z.3
0.18145 0.23218 0.27660
```

```
> scale(z)
```

```
2009-01-01 -1.06742
2009-02-01  1.34343
2009-03-01  0.16964
2009-04-01 -1.28742
2009-05-01  0.33347
2009-06-01  0.50829
attr(,"scaled:center")
[1] 0.41652
attr(,"scaled:scale")
[1] 0.18145
```

xts:

```
> scale(X)
```

```
                X.1       X.2      X.3
2009-01-01 -1.06742 -1.534521 -1.36437
2009-02-01  1.34343  0.021305 -0.31667
2009-03-01  0.16964  1.623869  0.10073
2009-04-01 -1.28742  0.013175  1.00683
2009-05-01  0.33347 -0.062703 -0.67501
2009-06-01  0.50829 -0.061125  1.24849
```

```
> scale(x)
```

```
                X.1
2009-01-01 -1.06742
2009-02-01  1.34343
2009-03-01  0.16964
2009-04-01 -1.28742
2009-05-01  0.33347
2009-06-01  0.50829
```

timeSeries:

```
> scale(S)
```

```
GMT
                S.1       S.2      S.3
2009-06-01  0.50829 -0.061125  1.24849
2009-05-01  0.33347 -0.062703 -0.67501
2009-04-01 -1.28742  0.013175  1.00683
2009-03-01  0.16964  1.623869  0.10073
2009-02-01  1.34343  0.021305 -0.31667
2009-01-01 -1.06742 -1.534521 -1.36437
```

```
> scale(s)
```

```
GMT
                S.1
2009-06-01  0.50829
2009-05-01  0.33347
2009-04-01 -1.28742
2009-03-01  0.16964
2009-02-01  1.34343
2009-01-01 -1.06742
```

Can I use sort like in R's base package?

The function `sort` (or `order`) sorts a vector or factor (partially) into ascending (or descending) order. For ordering along more than one variable, e.g., for sorting data frames, see order.

zoo:

```
> print(try(sort(Z)))
```

```
[1] "Error in `[.zoo`(x, order(x, na.last = na.last, decreasing = decreasing))
 : \n  subscript out of bounds\n"
attr(,"class")
[1] "try-error"
```

```
> sort(z)
```

```
2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
   0.22283    0.66028    0.44730    0.18291    0.47702    0.50875
```

xts:

```
> print(try(sort(X)))
```

```
[1] "Error in `[.xts`(x, order(x, na.last = na.last, decreasing = decreasing))
 : \n  subscript out of bounds\n"
attr(,"class")
[1] "try-error"

> sort(x)

                X.1
2009-01-01 0.22283
2009-02-01 0.66028
2009-03-01 0.44730
2009-04-01 0.18291
2009-05-01 0.47702
2009-06-01 0.50875
```

timeSeries:

```
> sort(S)

GMT
                S.1     S.2     S.3
2009-01-01 0.22283 0.23193 0.20833
2009-02-01 0.66028 0.59316 0.49813
2009-03-01 0.44730 0.96524 0.61359
2009-04-01 0.18291 0.59127 0.86422
2009-05-01 0.47702 0.57365 0.39901
2009-06-01 0.50875 0.57402 0.93106

> sort(s)

GMT
                S.1
2009-01-01 0.22283
2009-02-01 0.66028
2009-03-01 0.44730
2009-04-01 0.18291
2009-05-01 0.47702
2009-06-01 0.50875
```

Can I use start and end like in R's base package?

The functions `start()` and `end()` extract and encode the times the first and last observations were taken.

zoo:

```
> start(Z); end(Z)

[1] "2009-01-01"

[1] "2009-06-01"

> start(z); end(z)

[1] "2009-01-01"

[1] "2009-06-01"
```

xts:

```
> start(X); end(X)

[1] "2009-01-01"

[1] "2009-06-01"

> start(x); end(x)

[1] "2009-01-01"

[1] "2009-06-01"
```

timeSeries:

```
> start(S); end(S)

GMT
[1] [2009-01-01]

GMT
[1] [2009-06-01]

> start(s); end(s)

GMT
[1] [2009-01-01]

GMT
[1] [2009-06-01]
```

3.2 Functions and Methods from stats R

Required R package(s):

```
> library(zoo)
> library(xts)
> library(timeSeries)
```

In the following we test selected functions which are relevant for use in financial time series analysis.

Common Data:

We test them on multivariate

```
> set.seed(1953)
> data <- matrix(runif(18), ncol = 3)
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec

[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> Z <- zoo(data, as.Date(charvec))
> colnames(Z) = paste("Z", 1:3, sep = ".")
> Z

               Z.1     Z.2     Z.3
2009-01-01 0.50875 0.57402 0.93106
2009-02-01 0.47702 0.57365 0.39901
2009-03-01 0.18291 0.59127 0.86422
2009-04-01 0.44730 0.96524 0.61359
2009-05-01 0.66028 0.59316 0.49813
2009-06-01 0.22283 0.23193 0.20833
```

xts:

```
> X <- xts(data, as.Date(charvec))
> colnames(X) = paste("X", 1:3, sep = ".")
> X

               X.1     X.2     X.3
2009-01-01 0.50875 0.57402 0.93106
2009-02-01 0.47702 0.57365 0.39901
2009-03-01 0.18291 0.59127 0.86422
2009-04-01 0.44730 0.96524 0.61359
2009-05-01 0.66028 0.59316 0.49813
2009-06-01 0.22283 0.23193 0.20833
```

timeSeries:

```
> S <- timeSeries(data, charvec)
> colnames(S) = paste("S", 1:3, sep = ".")
> S

GMT
               S.1     S.2     S.3
2009-01-01 0.50875 0.57402 0.93106
2009-02-01 0.47702 0.57365 0.39901
2009-03-01 0.18291 0.59127 0.86422
2009-04-01 0.44730 0.96524 0.61359
2009-05-01 0.66028 0.59316 0.49813
2009-06-01 0.22283 0.23193 0.20833
```

and/or univariate time series objects.

zoo:

```
> z <- Z[, 1]
> z

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
   0.50875    0.47702    0.18291    0.44730    0.66028    0.22283
```

xts:

```
> x <- X[, 1]
> x
```

```
               X.1
2009-01-01 0.50875
2009-02-01 0.47702
2009-03-01 0.18291
2009-04-01 0.44730
2009-05-01 0.66028
2009-06-01 0.22283
```

timeSeries:

```
> s <- S[, 1]
> s

GMT
               S.1
2009-01-01 0.50875
2009-02-01 0.47702
2009-03-01 0.18291
2009-04-01 0.44730
2009-05-01 0.66028
2009-06-01 0.22283
```

Can I use arima like in R's base package?

The function `arima()` fits an ARIMA model to a univariate time series.

zoo:

```
> arima(z)

Call:
arima(x = z)

Coefficients:
      intercept
          0.417
s.e.      0.068

sigma^2 estimated as 0.0274:  log likelihood = 2.27,  aic = -0.55
```

xts:

```
> arima(x)

Call:
arima(x = x)

Coefficients:
      intercept
          0.417
s.e.      0.068

sigma^2 estimated as 0.0274:  log likelihood = 2.27,  aic = -0.55
```

timeSeries:

```
> arima(s)

Call:
arima(x = s)

Coefficients:
      intercept
          0.417
s.e.      0.068

sigma^2 estimated as 0.0274:  log likelihood = 2.27,  aic = -0.55
```

Can I use acf like in R's stats package?

The function `acf()` computes (and by default plots) estimates of the autocovariance or autocorrelation function. Function `pacf()` is the function used for the partial autocorrelations. Function `ccf()` computes the cross-correlation or cross-covariance of two univariate series.

zoo:

```
> print(try(acf(z)))

[1] "Error in na.fail.default(as.ts(x)) : missing values in object\n"
attr(,"class")
[1] "try-error"
```

xts:

```
> print(try(acf(x)))

Autocorrelations of series 'x', by lag

     0      1      2      3      4      5 
 1.000 -0.337 -0.502  0.382  0.065 -0.109 
```

timeSeries:

```
> print(acf(s))

Autocorrelations of series 's', by lag

0.0000 0.0833 0.1667 0.2500 0.3333 0.4167 
 1.000 -0.337 -0.502  0.382  0.065 -0.109 
```

Can I use cov like in R's stats package?

The functions `var()`, `cov()` and `cor()` compute the variance of `x` and the covariance or correlation of `x` and `y` if these are vectors. If `x` and `y` are matrices then the covariances (or correlations) between the columns of `x` and the columns of `y` are computed. Applied to zoo, xts, and timeSeries objects these functions behave in the following way.

zoo:

```
> var(z)

[1] 0.032925

> var(Z)

          Z.1      Z.2       Z.3
Z.1 0.0329245 0.015783 0.0016191
Z.2 0.0157826 0.053906 0.0286396
Z.3 0.0016191 0.028640 0.0765103

> cov(Z)
```

```
            Z.1      Z.2       Z.3
Z.1 0.0329245 0.015783 0.0016191
Z.2 0.0157826 0.053906 0.0286396
Z.3 0.0016191 0.028640 0.0765103
```

```
> cor(Z)
```

```
          Z.1     Z.2      Z.3
Z.1 1.000000 0.37463 0.032259
Z.2 0.374627 1.00000 0.445951
Z.3 0.032259 0.44595 1.000000
```

xts:

```
> var(x)
```

```
          X.1
X.1 0.032925
```

```
> var(X)
```

```
            X.1      X.2       X.3
X.1 0.0329245 0.015783 0.0016191
X.2 0.0157826 0.053906 0.0286396
X.3 0.0016191 0.028640 0.0765103
```

```
> cov(X)
```

```
            X.1      X.2       X.3
X.1 0.0329245 0.015783 0.0016191
X.2 0.0157826 0.053906 0.0286396
X.3 0.0016191 0.028640 0.0765103
```

```
> cor(X)
```

```
          X.1     X.2      X.3
X.1 1.000000 0.37463 0.032259
X.2 0.374627 1.00000 0.445951
X.3 0.032259 0.44595 1.000000
```

timeSeries:

```
> var(s)
```

```
          S.1
S.1 0.032925
```

```
> var(S)

          S.1      S.2       S.3
S.1 0.0329245 0.015783 0.0016191
S.2 0.0157826 0.053906 0.0286396
S.3 0.0016191 0.028640 0.0765103

> cov(S)

          S.1      S.2       S.3
S.1 0.0329245 0.015783 0.0016191
S.2 0.0157826 0.053906 0.0286396
S.3 0.0016191 0.028640 0.0765103

> cor(S)

         S.1     S.2      S.3
S.1 1.000000 0.37463 0.032259
S.2 0.374627 1.00000 0.445951
S.3 0.032259 0.44595 1.000000
```

Can I use dist like in R's stats package?

This function `dist()` computes and returns the distance matrix computed by using the specified distance measure to compute the distances between the rows of a data matrix.

zoo:

```
> dist(Z)

           2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01
2009-02-01    0.53300
2009-03-01    0.33307    0.55066
2009-04-01    0.50756    0.44751    0.52208
2009-05-01    0.45908    0.20926    0.60159    0.44400
2009-06-01    0.84918    0.46663    0.74894    0.86738    0.63705

> dist(t(Z))

        Z.1     Z.2
Z.2 0.67321
Z.3 0.83831 0.60475
```

xts:

```
> dist(X)

           2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01
2009-02-01    0.53300
2009-03-01    0.33307    0.55066
2009-04-01    0.50756    0.44751    0.52208
2009-05-01    0.45908    0.20926    0.60159    0.44400
2009-06-01    0.84918    0.46663    0.74894    0.86738    0.63705

> dist(t(X))

        X.1     X.2
X.2 0.67321
X.3 0.83831 0.60475
```

timeSeries:

```
> dist(S)

           2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01
2009-02-01    0.53300
2009-03-01    0.33307    0.55066
2009-04-01    0.50756    0.44751    0.52208
2009-05-01    0.45908    0.20926    0.60159    0.44400
2009-06-01    0.84918    0.46663    0.74894    0.86738    0.63705

> dist(t(S))

        S.1     S.2
S.2 0.67321
S.3 0.83831 0.60475
```

Can I use dnorm like in R's stats package?

The function `dnorm()` computes the density for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`.

zoo:

```
> dnorm(z, mean = 0, sd = 1)
```

```
2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
   0.35052    0.35604    0.39232    0.36096    0.32080    0.38916
```

xts:

```
> dnorm(x, mean = 0, sd = 1)

               X.1
2009-01-01 0.35052
2009-02-01 0.35604
2009-03-01 0.39232
2009-04-01 0.36096
2009-05-01 0.32080
2009-06-01 0.38916
```

timeSeries:

```
> dnorm(s, mean = 0, sd = 1)

GMT
               S.1
2009-01-01 0.35052
2009-02-01 0.35604
2009-03-01 0.39232
2009-04-01 0.36096
2009-05-01 0.32080
2009-06-01 0.38916
```

Can I use filter like in R's stats package?

The function `filter()` applies linear filtering to a univariate time series or to each series separately of a multivariate time series.

zoo:

```
> print(try(filter(z)))
```

```
[1] "Error in filter(z) : element 1 is empty;\n   the part of the args list of
 'length' being evaluated was:\n   (filter)\n"
attr(,"class")
[1] "try-error"
```

xts:

```
> print(try(filter(x)))
```

```
[1] "Error in filter(x) : element 1 is empty;\n   the part of the args list of
 'length' being evaluated was:\n   (filter)\n"
attr(,"class")
[1] "try-error"
```

timeSeries:

```
> filter(s, rep(1,3))
```

```
GMT
              S.1
2009-01-01     NA
2009-02-01 1.1687
2009-03-01 1.1072
2009-04-01 1.2905
2009-05-01 1.3304
2009-06-01     NA
```

Can I use fivenum like in R's stats package?

The function `fivenum` returns Tukey's five number summary (minimum, lower-hinge, median, upper-hinge, maximum) for the input data.

zoo:

```
> fivenum(z)
```

```
2009-01-01 2009-02-01 2009-05-01 2009-06-01
   0.50875    0.47702    0.66028    0.22283
```

xts:

```
> fivenum(x)

               X.1
2009-01-01 0.50875
2009-02-01 0.47702
2009-05-01 0.66028
2009-06-01 0.22283
```

timeSeries:

```
> fivenum(s)

[1] 0.18291 0.22283 0.46216 0.50875 0.66028
```

Can I use hist like in R's stats package?

The generic function `hist()` computes a histogram of the given data values. If `plot=TRUE`, the resulting object of class `"histogram"` is plotted by `plot.histogram`, before it is returned.

zoo:

```
> hist(z)$density

[1] 1.6667 1.6667 0.0000 3.3333 1.6667 1.6667
```

xts:

```
> hist(x)$density

[1] 1.6667 1.6667 0.0000 3.3333 1.6667 1.6667
```

timeSeries:

```
> hist(s)$density

[1] 1.6667 1.6667 0.0000 3.3333 1.6667 1.6667
```

Can I use lag like in R's stats package?

The function `lag()` computes a lagged version of a time series, shifting the time base back by a given number of observations.

zoo:

```
> lag(Z)

                Z.1     Z.2     Z.3
2009-01-01 0.47702 0.57365 0.39901
2009-02-01 0.18291 0.59127 0.86422
2009-03-01 0.44730 0.96524 0.61359
2009-04-01 0.66028 0.59316 0.49813
2009-05-01 0.22283 0.23193 0.20833
```

xts:

```
> lag(X)

                X.1     X.2     X.3
2009-01-01      NA      NA      NA
2009-02-01 0.50875 0.57402 0.93106
2009-03-01 0.47702 0.57365 0.39901
2009-04-01 0.18291 0.59127 0.86422
2009-05-01 0.44730 0.96524 0.61359
2009-06-01 0.66028 0.59316 0.49813
```

timeSeries:

```
> lag(S)

GMT
             S.1[1]  S.2[1]  S.3[1]
2009-01-01      NA      NA      NA
2009-02-01 0.50875 0.57402 0.93106
2009-03-01 0.47702 0.57365 0.39901
2009-04-01 0.18291 0.59127 0.86422
2009-05-01 0.44730 0.96524 0.61359
2009-06-01 0.66028 0.59316 0.49813
```

Can I use lm like in R's stats package?

The function `lm()` is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although aov may provide a more convenient interface for these).

zoo:

```
> fit <- lm( Z.1 ~ Z.2 + Z.3, data = Z)
> fit

Call:
lm(formula = Z.1 ~ Z.2 + Z.3, data = Z)

Coefficients:
(Intercept)          Z.2          Z.3
      0.274        0.351       -0.110

> resid(fit)

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
  0.135337   0.045015  -0.203938  -0.098638   0.232361  -0.110136
```

xts:

```
> fit <- lm( X.1 ~ X.2 + X.3, data = X)
> fit

Call:
lm(formula = X.1 ~ X.2 + X.3, data = X)

Coefficients:
(Intercept)          X.2          X.3
      0.274        0.351       -0.110

> resid(fit)

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
  0.135337   0.045015  -0.203938  -0.098638   0.232361  -0.110136
```

timeSeries:

```
> fit <- lm( S.1 ~ S.2 + S.3, data = S)
> fit

Call:
lm(formula = S.1 ~ S.2 + S.3, data = S)

Coefficients:
(Intercept)          S.2          S.3
      0.274        0.351       -0.110

> resid(fit)

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
  0.135337   0.045015  -0.203938  -0.098638   0.232361  -0.110136
```

Can I use lowess like in R's stats package?

The function `lowess()` performs the computations for the LOWESS smoother which uses locally-weighted polynomial regression.

zoo:

```
> lowess(z)

$x
[1] 1 2 3 4 5 6

$y
[1] 0.53355 0.40950 0.37472 0.43720 0.41244 0.26854
```

xts:

```
> lowess(x)

$x
[1] 1 2 3 4 5 6

$y
[1] 0.53355 0.40950 0.37472 0.43720 0.41244 0.26854
```

timeSeries:

```
> lowess(s)

$x
[1] 1 2 3 4 5 6

$y
[1] 0.53355 0.40950 0.37472 0.43720 0.41244 0.26854
```

Can I use mad like in R's stats package?

The function `mad()` computes the median absolute deviation, i.e., the (lo-/hi-) median of the absolute deviations from the median, and (by default) adjust by a factor for asymptotically normal consistency.

zoo:

```
> mad(z)

[1] 0.19599
```

xts:

```
> mad(x)

[1] 0.19599
```

timeSeries:

```
> mad(s)

[1] 0.19599
```

Can I use median like in R's stats package?

The function `median()` computes the sample median.

zoo:

```
> median(x)

[1] 0.31510
```

xts:

```
> median(x)

[1] 0.31510
```

timeSeries:

```
> median(s)

[1] 0.31510
```

Can I use qqnorm like in R's stats package?

The function `qqnorm()` is a generic function the default method of which produces a normal QQ plot of the values in `y`. `qqline()` adds a line to a normal quantile-quantile plot which passes through the first and third quartiles.

zoo:

```
> print(qqnorm(z))

$x
[1]  0.64335  0.20189 -1.28155 -0.20189  1.28155 -0.64335

$y
2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
   0.50875    0.47702    0.18291    0.44730    0.66028    0.22283
```

xts:

```
> print(qqnorm(x))

$x
[1]  0.64335  0.20189 -1.28155 -0.20189  1.28155 -0.64335

$y
               X.1
2009-01-01 0.50875
2009-02-01 0.47702
2009-03-01 0.18291
2009-04-01 0.44730
2009-05-01 0.66028
2009-06-01 0.22283
```

timeSeries:

```
> print(qqnorm(s))

$x
[1]  0.64335  0.20189 -1.28155 -0.20189  1.28155 -0.64335

$y
GMT
               S.1
2009-01-01 0.50875
2009-02-01 0.47702
2009-03-01 0.18291
2009-04-01 0.44730
2009-05-01 0.66028
2009-06-01 0.22283
```

Can I use smooth like in R's stats package?

The functions `smooth()` performs running median smoothing. The function implements Tukey's smoothers, 3RS3R, 3RSS, 3R, etc.

zoo:

```
> smooth(z)
```

```
3RS3R Tukey smoother resulting from  smooth(x = z)
 used 1 iterations
[1] 0.50875 0.47702 0.44730 0.44730 0.44730 0.44730
```

xts:

```
> smooth(x)
```

```
3RS3R Tukey smoother resulting from  smooth(x = x)
 used 1 iterations
[1] 0.50875 0.47702 0.44730 0.44730 0.44730 0.44730
```

timeSeries:

```
> smooth(s)
```

```
3RS3R Tukey smoother resulting from  smooth(x = s)
 used 1 iterations
[1] 0.50875 0.47702 0.44730 0.44730 0.44730 0.44730
```

Can I use spectrum like in R's stats package?

The function `spectrum()` estimates the spectral density of a time series.

zoo:

```
> print(try(spectrum(z)[1:2]))
```

```
[1] "Error in na.fail.default(as.ts(x)) : missing values in object\n"
attr(,"class")
[1] "try-error"
```

xts:

```
> print(spectrum(x)[1:2])
```

```
$freq
[1] 0.16667 0.33333 0.50000

$spec
[1] 0.0161880 0.0725889 0.0044029
```

timeSeries:

```
> print(spectrum(s)[1:2])
```

```
$freq
[1] 2 4 6

$spec
[1] 0.00134900 0.00604908 0.00036691
```

3.3 Functions and Methods from utils R

Required R package(s):

```
> library(zoo)
> library(xts)
> library(timeSeries)
```

In the following we test selected functions which are relevant for use in financial time series analysis.

Common Data:

We test them on multivariate

```
> set.seed(1953)
> data <- matrix(runif(18), ncol = 3)
> charvec <- paste("2009-0", 1:6, "-01", sep = "")
> charvec
```

```
[1] "2009-01-01" "2009-02-01" "2009-03-01" "2009-04-01" "2009-05-01"
[6] "2009-06-01"
```

zoo:

```
> Z <- zoo(data, as.Date(charvec))
> colnames(Z) = paste("Z", 1:3, sep = ".")
> Z
```

```
               Z.1     Z.2     Z.3
2009-01-01 0.50875 0.57402 0.93106
2009-02-01 0.47702 0.57365 0.39901
2009-03-01 0.18291 0.59127 0.86422
2009-04-01 0.44730 0.96524 0.61359
2009-05-01 0.66028 0.59316 0.49813
2009-06-01 0.22283 0.23193 0.20833
```

xts:

```
> X <- xts(data, as.Date(charvec))
> colnames(X) = paste("X", 1:3, sep = ".")
> X

                X.1     X.2     X.3
2009-01-01 0.50875 0.57402 0.93106
2009-02-01 0.47702 0.57365 0.39901
2009-03-01 0.18291 0.59127 0.86422
2009-04-01 0.44730 0.96524 0.61359
2009-05-01 0.66028 0.59316 0.49813
2009-06-01 0.22283 0.23193 0.20833
```

timeSeries:

```
> S <- timeSeries(data, charvec)
> colnames(S) = paste("S", 1:3, sep = ".")
> S

GMT
                S.1     S.2     S.3
2009-01-01 0.50875 0.57402 0.93106
2009-02-01 0.47702 0.57365 0.39901
2009-03-01 0.18291 0.59127 0.86422
2009-04-01 0.44730 0.96524 0.61359
2009-05-01 0.66028 0.59316 0.49813
2009-06-01 0.22283 0.23193 0.20833
```

and/or univariate time series objects.

zoo:

```
> z <- Z[, 1]
> z

2009-01-01 2009-02-01 2009-03-01 2009-04-01 2009-05-01 2009-06-01
   0.50875    0.47702    0.18291    0.44730    0.66028    0.22283
```

xts:

```
> x <- X[, 1]
> x
```

```
               X.1
2009-01-01 0.50875
2009-02-01 0.47702
2009-03-01 0.18291
2009-04-01 0.44730
2009-05-01 0.66028
2009-06-01 0.22283
```

timeSeries:

```
> s <- S[, 1]
> s

GMT
               S.1
2009-01-01 0.50875
2009-02-01 0.47702
2009-03-01 0.18291
2009-04-01 0.44730
2009-05-01 0.66028
2009-06-01 0.22283
```

Can I use head like in R's utils package?

The function `head()` returns the first or last parts of a vector, matrix, table, data frame or function.

ts:

«head.ts» ts <- rnorm(ts(rnorm(12))) ts head(ts)

For a regular time series of class `ts`, the attributes are lost.

zoo:

```
> head(Z, 3)

               Z.1     Z.2     Z.3
2009-01-01 0.50875 0.57402 0.93106
```

```
2009-02-01 0.47702 0.57365 0.39901
2009-03-01 0.18291 0.59127 0.86422
```

```
> head(z, 3)
```

```
2009-01-01 2009-02-01 2009-03-01
   0.50875    0.47702    0.18291
```

xts:

```
> head(X, 3)
```

```
               X.1     X.2     X.3
2009-01-01 0.50875 0.57402 0.93106
2009-02-01 0.47702 0.57365 0.39901
2009-03-01 0.18291 0.59127 0.86422
```

```
> head(x, 3)
```

```
               X.1
2009-01-01 0.50875
2009-02-01 0.47702
2009-03-01 0.18291
```

timeSeries:

```
> head(S, 3)
```

```
GMT
               S.1     S.2     S.3
2009-01-01 0.50875 0.57402 0.93106
2009-02-01 0.47702 0.57365 0.39901
2009-03-01 0.18291 0.59127 0.86422
```

```
> head(s, 3)
```

```
GMT
               S.1
2009-01-01 0.50875
2009-02-01 0.47702
2009-03-01 0.18291
```

Chapter 4
Performance Measurements

4.1 Performance of Time Series Creation

Required R package(s):

```
> library(zoo)
> library(xts)
> library(timeSeries)
```

We define the following function to measure the performance of the different functions with

```
> systemTime <- function(expr, gcFirst = TRUE, n = 20) {
      time <- sapply(integer(n), eval.parent(substitute(function(...)
          system.time(expr, gcFirst = gcFirst))))
      structure(apply(time, 1, median), class = "proc_time")
  }
```

How long does it take to create a one hundred year 5-column series with daily records from character time stamps?

If we read time series information from an ASCII, or extract it from a downloaded html file from the Web, or import it from a xls or csv file then we have the time usually as character strings. To convert this to a zoo, xts or timeSeries object, one has to transform the character time stamps in an appropriate index. Here we consider abot 35'000 daily records which represent about 100 year of daily data.

Common Data:

```
> charvec <- format(seq(from = as.Date("1901-01-01"),
      to = as.Date("1999-12-31"), by = "day"))
> data <- matrix(rnorm(length(charvec)*5), ncol = 5)
```

zoo:

```
> systemTime(zoo(data, charvec))

   user  system elapsed
   0.25    0.00    0.25
```

xts:

```
> print(try(xts(data, charvec)))
```

```
[1] "Error in xts(data, charvec) : \n  order.by requires an appropriate time-
based object\n"
attr(,"class")
[1] "try-error"
```

timeSeries:

```
> systemTime(timeSeries(data, charvec))
```

```
   user  system elapsed
  0.230   0.000   0.235
```

How long does it take to create a one hundred year 5-column series with daily records from Date time stamps?

Common Data:

```
> charvec <- format(seq(from = as.Date("1901-01-01"),
      to = as.Date("1999-12-31"), by = "day"))
> data <- matrix(rnorm(length(charvec)*5), ncol = 5)
> index <- as.Date(charvec)
> class(index)
```

```
[1] "Date"
```

zoo:

```
> systemTime(zoo(data, index))
```

```
   user  system elapsed
   0.03    0.00    0.03
```

This is the generic case to create daily recorded time series with `zoo`.

xts:

```
> systemTime(xts(data, index))

   user  system elapsed
   0.42    0.00    0.43
```

This is the generic case to create daily recorded time series with `xts`.

timeSeries:

```
> systemTime(timeSeries(data, index))

   user  system elapsed
   0.03    0.00    0.03
```

Since we are always working with `timeDate` object, the following test is the proper benchmark.

How long does it take to create a one hundred year 5-column series with daily records from GMT POSIXct time stamps?

Common Data:

```
> charvec <- format(seq(from = as.Date("1901-01-01"),
     to = as.Date("1999-12-31"), by = "day"))
> data <- matrix(rnorm(length(charvec)*5), ncol = 5)
> index <- as.POSIXct(charvec, tz = "GMT")
> class(index)

[1] "POSIXt"  "POSIXct"
```

zoo:

```
> systemTime(zoo(data, index))

   user  system elapsed
   0.03    0.00    0.03
```

xts:

```
> systemTime(xts(data, index))
```

```
   user  system elapsed
      0       0       0
```

timeSeries:

```
> systemTime(timeSeries(data, index))
```

```
   user  system elapsed
  0.020   0.000   0.025
```

How long does it take to create a one hundred year 5-column series with daily records from non-GMT POSIXct time stamps?

Common Data:

```
> charvec <- format(seq(from = as.Date("1901-01-01"),
     to = as.Date("1999-12-31"), by = "day"))
> data <- matrix(rnorm(length(charvec)*5), ncol = 5)
> index <- as.POSIXct(charvec, tz = "CET")
> class(index)
```

```
[1] "POSIXt"  "POSIXct"
```

zoo:

```
> systemTime(zoo(data, index))
```

```
   user  system elapsed
   0.03    0.00    0.03
```

xts:

```
> systemTime(xts(data, index))
```

```
   user  system elapsed
      0       0       0
```

timeSeries:

```
> systemTime(timeSeries(data, index))

   user  system elapsed
  0.025   0.000   0.030
```

How long does it take to create a one hundred year 5-column series with daily records from timeDate time stamps?

Common Data:

```
> charvec <- format(seq(from = as.Date("1901-01-01"),
      to = as.Date("1999-12-31"), by = "day"))
> data <- matrix(rnorm(length(charvec)*5), ncol = 5)
> index <- timeDate(charvec)
> class(index)

[1] "timeDate"
attr(,"package")
[1] "timeDate"
```

zoo:

```
> systemTime(zoo(data, index))

   user  system elapsed
   0.03    0.00    0.03
```

xts:

```
> systemTime(xts(data, index))

   user  system elapsed
   0.01    0.00    0.01
```

timeSeries:

```
> systemTime(timeSeries(data, index))
```

```
   user  system elapsed
      0       0       0
```

4.2 Performance of Time Series subsetting

Required R package(s):

```
> library(zoo)
> library(xts)
> library(timeSeries)
```

We define the following function to measure the performance of the different functions with

```
> systemTime <- function(expr, gcFirst = TRUE, n = 20) {
     time <- sapply(integer(n), eval.parent(substitute(function(...)
         system.time(expr, gcFirst = gcFirst))))
     structure(apply(time, 1, median), class = "proc_time")
 }
```

How long does it take to subset by integers a one hundred year 5-column series with daily records?

Common data:

```
> charvec <- format(seq(from = as.Date("1901-01-01"),
     to = as.Date("1999-12-31"), by = "day"))
> data <- matrix(rnorm(length(charvec)*5), ncol = 5)
> index <- charvec
> length <- floor(length(charvec)/2)
> subset <- sample(charvec)[1:length]
```

zoo:

```
> z <- zoo(data, as.Date(charvec))
> systemTime(z[subset, ])
```

```
   user  system elapsed
   0.14    0.00    0.14
```

xts:

```
> x <- xts(data, as.Date(charvec))
```

This cannot be done in a reasonable amount of time!

timeSeries:

```
> s <- timeSeries(data, charvec)
> systemTime(s[subset, ])

   user  system elapsed
   0.13    0.00    0.13
```

How long does it take to subset by Date objects a one hundred year 5-column series with daily records?

Common data:

```
> charvec <- format(seq(from = as.Date("1901-01-01"),
     to = as.Date("1999-12-31"), by = "day"))
> data <- matrix(rnorm(length(charvec)*5), ncol = 5)
> index <- as.Date(charvec)
> length <- floor(length(charvec)/2)
> subset <- as.Date(sample(charvec)[1:length])
```

zoo:

```
> z <- zoo(data, index)
> systemTime(z[subset, ])

   user  system elapsed
   0.03    0.00    0.03
```

xts:

```
> x <- xts(data, index)
```

This cannot be done in a reasonable amount of time!

timeSeries:

```
> s <- timeSeries(data, index)
> systemTime(s[subset, ])

   user  system elapsed
   0.02    0.00    0.02
```

How long does it take to subset by Date objects a one hundred year 5-column series with GMT POSIXct records?

Common data:

```
> charvec <- format(seq(from = as.Date("1901-01-01"),
     to = as.Date("1999-12-31"), by = "day"))
> data <- matrix(rnorm(length(charvec)*5), ncol = 5)
> index <- as.POSIXct(charvec, tz = "GMT")
> length <- floor(length(charvec)/2)
> subset <- as.POSIXct(sample(charvec)[1:length], tz = "GMT")
```

zoo:

```
> z <- zoo(data, index)
> systemTime(z[subset, ])

   user  system elapsed
   0.03    0.00    0.03
```

xts:

```
> x <- xts(data, index)
```

This cannot be done in a reasonable amount of time!

timeSeries:

```
> s <- timeSeries(data, index)
> systemTime(s[subset, ])
```

```
   user  system elapsed 
   0.02    0.00    0.02 
```

How long does it take to subset by Date objects a one hundred year 5-column series with non-GMT POSIXct records?

Common data:

```
> charvec <- format(seq(from = as.Date("1901-01-01"),
     to = as.Date("1999-12-31"), by = "day"))
> data <- matrix(rnorm(length(charvec)*5), ncol = 5)
> index <- as.POSIXct(charvec, tz = "CET")
> length <- floor(length(charvec)/2)
> subset <- as.POSIXct(sample(charvec)[1:length], tz = "GMT")
```

zoo:

```
> z <- zoo(data, index)
> systemTime(z[subset, ])
```

```
   user  system elapsed 
   0.01    0.00    0.01 
```

xts:

```
> x <- xts(data, index)
```

This cannot be done in a reasonable amount of time!

timeSeries:

```
> s <- timeSeries(data, index)
```

Here is a problem ?

How long does it take to subset by Date objects a one hundred year 5-column series with GMT timeDate records?

Common data:

```
> charvec <- format(seq(from = as.Date("1901-01-01"),
     to = as.Date("1999-12-31"), by = "day"))
> data <- matrix(rnorm(length(charvec)*5), ncol = 5)
> index <- timeDate(charvec)
> length <- floor(length(charvec)/2)
> subset <- timeDate(sample(charvec)[1:length])
```

zoo:

```
> z <- zoo(data, index)
> systemTime(z[subset, ])

   user  system elapsed
   0.03    0.00    0.03

> try(detach("package:fCalendar"))
```

Note that `zoo` attaches old version `timeDate()` from package `fCalendar`.

xts:

This cannot be done in a reasonable amount of time!

timeSeries:

```
> s <- timeSeries(data, index)
> systemTime(s[subset, ])

   user  system elapsed
   0.02    0.00    0.01
```

How long does it take to subset by Date objects a one hundred year 5-column series with non-GMT timeDate records?

Common data:

```
> charvec <- format(seq(from = as.Date("1901-01-01"),
     to = as.Date("1999-12-31"), by = "day"))
> data <- matrix(rnorm(length(charvec)*5), ncol = 5)
> index <- timeDate(charvec, zone = "Zurich", FinCenter = "Zurich")
> length <- floor(length(charvec)/2)
> subset <- timeDate(sample(charvec)[1:length], zone = "Zurich", FinCenter = "
Zurich")
```

zoo:

```
> z <- zoo(data, index)
> systemTime(z[subset, ])
```

```
   user  system elapsed
   0.03    0.00    0.03
```

xts:

```
> x <- xts(data, index)
> #systemTime(x[subset, ])
> NA
```

```
[1] NA
```

timeSeries:

```
> s <- timeSeries(data, index)
> systemTime(s[subset, ])
```

```
   user  system elapsed
   0.02    0.00    0.01
```

Appendix A
Packages Required for this Ebook

Required R package(s):

```
> library(fBasics)
```

Contributed R Package: zoo

The description file of the `zoo` package used in this version of the ebook.

```
> listDescription(zoo)

 zoo Description:

  Package:            zoo
  Version:            1.5-8
  Date:               2009-07-22
  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. zoo's key design
                        goals are independence of a particular
                        index/date/time class and consistency with ts
                        and base R by providing methods to extend
                        standard generics.
  Depends:            R (>= 2.4.1), stats
```

```
Suggests:           coda, chron, DAAG, fame, fCalendar, fSeries,
                      fts, its, lattice, strucchange, timeDate,
                      timeSeries, tseries, xts
Imports:            stats, utils, graphics, grDevices, lattice
LazyLoad:           yes
License:            GPL-2
URL:                http://R-Forge.R-project.org/projects/zoo/
Packaged:           2009-07-22 17:21:04 UTC; zeileis
Repository:         CRAN
Date/Publication:   2009-07-22 19:20:13
Built:              R 2.9.1; ; 2009-08-23 16:38:13 UTC; windows
```

Contributed R Package: xts

The description file of the `xts` package used in this version of the ebook.

```
> listDescription(xts)

 xts Description:

  Package:            xts
  Type:               Package
  Title:              Extensible Time Series
  Version:            0.6-7
  Date:               2009-07-21
  Author:             Jeffrey A. Ryan, Josh M. Ulrich
  Depends:            zoo
  Suggests:           timeSeries,timeDate,tseries,its,chron,fts,tis
  LazyLoad:           yes
  Maintainer:         Jeffrey A. Ryan <jeff.a.ryan@gmail.com>
  Description:        Provide for uniform handling of R's different
                        time-based data classes by extending zoo,
                        maximizing native format information
                        preservation and allowing for user level
                        customization and extension, while simplifying
                        cross-class interoperability.
  License:            GPL-3
  URL:                http://r-forge.r-project.org/projects/xts/
  Packaged:           2009-07-21 20:27:23 UTC; jryan
  Repository:         CRAN
  Date/Publication:   2009-07-22 04:55:21
  Built:              R 2.9.1; i386-pc-mingw32; 2009-08-23 16:39:28
                        UTC; windows
```

Rmetrics Package: timeDate

The description file of the `timeDate` package used in this version of the ebook.

```
> listDescription(timeDate)

 timeDate Description:

  Package:       timeDate
  Version:       2100.86
  Revision:      4054
  Date:          2009-04-15
  Title:         Rmetrics - Chronological and Calendarical Objects
  Author:        Diethelm Wuertz and Yohan Chalabi with
                   contributions from Martin Maechler, Joe W. Byers,
                   and others
  Depends:       R (>= 2.6.0), graphics, utils, stats, methods
  Suggests:      RUnit
  Maintainer:    Rmetrics Core Team <Rmetrics-core@r-project.org>
  Description:   Environment for teaching "Financial Engineering and
                   Computational Finance"
  NOTE:          SEVERAL PARTS ARE STILL PRELIMINARY AND MAY BE
                   CHANGED IN THE FUTURE. THIS TYPICALLY INCLUDES
                   FUNCTION AND ARGUMENT NAMES, AS WELL AS DEFAULTS
                   FOR ARGUMENTS AND RETURN VALUES.
  LazyLoad:      yes
  LazyData:      yes
  License:       GPL (>= 2)
  URL:           http://www.rmetrics.org
  Built:         R 2.9.1; ; 2009-08-21 13:24:27 UTC; windows
```

Rmetrics Package: timeSeries

The description file of the `timeSeries` package used in this version of the ebook.

```
> listDescription(timeSeries)

 timeSeries Description:

  Package:       timeSeries
  Version:       2100.84
  Revision:      4093
  Date:          2009-04-19
```

```
Title:          Rmetrics - Financial Time Series Objects
Author:         Diethelm Wuertz and Yohan Chalabi
Depends:        R (>= 2.6.0), graphics, grDevices, methods, stats,
                  utils, timeDate (>= 2100.86)
Suggests:       robustbase, RUnit
Maintainer:     Rmetrics Core Team <Rmetrics-core@r-project.org>
Description:    Environment for teaching "Financial Engineering and
                  Computational Finance"
NOTE:           SEVERAL PARTS ARE STILL PRELIMINARY AND MAY BE
                  CHANGED IN THE FUTURE. THIS TYPICALLY INCLUDES
                  FUNCTION AND ARGUMENT NAMES, AS WELL AS DEFAULTS
                  FOR ARGUMENTS AND RETURN VALUES.
LazyLoad:       yes
LazyData:       yes
License:        GPL (>= 2)
URL:            http://www.rmetrics.org
Built:          R 2.9.1; ; 2009-08-21 13:29:41 UTC; windows
```

Appendix B
Function Listings

Required R package(s):

```
> library(zoo)
> library(xts)
> library(timeSeries)
> library(fBasics)
```

Contributed R Package: zoo

The functions available from the zoo package used in this version of the ebook.

```
> listFunctions(zoo)
```

```
 [1] "as.Date.numeric"    "as.yearmon"         "as.yearmon.default"
 [4] "as.yearqtr"         "as.yearqtr.default" "as.yearqtr.yearqtr"
 [7] "as.zoo"             "as.zoo.default"     "as.zooreg"
[10] "as.zooreg.default"  "cbind.zoo"          "coredata"
[13] "coredata.default"   "coredata<-"         "format.yearqtr"
[16] "frequency<-"        "index"              "index<-"
[19] "index2char"         "is.regular"         "is.zoo"
[22] "make.par.list"      "MATCH"              "MATCH.default"
[25] "merge.zoo"          "na.approx"          "na.approx.default"
[28] "na.locf"            "na.locf.default"    "na.spline"
[31] "na.spline.default"  "na.trim"            "na.trim.default"
[34] "ORDER"              "ORDER.default"      "panel.arrows.its"
[37] "panel.arrows.tis"   "panel.arrows.ts"    "panel.arrows.zoo"
```

```
[40] "panel.lines.its"     "panel.lines.tis"     "panel.lines.ts"
[43] "panel.lines.zoo"     "panel.plot.custom"   "panel.plot.default"
[46] "panel.points.its"    "panel.points.tis"    "panel.points.ts"
[49] "panel.points.zoo"    "panel.polygon.its"   "panel.polygon.tis"
[52] "panel.polygon.ts"    "panel.polygon.zoo"   "panel.rect.its"
[55] "panel.rect.tis"      "panel.rect.ts"       "panel.rect.zoo"
[58] "panel.segments.its"  "panel.segments.tis"  "panel.segments.ts"
[61] "panel.segments.zoo"  "panel.text.its"      "panel.text.tis"
[64] "panel.text.ts"       "panel.text.zoo"      "plot.zoo"
[67] "rbind.zoo"           "read.zoo"            "rollapply"
[70] "rollmax"             "rollmax.default"     "rollmean"
[73] "rollmean.default"    "rollmedian"          "rollmedian.default"
[76] "Sys.yearmon"         "Sys.yearqtr"         "time<-"
[79] "write.zoo"           "xtfrm.zoo"           "yearmon"
[82] "yearqtr"             "zoo"                 "zooreg"
```

Contributed R Package: xts

The functions available from the `xts` package used in this version of the ebook.

```
> listFunctions(xts)
```

```
 [1] ".index"            ".index<-"          ".indexDate"
 [4] ".indexday"         ".indexhour"        ".indexisdst"
 [7] ".indexmday"        ".indexmin"         ".indexmon"
[10] ".indexsec"         ".indexwday"        ".indexweek"
[13] ".indexyday"        ".indexyear"        ".subset.xts"
[16] ".xts"              "apply.daily"       "apply.monthly"
[19] "apply.quarterly"   "apply.weekly"      "apply.yearly"
[22] "as.fts.xts"        "as.xts"            "axTicksByTime"
[25] "c.xts"             "cbind.xts"         "CLASS"
[28] "CLASS<-"           "convertIndex"      "diff.xts"
[31] "dimnames.xts"      "dimnames<-.xts"    "endpoints"
[34] "first"             "firstof"           "indexClass"
[37] "indexClass<-"      "indexFormat"       "indexFormat<-"
[40] "indexTZ"           "is.timeBased"      "is.xts"
[43] "isOrdered"         "lag.xts"           "last"
[46] "lastof"            "lines.xts"         "merge.xts"
[49] "ndays"             "nhours"            "nminutes"
[52] "nmonths"           "nquarters"         "nseconds"
[55] "nweeks"            "nyears"            "period.apply"
[58] "period.max"        "period.min"        "period.prod"
[61] "period.sum"        "periodicity"       "plot.xts"
```

```
[64] "rbind.xts"       "reclass"         "Reclass"
[67] "split.xts"       "timeBased"       "timeBasedRange"
[70] "timeBasedSeq"    "to.daily"        "to.hourly"
[73] "to.minutes"      "to.minutes10"    "to.minutes15"
[76] "to.minutes3"     "to.minutes30"    "to.minutes5"
[79] "to.monthly"      "to.period"       "to.quarterly"
[82] "to.weekly"       "to.yearly"       "try.xts"
[85] "use.reclass"     "use.xts"         "xcoredata"
[88] "xcoredata<-"     "xts"             "xtsAttributes"
[91] "xtsAttributes<-" "xtsible"
```

Rmetrics Package: timeDate

The functions available from the `timeDate` package used in this version of the ebook.

```
> listFunctions(timeDate)

  [1] ".by2seconds"              ".day.of.week"
  [3] ".easter"                  ".easterSunday"
  [5] ".endpoints"               ".fjulian"
  [7] ".formatFinCenter"         ".formatFinCenterNum"
  [9] ".genDaylightSavingTime"   ".isPOSIX"
 [11] ".julian"                  ".JULIAN"
 [13] ".last.of.nday"            ".midnightStandard"
 [15] ".month.day.year"          ".monthlyRolling"
 [17] ".nth.of.nday"             ".on.or.after"
 [19] ".on.or.before"            ".periodicallyRolling"
 [21] ".periods"                 ".sdate"
 [23] ".sday.of.week"            ".sjulian"
 [25] ".sleap.year"              ".subsetByPython"
 [27] ".subsetBySpan"            ".subsetCode"
 [29] ".whichFormat"             "Abidjan"
 [31] "abline"                   "Accra"
 [33] "Adak"                     "Addis_Ababa"
 [35] "Adelaide"                 "Aden"
 [37] "Advent1st"                "Advent2nd"
 [39] "Advent3rd"                "Advent4th"
 [41] "Algiers"                  "align"
 [43] "AllSaints"                "AllSouls"
 [45] "Almaty"                   "Amman"
 [47] "Amsterdam"                "Anadyr"
 [49] "Anchorage"                "Andorra"
 [51] "Anguilla"                 "Annunciation"
 [53] "Antananarivo"                 "Antigua"
 [55] "Apia"                         "Aqtau"
 [57] "Aqtobe"                       "Araguaina"
 [59] "Aruba"                        "as.timeDate"
 [61] "Ascension"                    "Ashgabat"
 [63] "AshWednesday"                 "Asmara"
 [65] "AssumptionOfMary"             "Asuncion"
 [67] "Athens"                       "Atikokan"
 [69] "atoms"                        "Auckland"
 [71] "axis.timeDate"                "Azores"
 [73] "Baghdad"                      "Bahia"
 [75] "Bahrain"                      "Baku"
 [77] "Bamako"                       "Bangkok"
 [79] "Bangui"                       "Banjul"
 [81] "Barbados"                     "Beirut"
 [83] "Belem"                        "Belgrade"
 [85] "Belize"                       "Berlin"
 [87] "Bermuda"                      "BirthOfVirginMary"
 [89] "Bishkek"                      "Bissau"
 [91] "Blanc-Sablon"                 "Blantyre"
 [93] "blockEnd"                     "blockStart"
 [95] "Boa_Vista"                    "Bogota"
 [97] "Boise"                        "BoxingDay"
 [99] "Bratislava"                   "Brazzaville"
[101] "Brisbane"                     "Broken_Hill"
[103] "Brunei"                       "Brussels"
[105] "Bucharest"                    "Budapest"
[107] "Buenos_Aires"                 "BuenosAires"
[109] "Bujumbura"                    "CACanadaDay"
[111] "CACivicProvincialHoliday"     "CAFamilyDay"
[113] "Cairo"                        "CALabourDay"
[115] "Calcutta"                     "Cambridge_Bay"
[117] "Campo_Grande"                 "Canary"
[119] "Cancun"                       "Cape_Verde"
[121] "Caracas"                      "CaRemembranceDay"
[123] "Casablanca"                   "Casey"
[125] "Catamarca"                    "CAThanksgivingDay"
[127] "CAVictoriaDay"                "Cayenne"
[129] "Cayman"                       "CelebrationOfHolyCross"
[131] "Center"                       "Ceuta"
[133] "Chagos"                       "CHAscension"
[135] "Chatham"                      "CHBerchtoldsDay"
[137] "CHConfederationDay"           "Chicago"
[139] "Chihuahua"                    "Chisinau"
[141] "CHKnabenschiessen"            "Choibalsan"
[143] "Chongqing"                    "Christmas"
```

```
[145] "ChristmasDay"            "ChristmasEve"
[147] "ChristTheKing"           "CHSechselaeuten"
[149] "Cocos"                   "Colombo"
[151] "Comoro"                  "Conakry"
[153] "Copenhagen"              "Cordoba"
[155] "CorpusChristi"           "Costa_Rica"
[157] "Cuiaba"                  "Curacao"
[159] "Currie"                  "Dakar"
[161] "Damascus"                "Danmarkshavn"
[163] "Dar_es_Salaam"           "Darwin"
[165] "Davis"                   "Dawson"
[167] "Dawson_Creek"            "dayOfWeek"
[169] "dayOfYear"               "DEAscension"
[171] "DEChristmasEve"          "DECorpusChristi"
[173] "DEGermanUnity"           "DENewYearsEve"
[175] "Denver"                  "Detroit"
[177] "Dhaka"                   "difftimeDate"
[179] "Dili"                    "Djibouti"
[181] "Dominica"                "Douala"
[183] "Dubai"                   "Dublin"
[185] "DumontDUrville"          "Dushanbe"
[187] "Easter"                  "EasterMonday"
[189] "Eastern"                 "EasterSunday"
[191] "Edmonton"                "Efate"
[193] "Eirunepe"                "El_Aaiun"
[195] "El_Salvador"             "Enderbury"
[197] "Epiphany"                "Eucla"
[199] "Fakaofo"                 "Faroe"
[201] "Fiji"                    "finCenter"
[203] "finCenter<-"             "Fortaleza"
[205] "FRAllSaints"             "Frankfurt"
[207] "FRArmisticeDay"          "FRAscension"
[209] "FRAssumptionVirginMary"  "FRBastilleDay"
[211] "Freetown"                "FRFetDeLaVictoire1945"
[213] "Funafuti"                "Gaborone"
[215] "Galapagos"               "Gambier"
[217] "Gaza"                    "GBBankHoliday"
[219] "GBMayDay"                "GBMilleniumDay"
[221] "GBSummerBankHoliday"     "getRmetricsOptions"
[223] "Gibraltar"               "Glace_Bay"
[225] "Godthab"                 "GoodFriday"
[227] "Goose_Bay"               "Grand_Turk"
[229] "Grenada"                 "Guadalcanal"
[231] "Guadeloupe"              "Guam"
[233] "Guatemala"               "Guayaquil"
[235] "Guernsey"                "Guyana"
```

```
[237] "Halifax"                    "Harare"
[239] "Harbin"                     "Havana"
[241] "Helsinki"                   "Hermosillo"
[243] "Hobart"                     "holiday"
[245] "holidayNERC"                "holidayNYSE"
[247] "holidayTSX"                 "holidayZURICH"
[249] "Hong_Kong"                  "HongKong"
[251] "Honolulu"                   "Hovd"
[253] "Indianapolis"               "Inuvik"
[255] "Iqaluit"                    "Irkutsk"
[257] "isBizday"                   "isHoliday"
[259] "Isle_of_Man"                "Istanbul"
[261] "isWeekday"                  "isWeekend"
[263] "ITAllSaints"                "ITAssumptionOfVirginMary"
[265] "ITEpiphany"                 "ITImmaculateConception"
[267] "ITLiberationDay"            "ITStAmrose"
[269] "Jakarta"                    "Jamaica"
[271] "Jayapura"                   "Jersey"
[273] "Jerusalem"                  "Johannesburg"
[275] "Johnston"                   "JPAutumnalEquinox"
[277] "JPBankHolidayDec31"         "JPBankHolidayJan2"
[279] "JPBankHolidayJan3"          "JPBunkaNoHi"
[281] "JPChildrensDay"             "JPComingOfAgeDay"
[283] "JPConstitutionDay"          "JPEmperorsBirthday"
[285] "JPGantan"                   "JPGreeneryDay"
[287] "JPHealthandSportsDay"       "JPKeirouNOhi"
[289] "JPKenkokuKinenNoHi"         "JPKenpouKinenBi"
[291] "JPKinrouKanshaNoHi"         "JPKodomoNoHi"
[293] "JPKokuminNoKyujitu"         "JPMarineDay"
[295] "JPMidoriNoHi"               "JPNatFoundationDay"
[297] "JPNationalCultureDay"       "JPNationHoliday"
[299] "JPNewYearsDay"              "JPRespectForTheAgedDay"
[301] "JPSeijinNoHi"               "JPShuubunNoHi"
[303] "JPTaiikuNoHi"               "JPTennouTanjyouBi"
[305] "JPThanksgivingDay"          "JPUmiNoHi"
[307] "JPVernalEquinox"            "Jujuy"
[309] "julian"                     "Juneau"
[311] "Kabul"                      "Kaliningrad"
[313] "Kamchatka"                  "Kampala"
[315] "Karachi"                    "Kashgar"
[317] "Katmandu"                   "Kerguelen"
[319] "Khartoum"                   "Kiev"
[321] "Kigali"                     "Kinshasa"
[323] "Kiritimati"                 "Knox"
[325] "Kosrae"                     "Krasnoyarsk"
[327] "Kuala_Lumpur"               "KualaLumpur"
```

```
[329] "Kuching"              "kurtosis"
[331] "Kuwait"               "Kwajalein"
[333] "La_Paz"               "La_Rioja"
[335] "LaborDay"             "Lagos"
[337] "Libreville"           "Lima"
[339] "Lindeman"             "lines"
[341] "Lisbon"               "listFinCenter"
[343] "listHolidays"         "Ljubljana"
[345] "Lome"                 "London"
[347] "Longyearbyen"         "Lord_Howe"
[349] "Los_Angeles"          "LosAngeles"
[351] "Louisville"           "Luanda"
[353] "Lubumbashi"           "Lusaka"
[355] "Luxembourg"           "Macau"
[357] "Maceio"               "Madeira"
[359] "Madrid"               "Magadan"
[361] "Mahe"                 "Majuro"
[363] "Makassar"             "Malabo"
[365] "Maldives"             "Malta"
[367] "Managua"              "Manaus"
[369] "Manila"               "Maputo"
[371] "Marengo"              "Mariehamn"
[373] "Marigot"              "Marquesas"
[375] "Martinique"           "Maseru"
[377] "MassOfArchangels"     "Mauritius"
[379] "Mawson"               "Mayotte"
[381] "Mazatlan"             "Mbabane"
[383] "McMurdo"              "Melbourne"
[385] "Mendoza"              "Menominee"
[387] "Merida"               "Mexico_City"
[389] "MexicoCity"           "midnightStandard"
[391] "Midway"               "Minsk"
[393] "Miquelon"             "Mogadishu"
[395] "Monaco"               "Moncton"
[397] "Monrovia"             "Monterrey"
[399] "Montevideo"           "monthlyRolling"
[401] "months"               "Monticello"
[403] "Montreal"             "Montserrat"
[405] "Moscow"               "Muscat"
[407] "Nairobi"              "Nassau"
[409] "Nauru"                "Ndjamena"
[411] "New_Salem"            "New_York"
[413] "NewYearsDay"          "NewYork"
[415] "Niamey"               "Nicosia"
[417] "Nipigon"              "Niue"
[419] "Nome"                 "Norfolk"
```

```
[421] "Noronha"                    "Nouakchott"
[423] "Noumea"                     "Novosibirsk"
[425] "Omsk"                       "Oral"
[427] "Oslo"                       "Ouagadougou"
[429] "Pacific"                    "Pago_Pago"
[431] "Palau"                      "Palmer"
[433] "PalmSunday"                 "Panama"
[435] "Pangnirtung"                "Paramaribo"
[437] "Paris"                      "Pentecost"
[439] "PentecostMonday"            "periodicallyRolling"
[441] "periods"                    "Perth"
[443] "Petersburg"                 "Phnom_Penh"
[445] "Phoenix"                    "Pitcairn"
[447] "plot"                       "Podgorica"
[449] "points"                     "Ponape"
[451] "Pontianak"                  "Port-au-Prince"
[453] "Port_Moresby"               "Port_of_Spain"
[455] "Porto-Novo"                 "Porto_Velho"
[457] "Prague"                     "PresentationOfLord"
[459] "Puerto_Rico"                "Pyongyang"
[461] "Qatar"                      "Quinquagesima"
[463] "Qyzylorda"                  "Rainy_River"
[465] "Rangoon"                    "Rankin_Inlet"
[467] "Rarotonga"                  "Recife"
[469] "Regina"                     "Resolute"
[471] "Reunion"                    "Reykjavik"
[473] "Riga"                       "Rio_Branco"
[475] "Rio_Gallegos"               "Riyadh"
[477] "RogationSunday"             "Rome"
[479] "Rothera"                    "rulesFinCenter"
[481] "Saigon"                     "Saipan"
[483] "Sakhalin"                   "Samara"
[485] "Samarkand"                  "sample"
[487] "San_Juan"                   "San_Marino"
[489] "Santiago"                   "Santo_Domingo"
[491] "Sao_Paulo"                  "Sao_Tome"
[493] "Sarajevo"                   "Scoresbysund"
[495] "Seoul"                      "Septuagesima"
[497] "setRmetricsOptions"         "Shanghai"
[499] "Shiprock"                   "Simferopol"
[501] "Singapore"                  "skewness"
[503] "Skopje"                     "Sofia"
[505] "SolemnityOfMary"            "South_Georgia"
[507] "South_Pole"                 "St_Barthelemy"
[509] "St_Helena"                  "St_Johns"
[511] "St_Kitts"                   "St_Lucia"
```

```
[513] "St_Thomas"                "St_Vincent"
[515] "Stanley"                  "Stockholm"
[517] "strptimeDate"             "Swift_Current"
[519] "Sydney"                   "Syowa"
[521] "Sys.timeDate"             "Tahiti"
[523] "Taipei"                   "Tallinn"
[525] "Tarawa"                   "Tashkent"
[527] "Tbilisi"                  "Tegucigalpa"
[529] "Tehran"                   "Tell_City"
[531] "Thimphu"                  "Thule"
[533] "Thunder_Bay"              "Tijuana"
[535] "timeCalendar"             "timeDate"
[537] "timeFirstDayInMonth"      "timeFirstDayInQuarter"
[539] "timeLastDayInMonth"       "timeLastDayInQuarter"
[541] "timeLastNdayInMonth"      "timeNdayOnOrAfter"
[543] "timeNdayOnOrBefore"       "timeNthNdayInMonth"
[545] "timeSequence"             "Tirane"
[547] "Tokyo"                    "Tongatapu"
[549] "Toronto"                  "Tortola"
[551] "TransfigurationOfLord"    "TrinitySunday"
[553] "Tripoli"                  "Truk"
[555] "Tucuman"                  "Tunis"
[557] "Ulaanbaatar"              "Urumqi"
[559] "USChristmasDay"           "USColumbusDay"
[561] "USCPulaskisBirthday"      "USDecorationMemorialDay"
[563] "USElectionDay"            "USGoodFriday"
[565] "Ushuaia"                  "USInaugurationDay"
[567] "USIndependenceDay"        "USLaborDay"
[569] "USLincolnsBirthday"       "USMemorialDay"
[571] "USMLKingsBirthday"        "USNewYearsDay"
[573] "USPresidentsDay"          "USThanksgivingDay"
[575] "USVeteransDay"            "USWashingtonsBirthday"
[577] "Uzhgorod"                 "Vaduz"
[579] "Vancouver"                "Vatican"
[581] "Vevay"                    "Vienna"
[583] "Vientiane"                "Vilnius"
[585] "Vincennes"                "Vladivostok"
[587] "Volgograd"                "Vostok"
[589] "Wake"                     "Wallis"
[591] "Warsaw"                   "whichFormat"
[593] "Whitehorse"               "Winamac"
[595] "Windhoek"                 "Winnipeg"
[597] "Yakutat"                  "Yakutsk"
[599] "Yekaterinburg"            "Yellowknife"
[601] "Yerevan"                  "Zagreb"
[603] "Zaporozhye"               "Zurich"
```

Rmetrics Package: timeSeries

The functions available from the `timeSeries` package used in this version of the ebook.

```
> listFunctions(timeSeries)

 [1] ".aggregate.timeSeries"      ".align.timeSeries"
 [3] ".applySeries"               ".as.data.frame.timeSeries"
 [5] ".as.list.timeSeries"        ".as.matrix.timeSeries"
 [7] ".as.ts.timeSeries"          ".cut.timeSeries"
 [9] ".description"               ".diff.timeSeries"
[11] ".dollar_assign"             ".end.timeSeries"
[13] ".endOfPeriodBenchmarks"     ".endOfPeriodSeries"
[15] ".endOfPeriodStats"          ".extract.turnpointsPastecs"
[17] ".fapply"                    ".findIndex"
[19] ".frequency.timeDate"        ".frequency.timeSeries"
[21] ".head.timeSeries"           ".isDaily"
[23] ".isDaily.default"           ".isDaily.timeDate"
[25] ".isDaily.timeSeries"        ".isMonthly"
[27] ".isMonthly.default"         ".isMonthly.timeDate"
[29] ".isMonthly.timeSeries"      ".isOHLC"
[31] ".isOHLCV"                   ".isQuarterly"
[33] ".isQuarterly.default"       ".isQuarterly.timeDate"
[35] ".isQuarterly.timeSeries"    ".lines.timeSeries"
[37] ".lowessSmoother"            ".merge.timeSeries"
[39] ".na.omit.timeSeries"        ".naOmitMatrix"
[41] ".old2newRda"                ".old2newTimeSeries"
[43] ".plot.timeSeries"           ".plot.turnpointsPastecs"
[45] ".plotOHLC"                  ".plotTimeSeries"
[47] ".points.timeSeries"         ".print.timeSeries"
[49] ".rev.timeSeries"            ".scale.timeSeries"
[51] ".signalCounts"              ".signalSeries"
[53] ".sort.timeSeries"           ".splineSmoother"
[55] ".start.timeSeries"          ".str.timeSeries"
[57] ".subset_timeSeries"         ".summary.turnpointsPastecs"
[59] ".supsmuSmoother"            ".tail.timeSeries"
[61] ".time.timeSeries"           ".timeSeries"
[63] ".turnpoints2"               ".turnpointsPastecs"
[65] ".turnpointsSeries"          ".turnpointsStats"
[67] ".window.timeSeries"         "aggregate"
[69] "aggregate.timeSeries"       "alignDailySeries"
[71] "apply"                      "applySeries"
[73] "as.data.frame"              "as.list"
[75] "as.list.timeSeries"         "as.matrix"
[77] "as.timeSeries"              "as.ts"
```

```
 [79] "as.ts.timeSeries"          "attach"
 [81] "colAvgs"                   "colCummaxs"
 [83] "colCummins"                "colCumprods"
 [85] "colCumreturns"             "colCumsums"
 [87] "colKurtosis"               "colMaxs"
 [89] "colMeans"                  "colMins"
 [91] "colnames"                  "colnames<-"
 [93] "colProds"                  "colQuantiles"
 [95] "colSds"                    "colSkewness"
 [97] "colStats"                  "colStdevs"
 [99] "colSums"                   "colVars"
[101] "comment"                   "comment<-"
[103] "countMonthlyRecords"       "cumulated"
[105] "cut"                       "cut.timeSeries"
[107] "description"               "diff"
[109] "diff.timeSeries"           "drawdowns"
[111] "drawdownsStats"            "dummyDailySeries"
[113] "dummySeries"               "durations"
[115] "durationSeries"            "end"
[117] "end.timeSeries"            "fapply"
[119] "getDataPart"               "getReturns"
[121] "hclustColnames"            "head"
[123] "head.timeSeries"           "interpNA"
[125] "is.signalSeries"           "is.timeSeries"
[127] "is.unsorted"               "isMonthly"
[129] "isMultivariate"            "isUnivariate"
[131] "lag"                       "lag.timeSeries"
[133] "merge"                     "merge.timeSeries"
[135] "midquotes"                 "midquoteSeries"
[137] "na.omit"                   "na.omit.timeSeries"
[139] "newPositions<-"            "ohlcDailyPlot"
[141] "orderColnames"             "orderStatistics"
[143] "outlier"                   "pcaColnames"
[145] "print"                     "quantile"
[147] "rank"                      "readSeries"
[149] "removeNA"                  "returns"
[151] "returnSeries"              "rev"
[153] "rev.timeSeries"            "rollDailySeries"
[155] "rollMonthlySeries"         "rollMonthlyWindows"
[157] "rowCumsums"                "rownames"
[159] "rownames<-"                "sampleColnames"
[161] "scale"                     "scale.timeSeries"
[163] "series"                    "series<-"
[165] "seriesData"                "seriesPositions"
[167] "setDataPart"               "sort"
[169] "sort.timeSeries"           "sortColnames"
```

```
[171] "spreads"                     "spreadSeries"
[173] "start"                       "start.timeSeries"
[175] "statsColnames"               "str"
[177] "substituteNA"                "tail"
[179] "tail.timeSeries"             "time"
[181] "time.timeSeries"             "time<-"
[183] "timeSeries"                  "window"
[185] "window.timeSeries"
```

Bibliography

zoo Reference Manual
Achim Zeileis, Gabor Grothendieck
http://cran.r-project.org/web/packages/zoo/zoo.pdf

zoo FAQ, Vignette
Gabor Grothendieck, Achim Zeileis
http://cran.r-project.org/web/packages/zoo/vignettes/zoo-faq.pdf

zoo Quick Reference, Vignette
Ajay Shah, Achim Zeileis, Gabor Grothendieck
http://cran.r-project.org/web/packages/zoo/vignettes/zoo-quickref.pdf

zoo: An S3 Class and Methods for Indexed Totally Ordered Observations, Vignette
Achim Zeileis, Gabor Grothendieck
http://cran.r-project.org/web/packages/zoo/vignettes/zoo.pdf

xts Reference Manual
Jeff Ryan, Josh Ulrich
http://cran.r-project.org/web/packages/xts/xts.pdf

xts: Extensible Time Series, Vignette
Jeff Ryan, Josh Ulrich
http://cran.r-project.org/web/packages/xts/vignettes/xts.pdf

timeDate Reference Manual
Diethelm Wuertz, Yohan Chalabi
http://cran.r-project.org/web/packages/timeDate/timeDate.pdf

timeDate in Rnews
Yohan Chalabi, Martin Maechler, Diethelm Wuertz

timeSeries Reference Manual
Diethelm Wuertz, Yohan Chalabi
http://cran.r-project.org/web/packages/timeSeries/timeSeries.pdf

Index

About the Authors

Diethelm Würtz is private lecturer at the Institute for Theoretical Physics, ITP, and for the Curriculum Computational Science and Engineering, CSE, at the Swiss Federal Institute of Technology in Zurich. He teaches Econophysics at ITP and supervises seminars in Financial Engineering at CSE. Diethelm is senior partner of Finance Online, an ETH spin-off company in Zurich, and co-founder of the Rmetrics Association.

Yohan Chalabi has a master in Physics from the Swiss Federal Institute of Technology in Lausanne. He is now a PhD student in the Econophysics group at ETH Zurich at the Institute for Theoretical Physics. Yohan is a co-maintainer of the Rmetrics packages.

Andrew Ellis read neuroscience and mathematics at the University in Zurich. He is working for Finance Online and is currently doing a 6 month Student Internship in the Econophysics group at ETH Zurich at the Institute for Theoretical Physics. Andrew is working on the Rmetrics documentation project.