# Chronological Objects with Rmetrics

Diethelm Würtz
Tobias Setz
Yohan Chalabi
Andrew Ellis

Rmetrics Association & Finance Online Publishing

# Chronological Objects for R/Rmetrics

Diethelm Würtz
Tobias Setz
Yohan Chalabi
Andrew Ellis

October 2014

*Series Editors:*
Professor Dr. Diethelm Würtz
Institute for Theoretical Physics and
Curriculum for Computational Science
ETH - Swiss Federal Institute of Technology
Hönggerberg, HIT G 32.3
8093 Zurich

Dr. Martin Hanf
Finance Online GmbH
Zeltweg 7
8032 Zurich

*Contact Address:*
Rmetrics Association
Zeltweg 7
8032 Zurich
info@rmetrics.org

*Publisher:*
Finance Online GmbH
Swiss Information Technologies
Zeltweg 7
8032 Zurich

*Authors and Contributors:*
Diethelm Würtz, ETH Zurich
Tobias Setz, ETH Zurich
Yohan Chalabi, ETH Zurich
Andrew Ellis, Finance Online GmbH Zurich

ISBN: 978-3-906041-03-2 (Update 2015)

# Dedication

*This book is dedicated to all those who*
*have helped make Rmetrics what it is today:*
*The leading open source software environment in*
*computational finance and financial engineering.*

# Preface

## About this Book

This is a book about the R packages timeDate and timeSeries.

## Computations

In this book we use the statistical software environment R to perform our computations. R is an advanced statistical computing system with very high quality graphics that is freely available for many computing platforms. It can be downloaded from the CRAN server[1] (central repository), and is distributed under the GNU Public Licence. The R project was started by Ross Ihaka and Robert Gentlemen at the University of Auckland. The R base system is greatly enhanced by extension packages. R provides a command line driven interpreter for the S language. The dialect supported is very close to that implemented in S-Plus. R is an advanced system and provides powerful state-of-the-art methods for almost every application in statistics.
Rmetrics is a collection of several hundreds of R functions and enhances the R environment for computational finance and financial engineering. Source packages of Rmetrics and compiled MS Windows and Mac OS X binaries can be downloaded from CRAN and the development branch of Rmetrics can be downloaded from the R-Forge repository [2].

## Audience Background

The material presented in this book was originally written for my students in the areas of empirical finance and financial econometrics. However, the audience is not restricted to academia; this book is also intended to offer researchers and practitioners an introduction to using the statistical environment of R and the Rmetrics packages.

[1]http://cran.r-project.org
[2]http://r-forge.r-project.org/projects/rmetrics/

It is assumed that the reader has a basic familiarity with the R statistical environment. A background in computational statistics and finance and in financial engineering will be helpful. Most importantly, the authors assume that the reader is interested in analyzing and modelling time series.
Note that the book is not only intended as a user guide or as a reference manual. The goal is also that you learn to interpret and to understand the output of the R functions and, even more importantly, that you learn how to modify and how to enhance functions to suit your personal needs. You will become an R developer and expert, which will allow you to rapidly prototype your models with a powerful scripting language and environment.

## GETTING HELP

There are various manuals available on the CRAN server as well as a list of frequently asked questions (FAQ). The FAQ document [3] ranges from basic syntax questions to help on obtaining R and downloading and installing R packages. The manuals [4] range from a basic introduction to R to detailed descriptions of the R language definition or how to create your own R packages. The manuals are described in more detail in Appendix E.
We also suggest having a look at the mailing lists [5] for R and reading the general instructions. If you need help for any kind of R and/or Rmetrics problems, we recommend consulting r-help [6], which is R's main mailing list. R-help has become quite an active list with often dozens of messages per day. r-devel [7] is a public discussion list for R 'developers' and 'pre-testers'. This list is for discussions about the future of R and pre-testing of new versions. It is meant for those who maintain an active position in the development of R. Also, all bug reports are sent there. And finally, r-sig-finance [8] is the 'Special Interest Group' for R in finance. Subscription requests to all mailing lists can be made by using the usual confirmation system employed by the mailman software.

## GETTING STARTED

When this eBook was last compiled, the most recent copy of R was version 3.1.2. It can be downloaded from the CRAN[9] (Comprehensive R Archive

[3]http://cran.r-project.org/doc/FAQ/R-FAQ.html
[4]http://cran.r-project.org/manuals.html
[5]http://www.r-project.org/mail.html
[6]https://stat.ethz.ch/mailman/listinfo/r-help
[7]ttps://stat.ethz.ch/mailman/listinfo/r-devel
[8]https://stat.ethz.ch/mailman/listinfo/r-sig-finance
[9]http://cran-r-project.org

Network) web site. Contributed R packages can also be downloaded from this site. Alternatively, packages can be installed directly in the R environment. A list of R packages accompanied by a brief description can be found on the web site itself, or, for financial and econometrics packages, from the CRAN Task View [10] in finance and econometrics. This task view contains a list of packages useful for empirical work in finance and econometrics grouped by topic.
To install all packages required for the examples of this eBook we recommend that you install the package fBasics including its dependencies. This can be done with the following command in the R environment.

```
> install.packages("fBasics", repos = "http://cran.r-project.org")
```

It is important that your installed packages are up to date.

```
> update.packages()
```

If there is no binary package for your operating system, you can install the package from source by using the argument `type = "source"`. The R Installation and Administration [11] manual has detailed instructions regarding the required tools to compile packages from source for different platforms.

## GETTING SUPPORT

Note that especially for Mac OS X the situation is not very satisfying for operating systems newer than Snow Leopard. This due to the extensive changes made to the Xcode environment. Many packages are not available as OS X binaries and installing them from source seems rather tricky. As longs as this situation doesn't change we can not give any guarantee for this book to work for Mac. One solution for Mac users is to install Windows or Linux as a virtual machine.
Internally we succesfully compiled all the necessary packages for newer OS X operating systems. If you need help in setting up an environment for Mac you can get support from the Rmetrics association. [12]

## ACKNOWLEDGEMENTS

This book would not be possible without the R environment developed by the R Development Core Team.
We are also grateful to many people who have read and commented on draft material and on previous manuscripts of this eBook. Thanks also to

[10]http://cran.r-project.org/web/views/Finance.html
[11]http://cran.r-project.org/doc/manuals/R-admin.html
[12]Terms and conditions may apply.

those who contribute to the R-sig-finance mailing list, helping us to test our software.
We cannot name all who have helped us but we would like to thank ADD NAMES,
the Institute for Theoretical Physics at ETH Zurich, and the participants and sponsors of the R/Rmetrics Meielisalp Workshops.

This book is the third in a series of Rmetrics eBooks. These eBooks will cover the whole spectrum of basic R and the Rmetrics packages; from managing chronological objects, to dealing with risk, to portfolio design. In this eBook we introduce those Rmetrics packages that deal with the creation and analysis of time series.

Enjoy it!

Diethelm Würtz
Zurich, January 2011

This book was written three years ago. Since then many changes have been made in the base R environment. Most of them had impact on our eBook and have been continuously updated. Now with R 3.X we have done a complete revison of the book. This refreshed version of the book should take care of all updates until Winter 2014.

Diethelm Würtz
Tobias Setz
Zurich, October 2014

# Contents

Dedication iii

Preface v

*About this Book* . . . . . v

*Computations* . . . . . v

*Audience Background* . . . . . v

*Getting Help* . . . . . vi

*Getting Started* . . . . . vi

*Getting Support* . . . . . vii

*Acknowledgements* . . . . . vii

Contents ix

List of Figures xv

List of Tables xvii

**I `timeDate` Objects 1**

1 Time and Date Standards 3

1.1 *The ISO-8601 Standard* . . . . . 3

1.2 *The ANSI and POSIX Standard* . . . . . 4

1.3 *Julian Dates and Counters* . . . . . 6

1.4 *POSIX Based Implementation in R* . . . . . 7

1.5 *Representation of POSIX Objects* . . . . . 7

1.6 *Generation of POSIX Objects* . . . . . 8

1.7 *Generation of 'Date' Objects* . . . . . 11

1.8 *Subtracting and Adding Date and Time* . . . . . 11

1.9 *Extractor Functions* . . . . . 12

1.10 *Value Matching* . . . . . 13

1.11 *Finding Intervals* . . . . . 15

2 Rmetrics 'timeDate' Objects 17

2.1 *Class Representation* . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
2.2 *Create 'timeDate' Objects from Scratch* . . . . . . . . . . . . . . . 18
2.3 *Create 'timeDate' Objects from Strings* . . . . . . . . . . . . . . . 20
2.4 *Creation of 'timDate' Objects Using* `strptimeDate` . . . . . . . 21
2.5 *Create 'timeDate' Sequences* . . . . . . . . . . . . . . . . . . . . . . 22
2.6 *Create 'timeDate' Objects from Calendar Atoms* . . . . . . . . . 24
2.7 *Handling non ISO-8601 Formats* . . . . . . . . . . . . . . . . . . . 25
2.8 *Midnight Standard* . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
2.9 *Printing and Formatting 'timeDate' Objects* . . . . . . . . . . . 26

3 FINANCIAL CENTERS 29
3.1 *Setting Financial Centers* . . . . . . . . . . . . . . . . . . . . . . . . 29
3.2 *List of Financial Centers* . . . . . . . . . . . . . . . . . . . . . . . . . 30
3.3 *Daylight Saving Time Rules* . . . . . . . . . . . . . . . . . . . . . . . 31

4 GROUP GENERIC OPERATIONS 33
4.1 *Rounding and Truncating 'timeDate' Objects* . . . . . . . . . . . 34
4.2 *Subtracting and Adding 'timeDate' Objects* . . . . . . . . . . . . . 34
4.3 *Comparing 'timeDate' Objects* . . . . . . . . . . . . . . . . . . . . . 35

5 SUBSETTING 'TIMEDATE' OBJECTS 37
5.1 *Rmetrics Functions for Subsetting 'timeDate Objects* . . . . . . 37
5.2 *Extracting Elements by Integer Subsetting* . . . . . . . . . . . . . 37
5.3 *Extracting Elements by Logical Predicates* . . . . . . . . . . . . 38
5.4 *Extracting Elements by Python Like Indexing* . . . . . . . . . . 39
5.5 *Extracting Elements by Span Indexes* . . . . . . . . . . . . . . . . 40
5.6 *Extracting the first, the last, and the range of 'timeDate' Objects* 40
5.7 *Extracting Elements by 'timeDate' Indexes* . . . . . . . . . . . . . 41
5.8 *Extracting elements Using the Window Function* . . . . . . . . 41
5.9 *Extracting Weekdays and Business Days* . . . . . . . . . . . . . . 42

6 BASE FUNCTIONS AND METHODS 45
6.1 `c()` *- Combining Objects* . . . . . . . . . . . . . . . . . . . . . . . . 45
6.2 `diff()` *- Differencing an Object* . . . . . . . . . . . . . . . . . . . . 46
6.3 `duplicated()` *- Determining Duplicate Elements* . . . . . . . 48
6.4 `head()` *- Extracting First and Last few Objects* . . . . . . . . . 49
6.5 `length()` *- Measuring the Length of an Object* . . . . . . . . . 49
6.6 `min()` *- Extracting First and Last Entries from an Object* . . 49
6.7 `range()` *- Extracting the Range of an Object* . . . . . . . . . . 50
6.8 `rep()` *- Replicating Elements* . . . . . . . . . . . . . . . . . . . . . 50
6.9 `sample()` *- Sampling Objects Randomly* . . . . . . . . . . . . . 50
6.10 `sort()` *Sorting or Ordering an Object* . . . . . . . . . . . . . . . . 51
6.11 `start()` *- Extracting Terminal Times of an Object* . . . . . . . 51
6.12 `unique()` *- Extracting Unique Elements of an Object* . . . . 52

6.13 `weekdays()` - *Extracting Parts of an Object* . . . . . . . . . . . 52
6.14 `window()` - *Subsetting a Time Window from an Object* . . . . 54

7 COERCIONS AND TRANSFORMATIONS 55
7.1 *Financial Center Transformations* . . . . . . . . . . . . . . . . . . 55
7.2 *Coercion to 'POSIXct' objects* . . . . . . . . . . . . . . . . . . . . . 56
7.3 *Coercion to 'POSIXlt' objects* . . . . . . . . . . . . . . . . . . . . . 57
7.4 *Coercion to 'Date' objects* . . . . . . . . . . . . . . . . . . . . . . . . 57
7.5 *Coercion to a character Vector* . . . . . . . . . . . . . . . . . . . . . 57

**II Calendars 59**

8 CALENDAR FUNCTIONS AND METHODS 61
8.1 `align()` - *Aligning 'timeDate' Objects* . . . . . . . . . . . . . . . . 61
8.2 `atoms()` - *Extracting Calendar Atoms* . . . . . . . . . . . . . . . 63
8.3 `dayOfWeek()` - *Returning the Day of the Week* . . . . . . . . . 63
8.4 `dayOfYear()` - *Returning the Day of the Year* . . . . . . . . . . . 64
8.5 `Easter()` - *Computing the Date of Easter* . . . . . . . . . . . . . 65
8.6 `endpoints()` - *Calculating Endpoint Indexes* . . . . . . . . . . 65
8.7 `isBizday()` - *Testing for Business Days* . . . . . . . . . . . . . . 65
8.8 `isRegular()` - *Checking if a Date Vector is Regular* . . . . . . . 67
8.9 `periods()` - *Composing Rolling Periods* . . . . . . . . . . . . . . 69
8.10 `timeLastDayInMonth()` - *Computing First and Last Days* . . 74
8.11 `timeNdayOnOrAfter()` - *Computing On-Or-After Dates* . . 75
8.12 `timeNthNdayInMonth()` - *Computing the n-th n-Day* . . . . . 77

9 ECCLESIASTICAL AND PUBLIC HOLIDAYS 79
9.1 *Ecclesiastical Holidays* . . . . . . . . . . . . . . . . . . . . . . . . . . 79
9.2 *Public and Federal Holidays* . . . . . . . . . . . . . . . . . . . . . . . 81
9.3 *Listing Holidays* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 85

10 HOLIDAY CALENDARS 89
10.1 *Public Holiday Calendars* . . . . . . . . . . . . . . . . . . . . . . . . 89
10.2 *Exchange Calendars* . . . . . . . . . . . . . . . . . . . . . . . . . . . . 90

**III `timeSeries` Objects 93**

11 RMETRICS 'TIMESERIES' OBJECTS 95
11.1 *Class Representation* . . . . . . . . . . . . . . . . . . . . . . . . . . . 95
11.2 *Creating 'timeSeries' Objects from Scratch* . . . . . . . . . . . . 96
11.3 *Importing 'timeSeries' Objects* . . . . . . . . . . . . . . . . . . . . . 106
11.4 *Extracting and Modifying the Slots of a Time Series* . . . . . . . 107
11.5 *Printing 'timeSeries' Objects* . . . . . . . . . . . . . . . . . . . . . . 110

11.6 *Plotting 'timeSeries' Objects* . . . . . . . . . . . . . . . . . . . . . . . . 112
11.7 *Signal Series: Time Series without Time Stamps* . . . . . . . . 115
11.8 *Regular Time Series* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117

12 GROUP GENERIC OPERATIONS 125
12.1 *Ops: Aritmetic, Comparison and Logical Operations* . . . . . 125
12.2 *Math: Mathematical Operations* . . . . . . . . . . . . . . . . . . . . . 127
12.3 *Summary: Summary Operations* . . . . . . . . . . . . . . . . . . . . 128

13 SUBSETTING OPERATIONS 129
13.1 *Get the first and last Records of Time Series Objects* . . . . . . 129
13.2 *Extract Elements by Integer Subsetting* . . . . . . . . . . . . . . . 132
13.3 *Extract Elements by Column Names* . . . . . . . . . . . . . . . . . . 134
13.4 *Extract Elements Using the 'window' Function* . . . . . . . . . 136
13.5 *Extract Weekdays and Business Days* . . . . . . . . . . . . . . . . . 137
13.6 *Extract Data Records with Missing Values* . . . . . . . . . . . . 140

14 COERCIONS AND TRANSFORMATIONS 143
14.1 *Coercion to Vector, Matrix, and Data Frames* . . . . . . . . . . 143
14.2 *Column Statistics of 'timeSeries' Objects* . . . . . . . . . . . . . . 145
14.3 *Cumulated Column Statistics of* `timeSeries` *Objects* . . . . 146
14.4 *Rolling Statistics of 'timeSeries' Objects* . . . . . . . . . . . . . . . 146

**IV Base, Stats and Financial Functions 149**

15 BASE FUNCTIONS AND METHODS 151
15.1 `apply()` *- Applying Functions Over Time Series Margins* . . . 151
15.2 `attach()` *- Attaching a Time Series to the Search Path* . . . 153
15.3 `cbind()` *- Binding Time Series Together* . . . . . . . . . . . . . . . 154
15.4 `diff()` *- Differencing a Time Series* . . . . . . . . . . . . . . . . . 156
15.5 `dim()` *- Getting the Size of a Time Series* . . . . . . . . . . . . . . . 157
15.6 `length()` *- Returns the Length of a Time Series* . . . . . . . . . . 157
15.7 `merge()` *- Merging two Time Series Together* . . . . . . . . . . . 158
15.8 `min()` *- Getting the Extremes of Data from a Time Series* . . 159
15.9 `range()` *- Getting the Range of Data from a Time Series* . . 160
15.10 `rev()` *- Reverse Order of Time Stamps of a Time Series* . . . 160
15.11 `sample()` *- Sample the Time Stamps of a Time Series* . . . . 160
15.12 `scale()` *- Scales a Centers the Values of a Time Series* . . . . . 161
15.13 `sort()` *- Sorting a Time Series* . . . . . . . . . . . . . . . . . . . . . . . 161

16 STATS FUNCTIONS AND METHODS 163
16.1 `acf()` *- Computing Autocorrelations* . . . . . . . . . . . . . . . . . . 164
16.2 `aggregate()` *- Aggregating a Time Series* . . . . . . . . . . . . . . . 164
16.3 `arima()` *- Fitting an ARIMA Model* . . . . . . . . . . . . . . . . . . 166

16.4 *`cov()` - Computing the Covariance Matrix* . . . . . 167
16.5 *`dist()` - Computing the Distance Matrix* . . . . . 167
16.6 *`dnorm()` - Computing Normal Density* . . . . . 168
16.7 *`filter()` - Applying Linear Filtering* . . . . . 168
16.8 *`fivenum()` - Returning Five Number Summary* . . . . . 168
16.9 *`hist()` - Computing a Histogram* . . . . . 169
16.10 *`lag()` - Creating Lagged Versions of a timeSeries* . . . . . 169
16.11 *`lm()` - Fitting Linear Models* . . . . . 170
16.12 *`lowess()` - Smoothing by Polynomial Regression* . . . . . 171
16.13 *`mad()` - Computing Median Absolute Deviation* . . . . . 171
16.14 *`median()` - Computing the Sample Median* . . . . . 171
16.15 *`na.omit()` - Handling NAs in Time Series* . . . . . 172
16.16 *`qqnorm()` - Producing a Normal QQ Plot* . . . . . 176
16.17 *`smooth()` - Performing Running Median Smoothing* . . . . . 176
16.18 *`spectrum()` - Estimating the Sprectral Density* . . . . . 177
16.19 *`window()` - Subsetting by Windows* . . . . . 178

17 FINANCIAL FUNCTIONS AND METHODS 179
17.1 *`align()` - Aligning a 'timeSeries' Objects* . . . . . 179
17.2 *`cumulated()` - Cumulating a Financial Series* . . . . . 181
17.3 *`daily()` - Specific Daily Time Series Handling* . . . . . 182
17.4 *`drawdowns()` - Calculating Drawdowns* . . . . . 182
17.5 *`durations()` - Measuring Durations* . . . . . 184
17.6 *`monthly()` - Handling Monthly Series* . . . . . 184
17.7 *`orderColnames()` - Ordering Column Names* . . . . . 186
17.8 *`orderStatistics()` - Computing Order Statistics* . . . . . 187
17.9 *`periodical()` - Computing Periodical Performance* . . . . . 187
17.10 *`returns()` - Computing Financial Returns* . . . . . 188
17.11 *`smooth()` - Smoothing a Time Series* . . . . . 190
17.12 *`splits()` - Removing Outliers from Splits* . . . . . 191
17.13 *`spreads()` - Computing Spreads and Midquites* . . . . . 191
17.14 *`turnpoints()` - Searching for Peaks and Pits* . . . . . 191

**V Appendix 195**

A FAQ: TIME SERIES BASICS 197
A.1 *Creating Time Series Objects* . . . . . 197
A.2 *Regular Time Series Objects* . . . . . 198
A.3 *Time Zone and Daylight Saving Time* . . . . . 203
A.4 *Ordering and Overlapping of Time Series* . . . . . 209
A.5 *Binding and Merging of Time Series* . . . . . 214
A.6 *Subsetting Time Series Objects* . . . . . 223
A.7 *Missing Data Handling* . . . . . 228

B FAQ: BASE, STATS, AND UTILITY FUNCTIONS 235
B.1 *Functions and Methods from base R* . . . . . . . . . . . . . . . . 235
B.2 *Functions and Methods from stats R* . . . . . . . . . . . . . . . . . 241

C FAQ: PRINTING AND PLOTTING 251
C.1 *Printing Time Series Objects* . . . . . . . . . . . . . . . . . . . . . . . 251
C.2 *Plotting Time Series Objects* . . . . . . . . . . . . . . . . . . . . . . . 255

D PACKAGES REQUIRED FOR THIS EBOOK 261
D.1 *Rmetrics Package:* `timeDate` . . . . . . . . . . . . . . . . . . . . . . . 261
D.2 *Rmetrics Package:* `timeSeries` . . . . . . . . . . . . . . . . . . . . 262
D.3 *Rmetrics Package:* `fBasics` . . . . . . . . . . . . . . . . . . . . . . . 263

E R MANUALS ON CRAN 265

F RMETRICS ASSOCIATION 267

G RMETRICS TERMS OF LEGAL USE 271

BIBLIOGRAPHY 273

INDEX 275

ABOUT THE AUTHORS 281

# LIST OF FIGURES

11.1 Multiple time series plot of the Swiss pension fund assets. . 113
11.2 Single plot of the swiss pension fund asset returns. . . . . . . . . 114
11.3 Scatter plot of Swiss bond versus equities returns. . . . . . . . 115
11.4 Display of available plotting symbols. . . . . . . . . . . . . . . . . . 116
11.5 Display of available characters. . . . . . . . . . . . . . . . . . . . . . . . 117
11.6 Display of available colors. . . . . . . . . . . . . . . . . . . . . . . . . . 118

14.1 Statistical Plots. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 144
14.2 Pairs Plot. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 145

C.1 Plot 1 - Example of a single panel plot . . . . . . . . . . . . . . . . . 256
C.2 Plot 2b - Multivariate Time Series plots in multiple graphs . . 257
C.3 Plot 3b - Example of a single panel plot from timeSeries . . . 258

# List of Tables

1.1 POSIX Functions . . . . . 8

2.1 POSIX Format Identifiers . . . . . 25
2.2 ISO Format Summary . . . . . 26

4.1 timeDate Group Generic Functions . . . . . 33

5.1 Subsetting timeDate Objects . . . . . 37

6.1 timeDate Functions known from R's Base Package . . . . . 45

9.1 Dates of Easter Related Feasts . . . . . 80
9.2 Dates of Sundays in Advent . . . . . 81
9.3 Ecclesiastical Feasts with Fix Dates . . . . . 81
9.4 Federal Holidays . . . . . 82
9.5 Additional Feasts . . . . . 82
9.6 Dates of London Holidays . . . . . 83
9.7 Dates of New York and Chicago Holidays . . . . . 83
9.8 Dates of Additional Chicago Holidays . . . . . 83
9.9 Dates of Frankfurt Holidays . . . . . 83
9.10 Dates of Additional Paris Holidays . . . . . 84
9.11 Dates of Additional Milano Holidays . . . . . 84
9.12 Dates of Toronto and Montreal Holidays . . . . . 84
9.13 Dates of Tokyo and Osaka Holidays . . . . . 85

11.1 timeSeries Class Representation . . . . . 95

15.1 timeSeries Functions known from R's Base Package . . . . . 151

16.1 Functions known from R's Stats Package . . . . . 163

# PART I

# timeDate OBJECTS

# CHAPTER 1

# TIME AND DATE STANDARDS

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

## 1.1 THE ISO-8601 STANDARD

The international standard ISO-8601 (ISO-8601, 1988) defines formats for the numerical representation of dates and times. For date and time, ISO-8061 mandates a `"YYYY-MM-DD hh:mm:ss"` format, where `"YYYY"` denotes the 4-digit year including the century, `"MM"` the month, `"DD"` the day, `"hh"` hours, `"mm"` minutes, and `"ss"` seconds. The standard allows the omission of the field separators `"-"`, `" "`, and `":"` if compactness of the representation is more important than human readability, yielding `"YYYYMMDDhhmmss"`.

- `Basic Date Format` - YYYYMMDD,
  e.g. 20050101
- `Extended Date Format` - YYYY-MM-DD,
  e.g. 2005-01-01
- `Basic Date/Time Format` - YYYYMMDDhhmmss,
  e.g. 20050101161500
- `Extended Date/Time Format` - YYYY-MM-DD hh:mm:ss,
  e.g. 2005-01-01 16:15:00

The ISO-8601 notation is the most commonly recommended format of representing date and time as human-readable strings in use today. The notation allows the time-stamping of data, events and information, so that anyone in the world, of any nationalit,y will know, without ambiguity, which date this identifier relates to. Writing date and time in such a standardized way has many advantages:

- Easily readable and writeable by software, no month name to number conversion is necessary, and thus it is language-independent.
- Consistency with the common time notation system, where the larger unit (hour) is written in front of the smaller ones (minutes and seconds).
- Since the century is explicitly specified sorting tools can easily applied to date vectors since the individual date strings are ordered in a descending resolution from centuries to seconds.
- The notation is short and has constant length, which makes both keyboard data entry and table layout easier.

Here we presented only two aspects of the ISO-8601 standard, representing calendar dates and 24-hour timekeeping. However, ISO-8601 also standardizes the representation of ordinal dates expressed in terms of year and day of the year, of week dates expressed in terms of year, week number and day of the week, of differences between local time and Coordinated Universal Time, and of time intervals. The interested reader should consult the ISO-8601 (ISO-8601, 1988) document in its final version.

## 1.2 THE ANSI AND POSIX STANDARD

The most important and most complete source of date and time functionality is contained in the standard GNU C library (Bateman, 2000). The purpose of this section is to outline briefly what functionalities the ANSI and POSIX standards, upon which the C library is based, offer.
Let us summarize some of the important terms used by the documents that drive the date/time standard. These terms, such as *GMT*, *Local Time*, *DST*, and *Time Zone*, are the same as those we are confronted with when we define date and time classes for `R`.

- `GMT` - Greenwich Mean Time is the time at the Greenwich Meridian. There is no intrinsic geographical reason for the choice made quite arbitrarily in 1883, it just reflects Great Britain's position as a maritime power at that time.
- `Local Time` - is the time for the current, geographical locale, usually adjusted for daylight or summer time.

- `DST` - Daylight Savings Time, is called summer time in most countries. It is the local time during a part of the year when that locale finds it advantageous to advance the clock artificially, usually by one hour. Most, but not all locales of the world are plagued by this phenomenon. Today rules exist for many locales, but until the 1980s, time was an annual subject of Parliament debates and no rule could describe it.
- `Time Zone` - describes the recognition of the problem of longitude across the face of the planet, namely that the cadastral position of the Sun is not the same for every locale around the world at precisely the same moment in time. There are roughly, twenty-four time zones separated by one hour from each other around the world, but practically speaking, many more exist that calculate their current time offset from UTC as fractional. Note that time zones may even be overlapping.

In computer programs, standard time functionality is the object of the standard (ANSI) header definition. These definitions are also at the heart of the POSIX named date and time implementation in `R`. It is important to note how local time is counted.

- `localtime` - This data type consists of the following fields:
  - `int tm_sec;` - seconds after the minute [0..59]
  - `int tm_min;` - minutes after the hour [0..59]
  - `int tm_hour;` - hours since midnight [0..23]
  - `int tm_mday;` - days of the month [1..31]
  - `int tm_mon;` - months since Jan [0..11], zero-based
  - `int tm_year;` - years since 1900 [0..INF]
  - `int tm_wday;` - days since Sunday [0..6], zero-based
  - `int tm_yday;` - days since first January [0..365]
  - `int tm_isdst;` - summer time (1 in effect, 0 not, -1 unknown)

The *time zone information* is extremely complex in the library and has been the main source of errors found within the C library.
The *public-domain time zone database* contains code and data that represent the history of local time for many representative locations around the globe. It is updated periodically to reflect changes made by political bodies to UTC offsets and daylight-saving rules. This database, often called `tz` or `zoneinfo`, is used by several implementations, including the GNU C Library used for example in the Linux and Mac OS X operating systems. Each

location in the database represents a national region. Locations are identified by continent or ocean and then by the name of the location, which is typically the largest city within the region, e.g. `TZ="Europe/Zurich"`. Rmetrics brings the *Olson's tz database* into R since it holds all the information in easily accessible ASCII files. The library thus allows cross-platform time zone calculations using Rmetrics packages.
This short introduction gives us the impression that we cannot expect computers to always act in the same way on different operating systems, using different versions of C library implementations, and different versions of the R software itself. For example, if we analyse high-frequency data from worldwide markets acting in different time zones according to different DST rules, we would expect that we obtain possibly different results on different computers. Therefore, with Rmetrics packages, we have decided to always work internally with dates and times based on *GMT*. That way, we can be quite sure that every computer delivers the same results, since we eliminate all the complexities arising from time zones and daylight savings time rules. The local time management will then be done by R functions implemented in Rmetrics packages using TZ and DST information stored in a local ASCII database. This guarantees that that we can work under the same conditions on every computer. *GMT* is thus our reference date and time system, and date time is measured by counters relatively to an GMT specified offset.

## 1.3 JULIAN DATES AND COUNTERS

*Julian Dates* are simply a continuous count of days and fractions since noon on January 1, 4713 BC, in Universal Time. Note that astronomical convention starts the day at noon, in contrast with the civil practice where the day starts with midnight. Almost 2.5 million days have transpired since this date. Julian dates have their origin in astronomy and thus are widely used as time variables within astronomical software. Statistical software has incorporated the idea and concept of the internal representation of date (and time) by *Julian Counters*. In statistical software packages these counters assign a unique number to every day since a predetermined date. By default the R functions from the contributed `chron` package use as Julian origin `1970-01-01` and SPlus uses `1960-01-01`, but in both environments the date of origin can be user defined.
For the calculation of Julian dates rather elaborate logic is needed to take into account the varying number of days in each month, plus the occurrence of leap years. Fliegel and van Flandern [1968] and Becker, Chambers, and Wilks [1988] report algorithms that take advantage of the truncation feature of integer arithmetic to solve the problem in a very compact way. Both codes convert any given calendar date to a Julian Date. For example, noon GMT on January 1, 1970, is the beginning of the Julian Date

2’440’588. Now, the interval between any two calendar dates can be found by obtaining the Julian Date for both dates, and finding their difference. This allows you to easily write functions to convert calendar dates to Julian counts and vice versa, to extract the day of the week, and to decide if a year is a leap year or not.

## 1.4 POSIX BASED IMPLEMENTATION IN R

Date and time objects were originally made available for R through the `chron` package which was written by James and Pregibon [1992] for the S environment. Kurt Hornik [2001] ported the package to R. The package is very popular and heavily used under both R and SPlus, and there are only minor differences in the R and SPlus implementation.
In the `chron` package, chronological objects are numeric vectors that represent number of days. There are three classes of chronological objects: `times`, `dates`, and `chron`. A `times` object represents elapsed number of days, while `dates` and `chron` objects represent number of days from a specified origin which may be arbitrary, defined through the function `origin`. The difference between `dates` and `chron` objects is that the latter represents time-of-day in addition to dates. A `chron` object inherits from `dates`, and `dates` objects inherit from `times`. The disadvantage of the `chron` implementation seems to be twofold. First, date and time are not represented in the ISO-8601 standard, and second, time zones and daylight savings time are not supported. However, if this is not the issue in your statistical analysis, then `chron` objects may offer the facilities you need. This is the reason, that many people still make the `chron` objects their first choice when working with date and time objects.
However, acting on financial markets located in financial centers around the globe in different time zones and relying on different DST rules require a more sophisticated date and time management system. R has two powerful basic classes of date/time implemented, which are based on the POSIX standard. They were included into R Version 1.2 by B.D. Ripley and K. Hornik [2001] as part of the base package. Supported are a time zone attribute and a component indicating whether daylight savings time is active or not. With R Version 1.3, facilities for handling time differences were added. Today, R’s POSIX implementation comes with many convenience functions which make use of information from the POSIX based date/time classes.

## 1.5 REPRESENTATION OF POSIX OBJECTS

Class `POSIXct` represents the (signed) number of seconds since the beginning of 1970 as a numeric vector.

- signed number of seconds since 1970-01-01 00:00:00

Class `POSIXlt` is a named list of vectors.

- `sec` 0–61: seconds
- `min` 0–59: minutes
- `hour` 0–23: hours
- `mday` 1–31: day of the month
- `mon` 0–11: months after the first of the year.
- `year` Years since 1900.
- `wday` 0–6 day of the week, starting on Sunday.
- `yday` 0–365: day of the year.
- `isdst` Daylight savings time flag. Positive if in force, zero if not, negative if unknown.

`POSIXct` is more convenient for including in data frames, and `POSIXlt` is closer to human-readable forms. A virtual class `POSIXt` inherits from both of the classes; it is used to allow operations to mix the two classes.

## 1.6 GENERATION OF POSIX OBJECTS

There are several functions for the generation and manipulation of `POSIXt` objects. The major functions are listed in the following table:

LISTING 1.1: POSIX FUNCTIONS.

```
Function:
as.POSIXct          for conversion between the classes
as.POSIXlt          for conversion between the classes
strptime            for conversion to and from character
Sys.time            for clock time as a POSIXct object
difftime            for creating time intervals
cut.POSIXt          for dividing the range into intervals
seq.POSIXt          for creating a sequence of POSIXt objects
round.POSIXt        for rounding a POSIXt object
trunc.POSIXt        for methods for these classes
weekdays.POSIXt     for convenience extraction functions
Dates               for dates without times
```

The function `strptime()` and the related functions and methods `format()`, `as.character()`, `strftime()`, `ISOdatetime()`, and `ISOdate()` are functions for date-time POSIXt conversion to and from character string.

The following example shows how to create a 'POSIXlt'

```
> # Create time and date Vectors:
> dts <- c("1989-09-28", "2001-09-15", "2004-08-30", "1990-02-09")
> tms <- c("23:12:55",   "10:34:02",   "08:30:00",   "11:18:23")
> # Create POSIXlt Object:
> lt <- strptime(paste(dts, tms), format="%Y-%m-%d %H:%M:%S")
> lt
[1] "1989-09-28 23:12:55 CET"  "2001-09-15 10:34:02 CEST"
[3] "2004-08-30 08:30:00 CEST" "1990-02-09 11:18:23 CET"
> class(lt)
[1] "POSIXlt" "POSIXt"
```

Here, the function `class()` prints the vector of names of classes the `lt` object inherits from. In the next example the function `unclass()` returns a copy of its argument with its class attribute removed.

```
> unclass(lt)
$sec
[1] 55  2  0 23

$min
[1] 12 34 30 18

$hour
[1] 23 10  8 11

$mday
[1] 28 15 30  9

$mon
[1] 8 8 7 1

$year
[1]  89 101 104  90

$wday
[1] 4 6 1 5

$yday
[1] 270 257 242  39

$isdst
[1] 0 1 1 0

$zone
[1] "CET"  "CEST" "CEST" "CET"

$gmtoff
[1] NA NA NA NA
```

As we see, an object of class `POSIXlt` is a named list of vectors representing the calendar atoms seconds (sec), minutes (min), hours (hour), day of

the month (mday), month (mon), day of the year (yday), day of the weak (wday), and the DST flag (isdst).
To create a 'POSIXct' object from scratch, we just convert the object `lt` into a `POSIXct` object

```
> # Create POSIXct Date/Time Object:
> ct <- as.POSIXct(lt)
> ct
[1] "1989-09-28 23:12:55 CET"  "2001-09-15 10:34:02 CEST"
[3] "2004-08-30 08:30:00 CEST" "1990-02-09 11:18:23 CET"
> unclass(ct)
[1]  623023975 1000542842 1093847400  634558703
attr(,"tzone")
[1] ""
```

Unclassing this object we discover that `POSIXct` is internally represented by a counter, representing the (signed) number of seconds since the beginning of 1970 as a numeric vector.
The functions `ISOdate()` and `ISOdatetime()` are convenience wrappers for the function `strptime()`, that differ only in their defaults and return an object of class `POSIXct`. The arguments for these functions are the calendar and time atoms `year`, `month`, `day`, `hour`, `min`, and `sec`.
The function `Sys.time()` returns the system's idea of the current date and time, an object of class `POSIXct`.

```
> Sys.time()
[1] "2014-12-08 11:46:32 CET"
```

The function `Sys.timezone()` returns an operating system-specific character string for the time zone in use, possibly an empty string.

```
> Sys.timezone()
[1] "Europe/Zurich"
```

Another way to generate POSIX objects uses the generic function `seq()`.

```
> args(seq.POSIXt)
function (from, to, by, length.out = NULL, along.with = NULL,
    ...)
NULL
```

The function allows through the arguments `from` and `to` to specify start and end date of the sequence, or alternative the start date and the desired length of the sequence using the argument `length.out`. The intervals can be denoted by a character string with the argument `by` specifying time units ranging from seconds to years. Arbitrary lengths of intervals can be created by assigning a numeric value to the argument `by`, e.g. `by=1800` creates half hourly time intervals.
Create times and dates as character vectors

```
> dts <- c("1989-09-28", "2001-01-15", "2004-08-30", "1990-02-09")
> tms <- c(  "23:12:55",   "10:34:02",   "08:30:00",   "11:18:23")
> ct <- as.POSIXct(strptime(paste(dts, tms), "%Y-%m-%d %H:%M:%S"))
> # by - A number, taken to be in seconds, use one day:
> seq(min(ct), by=24*3600, length=4)

[1] "1989-09-28 23:12:55 CET" "1989-09-29 23:12:55 CET"
[3] "1989-09-30 23:12:55 CET" "1989-10-01 23:12:55 CET"

> # by - A object of class 'difftime':
> seq(min(ct), by=diff(ct)[1], length=4)

[1] "1989-09-28 23:12:55 CET"  "2001-01-15 10:34:02 CET"
[3] "2012-05-03 22:55:09 CEST" "2023-08-21 10:16:16 CEST"

> # by - A character string, containing one of 'sec', 'min',
> # 'hour', 'day', 'DSTday', 'week', 'month' or 'year'. This
> # can optionally be preceded by an integer and a space, or
> # followed by 's'.
> seq(min(ct), by="months", length=4)

[1] "1989-09-28 23:12:55 CET" "1989-10-28 23:12:55 CET"
[3] "1989-11-28 23:12:55 CET" "1989-12-28 23:12:55 CET"
```

## 1.7 GENERATION OF 'DATE' OBJECTS

With R version 1.9 the `Date` class was introduced into the base package. G. Grothendieck and T. Petzoldt [2004] write about this class of objects: The class supports dates without times. Eliminating times simplifies dates substantially since potential complications arising from time zones and daylight savings time need not be considered. Dates are represented internally as days since January 1, 1970. Information on Date can be found in `?Dates`. The function `Sys.Date()` returns the system's current date.

```
> Sys.Date()

[1] "2014-12-08"
```

and the returned object has the class

```
> class(Sys.Date())

[1] "Date"
```

## 1.8 SUBTRACTING AND ADDING DATE AND TIME

R allows math operations on the 'POSIXt' and 'Date' classes. The minus "-", and "+" sign also operates on date/time stamps. The subtraction or addition of an integer representing seconds from a POSIXt date returns a POSIXt object with the corresponding date and time. On the other hand, the subtraction of two time/date objects results in a time difference, represented by an object of class `difftime()`. The addition of two date/time objects is not allowed. The following example demonstrates how to subtract one hour from a POSIX date/time object, and how to subtract two POSIX date/time objects. Print current time and time one hour ago

```
> end <- Sys.time()
> start <- end - 3600
> c(start, end)

[1] "2014-12-08 10:46:32 CET" "2014-12-08 11:46:32 CET"
```

and its time interval

```
> end - start

Time difference of 1 hours
```

### *Logical Operations on Date and Time*

`POSIXt` and `Date` objects also allow for logical operations. Logical operations can test the ordering of date/time objects, and determine whether an event is earlier than another, at the same time, or later. An equal sign means at the same time, a smaller and greater sign mean before and after, etc.

```
> ct

[1] "1989-09-28 23:12:55 CET"  "2001-01-15 10:34:02 CET"
[3] "2004-08-30 08:30:00 CEST" "1990-02-09 11:18:23 CET"

> ct[1] < ct[2]

[1] TRUE

> ct == ct[3]

[1] FALSE FALSE  TRUE FALSE
```

## 1.9 EXTRACTOR FUNCTIONS

`R` allows extracting parts of ’POSIXt’ and ’Date’ objects in several ways.

### *How to extract weekdays*

The function `weekdays()` extracts weekdays, i.e. days during a week from Monday to Friday. The function returns a character vector of names in the locale in use.

```
> weekdays(lt)

[1] "Thursday" "Saturday" "Monday"   "Friday"
```

*How to extract calendar atoms*

The functions `months()` and `quarters()` extract month or quarter calendar atoms. These are generic functions. `months` returns a character vector of names in the locale in use, `quarters()` returns a character vector of `"Q1"` to `"Q4"`.

```
> months(lt)
[1] "September" "September" "August"    "February"
> quarters(lt)
[1] "Q3" "Q3" "Q3" "Q1"
```

*How to convert to Julian counts*

The function `julian()` extracts the Julian time, the number of days since some origin, possibly fractional. The origin is specified through the `"origin"` attribute.

```
> julian(lt)
Time differences in days
[1]  7210.9 11580.4 12660.3  7344.4
attr(,"origin")
[1] "1970-01-01 GMT"
```

*Are some values true*

The generic function `any()` becomes useful when we have to answer the question: *Given a set of logical vectors, is at least one of the values true?*

```
> tC <- timeCalendar()
> any(tC > tC[6])
[1] TRUE
```

The function `all()` is the complement to the function `any()`.

## 1.10 VALUE MATCHING

*How to find first matches*

The function `match()` returns a vector of the positions of (first) matches of its first argument in its second.

```
> dates <- seq(as.Date("2009-01-01"), by = "month", length.out = 12)

> Date1 <- as.Date("2009-05-01")
> match(Date1, dates)
[1] 5
```

```
> Date2 <- as.Date("2009-05-15")
> match(Date2, dates)

[1] NA
```

### *How to find if there are matches or not*

The function `%in%()` is a more intuitive interface as a binary operator, which returns a logical vector indicating whether or not there is a match for its left operand.
In the help page we find that factors, raw vectors and lists are converted to character vectors, and then `x` and table are coerced to a common type (the later of the two types in R's ordering, logical < integer < numeric < complex < character) before matching. If incomparables have positive length they is coerced to the common type. Matching for lists is potentially very slow and best avoided except in simple cases. Exactly what matches what is to some extent a matter of definition. For all types, `NA` matches `NA` and no other value. For real and complex values, `NaN` values are regarded as matching any other `NaN` value, but not matching `NA`. Notice that the function also accepts date and time objects.

```
> dates <- seq(as.Date("2009-01-01"), by = "month", length.out = 12)


> Date <- as.Date("2009-05-01")
> Date %in% dates

[1] TRUE


> Date <- as.Date("2009-05-15")
> Date %in% dates

[1] FALSE
```

### *How to find matches among arguments*

The function `pmatch()` seeks matches for the elements of its first argument among those of its second.

```
> firstInMonth <- seq(as.Date("2009-01-01"), by = "month", length.out = 12)


> pmatch("2009-05-01", firstInMonth)

[1] 5

> pmatch("2009-05-15", firstInMonth)

[1] NA
```

The first match returns as index a 5, the second an `NA` because the match failed.

## 1.11 FINDING INTERVALS

The function `findInterval()`

```
> args(findInterval)

function (x, vec, rightmost.closed = FALSE, all.inside = FALSE)
NULL
```

helps to find interval numbers or indices.
`findIntervals()` returns a `vector` of length `length(x)` with values in `0:N` (and `NA`) where `N <- length(vec)`, or values coerced to `1:(N-1)` if and only if `all.inside=TRUE` (equivalently coercing all `x` values inside the intervals). Note that `NA`s are propagated from `x`, and `Inf` values are allowed in both `x` and `vec`.
The first example counts the number of 40'000 normal random variates in the four quartiles

```
> set.seed(1953)
> N <- 40000
> vec <- sort(rnorm(N))
> x <- qnorm(c(0.25, 0.5, 0.75))
> int <- c(0, findInterval(x, vec), N)
> diff(int)

[1]  9990  9951 10008 10051
```

This example counts the number of days per month in the daily sequence of dates in the year 2009.

```
> timeStamps <- seq(as.Date("2009-01-01"), by = "day", length.out = 365)
> endOfMonth <- seq(as.Date("2009-02-01"), by = "month", length.out = 12) -
      1
> int <- findInterval(endOfMonth, timeStamps)


> daysInMonth <- c(int[1], diff(int))
> names(daysInMonth) <- 1:12
> daysInMonth

 1  2  3  4  5  6  7  8  9 10 11 12
31 28 31 30 31 30 31 31 30 31 30 31
```

Note that this approach can be applied to many date/time extraction problems with arbitrary intervals and not only to monthly ones.

# CHAPTER 2

# RMETRICS 'TIMEDATE' OBJECTS

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

## 2.1 CLASS REPRESENTATION

The `timeDate` class in Rmetrics packages can be formally expressed as the combination of the following three major components, *POSIXct* time stamps always within the "GMT" time zone, DST the `time zone` and *Daylight Saving Rules* as given in Olson's time zone database specified by a *financial center* name, where the time stamps belong, and *ISO-8601*, the human readable standard format, which expresses dates as `"%Y-%m-%d"` and dates/times as `"%Y-%m-%d %H:%M-%S"`. There is no explicit distinction between the formats, it is only the view which makes the difference.
`timeDate()` objects fulfil the conventions of the ANSI C and POSIX standard as well as of the ISO 8601 standard. Beyond these standards Rmetrics has added the "Financial Center" concept which references Olson's time zone data base and allows to handle data records collected in different time zones and mix them up to have always the proper time stamps with respect to your personal financial center, or alternatively to the "GMT" reference time. It can thus also handle time stamps from historical data records from the same time zone, even if the financial centers changed day light saving times at different calendar dates.
Moreover Rmetrics' `timeDate()` is almost compatible with the `timeDate` class in Insightful's SPlus. If you move between the two worlds of R and SPlus, you will not have to rewrite your code. This is important mostly for business applications.

The `timeDate()` class offers not only date and time functionality but also sophisticated calendar manipulations for business days, weekends, as well as public and ecclesiastical holidays. This allows to handle easily day count conventions and rolling business conventions according to the rules of ICMA, ISDA, and SIFMA.
Date and time stamps are represented by an S4 object of class `timeDate`.

```
> getClass("timeDate")
Class "timeDate" [package "timeDate"]

Slots:

Name:       Data    format FinCenter
Class:   POSIXct character character
```

They have three slots. `@Data` holds time stamps which are `POSIXct` formatted as specified in the `@format` slot. Time stamps are local belonging to the financial center expressed through the slot `@FinCenter`.
There are several possibilities to generate a `timeDate` object. The most forward procedure is to use one of the following functions: `timeDate()` creates a `timeDate` object from scratch, `timeSequence()` creates a sequence of `timeDate` objects, `timeCalendar()` creates a `timeDate` object from calendar atoms.

## 2.2 CREATE 'TIMEDATE' OBJECTS FROM SCRATCH

With the function `timeDate()`

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

we can create `timeDate` objects from scratch by specifying a just a character value or vector `charvec` of time stamps. The remaining arguments are optional. Instead of a character value or vector also time series or related objects can be give as far as they come with a method for the generic function `format`. For example, allowed `charvec` arguments are also `timeDate` objects, or R's 'POSIXct', 'POSIXlt', or 'Date' objects. The remaining arguments are all optional, we will discuss them later. The default setting for the argument `format` is `NULL` which means that an auto-detection meachanism to recognize date/time formats is effective, and that time `zone` and the financial center argument named `FinCenter` are set both to the default value of the financial center which is "GMT". To check this, type

```
> getRmetricsOptions("myFinCenter")
myFinCenter
      "GMT"
```

*Get Current Date and Time as a 'timeDate' Object*

The function `timeDate()` with its default settings returns the date and time as a `timeDate` object with repsect to the financial center specified by the string variable `myFinCenter`, thus

```
> timeDate()

GMT
[1] [2014-12-08 10:46:30]
```

returns the time just now in "GMT". Specifying a financial center argument we can get the time now for example in Zurich or Tokyo

```
> timeDate(FinCenter = "Zurich")

Zurich
[1] [2014-12-08 11:46:30]

> timeDate(, , , "Tokyo")

Tokyo
[1] [2014-12-08 19:46:30]
```

to simplify this we can use the function `timeNow`

```
> timeNow <- function(FinCenter) timeDate(, , "GMT", FinCenter)
> timeNow("Zurich")

Zurich
[1] [2014-12-08 11:46:30]
```

and the time difference between Zurich and Tokyo will be

```
> timeNow("Zurich") - timeNow("Tokyo")

Time difference of -0.0009079 secs
```

This would possibly not what you would have been expected. But it is true, since we are now at the same time in both centers. To get the time zone difference, we need a reference point, or rexpress the time stamps for both financial centers as objects of class `POSIX`.

```
> # Function to compute time zone difference:
> tzDiff <- function(FinCenter1, FinCenter2)
 {
     as.POSIXct(timeDate(,,FinCenter1)) -
     as.POSIXct(timeDate(,,FinCenter2))
 }
```

The function `tzDiff()` allows us now to compute the current time zone difference between Tokyo and Zurich.

```
> # Time zone difference:
> tzDiff("Tokyo", "Zurich")

Time difference of -8 hours
```

## 2.3 CREATE 'TIMEDATE' OBJECTS FROM STRINGS

In general `timeDate` objects can be generated from a character value or vector of arbitrary chosen ISO-8601 formatted dates `ds`, e.g.

```
> ds <- c("1989-09-28", "2001-01-15", "2004-08-30")
> ds
[1] "1989-09-28" "2001-01-15" "2004-08-30"

> tD <- timeDate(ds)
> tD

GMT
[1] [1989-09-28] [2001-01-15] [2004-08-30]
```

What we get displayed typing `tD` or `print(tD)` is a vector of dates printed in square brackets, e.g. `[1989-09-28]`, to distinguish the output easily from a character vector `"1989-09-28"` enclosed in double quotes, or from the output of object of other date/time classes. In addition on top of the displayed date/time stamp(s) the financial center is shown, here, `"GMT"` giving us the name of the financial center to which the time stamps have to be referenced. If time is missing, we assume silently midnight, i.e. `"00:00:00"`. To create a date/time object we have to specify also the time stamps `ts`

```
> ts <- c("23:12:55", "10:34:02", "08:30:00")
> DatesTimes <- paste(ds, ts)
> DatesTimes
[1] "1989-09-28 23:12:55" "2001-01-15 10:34:02" "2004-08-30 08:30:00"

> GMT <- timeDate(DatesTimes)
> GMT

GMT
[1] [1989-09-28 23:12:55] [2001-01-15 10:34:02] [2004-08-30 08:30:00]
```

The slot names of the `GMT` `timeDate` object are

```
> slotNames(GMT)
[1] "Data"      "format"    "FinCenter"
```

the attributes of `GMT` `timeDate` object are

```
> attributes(GMT)
$Data
[1] "1989-09-28 23:12:55 GMT" "2001-01-15 10:34:02 GMT"
[3] "2004-08-30 08:30:00 GMT"

$format
[1] "%Y-%m-%d %H:%M:%S"

$FinCenter
[1] "GMT"

$class
```

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

and, the structure `GMT` `timeDate` object can be described as

```
> str(GMT)
Formal class 'timeDate' [package "timeDate"] with 3 slots
  ..@ Data     : POSIXct[1:3], format: "1989-09-28 23:12:55" "2001-01-15 10:34:02" ...
  ..@ format   : chr "%Y-%m-%d %H:%M:%S"
  ..@ FinCenter: chr "GMT"
```

The `@Data` slot is an object of class 'POSIXct' and can be retrieved by typing one of the following three commands

```
> GMT@Data
[1] "1989-09-28 23:12:55 GMT" "2001-01-15 10:34:02 GMT"
[3] "2004-08-30 08:30:00 GMT"
> slot(GMT, "Data")
[1] "1989-09-28 23:12:55 GMT" "2001-01-15 10:34:02 GMT"
[3] "2004-08-30 08:30:00 GMT"
> as.POSIXct(GMT)
[1] "1989-09-28 23:12:55 GMT" "2001-01-15 10:34:02 GMT"
[3] "2004-08-30 08:30:00 GMT"
```

## 2.4 CREATION OF 'TIMDATE' OBJECTS USING `strptimeDate`

Like the function `strptime()` for generating 'POSIXlt' objects, the function `strptimeDate()` can be used to generate `timeDate` objects

```
> args(strptimeDate)
function (x, format = whichFormat(x), tz = "")
NULL
```

The arguments have the same meaning as for the function `strptime()`: `x` is a value or vector of character strings, `format` is the format specification string, and `tz` denotes the time zone specified by a Financial Center or "GMT".

```
> strptimeDate("2008-01-01 16:00", tz = "GMT")
GMT
[1] [2008-01-01 16:00:00]
> strptimeDate("2008-01-01 16:00", tz = "New_York")
New_York
[1] [2008-01-01 16:00:00]
```

Note, the time stamp strings are local describing date and time in GMT or for the Zurich financial center.

*Moving between Time Zones and Financial Centers*

In the examples above the printed `timeDate` object was in `"GMT"`. If we want to express `timeDate` objects in local `"Zurich"` time, we have to specify the arguments `zone` and `FinCenter`

```
> ZRH <- timeDate(GMT, zone = "GMT", FinCenter = "Zurich")
> ZRH
Zurich
[1] [1989-09-29 00:12:55] [2001-01-15 11:34:02] [2004-08-30 10:30:00]
```

To convert the date/time stamps from `"Zurich"` into local "NewYork" date/time stamps, we type

```
> NYC <- timeDate(ZRH, zone = "Zurich", FinCenter = "NewYork")
> NYC
NewYork
[1] [1989-09-28 19:12:55] [2001-01-15 05:34:02] [2004-08-30 04:30:00]
```

Note, converting time zones and changing financial centers will be discussed in more detail in section X.

## 2.5 CREATE 'TIMEDATE' SEQUENCES

Alternatively we can create a sequence of `timeDate` objects with the help of the function `timeSequence()`

```
> args(timeSequence)
function (from, to = Sys.timeDate(), by, length.out = NULL, format = NULL,
    zone = "", FinCenter = "")
NULL
```

or using the generic function `seq()` alternatively, as a matter of taste.

```
> args(timeDate:::seq.timeDate)
function (from, to, by, length.out = NULL, along.with = NULL,
    ...)
NULL
```

Here, the arguments have the following meaning: `from` and `to` are the starting date (required) and end date (optional). If supplied `to` must be after (later than) `from`. Using seq, then in this case the `from` and `to` dates must be objects of class `timeDate`. `by` is a character string, containing one of "sec", "min", "hour", "day", "week", "month" or "year". This can optionally be preceded by an integer and a space, or followed by "s". `length.out` is an integer, optionally specified. the variable denotes the desired length of the sequence, if specified then `to` will be ignored. `format` provides the format specification of the input character vector(s) for `from` and `to`. `zone` and `FinCenter` describe as usual the time zone and the financial use in use.

### *Start and End Time of Sequences*

Generating a time sequence can thus be done in several ways. As a first example we specify the range of the time stamps through the arguments `from` and `to` and generate a monthly series for the current year

```
> # Monthly 2008 Sequence:
> setRmetricsOptions(myFinCenter="GMT")
> tS <- timeSequence(from="2008-01-01", to="2008-12-31", by="month")
> tS
GMT
 [1] [2008-01-01] [2008-02-01] [2008-03-01] [2008-04-01] [2008-05-01]
 [6] [2008-06-01] [2008-07-01] [2008-08-01] [2008-09-01] [2008-10-01]
[11] [2008-11-01] [2008-12-01]
```

Note, in the case of a monthly and quarterly sequences, we have further options. For example you can generate the series with the first or last day in each month, or use more complex rules like the last or n-th Friday in every month.
The following example shows how to create a vector of real end of month time stamps

```
> # Do you want the last Day or the last
> timeLastDayInMonth(tS)
GMT
 [1] [2008-01-31] [2008-02-29] [2008-03-31] [2008-04-30] [2008-05-31]
 [6] [2008-06-30] [2008-07-31] [2008-08-31] [2008-09-30] [2008-10-31]
[11] [2008-11-30] [2008-12-31]
```

or if we like to return a vector of time stamps with the dates of the last Friday in each month. For further options on working with special dates we refer to section XX. Now we proceed as

```
> # Do you want the last Friday in Month Data ?
> timeLastNdayInMonth(tS, nday=5)
GMT
 [1] [2008-02-01] [2008-02-29] [2008-04-04] [2008-05-02] [2008-06-06]
 [6] [2008-07-04] [2008-08-01] [2008-09-05] [2008-10-03] [2008-10-31]
[11] [2008-12-05] [2009-01-02]
```

### *Length of the Time Sequence*

Beside specifying the start date of the series, we can alternatively provide its length

```
> # Monthly 2008 Sequence:
> tS <- timeSequence(to="2008-12-31", by="month", length.out=12)
> tS
GMT
 [1] [2008-12-02] [2009-01-02] [2009-02-02] [2009-03-02] [2009-04-02]
 [6] [2009-05-02] [2009-06-02] [2009-07-02] [2009-08-02] [2009-09-02]
[11] [2009-10-02] [2009-11-02]
```

## 2.6 CREATE 'TIMEDATE' OBJECTS FROM CALENDAR ATOMS

A third possibility is to create `timeDate` objects from calendar atoms. You can specify values or vectors of equal length of integers denoting year, month, day, hour, minute and seconds. If every day has the same time stamp, you can just add an offset.
The default settings create a monthly `timeDate` object for the current year:

```
> currentYear <- getRmetricsOptions("currentYear")
> timeCalendar()

GMT
 [1] [2014-01-01] [2014-02-01] [2014-03-01] [2014-04-01] [2014-05-01]
 [6] [2014-06-01] [2014-07-01] [2014-08-01] [2014-09-01] [2014-10-01]
[11] [2014-11-01] [2014-12-01]
```

To generate a daily `timeDate` object for January data of Tokyo local time 16:00 for use in Zurich we proceed as

```
> timeCalendar(currentYear, m = 1, d = 1:31, h = 16, zone = "Tokyo",
     FinCenter = "Zurich")

Zurich
 [1] [2014-01-01 08:00:00] [2014-01-02 08:00:00] [2014-01-03 08:00:00]
 [4] [2014-01-04 08:00:00] [2014-01-05 08:00:00] [2014-01-06 08:00:00]
 [7] [2014-01-07 08:00:00] [2014-01-08 08:00:00] [2014-01-09 08:00:00]
[10] [2014-01-10 08:00:00] [2014-01-11 08:00:00] [2014-01-12 08:00:00]
[13] [2014-01-13 08:00:00] [2014-01-14 08:00:00] [2014-01-15 08:00:00]
[16] [2014-01-16 08:00:00] [2014-01-17 08:00:00] [2014-01-18 08:00:00]
[19] [2014-01-19 08:00:00] [2014-01-20 08:00:00] [2014-01-21 08:00:00]
[22] [2014-01-22 08:00:00] [2014-01-23 08:00:00] [2014-01-24 08:00:00]
[25] [2014-01-25 08:00:00] [2014-01-26 08:00:00] [2014-01-27 08:00:00]
[28] [2014-01-28 08:00:00] [2014-01-29 08:00:00] [2014-01-30 08:00:00]
[31] [2014-01-31 08:00:00]
```

or as

```
> timeCalendar(currentYear, m = 1, d = 1:31, zone = "Tokyo", FinCenter = "Zurich") +
     16 * 3600

Zurich
 [1] [2014-01-01 08:00:00] [2014-01-02 08:00:00] [2014-01-03 08:00:00]
 [4] [2014-01-04 08:00:00] [2014-01-05 08:00:00] [2014-01-06 08:00:00]
 [7] [2014-01-07 08:00:00] [2014-01-08 08:00:00] [2014-01-09 08:00:00]
[10] [2014-01-10 08:00:00] [2014-01-11 08:00:00] [2014-01-12 08:00:00]
[13] [2014-01-13 08:00:00] [2014-01-14 08:00:00] [2014-01-15 08:00:00]
[16] [2014-01-16 08:00:00] [2014-01-17 08:00:00] [2014-01-18 08:00:00]
[19] [2014-01-19 08:00:00] [2014-01-20 08:00:00] [2014-01-21 08:00:00]
[22] [2014-01-22 08:00:00] [2014-01-23 08:00:00] [2014-01-24 08:00:00]
[25] [2014-01-25 08:00:00] [2014-01-26 08:00:00] [2014-01-27 08:00:00]
[28] [2014-01-28 08:00:00] [2014-01-29 08:00:00] [2014-01-30 08:00:00]
[31] [2014-01-31 08:00:00]
```

where we have added 16 hours in seconds.

## 2.7 HANDLING NON ISO-8601 FORMATS

How to proceed when our character vector of dates is not ISO-8601 formatted, for example if we are concerned with American date formats? For this we can specify the format string explicitly

```
> dsAmerican <- c("9/28/1989", "1/15/2001", "8/30/2004")
> timeDate(dsAmerican, format = "%m/%d/%Y")
GMT
[1] [1989-09-28] [2001-01-15] [2004-08-30]
```

Note, the function `timeDate` has implemented an auto-detection for non ISO-8601 formats including American date formats. Thus you can just type

```
> timeDate(dsAmerican)
GMT
[1] [1989-09-28] [2001-01-15] [2004-08-30]
```

Unfortunately, the details of the formats are system-specific, but the following strings are defined by the ISO C / POSIX standard and are likely to be widely available. A conversion specification is introduced by `%`, usually followed by a single letter or `O` or `E` and then a single letter. Any character in the format string not part of a conversion specification is interpreted literally (and %% gives %). Widely implemented conversion specifications include

LISTING 2.1: POSIX FORMAT IDENTIFIERS.

```
Format
%a      Abbreviated weekday name
%b      Abbreviated month name
%d      Day of the month as decimal number (01-31)
%H      Hours as decimal number (00-23)
%j      Day of year as decimal number (001-366)
%m      Month as decimal number (01-12)
%M      Minute as decimal number (00-59)
%S      Second as decimal number (00-61), up to 2 leap-seconds.
%U      Week of the year as decimal number (00-53)
%y      Year without century (00-99)
%Y      Year with century.
%z      Offset from Greenwich, so '-0800' is 8 hours west
%Z      Time zone as a character string
%F      Equivalent to %Y-%m-%d (the ISO 8601 date format)
%u      Weekday as a decimal number (1-7, Monday is 1)
%D      Locale-specific date format such as '%m/%d/%y'
%T      Equivalent to '%H:%M:%S'
```

For a detailed description of the format conversion specifications we refer to R's help page sgtrptime(base).

## 2.8 MIDNIGHT STANDARD

The human readible international standard notation for the time of day is "hh:mm:ss" where hh is the number of complete hours that have passed since midnight (00-24), mm is the number of complete minutes that have passed since the start of the hour (00-59), and ss is the number of complete seconds since the start of the minute (00-60). Note, if the hour value is 24, then the minute and second values must be zero.
As every day both starts and ends with midnight, the two notations 00:00 and 24:00 are available to distinguish the two midnights that can be associated with one date. This means that the following two notations refer to exactly the same point in time: "2008-12-31 24:00:00" and "2009-01-011 00:00:00" The midnight standard function in Rmetrics takes care of this and reverts the first midnigth date/time stamp to the second one, i.e.

```
> midnightStandard("2008-12-31 24:00:00")
[1] "2009-01-01 00:00:00"
```

## 2.9 PRINTING AND FORMATTING 'TIMEDATE' OBJECTS

In format() is a S3 generic function for pretty printing or in other words for encoding objects in a common format.

### *whichFormat: Auto-detection of 'timeDate' Formats*

Which format a timeDate objects have can be detected by the function whichFormat.

```
> args(whichFormat)
function (charvec, silent = FALSE)
NULL
```

The first argument of the function requires a value or vector of timeDate objects. It allows to print a warning, if the format could not be recognized. The following formats can be auto-detected

LISTING 2.2: ISO FORMAT SUMMARY.

```
Human readable ISO:
    "%Y"
    "%Y-%m"
    "%Y-%m-%d"
    "%Y-%m-%d %H"
    "%Y-%m-%d %H:%M"
    "%Y-%m-%d %H:%M:%S"
Short ISO:
    "%Y%m"
    "%Y%m%d"
    "%Y%m%d%H"
    "%Y%m%d%H%M"
    "%Y%m%d%H%M%S"
American Formats:
    "%m/%d/%Y"
    "%d-%b-%Y"
```

Note, misspecification may occur, if the interpretation of the format is not unique. Here, are three examples:

```
> whichFormat("2008-01-01")
[1] "%Y-%m-%d"
> whichFormat("1/1/2008")
[1] "%m/%d/%Y"
```

where the last returns "unknown", i.e. the time stamp format could not be auto-detected.

*format: Non ISO-8601 Reformatting and Printing*

Reformatting and printing in other than ISO-8601 formats can be done using the function format, and an appropriate format specification string. Here comes an example for a local specific version of `timeDate()`

```
> # locale-specific version of timeDate()
> format(Sys.timeDate(), format="%a %b %d %X %Y %Z")
[1] "Mon Dec 08 10:46:31 2014 GMT"
```

*print: The Printing Command*

Printing with the print command is done in the usual way, calling an S4 method

```
> print(timeSequence())
GMT
 [1] [2014-11-09 10:46:31] [2014-11-10 10:46:31] [2014-11-11 10:46:31]
 [4] [2014-11-12 10:46:31] [2014-11-13 10:46:31] [2014-11-14 10:46:31]
 [7] [2014-11-15 10:46:31] [2014-11-16 10:46:31] [2014-11-17 10:46:31]
[10] [2014-11-18 10:46:31] [2014-11-19 10:46:31] [2014-11-20 10:46:31]
[13] [2014-11-21 10:46:31] [2014-11-22 10:46:31] [2014-11-23 10:46:31]
[16] [2014-11-24 10:46:31] [2014-11-25 10:46:31] [2014-11-26 10:46:31]
[19] [2014-11-27 10:46:31] [2014-11-28 10:46:31] [2014-11-29 10:46:31]
[22] [2014-11-30 10:46:31] [2014-12-01 10:46:31] [2014-12-02 10:46:31]
[25] [2014-12-03 10:46:31] [2014-12-04 10:46:31] [2014-12-05 10:46:31]
[28] [2014-12-06 10:46:31] [2014-12-07 10:46:31] [2014-12-08 10:46:31]
> show(timeCalendar())
```

```
GMT
 [1] [2014-01-01] [2014-02-01] [2014-03-01] [2014-04-01] [2014-05-01]
 [6] [2014-06-01] [2014-07-01] [2014-08-01] [2014-09-01] [2014-10-01]
[11] [2014-11-01] [2014-12-01]
```

Note, in the first line we find the name of the financial center, or "GMT", and in the following line(s) the formatted date/time stamps enclosed in square brackets, which makes the identification as `timeDate` object quite easy.

# CHAPTER 3

# FINANCIAL CENTERS

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

## 3.1 SETTING FINANCIAL CENTERS

Each financial center worldwide has a function which returns Daylight Saving Time Rules. Almost 400 prototypes are made available through the Olson time zone data base.
The cities and regions can be listed using the command `listFinCenter()`. The DST rules for specific financial centers can be viewed by their name, e.g. `Zurich()`. Additional financial centers can be added by the user taking care of the format specification of the DST functions.
All time stamps are handled according to the time zone and daylight saving time rules specified by the center through the variable `myFinCenter`. This variable is set by default to `"GMT"` but can be changed to your local financial center or to any other financial center you want to use.
NOTE: By setting the financial center to a continent/city which lies outside of the time zone used by your computer does not change any time settings or environment variables used by your computer.
To change the name of a financial center from one setting to another just assign to the variable `myFinCenter` the desired name of the city:

```
> # What is my current Financial Center ?
> getRmetricsOptions("myFinCenter")
myFinCenter
      "GMT"
```

```
> # Change to Zurich:
> setRmetricsOptions(myFinCenter="Zurich")
> getRmetricsOptions("myFinCenter")

myFinCenter
   "Zurich"
```

From now on, all dates and times are handled within the middle European time zone and the DST rules which are valid for Zurich.

## 3.2 LIST OF FINANCIAL CENTERS

There are many other financial centers supported by Rmetrics. They can be displayed by the function `listFinCenter()`. You can also display partial lists with wildcards and regular expressions. To count all supported financial centers worldwide, type

```
> allCenters <- listFinCenter()
> length(allCenters)

[1] 397
```

To list all centers in Asia

```
> listFinCenter("Asia/*")

 [1] "Asia/Aden"          "Asia/Almaty"        "Asia/Amman"
 [4] "Asia/Anadyr"        "Asia/Aqtau"         "Asia/Aqtobe"
 [7] "Asia/Ashgabat"      "Asia/Baghdad"       "Asia/Bahrain"
[10] "Asia/Baku"          "Asia/Bangkok"       "Asia/Beirut"
[13] "Asia/Bishkek"       "Asia/Brunei"        "Asia/Calcutta"
[16] "Asia/Choibalsan"    "Asia/Chongqing"     "Asia/Colombo"
[19] "Asia/Damascus"      "Asia/Dhaka"         "Asia/Dili"
[22] "Asia/Dubai"         "Asia/Dushanbe"      "Asia/Gaza"
[25] "Asia/Harbin"        "Asia/Hong_Kong"     "Asia/Hovd"
[28] "Asia/Irkutsk"       "Asia/Jakarta"       "Asia/Jayapura"
[31] "Asia/Jerusalem"     "Asia/Kabul"         "Asia/Kamchatka"
[34] "Asia/Karachi"       "Asia/Kashgar"       "Asia/Katmandu"
[37] "Asia/Krasnoyarsk"   "Asia/Kuala_Lumpur"  "Asia/Kuching"
[40] "Asia/Kuwait"        "Asia/Macau"         "Asia/Magadan"
[43] "Asia/Makassar"      "Asia/Manila"        "Asia/Muscat"
[46] "Asia/Nicosia"       "Asia/Novosibirsk"   "Asia/Omsk"
[49] "Asia/Oral"          "Asia/Phnom_Penh"    "Asia/Pontianak"
[52] "Asia/Pyongyang"     "Asia/Qatar"         "Asia/Qyzylorda"
[55] "Asia/Rangoon"       "Asia/Riyadh"        "Asia/Saigon"
[58] "Asia/Sakhalin"      "Asia/Samarkand"     "Asia/Seoul"
[61] "Asia/Shanghai"      "Asia/Singapore"     "Asia/Taipei"
[64] "Asia/Tashkent"      "Asia/Tbilisi"       "Asia/Tehran"
[67] "Asia/Thimphu"       "Asia/Tokyo"         "Asia/Ulaanbaatar"
[70] "Asia/Urumqi"        "Asia/Vientiane"     "Asia/Vladivostok"
[73] "Asia/Yakutsk"       "Asia/Yekaterinburg" "Asia/Yerevan"
```

To list all centers worldwide in cities starting with a "Z"

```
> listFinCenter(".*/Z")
```

```
[1] "Europe/Zagreb"     "Europe/Zaporozhye" "Europe/Zurich"
```

## 3.3 DAYLIGHT SAVING TIME RULES

For each financial center a function is available. It keeps the information of the time zones and the DST rules. The functions return a data.frame with 4Columns:

```
Zurich offSet isdst TimeZone
...
62  2008-03-30 01:00:00   7200     1     CEST
63  2008-10-26 01:00:00   3600     0      CET
64  2009-03-29 01:00:00   7200     1     CEST
65  2009-10-25 01:00:00   3600     0      CET
...
```

The first column describes when the time was changed, the second gives the offset to "GMT", the third returns the daylight savings time flag which is positive if in force, zero if not, and negative if unknown. The last column gives the name of the time zone. You can have a look at the function `Zurich()()`.

```
> Zurich()[64:74, ]
                Zurich offSet isdst TimeZone    numeric
64 2010-03-28 01:00:00   7200     1     CEST 1269738000
65 2010-10-31 01:00:00   3600     0      CET 1288486800
66 2011-03-27 01:00:00   7200     1     CEST 1301187600
67 2011-10-30 01:00:00   3600     0      CET 1319936400
68 2012-03-25 01:00:00   7200     1     CEST 1332637200
69 2012-10-28 01:00:00   3600     0      CET 1351386000
70 2013-03-31 01:00:00   7200     1     CEST 1364691600
71 2013-10-27 01:00:00   3600     0      CET 1382835600
72 2014-03-30 01:00:00   7200     1     CEST 1396141200
73 2014-10-26 01:00:00   3600     0      CET 1414285200
74 2015-03-29 01:00:00   7200     1     CEST 1427590800
```

Here the index extracts the years 2010 to 2015, have a look on the complete data frame.

# CHAPTER 4

# GROUP GENERIC OPERATIONS

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

Many operations can be performed on `timeDate` objects. You can add and subtract, round and truncate, subset, coerce or transform them to other objects. These are only few options among many others. One of the most important operations are mathematical operations which allow to extract, add and subtract dates and times, and to perform logical operations on `timeDate` objects.

Group generic methods are defined related to R's four prespecified groups of functions, Math, Ops, Summary and Complex. You will find these objects in R's methods package.

LISTING 4.1: TIMEDATE GROUP GENERIC FUNCTIONS.

```
Group Math
    includes the functions round() and trunc()
Group Ops
    (i) "+", "-",  % "*", "/", "^", "%%", "%/%",
    (ii) "&", "|", "!",
    (iii) "==", "!=", "<", "<=", ">=", ">".
Group Summary
    all}, any, sum, prod, min, max, range
Group Complex
```

## 4.1 Rounding and Truncating 'timeDate' Objects

Dates and times can be rounded and/or truncated. To do this, use the functions `round` and `trunc`.
For example round random time stamps to the nearest hour:

```
> tC <- timeCalendar()
> tR <- tC + round(3600 * rnorm(12))
> tR

GMT
 [1] [2014-01-01 00:09:08] [2014-01-31 23:44:03] [2014-03-01 00:13:17]
 [4] [2014-04-01 02:07:50] [2014-05-01 01:11:30] [2014-05-31 23:30:28]
 [7] [2014-07-01 00:41:07] [2014-08-01 00:15:37] [2014-09-01 01:23:17]
[10] [2014-10-01 00:55:52] [2014-10-31 23:31:28] [2014-12-01 00:53:39]

> round(tR, "h")

GMT
 [1] [2014-01-01 00:00:00] [2014-02-01 00:00:00] [2014-03-01 00:00:00]
 [4] [2014-04-01 02:00:00] [2014-05-01 01:00:00] [2014-06-01 00:00:00]
 [7] [2014-07-01 01:00:00] [2014-08-01 00:00:00] [2014-09-01 01:00:00]
[10] [2014-10-01 01:00:00] [2014-11-01 00:00:00] [2014-12-01 01:00:00]
```

or as another example truncate random time stamps to the next hour:

```
> trunc(tR + 3600, "h")

GMT
 [1] [2014-01-01 01:00:00] [2014-02-01 00:00:00] [2014-03-01 01:00:00]
 [4] [2014-04-01 03:00:00] [2014-05-01 02:00:00] [2014-06-01 00:00:00]
 [7] [2014-07-01 01:00:00] [2014-08-01 01:00:00] [2014-09-01 02:00:00]
[10] [2014-10-01 01:00:00] [2014-11-01 00:00:00] [2014-12-01 01:00:00]
```

## 4.2 Subtracting and Adding 'timeDate' Objects

`timeDate` objects can be subtracted. Let us calculate the end of day 16:00 difference between Zurich and New York:

```
> ZRH <- timeDate(c("2009-03-08 16:00", "2009-03-29 16:00"), zone = "Zurich",
     FinCenter = "Zurich")
> ZRH

Zurich
[1] [2009-03-08 16:00:00] [2009-03-29 16:00:00]

> NYC <- timeDate(c("2009-03-08 16:00", "2009-03-29 16:00"), zone = "New_York",
     FinCenter = "Zurich")
> NYC

Zurich
[1] [2009-03-08 21:00:00] [2009-03-29 22:00:00]

> NYC - ZRH

Time differences in hours
[1] 5 6
```

Note, he result expresses the fact that DST becomes effective at two different dates in Zurich and New York.
On the other hand from a `timeDate` object we can subtract or we can add a given time span, say 1 hour which corresponds to 3600 seconds

```
> oneHour <- 3600
> ZRH

Zurich
[1] [2009-03-08 16:00:00] [2009-03-29 16:00:00]

> ZRH - oneHour

Zurich
[1] [2009-03-08 15:00:00] [2009-03-29 15:00:00]

> ZRH + oneHour

Zurich
[1] [2009-03-08 17:00:00] [2009-03-29 17:00:00]
```

*Getting Date and Time as 'timeDate' Objects*

What is the current date and time returned as a `timeDate` object? Current date and time as a `timeDate` object can be obtained from the function `Sys.timeDate`:

```
> # Date and Time Now:
> Sys.timeDate()

GMT
[1] [2014-12-08 10:46:29]
```

The date and time one hour later can just be obtained adding one hour or equivalently 3'600 seconds:

```
> # Date and Time One Hour Later:
> oneDay <- 24 * oneHour
> Sys.timeDate() + oneHour

GMT
[1] [2014-12-08 11:46:29]

> Sys.timeDate() + oneDay

GMT
[1] [2014-12-09 10:46:29]
```

## 4.3 COMPARING 'TIMEDATE' OBJECTS

Sometimes we are interested if two events happen at the same time.

```
> Sys.timeDate("Zurich") == Sys.timeDate("Tokyo")

[1] TRUE
```

# Chapter 5

# Subsetting 'timeDate' Objects

```
> library(timeDate)
```

## 5.1 Rmetrics Functions for Subsetting 'timeDate Objects

Subsetting a `timeDate` is a very important issue in the management of dates and times. Rmetrics offers several functions which are useful in this context.

Listing 5.1: Subsetting timeDate Objects.

```
Option:
Extracting elements by integer subsetting
Extracting elements by logical predicates
Extracting elements by python like indexing
Extracting elements by span indexes
Extracting the first, the last, and the range
Extracting elements by timeDate indexes
Extracting elements using the window function
Extracting weekdays and business days
```

The most common type of subsetting a `timeDate` object is done by integer indexing. The following example shows three alternatives how to return the last element in 'timeDate' vector. First create a dummy calendar series

## 5.2 Extracting Elements by Integer Subsetting

```
> tC <- timeCalendar()
> tC
```

```
GMT
 [1] [2014-01-01] [2014-02-01] [2014-03-01] [2014-04-01] [2014-05-01]
 [6] [2014-06-01] [2014-07-01] [2014-08-01] [2014-09-01] [2014-10-01]
[11] [2014-11-01] [2014-12-01]
```

and then retrieve the first and last date:

```
> c(tC[12], tC[length(tC)], rev(tC)[1])

GMT
[1] [2014-12-01] [2014-12-01] [2014-12-01]
```

For example, if we have a monthly series like `tC`, we can simply generate a quarterly series by subsetting

```
> tC[c(1, 4, 7, 10)]

GMT
[1] [2014-01-01] [2014-04-01] [2014-07-01] [2014-10-01]

> tC[seq(1, 12, by = 3)]

GMT
[1] [2014-01-01] [2014-04-01] [2014-07-01] [2014-10-01]
```

Like for a numeric vector, negative indexes can be used to delete part of a series, e.g. removing the data of the first half year we proceed in the following way

```
> tC[-(1:6)]

GMT
[1] [2014-07-01] [2014-08-01] [2014-09-01] [2014-10-01] [2014-11-01]
[6] [2014-12-01]
```

## 5.3 EXTRACTING ELEMENTS BY LOGICAL PREDICATES

Logical subsetting is in many cases useful if we like for example to extract from a vector those time stamps which are before or after a given date and time

```
> tC[tC > tC[6]]

GMT
[1] [2014-07-01] [2014-08-01] [2014-09-01] [2014-10-01] [2014-11-01]
[6] [2014-12-01]
```

We can also create constructs as *on or before* or *on or after*

```
> tC[tC >= tC[6]]

GMT
[1] [2014-06-01] [2014-07-01] [2014-08-01] [2014-09-01] [2014-10-01]
[6] [2014-11-01] [2014-12-01]
```

## 5.4 EXTRACTING ELEMENTS BY PYTHON LIKE INDEXING

Subsetting via "[" methods offers also the ability to specify dates by range, if they are enclosed in quotes. This type of subsetting creates ranges with a double colon "::" operator. Each side of the operator may be left blank, which would then default to the beginning and end of the data, respectively. To specify a subset of times, it is only required that the time specified be in the human readible form of the standard ISO format. The time must be 'left-filled', that is to specify a full year one needs only to provide the year, a month would require the full year and the integer of the month requested.

Subset all records in 2014

```
> tC["2014"]

GMT
 [1] [2014-01-01] [2014-02-01] [2014-03-01] [2014-04-01] [2014-05-01]
 [6] [2014-06-01] [2014-07-01] [2014-08-01] [2014-09-01] [2014-10-01]
[11] [2014-11-01] [2014-12-01]
```

Subset all records from 2014-07:

```
> tC["2014-07::"]

GMT
[1] [2014-07-01] [2014-08-01] [2014-09-01] [2014-10-01] [2014-11-01]
[6] [2014-12-01]
```

Subset all records on or after 2014-08-25

```
> tC["2014-08-25::"]

GMT
[1] [2014-09-01] [2014-10-01] [2014-11-01] [2014-12-01]
```

Subset all records from April to August 2014

```
> tC["2014-04::2014-08"]

GMT
[1] [2014-04-01] [2014-05-01] [2014-06-01] [2014-07-01] [2014-08-01]
```

Subset all records up to August 2014

```
> tC["::2014-08"]

GMT
[1] [2014-01-01] [2014-02-01] [2014-03-01] [2014-04-01] [2014-05-01]
[6] [2014-06-01] [2014-07-01] [2014-08-01]
```

## 5.5 Extracting Elements by Span Indexes

We can also subset defining time spans, the origin of the span, and the direction of the span. To subset the last three months, we type,

```
> tC["last 3 months"]
GMT
[1] [2014-09-01] [2014-10-01] [2014-11-01] [2014-12-01]
```

Note, if we ask for the last 5 days we end up with one calendar date, since the other 4 days are missing in the monthly calendar series.

```
> tC["last 5 days"]
GMT
[1] [2014-12-01]
```

## 5.6 Extracting the first, the last, and the range of 'timeDate' Objects

To extract the first element, the last element and the range of a vector of `timeDate` objects we can use the functions `start`, `end`, and `range`. The functions `start` sorts the `timeDate` objects in increasing order and returns the first element of the sorted vector

```
> start(tC)
GMT
[1] [2014-01-01]
> start(rev(tC))
GMT
[1] [2014-01-01]
```

To demonstrate this we have reverted the time sequence `tC` and calculated the starting value again. Similarily, the function `end` returns the latest date/time stamp in the 'timeDate' vector, e.g.

```
> end(tC)
GMT
[1] [2014-12-01]
```

Finally, the function `range()` keeps just the start and end values in a `timeDate` vector of two elements

```
> range(tC)
GMT
[1] [2014-01-01] [2014-12-01]
> c(start(tC), end(tC))
GMT
[1] [2014-01-01] [2014-12-01]
```

## 5.7 EXTRACTING ELEMENTS BY 'TIMEDATE' INDEXES

We can subset by character strings, or by functions which create such strings

```
> tC["2014-01-01"]

GMT
[1] [2014-01-01]

> tC["2014-01-02"]

GMT
[1] [NA]
```

if the date is not available NA will be returned. Using `timeDate` objects instead of character strings is not possible, you first have to convert the `timeDate` objects to type character using the functions `as.character()`

```
> class(start(tC))

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

> tC[as.character(start(tC))]

GMT
[1] [2014-01-01]
```

## 5.8 EXTRACTING ELEMENTS USING THE WINDOW FUNCTION

`window()` is a generic function which extracts the subset of its input argument observed between two times, the `start` and `end` time. For example to subset all data records between the 4th and 9th date stamp of the calendar series `tC`

```
> start <- tC[4]
> end <- tC[9]
> c(start, end)

GMT
[1] [2014-04-01] [2014-09-01]

> window(tC, start = start, end = end)

GMT
[1] [2014-04-01] [2014-05-01] [2014-06-01] [2014-07-01] [2014-08-01]
[6] [2014-09-01]
```

## 5.9 EXTRACTING WEEKDAYS AND BUSINESS DAYS

Weekdays, weekends, business days, and holidays can be easily obtained with the following functions: `isWeekday()` and `isWeekend()` test if a 'timeDate' is a weekday or not, `isBizday()` and `isHoliday()` test if a date is a business day or not.
Let us start with a sequence of `timeDate` objects around Easter 2014. The function Easter allows not only to compute the date of Easter but also dates shifted by a given number of days around Easter. E.g. `Easter(2014, -14)` returns the date two weeks before Easter Sunday.
First we generate a time sequence around Easter 2014

```
> Easter(2014)
GMT
[1] [2014-04-20]
> tS <- timeSequence(Easter(2014, -14), Easter(2014, +14))
> tS
GMT
 [1] [2014-04-06] [2014-04-07] [2014-04-08] [2014-04-09] [2014-04-10]
 [6] [2014-04-11] [2014-04-12] [2014-04-13] [2014-04-14] [2014-04-15]
[11] [2014-04-16] [2014-04-17] [2014-04-18] [2014-04-19] [2014-04-20]
[16] [2014-04-21] [2014-04-22] [2014-04-23] [2014-04-24] [2014-04-25]
[21] [2014-04-26] [2014-04-27] [2014-04-28] [2014-04-29] [2014-04-30]
[26] [2014-05-01] [2014-05-02] [2014-05-03] [2014-05-04]
```

For the sequence `tS` of `timeDate` objects we now subset the weekdays and show the day of the week using the function `dayOfWeek()` to check the result

```
> tW <- tS[isWeekday(tS)]
> tW
GMT
 [1] [2014-04-07] [2014-04-08] [2014-04-09] [2014-04-10] [2014-04-11]
 [6] [2014-04-14] [2014-04-15] [2014-04-16] [2014-04-17] [2014-04-18]
[11] [2014-04-21] [2014-04-22] [2014-04-23] [2014-04-24] [2014-04-25]
[16] [2014-04-28] [2014-04-29] [2014-04-30] [2014-05-01] [2014-05-02]
> dayOfWeek(tW)
2014-04-07 2014-04-08 2014-04-09 2014-04-10 2014-04-11 2014-04-14 2014-04-15
     "Mon"      "Tue"      "Wed"      "Thu"      "Fri"      "Mon"      "Tue"
2014-04-16 2014-04-17 2014-04-18 2014-04-21 2014-04-22 2014-04-23 2014-04-24
     "Wed"      "Thu"      "Fri"      "Mon"      "Tue"      "Wed"      "Thu"
2014-04-25 2014-04-28 2014-04-29 2014-04-30 2014-05-01 2014-05-02
     "Fri"      "Mon"      "Tue"      "Wed"      "Thu"      "Fri"
```

or in the case of weekend days

```
> tW <- tS[isWeekend(tS)]
> tW
GMT
[1] [2014-04-06] [2014-04-12] [2014-04-13] [2014-04-19] [2014-04-20]
[6] [2014-04-26] [2014-04-27] [2014-05-03] [2014-05-04]
```

```
> dayOfWeek(tW)

2014-04-06 2014-04-12 2014-04-13 2014-04-19 2014-04-20 2014-04-26 2014-04-27
     "Sun"      "Sat"      "Sun"      "Sat"      "Sun"      "Sat"      "Sun"
2014-05-03 2014-05-04
     "Sat"      "Sun"
```

In the same way we can calculate business days in Zurich using the holiday calendar for Zurich, `holidayZURICH()`,

```
> # Subset Business Days:
> tB <- tS[isBizday(tS, holidayZURICH())]
> tB

GMT
 [1] [2014-04-07] [2014-04-08] [2014-04-09] [2014-04-10] [2014-04-11]
 [6] [2014-04-14] [2014-04-15] [2014-04-16] [2014-04-17] [2014-04-22]
[11] [2014-04-23] [2014-04-24] [2014-04-25] [2014-04-28] [2014-04-29]
[16] [2014-04-30] [2014-05-02]

> dayOfWeek(tB)

2014-04-07 2014-04-08 2014-04-09 2014-04-10 2014-04-11 2014-04-14 2014-04-15
     "Mon"      "Tue"      "Wed"      "Thu"      "Fri"      "Mon"      "Tue"
2014-04-16 2014-04-17 2014-04-22 2014-04-23 2014-04-24 2014-04-25 2014-04-28
     "Wed"      "Thu"      "Tue"      "Wed"      "Thu"      "Fri"      "Mon"
2014-04-29 2014-04-30 2014-05-02
     "Tue"      "Wed"      "Fri"
```

# CHAPTER 6

# BASE FUNCTIONS AND METHODS

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

In this chapter we present functions for time series objects which we know from R's base package for vectors, matrices, and/or data.frames.

LISTING 6.1: TIMEDATE FUNCTIONS KNOWN FROM R'S BASE PACKAGE.

```
Function:
align           Aligning a timeDate Object
c               Combining timeDate Objects
diff            Differencing a timeDate Object
duplicated      Determining Duplicates in a timeDate Objects
length          Measuring the Length of timeDate Objects
head            Extracting First and Last few timeDate Objects
min             Extracting First and Last Entries from a timeDate Object
range           Extracting the Range of a timeDate Object
rep             Replicating Elements of a timeDate Object
sample          Sampling randomly a timeDate Object
sort            Sorting or Ordering a timeDate Object
start           Extracting Terminal Times of a timeDate Object
unique          Extracting Unique Elements of a timeDate Object
weekdays        Extracting Parts of a timeDate Object
window          Subsetting a time window from a timeDate Object
```

## 6.1 `c()` - COMBINING OBJECTS

The function `c()` is a generic function in R(base) which combines its arguments. The 'timeDate' method of the generic function `c()` combines or in other words concatenates `timeDate` objects.

```
> getArgs("c")

function (..., recursive = FALSE)
NULL
c,:
NULL
```

The example shows how to combine the last and first quarter of two time calendar stamps calculated for the years 2008 and 2009 using Python like subsetting of time stamps.

```
> tC08 <- timeCalendar(2008)
> tC09 <- timeCalendar(2009)
> c(tC08["2008-10::"], tC09["::2009-12"])

GMT
 [1] [2008-10-01] [2008-11-01] [2008-12-01] [2009-01-01] [2009-02-01]
 [6] [2009-03-01] [2009-04-01] [2009-05-01] [2009-06-01] [2009-07-01]
[11] [2009-08-01] [2009-09-01] [2009-10-01] [2009-11-01] [2009-12-01]
```

What happens when we have `timeDate` objects from two different financial centers? In this case the first determines the financial center setting of the combind ‘timeDate’ vector:

```
> ZRH <- timeDate("2008-03-15 16:00", zone = "Zurich", FinCenter = "Zurich")
> ZRH

Zurich
[1] [2008-03-15 16:00:00]

> NYC <- timeDate("2008-03-15 16:00", zone = "New_York", FinCenter = "New_York")
> NYC

New_York
[1] [2008-03-15 16:00:00]

> c(ZRH, NYC)

Zurich
[1] [2008-03-15 16:00:00] [2008-03-15 21:00:00]

> c(NYC, ZRH)

New_York
[1] [2008-03-15 16:00:00] [2008-03-15 11:00:00]
```

## 6.2 `diff()` - DIFFERENCING AN OBJECT

The generic function `diff()` returns suitably lagged and iterated differences. It comes with a default method in R (base) and ones for classes `ts`, `POSIXt` and `Date` as well as one for `timeDate`.

```
> tC <- timeCalendar(2009)
> tC
```

```
GMT
 [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]
```

The number of days in each month are

```
> daysInMonth <- diff(c(timeDate("2008-12-01"), tC))
> daysInMonth
```

```
Time differences in days
 [1] 31 31 28 31 30 31 30 31 31 30 31 30
```

What type of class is it, and convert it to type integer

```
> class(daysInMonth)
```

```
[1] "difftime"
```

```
> as.integer(daysInMonth)
```

```
 [1] 31 31 28 31 30 31 30 31 31 30 31 30
```

This example returns the number of days in each month for the year 2009. The class of the returned object by the function `diff()` is an object of class `difftime`. To derive integer values from the returned object `DaysInMonth` we can just use the function `as.integer()` as it is shown in the example above.
The function `difftimeDate()` in Rmetrics works like the function `difftime()` in R (base). The function calculates a difference of two `timeDate` objects and returns an object of class `difftime` with an attribute indicating the units.
There is a round method for objects of this class, as well as methods for the group-generic (see Ops) logical and arithmetic operations. If its argument `units="auto"`, a suitable set of units is chosen, the largest possible (excluding "weeks") in which all the absolute differences are greater than one.
Subtraction of `timeDate` objects gives an object of this class, by calling `difftimDate()` with argument `units="auto"`.

```
> timeCalendar(2009) - timeCalendar(2008)
```

```
Time differences in days
 [1] 366 366 365 365 365 365 365 365 365 365 365 365
```

```
> difftimeDate(timeCalendar(2009), timeCalendar(2008))
```

```
Time differences in days
 [1] 366 366 365 365 365 365 365 365 365 365 365 365
```

Limited arithmetic is available on `difftime` objects: they can be added or subtracted, and multiplied or divided by a numeric vector. In addition, adding or subtracting a numeric vector implicitly converts the numeric vector to a 'difftime' object with the same units as the 'difftime' object.

The units of a `difftime` object can be extracted by the units function, which also has an replacement form. If the units are changed, the numerical value is scaled accordingly.

```
> DaysInMonth <- diff(timeCalendar(2009))
> DaysInMonth
Time differences in days
 [1] 31 28 31 30 31 30 31 31 30 31 30
> units(DaysInMonth)
[1] "days"
```

The `as.double()` method returns the numeric value expressed in the specified units. Using `units="auto"` means the units of the object. The format method simply formats the numeric value and appends the units as a text string.

## 6.3 `duplicated()` - DETERMINING DUPLICATE ELEMENTS

The function `duplicated()` is a S3 generic function in R(base)

```
> methods(duplicated)
[1] duplicated.array           duplicated.data.frame
[3] duplicated.default         duplicated.matrix
[5] duplicated.numeric_version duplicated.POSIXlt
```

which determines which elements of a vector or data frame are duplicates of elements with smaller subscripts, and returns a logical vector indicating which elements (rows) are duplicates. We can first format the `timeDate` and then apply the function `duplicated()`

```
> tC <- c(timeCalendar(2008, 1:3), timeCalendar(2008)[1:3])
> tC
GMT
[1] [2008-01-01] [2008-02-01] [2008-03-01] [2008-01-01] [2008-02-01]
[6] [2008-03-01]
> duplicated(format(tC))
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

or we can add a new S3 method for the function `duplicated()` which handles `timeDate` objects

```
> duplicated.timeDate <- function(x, ...) duplicated(format(x),
     ...)
> tC
GMT
[1] [2008-01-01] [2008-02-01] [2008-03-01] [2008-01-01] [2008-02-01]
[6] [2008-03-01]
> duplicated(tC)
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

## 6.4 head() - EXTRACTING FIRST AND LAST FEW OBJECTS

The functions `head()` and `tail()` are generic functions defined from R (utils). We can use them to extract the first (last) `n` entries from a vector of `timeDate` objects

```
> head(timeCalendar())

GMT
[1] [2014-01-01] [2014-02-01] [2014-03-01] [2014-04-01] [2014-05-01]
[6] [2014-06-01]

> tail(timeCalendar(), n = 3)

GMT
[1] [2014-10-01] [2014-11-01] [2014-12-01]
```

Note, head and tail return the first and last records of a timeDate object, the returned values are not ordered in time

```
> tS <- sample(timeCalendar())
> head(tS)

GMT
[1] [2014-04-01] [2014-12-01] [2014-01-01] [2014-05-01] [2014-06-01]
[6] [2014-07-01]

> tail(tS)

GMT
[1] [2014-02-01] [2014-08-01] [2014-09-01] [2014-11-01] [2014-10-01]
[6] [2014-03-01]
```

otherwise first sort the data using the function `sort()`.

## 6.5 length() - MEASURING THE LENGTH OF AN OBJECT

The function `length()` is a `.Primitive` function in R (base), and can be used for measuring the length of a vector of `timeDate` objects

```
> length(timeCalendar())

[1] 12
```

## 6.6 min() - EXTRACTING FIRST AND LAST ENTRIES FROM AN OBJECT

The functions `min()` and `max()` are generic functions and can be used in the same sense as the functions `start()` and `end()`, to return the earliest and latest entry of a `timeDate` object. The example returns the range:

```
> c(min(tC), max(tC))

GMT
[1] [2008-01-01] [2008-03-01]
```

## 6.7 `range()` - Extracting the Range of an Object

The function `range()` is a generic function which returns a two element vector containing the minimum and maximum of all the given arguments, i.e. the range of the vector of `timeDate` objects

```
> range(timeCalendar(2008))

GMT
[1] [2008-01-01] [2008-12-01]
```

## 6.8 `rep()` - Replicating Elements

The function `rep()` is a generic function in R(base) which replicates the values of its first argument in `x`.
To replicate a `timeDate` object proceed as follows. First extract the end of month dates

```
> tc <- timeCalendar()
> tEOM <- timeLastDayInMonth(tC)
> tEOM

GMT
[1] [2008-01-31] [2008-02-29] [2008-03-31] [2008-01-31] [2008-02-29]
[6] [2008-03-31]
```

Then compute the differences

```
> difftimeDate(tEOM, rep(tC[1], times = 12), units = "days")

Time differences in days
 [1] 30 59 90 30 59 90 30 59 90 30 59 90

> difftimeDate(tEOM, rep(tC[c(1, 4, 7, 10)], each = 3), units = "d")

Time differences in days
 [1] 30 59 90 30 59 90 NA NA NA NA NA NA
```

The value of `tC` the first date in month for the current year, and `tEOM` the date of the last day in each month. The first example returns the cumulated number of days during the year, and the example again the cumulated number of days, but with respect to the beginning of each quarter.

## 6.9 `sample()` - Sampling Objects Randomly

The function `sample()` is a non-generic function in R(base) made generic by Rmetrics. `sample()` takes a sample of the specified size from the elements of its argument, either with or without replacement. The following example shows how to sample a monthly `timeDate` vector:

```
> tC <- timeCalendar()
> sample(tC)
```

```
GMT
 [1] [2014-03-01] [2014-10-01] [2014-09-01] [2014-02-01] [2014-07-01]
 [6] [2014-06-01] [2014-01-01] [2014-04-01] [2014-05-01] [2014-12-01]
[11] [2014-08-01] [2014-11-01]
```

## 6.10 `sort()` SORTING OR ORDERING AN OBJECT

The function `sort()` is a generic function in R(base). The function sorts (or orders) 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()`.

*Sort in increasing order*

In Rmetrics `sort` can be used to sort a `timeDate` vector in increasing or ascending order

```
> tC <- timeCalendar()
> tS <- sample(tC)
> tS
GMT
 [1] [2014-08-01] [2014-11-01] [2014-03-01] [2014-10-01] [2014-02-01]
 [6] [2014-12-01] [2014-01-01] [2014-05-01] [2014-07-01] [2014-09-01]
[11] [2014-06-01] [2014-04-01]
```

*Sort in increasing order*

The same in decreasing order

```
> sort(tS)
GMT
 [1] [2014-01-01] [2014-02-01] [2014-03-01] [2014-04-01] [2014-05-01]
 [6] [2014-06-01] [2014-07-01] [2014-08-01] [2014-09-01] [2014-10-01]
[11] [2014-11-01] [2014-12-01]
> sort(tS, decreasing = TRUE)
GMT
 [1] [2014-12-01] [2014-11-01] [2014-10-01] [2014-09-01] [2014-08-01]
 [6] [2014-07-01] [2014-06-01] [2014-05-01] [2014-04-01] [2014-03-01]
[11] [2014-02-01] [2014-01-01]
```

## 6.11 `start()` - EXTRACTING TERMINAL TIMES OF AN OBJECT

The functions `start()` and `end()` are generic functions in R (stats) extracting and encoding the times the first and last observations were taken. Note, R provides them only for compatibility with S version 2, we recommend to use the functions `min()` and `max()`.

```
> tC <- timeCalendar()
> c(start(tC), end(tC))

GMT
[1] [2014-01-01] [2014-12-01]
```

See also the functions `min()` and `max()` which behave in the same way.

## 6.12 `unique()` - EXTRACTING UNIQUE ELEMENTS OF AN OBJECT

The function `unique()` is a generic function in R's stats package which returns a vector, data frame or array like object but with duplicate elements/rows removed.
Create a calendar series

```
> tC <- c(timeCalendar()[3:12], timeCalendar()[1:6])
> tC

GMT
 [1] [2014-03-01] [2014-04-01] [2014-05-01] [2014-06-01] [2014-07-01]
 [6] [2014-08-01] [2014-09-01] [2014-10-01] [2014-11-01] [2014-12-01]
[11] [2014-01-01] [2014-02-01] [2014-03-01] [2014-04-01] [2014-05-01]
[16] [2014-06-01]
```

and make it unique

```
> unique(tC)

GMT
 [1] [2014-03-01] [2014-04-01] [2014-05-01] [2014-06-01] [2014-07-01]
 [6] [2014-08-01] [2014-09-01] [2014-10-01] [2014-11-01] [2014-12-01]
[11] [2014-01-01] [2014-02-01]
```

Note, the returned vector is not sorted. See also the functions `duplicated()` which returns duplicated records in a 'timeDate' vector.

## 6.13 `weekdays()` - EXTRACTING PARTS OF AN OBJECT

The functions `weekdays()`, `months()`, `quarters()`, and `julian()` are generic functions in R(base) which allow to extract parts of a `timeDate` object.

*Extract the weekdays*

The weekdays are the days from Monday to Friday. Create a daily sequence from March 28 tio April 7, 2008

```
> tD <- timeSequence(from = "2008-03-28", to = "2008-04-07")
> tD

GMT
 [1] [2008-03-28] [2008-03-29] [2008-03-30] [2008-03-31] [2008-04-01]
 [6] [2008-04-02] [2008-04-03] [2008-04-04] [2008-04-05] [2008-04-06]
[11] [2008-04-07]
```

show the day of the week

```
> dayOfWeek(tD)
2008-03-28 2008-03-29 2008-03-30 2008-03-31 2008-04-01 2008-04-02 2008-04-03
     "Fri"      "Sat"      "Sun"      "Mon"      "Tue"      "Wed"      "Thu"
2008-04-04 2008-04-05 2008-04-06 2008-04-07
     "Fri"      "Sat"      "Sun"      "Mon"
```

and then extract the weekdays

```
> tW <- tD[isWeekday(tD)]
> tW
GMT
[1] [2008-03-28] [2008-03-31] [2008-04-01] [2008-04-02] [2008-04-03]
[6] [2008-04-04] [2008-04-07]
```

*Extract months from a 'timeDate' object*

To extract the months of timDate sequence,

```
> showMethods("months")
Function: months (package base)
x="ANY"
x="timeDate"
```

use the function months()

```
> months(tD)
 [1] 3 3 3 3 4 4 4 4 4 4 4
attr(,"control")
FinCenter
    "GMT"
```

The result is returned as a numeric vector of integers. To get the months use R's bultin constants month.name and month.abb

```
> month.abb
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

```
> month.name
 [1] "January"   "February"  "March"     "April"     "May"       "June"
 [7] "July"      "August"    "September" "October"   "November"  "December"
```

Thus to get the unique months type

```
> unique(month.abb[months(tD)])
[1] "Mar" "Apr"
```

```
> unique(month.name[months(tD)])
[1] "March" "April"
```

*Extract quarters from a 'timeDate' object*

To extract the quarter of `timDate` sequence, use the function `quarters()`,

```
> quarters(tD)
```

```
 [1] "Q1" "Q1" "Q1" "Q1" "Q2" "Q2" "Q2" "Q2" "Q2" "Q2" "Q2"
attr(,"control")
FinCenter
    "GMT"
```

*Extract julian counts from a 'timeDate' object*

To extract Julian counts from a `timDate` sequence, use the function `julian()`,

```
> julian(tD)
```

```
Time differences in days
 [1] 13966 13967 13968 13969 13970 13971 13972 13973 13974 13975 13976
attr(,"origin")
GMT
[1] [1970-01-01]
```

## 6.14 `window()` - SUBSETTING A TIME WINDOW FROM AN OBJECT

The function `window()` is a generic function in R(stats) which extracts the subset of the object x observed between the times `start()` and `end()`.

```
> tC <- timeCalendar()
> tC[c(4, 9)]
```

```
GMT
[1] [2014-04-01] [2014-09-01]
```

```
> window(tC, start = tC[4], end = tC[9])
```

```
GMT
[1] [2014-04-01] [2014-05-01] [2014-06-01] [2014-07-01] [2014-08-01]
[6] [2014-09-01]
```

# CHAPTER 7

# COERCIONS AND TRANSFORMATIONS

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

timeDate objects are not living in an isolated world. Coercions and transformations allow timeDate objects to communicate with other formatted time stamps. Be aware that in most cases information can be lost if the other date/time classes do not support this functionality.
There exist several methods to coerce and transform timeDate objects into other objects. For example as.timeDate.POSIXt returns a 'POSIX' object as timeDate object, as.timeDate.Date returns a Date object as timeDate object, or as.character.timeDate returns a timeDate object as 'character' string. On the other hand, for example as.POSIXct.timeDate converts a timeDate object into an object of class POSIXct, or as.Date.timeDate into an object as of class Date.

## 7.1 FINANCIAL CENTER TRANSFORMATIONS

First let us create for the year 2008 monthly GMT date/time stamps one time stamp on every first day in a month at 4 pm

```
> GMT <- timeCalendar(2008) + 16 * 3600
> GMT

GMT
 [1] [2008-01-01 16:00:00] [2008-02-01 16:00:00] [2008-03-01 16:00:00]
 [4] [2008-04-01 16:00:00] [2008-05-01 16:00:00] [2008-06-01 16:00:00]
 [7] [2008-07-01 16:00:00] [2008-08-01 16:00:00] [2008-09-01 16:00:00]
[10] [2008-10-01 16:00:00] [2008-11-01 16:00:00] [2008-12-01 16:00:00]
```

To find out and to print the time in Zurich we set the argument FinCenter="Zurich"

```
> ZRH <- timeDate(GMT, FinCenter = "Zurich")
> ZRH
Zurich
 [1] [2008-01-01 17:00:00] [2008-02-01 17:00:00] [2008-03-01 17:00:00]
 [4] [2008-04-01 18:00:00] [2008-05-01 18:00:00] [2008-06-01 18:00:00]
 [7] [2008-07-01 18:00:00] [2008-08-01 18:00:00] [2008-09-01 18:00:00]
[10] [2008-10-01 18:00:00] [2008-11-01 17:00:00] [2008-12-01 17:00:00]
```

and to express it in New York local time, we type

```
> NYC <- timeDate(GMT, FinCenter = "NewYork")
> NYC
NewYork
 [1] [2008-01-01 11:00:00] [2008-02-01 11:00:00] [2008-03-01 11:00:00]
 [4] [2008-04-01 12:00:00] [2008-05-01 12:00:00] [2008-06-01 12:00:00]
 [7] [2008-07-01 12:00:00] [2008-08-01 12:00:00] [2008-09-01 12:00:00]
[10] [2008-10-01 12:00:00] [2008-11-01 12:00:00] [2008-12-01 11:00:00]
```

```
> NYC <- timeDate(ZRH, zone = "Zurich", FinCenter = "NewYork")
> NYC
NewYork
 [1] [2008-01-01 11:00:00] [2008-02-01 11:00:00] [2008-03-01 11:00:00]
 [4] [2008-04-01 12:00:00] [2008-05-01 12:00:00] [2008-06-01 12:00:00]
 [7] [2008-07-01 12:00:00] [2008-08-01 12:00:00] [2008-09-01 12:00:00]
[10] [2008-10-01 12:00:00] [2008-11-01 12:00:00] [2008-12-01 11:00:00]
```

To summarize these examples, we state, that it is very easy in Rmetrics to express time in different time zones and financial center. We have just to know where the time stamp was recorded and in which financial center it will be used.

## 7.2 COERCION TO 'POSIXCT' OBJECTS

`timeDate` stamps can be coerced to `POSIXct` using the function `as.POSIXct()`

```
> as.POSIXct(GMT)
 [1] "2008-01-01 16:00:00 GMT" "2008-02-01 16:00:00 GMT"
 [3] "2008-03-01 16:00:00 GMT" "2008-04-01 16:00:00 GMT"
 [5] "2008-05-01 16:00:00 GMT" "2008-06-01 16:00:00 GMT"
 [7] "2008-07-01 16:00:00 GMT" "2008-08-01 16:00:00 GMT"
 [9] "2008-09-01 16:00:00 GMT" "2008-10-01 16:00:00 GMT"
[11] "2008-11-01 16:00:00 GMT" "2008-12-01 16:00:00 GMT"
```

```
> as.POSIXct(NYC)
 [1] "2008-01-01 16:00:00 GMT" "2008-02-01 16:00:00 GMT"
 [3] "2008-03-01 16:00:00 GMT" "2008-04-01 16:00:00 GMT"
 [5] "2008-05-01 16:00:00 GMT" "2008-06-01 16:00:00 GMT"
 [7] "2008-07-01 16:00:00 GMT" "2008-08-01 16:00:00 GMT"
 [9] "2008-09-01 16:00:00 GMT" "2008-10-01 16:00:00 GMT"
[11] "2008-11-01 16:00:00 GMT" "2008-12-01 16:00:00 GMT"
```

Both time stamp vectors agree as we expect.

## 7.3 COERCION TO 'POSIXLT' OBJECTS

The some holds for the coercion of `timeDate` objects to `POSIXlt` objects

```
> as.POSIXlt(GMT)[1:4]
[1] "2008-01-01 16:00:00 GMT" "2008-02-01 16:00:00 GMT"
[3] "2008-03-01 16:00:00 GMT" "2008-04-01 16:00:00 GMT"
> as.POSIXlt(NYC)[1:4]
[1] "2008-01-01 16:00:00 GMT" "2008-02-01 16:00:00 GMT"
[3] "2008-03-01 16:00:00 GMT" "2008-04-01 16:00:00 GMT"
```

## 7.4 COERCION TO 'DATE' OBJECTS

And again the some holds also for the coercion of `timeDate` objects to R's `Date` objects

```
> as.Date(GMT)[1:4]
[1] "2008-01-01" "2008-02-01" "2008-03-01" "2008-04-01"
> as.Date(NYC)[1:4]
[1] "2008-01-01" "2008-02-01" "2008-03-01" "2008-04-01"
```

## 7.5 COERCION TO A CHARACTER VECTOR

Character string can be obtained using the functions `as.character()` or `format()`

```
> as.character(GMT)[1:4]
[1] "2008-01-01 16:00:00" "2008-02-01 16:00:00" "2008-03-01 16:00:00"
[4] "2008-04-01 16:00:00"
> as.character(NYC)[1:4]
[1] "2008-01-01 11:00:00" "2008-02-01 11:00:00" "2008-03-01 11:00:00"
[4] "2008-04-01 12:00:00"
> format(GMT)[1:4]
[1] "2008-01-01 16:00:00" "2008-02-01 16:00:00" "2008-03-01 16:00:00"
[4] "2008-04-01 16:00:00"
> format(NYC)[1:4]
[1] "2008-01-01 11:00:00" "2008-02-01 11:00:00" "2008-03-01 11:00:00"
[4] "2008-04-01 12:00:00"
```

PART II

# CALENDARS

# CHAPTER 8

# CALENDAR FUNCTIONS AND METHODS

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

## 8.1 `align()` - ALIGNING 'TIMEDATE' OBJECTS

The function `align()` aligns a `timeDate` object to regular date and time stamps. To show the arguments of an S4 method we can use the Rmetrics function `getArgs()`

```
> getArgs("align", "timeDate")
function (x, by = "1d", offset = "0s")
NULL
align,timeDate:
NULL
```

The following example shows how to align biweekly time stamps for the calendar year 2008, starting on Monday, January 5

```
> tC <- timeCalendar(2009)
> tA <- align(tC, by = "2w", offset = "5d")
> tA
GMT
 [1] [2009-01-06] [2009-01-20] [2009-02-03] [2009-02-17] [2009-03-03]
 [6] [2009-03-17] [2009-03-31] [2009-04-14] [2009-04-28] [2009-05-12]
[11] [2009-05-26] [2009-06-09] [2009-06-23] [2009-07-07] [2009-07-21]
[16] [2009-08-04] [2009-08-18] [2009-09-01] [2009-09-15] [2009-09-29]
[21] [2009-10-13] [2009-10-27] [2009-11-10] [2009-11-24]
> dayOfWeek(tA[1])
2009-01-06
     "Tue"
```

The second example randomly generates 10 dates from the period January 11 to January 31, 2009, and completes them to a daily set of records

```
> tR <- timeDate(paste("2009-01-", round(runif(10, 11, 31)), sep = ""))
> tR

GMT
 [1] [2009-01-28] [2009-01-25] [2009-01-29] [2009-01-12] [2009-01-27]
 [6] [2009-01-28] [2009-01-24] [2009-01-20] [2009-01-12] [2009-01-25]

> tA <- align(sort(tR))
> tA

GMT
 [1] [2009-01-12] [2009-01-13] [2009-01-14] [2009-01-15] [2009-01-16]
 [6] [2009-01-17] [2009-01-18] [2009-01-19] [2009-01-20] [2009-01-21]
[11] [2009-01-22] [2009-01-23] [2009-01-24] [2009-01-25] [2009-01-26]
[16] [2009-01-27] [2009-01-28] [2009-01-29]
```

What happens when we have `timeDate` objects from two different financial centers, say from Zurich and New York, and we want to align them? Say, for example, we have end-of-day time stamps in Zurich and New York and we want to align these two time stamps hourly. In this case, we concatenate the time stamps using the function `c()` and we apply the `align()` function

```
> ZRH <- timeDate("2008-03-15 16:00", zone = "Zurich", FinCenter = "Zurich")
> ZRH

Zurich
[1] [2008-03-15 16:00:00]

> NYC <- timeDate("2008-03-15 16:00", zone = "New_York", FinCenter = "New_York")
> NYC

New_York
[1] [2008-03-15 16:00:00]

> tA <- align(c(ZRH, NYC), by = "1h")
> tA

Zurich
[1] [2008-03-15 16:00:00] [2008-03-15 17:00:00] [2008-03-15 18:00:00]
[4] [2008-03-15 19:00:00] [2008-03-15 20:00:00] [2008-03-15 21:00:00]
```

Since the function `c()` adapts its output to the first financial center of the concatenated vector of time stamps, the resulting time stamps are in Zurich local time. If you want them expressed in a different time zone, e.g. New York, they can easily be transformed

```
> timeDate(tA, FinCenter = "New_York")

New_York
[1] [2008-03-15 11:00:00] [2008-03-15 12:00:00] [2008-03-15 13:00:00]
[4] [2008-03-15 14:00:00] [2008-03-15 15:00:00] [2008-03-15 16:00:00]
```

## 8.2 atoms() - EXTRACTING CALENDAR ATOMS

The function `atoms()` extracts the calendar atoms from a `timeDate` object

```
> tsAtoms <- atoms(timeCalendar())
> class(tsAtoms)
[1] "data.frame"
> tsAtoms
      Y  m d H M S
1  2014  1 1 0 0 0
2  2014  2 1 0 0 0
3  2014  3 1 0 0 0
4  2014  4 1 0 0 0
5  2014  5 1 0 0 0
6  2014  6 1 0 0 0
7  2014  7 1 0 0 0
8  2014  8 1 0 0 0
9  2014  9 1 0 0 0
10 2014 10 1 0 0 0
11 2014 11 1 0 0 0
12 2014 12 1 0 0 0
```

and returns an object of class `data.frame`. The first column in the data frame are the years, the second the months, the third the days, and the last three columns are those for the hours, minutes, and seconds.
To print the second column with abbreviated month names, use the builtin constant `month.abb`, and just type

```
> tsAtoms[, 2] <- month.abb[tsAtoms[, 2]]
> tsAtoms
      Y   m d H M S
1  2014 Jan 1 0 0 0
2  2014 Feb 1 0 0 0
3  2014 Mar 1 0 0 0
4  2014 Apr 1 0 0 0
5  2014 May 1 0 0 0
6  2014 Jun 1 0 0 0
7  2014 Jul 1 0 0 0
8  2014 Aug 1 0 0 0
9  2014 Sep 1 0 0 0
10 2014 Oct 1 0 0 0
11 2014 Nov 1 0 0 0
12 2014 Dec 1 0 0 0
```

## 8.3 dayOfWeek() - RETURNING THE DAY OF THE WEEK

The function `dayOfWeek()` extracts the days of the week from a `timeDate` object

```
> dow <- dayOfWeek(timeCalendar())
> class(dow)
```

```
[1] "character"

> dow

2014-01-01 2014-02-01 2014-03-01 2014-04-01 2014-05-01 2014-06-01 2014-07-01
     "Wed"      "Sat"      "Sat"      "Tue"      "Thu"      "Sun"      "Tue"
2014-08-01 2014-09-01 2014-10-01 2014-11-01 2014-12-01
     "Fri"      "Mon"      "Wed"      "Sat"      "Mon"
```

and returns a character vector with the abbreviated month names. The vector is printed with names attributes given by the dates. To return the result in column mode, type

```
> matrix(dow, dimnames = list(names(dow), "DayOfWeek"))

           DayOfWeek
2014-01-01 "Wed"
2014-02-01 "Sat"
2014-03-01 "Sat"
2014-04-01 "Tue"
2014-05-01 "Thu"
2014-06-01 "Sun"
2014-07-01 "Tue"
2014-08-01 "Fri"
2014-09-01 "Mon"
2014-10-01 "Wed"
2014-11-01 "Sat"
2014-12-01 "Mon"
```

## 8.4 dayOfYear() - RETURNING THE DAY OF THE YEAR

The function dayOfYear() extracts the day number of the year of the dates from a timeDate object

```
> doy <- dayOfYear(timeCalendar())
> class(doy)

[1] "integer"

> doy

2014-01-01 2014-02-01 2014-03-01 2014-04-01 2014-05-01 2014-06-01 2014-07-01
         1         32         60         91        121        152        182
2014-08-01 2014-09-01 2014-10-01 2014-11-01 2014-12-01
       213        244        274        305        335
```

and returns an vector of integers.
To print the result as an integer index vector without name attributes, use the function as.vector()

```
> as.vector(doy)

 [1]   1  32  60  91 121 152 182 213 244 274 305 335
```

## 8.5 Easter() - COMPUTING THE DATE OF EASTER

The function `Easter()` returns the date of Easter for the specified years

```
> Easter(2000:2020)

GMT
 [1] [2000-04-23] [2001-04-15] [2002-03-31] [2003-04-20] [2004-04-11]
 [6] [2005-03-27] [2006-04-16] [2007-04-08] [2008-03-23] [2009-04-12]
[11] [2010-04-04] [2011-04-24] [2012-04-08] [2013-03-31] [2014-04-20]
[16] [2015-04-05] [2016-03-27] [2017-04-16] [2018-04-01] [2019-04-21]
[21] [2020-04-12]
```

## 8.6 endpoints() - CALCULATING ENDPOINT INDEXES

The function `endpoints()` returns a numeric vector corresponding to the last observation in each period specified by the argument `on`, with a zero added to the beginning of the vector, and the index of the last observation in the argument `x` at the end. This function behaves like the function `endpoints()` in the package `xts`.
The arguments `on` can take on the following values

```
> args(.endpoints)

function (x, on = c("months", "years", "quarters", "weeks", "days",
    "hours", "minutes", "seconds"), k = 1)
NULL
```

The function returns a numeric vector of endpoints beginning with 0 and ending with the a value equal to the length of the `x` argument.

```
> tS <- timeSequence(from = "2010-01-01", to = "2010-12-31")
> index <- .endpoints(tS)
> index

 [1]   0  31  59  90 120 151 181 212 243 273 304 334 365
```

The corresponding endpoint dates are

```
> tS[index]

GMT
 [1] [2010-01-31] [2010-02-28] [2010-03-31] [2010-04-30] [2010-05-31]
 [6] [2010-06-30] [2010-07-31] [2010-08-31] [2010-09-30] [2010-10-31]
[11] [2010-11-30] [2010-12-31]
```

## 8.7 isBizday() - TESTING FOR BUSINESS DAYS

The function `isBizday()` tests if a day is a business day or not, and the function `isHoliday()` tests its negation

*Check for business days*

```
> args(isBizday)

function (x, holidays = holidayNYSE(), wday = 1:5)
NULL
```

The argument x an object of class `timeDate` and the argument `holidays` is a vector of holiday dates from a holiday calendar, also an object of class `timeDate`. The function returns a logical vector indicating if a date is a business day, or a holiday.
As an example let us consider the calendar dates between March 20 and April 10, 2010, which includes Easter 2010

```
> year <- 2010
> Easter(2010)

GMT
[1] [2010-04-04]

> tS = timeSequence(from = paste(year, "-03-20", sep = ""), to = paste(year,
     "-04-10", sep = ""))
> tS

GMT
 [1] [2010-03-20] [2010-03-21] [2010-03-22] [2010-03-23] [2010-03-24]
 [6] [2010-03-25] [2010-03-26] [2010-03-27] [2010-03-28] [2010-03-29]
[11] [2010-03-30] [2010-03-31] [2010-04-01] [2010-04-02] [2010-04-03]
[16] [2010-04-04] [2010-04-05] [2010-04-06] [2010-04-07] [2010-04-08]
[21] [2010-04-09] [2010-04-10]
```

Then let us subset the business days based on the holiday calendar for the New York Stock Exchange

```
> Holidays <- holidayNYSE(2010)
> isBizday(tS, Holidays)

2010-03-20 2010-03-21 2010-03-22 2010-03-23 2010-03-24 2010-03-25 2010-03-26
     FALSE      FALSE       TRUE       TRUE       TRUE       TRUE       TRUE
2010-03-27 2010-03-28 2010-03-29 2010-03-30 2010-03-31 2010-04-01 2010-04-02
     FALSE      FALSE       TRUE       TRUE       TRUE       TRUE      FALSE
2010-04-03 2010-04-04 2010-04-05 2010-04-06 2010-04-07 2010-04-08 2010-04-09
     FALSE      FALSE       TRUE       TRUE       TRUE       TRUE       TRUE
2010-04-10
     FALSE

> tS[isBizday(tS, Holidays)]

GMT
 [1] [2010-03-22] [2010-03-23] [2010-03-24] [2010-03-25] [2010-03-26]
 [6] [2010-03-29] [2010-03-30] [2010-03-31] [2010-04-01] [2010-04-05]
[11] [2010-04-06] [2010-04-07] [2010-04-08] [2010-04-09]
```

*Check for holidays*

Alternatively, to return the holidays type

```
> isHoliday(tS, Holidays)

2010-03-20 2010-03-21 2010-03-22 2010-03-23 2010-03-24 2010-03-25 2010-03-26
      TRUE       TRUE      FALSE      FALSE      FALSE      FALSE      FALSE
2010-03-27 2010-03-28 2010-03-29 2010-03-30 2010-03-31 2010-04-01 2010-04-02
      TRUE       TRUE      FALSE      FALSE      FALSE      FALSE       TRUE
2010-04-03 2010-04-04 2010-04-05 2010-04-06 2010-04-07 2010-04-08 2010-04-09
      TRUE       TRUE      FALSE      FALSE      FALSE      FALSE      FALSE
2010-04-10
      TRUE

> tS[isHoliday(tS, Holidays)]

GMT
[1] [2010-03-20] [2010-03-21] [2010-03-27] [2010-03-28] [2010-04-02]
[6] [2010-04-03] [2010-04-04] [2010-04-10]

> dayOfWeek(tS[isHoliday(tS, Holidays)])

2010-03-20 2010-03-21 2010-03-27 2010-03-28 2010-04-02 2010-04-03 2010-04-04
     "Sat"      "Sun"      "Sat"      "Sun"      "Fri"      "Sat"      "Sun"
2010-04-10
     "Sat"
```

Note, Saturdays and Sundays are non business days and thus appear in the vector of holidays.

## 8.8 isRegular() - CHECKING IF A DATE VECTOR IS REGULAR

The function `isRegular()` checks if a date/time vector is regular, i.e. if it is a daily, a monthly, or a quarterly date/time vector. If the date/time vector is regular the frequency can determined calling the function `frequency()`. The rules are the following:

- A date/time vector is defined as daily if the vector has not more than one date/time stamp per day.
- A date/time vector is defined as monthly if the vector has not more than one date/time stamp per month.
- A date/time vector is defined as quarterly if the vector has not more than one date/time stamp per quarter.
- A monthly date/time vector is also a daily vector, a quarterly date/-time vector is also a monthly vector.
- A regular date/time vector is either a monthly or a quarterly vector.

Let us create a daily, a monthly and a quarterly `timeDate` vector

```
> tD <- timeSequence(from = "2010-01-01", length.out = 365)
> tM <- timeCalendar(2010)
> tQ <- tM[c(3, 6, 9, 12)]
```

The daily vector is irregular, the monthly and quarterly vectors are regular

```
> isRegular(tD)

[1] FALSE

> isRegular(tM)

[1] TRUE

> isRegular(tQ)

[1] TRUE
```

### *Daily 'timeDate' objects*

Check which of the vectors are daily vectors

```
> isDaily(tD)

[1] TRUE

> isDaily(tM)

[1] FALSE

> isDaily(tQ)

[1] FALSE
```

Note, that `tD` is no longer a daily series if date records are missing. Let us remove as an example record number 99.

```
> isDaily(tD[-99])

[1] FALSE
```

### *Monthly 'timeDate' objects*

Check which of the vectors are monthly vectors

```
> isMonthly(tD)

[1] FALSE

> isMonthly(tM)

[1] TRUE

> isMonthly(tQ)

[1] FALSE
```

*Quarterly 'timeDate' objects*

Check which of the vectors are quarterly vectors

```
> isQuarterly(tD)
[1] FALSE
> isQuarterly(tM)
[1] FALSE
> isQuarterly(tQ)
[1] TRUE
> isQuarterly(tM[c(2, 5, 8, 11)])
[1] TRUE
```

Note the vector `tM[c(2, 5, 8, 11)]` is also considered as a quarterly classtimeDate vector.

*Get the frequency of a regular 'timeDate' object*

The frequency of a daily `timeDate` vector is 1, of a monthly `timeDate` vector 12, and of of a quarterly `timeDate` vector 4

```
> frequency(tD)
[1] 1
> frequency(tM)
[1] 12
> frequency(tQ)
[1] 4
```

## 8.9 periods() - COMPOSING ROLLING PERIODS

The function `periods()` returns start and end dates for rolling periods

```
> args(periods)
function (x, period = "12m", by = "1m", offset = "0d")
NULL
```

The arguments `x` is an object of class `timeDate`, `period` is a span string, consisting of a length integer and a unit value, e.g. `"52w"` for 52 weeks, `by` is another span string, consisting of a length integer and a unit value, e.g. `"4w"` for 4 weeks, and `offset` is a span string, consisting of a length integer and a unit value, e.g. `"0d"` for no offset.
Let us create a daily time sequence

```
> x <- timeSequence(from = "2001-01-01", to = "2009-01-01", by = "day")
```

and then create rolling monthly calendar periods over 12 months

```
> periods(x, "12m", "1m")
$from
GMT
 [1] [2001-01-01] [2001-02-01] [2001-03-01] [2001-04-01] [2001-05-01]
 [6] [2001-06-01] [2001-07-01] [2001-08-01] [2001-09-01] [2001-10-01]
[11] [2001-11-01] [2001-12-01] [2002-01-01] [2002-02-01] [2002-03-01]
[16] [2002-04-01] [2002-05-01] [2002-06-01] [2002-07-01] [2002-08-01]
[21] [2002-09-01] [2002-10-01] [2002-11-01] [2002-12-01] [2003-01-01]
[26] [2003-02-01] [2003-03-01] [2003-04-01] [2003-05-01] [2003-06-01]
[31] [2003-07-01] [2003-08-01] [2003-09-01] [2003-10-01] [2003-11-01]
[36] [2003-12-01] [2004-01-01] [2004-02-01] [2004-03-01] [2004-04-01]
[41] [2004-05-01] [2004-06-01] [2004-07-01] [2004-08-01] [2004-09-01]
[46] [2004-10-01] [2004-11-01] [2004-12-01] [2005-01-01] [2005-02-01]
[51] [2005-03-01] [2005-04-01] [2005-05-01] [2005-06-01] [2005-07-01]
[56] [2005-08-01] [2005-09-01] [2005-10-01] [2005-11-01] [2005-12-01]
[61] [2006-01-01] [2006-02-01] [2006-03-01] [2006-04-01] [2006-05-01]
[66] [2006-06-01] [2006-07-01] [2006-08-01] [2006-09-01] [2006-10-01]
[71] [2006-11-01] [2006-12-01] [2007-01-01] [2007-02-01] [2007-03-01]
[76] [2007-04-01] [2007-05-01] [2007-06-01] [2007-07-01] [2007-08-01]
[81] [2007-09-01] [2007-10-01] [2007-11-01] [2007-12-01] [2008-01-01]
[86] [2008-02-01]

$to
GMT
 [1] [2001-12-31] [2002-01-31] [2002-02-28] [2002-03-31] [2002-04-30]
 [6] [2002-05-31] [2002-06-30] [2002-07-31] [2002-08-31] [2002-09-30]
[11] [2002-10-31] [2002-11-30] [2002-12-31] [2003-01-31] [2003-02-28]
[16] [2003-03-31] [2003-04-30] [2003-05-31] [2003-06-30] [2003-07-31]
[21] [2003-08-31] [2003-09-30] [2003-10-31] [2003-11-30] [2003-12-31]
[26] [2004-01-31] [2004-02-29] [2004-03-31] [2004-04-30] [2004-05-31]
[31] [2004-06-30] [2004-07-31] [2004-08-31] [2004-09-30] [2004-10-31]
[36] [2004-11-30] [2004-12-31] [2005-01-31] [2005-02-28] [2005-03-31]
[41] [2005-04-30] [2005-05-31] [2005-06-30] [2005-07-31] [2005-08-31]
[46] [2005-09-30] [2005-10-31] [2005-11-30] [2005-12-31] [2006-01-31]
[51] [2006-02-28] [2006-03-31] [2006-04-30] [2006-05-31] [2006-06-30]
[56] [2006-07-31] [2006-08-31] [2006-09-30] [2006-10-31] [2006-11-30]
[61] [2006-12-31] [2007-01-31] [2007-02-28] [2007-03-31] [2007-04-30]
[66] [2007-05-31] [2007-06-30] [2007-07-31] [2007-08-31] [2007-09-30]
[71] [2007-10-31] [2007-11-30] [2007-12-31] [2008-01-31] [2008-02-29]
[76] [2008-03-31] [2008-04-30] [2008-05-31] [2008-06-30] [2008-07-31]
[81] [2008-08-31] [2008-09-30] [2008-10-31] [2008-11-30] [2008-12-31]
[86] [2009-01-31]

attr(,"control")
GMT
       start          end
[2001-01-01] [2009-01-01]
```

Alternatively, we can create regular 4 weekly periods over 52 weeks

```
> periods(x, "52w", "4w")
$from
GMT
 [1] [2001-01-01] [2001-01-29] [2001-02-26] [2001-03-26] [2001-04-23]
 [6] [2001-05-21] [2001-06-18] [2001-07-16] [2001-08-13] [2001-09-10]
```

```
[11] [2001-10-08] [2001-11-05] [2001-12-03] [2001-12-31] [2002-01-28]
[16] [2002-02-25] [2002-03-25] [2002-04-22] [2002-05-20] [2002-06-17]
[21] [2002-07-15] [2002-08-12] [2002-09-09] [2002-10-07] [2002-11-04]
[26] [2002-12-02] [2002-12-30] [2003-01-27] [2003-02-24] [2003-03-24]
[31] [2003-04-21] [2003-05-19] [2003-06-16] [2003-07-14] [2003-08-11]
[36] [2003-09-08] [2003-10-06] [2003-11-03] [2003-12-01] [2003-12-29]
[41] [2004-01-26] [2004-02-23] [2004-03-22] [2004-04-19] [2004-05-17]
[46] [2004-06-14] [2004-07-12] [2004-08-09] [2004-09-06] [2004-10-04]
[51] [2004-11-01] [2004-11-29] [2004-12-27] [2005-01-24] [2005-02-21]
[56] [2005-03-21] [2005-04-18] [2005-05-16] [2005-06-13] [2005-07-11]
[61] [2005-08-08] [2005-09-05] [2005-10-03] [2005-10-31] [2005-11-28]
[66] [2005-12-26] [2006-01-23] [2006-02-20] [2006-03-20] [2006-04-17]
[71] [2006-05-15] [2006-06-12] [2006-07-10] [2006-08-07] [2006-09-04]
[76] [2006-10-02] [2006-10-30] [2006-11-27] [2006-12-25] [2007-01-22]
[81] [2007-02-19] [2007-03-19] [2007-04-16] [2007-05-14] [2007-06-11]
[86] [2007-07-09] [2007-08-06] [2007-09-03] [2007-10-01] [2007-10-29]
[91] [2007-11-26] [2007-12-24]

$to
GMT
 [1] [2001-12-31] [2002-01-28] [2002-02-25] [2002-03-25] [2002-04-22]
 [6] [2002-05-20] [2002-06-17] [2002-07-15] [2002-08-12] [2002-09-09]
[11] [2002-10-07] [2002-11-04] [2002-12-02] [2002-12-30] [2003-01-27]
[16] [2003-02-24] [2003-03-24] [2003-04-21] [2003-05-19] [2003-06-16]
[21] [2003-07-14] [2003-08-11] [2003-09-08] [2003-10-06] [2003-11-03]
[26] [2003-12-01] [2003-12-29] [2004-01-26] [2004-02-23] [2004-03-22]
[31] [2004-04-19] [2004-05-17] [2004-06-14] [2004-07-12] [2004-08-09]
[36] [2004-09-06] [2004-10-04] [2004-11-01] [2004-11-29] [2004-12-27]
[41] [2005-01-24] [2005-02-21] [2005-03-21] [2005-04-18] [2005-05-16]
[46] [2005-06-13] [2005-07-11] [2005-08-08] [2005-09-05] [2005-10-03]
[51] [2005-10-31] [2005-11-28] [2005-12-26] [2006-01-23] [2006-02-20]
[56] [2006-03-20] [2006-04-17] [2006-05-15] [2006-06-12] [2006-07-10]
[61] [2006-08-07] [2006-09-04] [2006-10-02] [2006-10-30] [2006-11-27]
[66] [2006-12-25] [2007-01-22] [2007-02-19] [2007-03-19] [2007-04-16]
[71] [2007-05-14] [2007-06-11] [2007-07-09] [2007-08-06] [2007-09-03]
[76] [2007-10-01] [2007-10-29] [2007-11-26] [2007-12-24] [2008-01-21]
[81] [2008-02-18] [2008-03-17] [2008-04-14] [2008-05-12] [2008-06-09]
[86] [2008-07-07] [2008-08-04] [2008-09-01] [2008-09-29] [2008-10-27]
[91] [2008-11-24] [2008-12-22]

attr(,"control")
GMT
       start          end
[2001-01-01] [2009-01-01]
```

The function `periods()` calls for monthly calendar periods the function `monthlyRolling()`,

```
> args(monthlyRolling)
function (x, period = "12m", by = "1m")
NULL
```

and for equidistand spaced periods the function `periodicallyRolling()`

```
> args(periodicallyRolling)
```

```
function (x, period = "52w", by = "4w", offset = "0d")
NULL
```

Here are two examples how to use these functions using the default settings of the arguments

```
> monthlyRolling(x)
$from
GMT
 [1] [2001-01-01] [2001-02-01] [2001-03-01] [2001-04-01] [2001-05-01]
 [6] [2001-06-01] [2001-07-01] [2001-08-01] [2001-09-01] [2001-10-01]
[11] [2001-11-01] [2001-12-01] [2002-01-01] [2002-02-01] [2002-03-01]
[16] [2002-04-01] [2002-05-01] [2002-06-01] [2002-07-01] [2002-08-01]
[21] [2002-09-01] [2002-10-01] [2002-11-01] [2002-12-01] [2003-01-01]
[26] [2003-02-01] [2003-03-01] [2003-04-01] [2003-05-01] [2003-06-01]
[31] [2003-07-01] [2003-08-01] [2003-09-01] [2003-10-01] [2003-11-01]
[36] [2003-12-01] [2004-01-01] [2004-02-01] [2004-03-01] [2004-04-01]
[41] [2004-05-01] [2004-06-01] [2004-07-01] [2004-08-01] [2004-09-01]
[46] [2004-10-01] [2004-11-01] [2004-12-01] [2005-01-01] [2005-02-01]
[51] [2005-03-01] [2005-04-01] [2005-05-01] [2005-06-01] [2005-07-01]
[56] [2005-08-01] [2005-09-01] [2005-10-01] [2005-11-01] [2005-12-01]
[61] [2006-01-01] [2006-02-01] [2006-03-01] [2006-04-01] [2006-05-01]
[66] [2006-06-01] [2006-07-01] [2006-08-01] [2006-09-01] [2006-10-01]
[71] [2006-11-01] [2006-12-01] [2007-01-01] [2007-02-01] [2007-03-01]
[76] [2007-04-01] [2007-05-01] [2007-06-01] [2007-07-01] [2007-08-01]
[81] [2007-09-01] [2007-10-01] [2007-11-01] [2007-12-01] [2008-01-01]
[86] [2008-02-01]

$to
GMT
 [1] [2001-12-31] [2002-01-31] [2002-02-28] [2002-03-31] [2002-04-30]
 [6] [2002-05-31] [2002-06-30] [2002-07-31] [2002-08-31] [2002-09-30]
[11] [2002-10-31] [2002-11-30] [2002-12-31] [2003-01-31] [2003-02-28]
[16] [2003-03-31] [2003-04-30] [2003-05-31] [2003-06-30] [2003-07-31]
[21] [2003-08-31] [2003-09-30] [2003-10-31] [2003-11-30] [2003-12-31]
[26] [2004-01-31] [2004-02-29] [2004-03-31] [2004-04-30] [2004-05-31]
[31] [2004-06-30] [2004-07-31] [2004-08-31] [2004-09-30] [2004-10-31]
[36] [2004-11-30] [2004-12-31] [2005-01-31] [2005-02-28] [2005-03-31]
[41] [2005-04-30] [2005-05-31] [2005-06-30] [2005-07-31] [2005-08-31]
[46] [2005-09-30] [2005-10-31] [2005-11-30] [2005-12-31] [2006-01-31]
[51] [2006-02-28] [2006-03-31] [2006-04-30] [2006-05-31] [2006-06-30]
[56] [2006-07-31] [2006-08-31] [2006-09-30] [2006-10-31] [2006-11-30]
[61] [2006-12-31] [2007-01-31] [2007-02-28] [2007-03-31] [2007-04-30]
[66] [2007-05-31] [2007-06-30] [2007-07-31] [2007-08-31] [2007-09-30]
[71] [2007-10-31] [2007-11-30] [2007-12-31] [2008-01-31] [2008-02-29]
[76] [2008-03-31] [2008-04-30] [2008-05-31] [2008-06-30] [2008-07-31]
[81] [2008-08-31] [2008-09-30] [2008-10-31] [2008-11-30] [2008-12-31]
[86] [2009-01-31]

attr(,"control")
GMT
       start          end
[2001-01-01] [2009-01-01]
```

and

```
> periodicallyRolling(x)

$from
GMT
 [1] [2001-01-01] [2001-01-29] [2001-02-26] [2001-03-26] [2001-04-23]
 [6] [2001-05-21] [2001-06-18] [2001-07-16] [2001-08-13] [2001-09-10]
[11] [2001-10-08] [2001-11-05] [2001-12-03] [2001-12-31] [2002-01-28]
[16] [2002-02-25] [2002-03-25] [2002-04-22] [2002-05-20] [2002-06-17]
[21] [2002-07-15] [2002-08-12] [2002-09-09] [2002-10-07] [2002-11-04]
[26] [2002-12-02] [2002-12-30] [2003-01-27] [2003-02-24] [2003-03-24]
[31] [2003-04-21] [2003-05-19] [2003-06-16] [2003-07-14] [2003-08-11]
[36] [2003-09-08] [2003-10-06] [2003-11-03] [2003-12-01] [2003-12-29]
[41] [2004-01-26] [2004-02-23] [2004-03-22] [2004-04-19] [2004-05-17]
[46] [2004-06-14] [2004-07-12] [2004-08-09] [2004-09-06] [2004-10-04]
[51] [2004-11-01] [2004-11-29] [2004-12-27] [2005-01-24] [2005-02-21]
[56] [2005-03-21] [2005-04-18] [2005-05-16] [2005-06-13] [2005-07-11]
[61] [2005-08-08] [2005-09-05] [2005-10-03] [2005-10-31] [2005-11-28]
[66] [2005-12-26] [2006-01-23] [2006-02-20] [2006-03-20] [2006-04-17]
[71] [2006-05-15] [2006-06-12] [2006-07-10] [2006-08-07] [2006-09-04]
[76] [2006-10-02] [2006-10-30] [2006-11-27] [2006-12-25] [2007-01-22]
[81] [2007-02-19] [2007-03-19] [2007-04-16] [2007-05-14] [2007-06-11]
[86] [2007-07-09] [2007-08-06] [2007-09-03] [2007-10-01] [2007-10-29]
[91] [2007-11-26] [2007-12-24]

$to
GMT
 [1] [2001-12-31] [2002-01-28] [2002-02-25] [2002-03-25] [2002-04-22]
 [6] [2002-05-20] [2002-06-17] [2002-07-15] [2002-08-12] [2002-09-09]
[11] [2002-10-07] [2002-11-04] [2002-12-02] [2002-12-30] [2003-01-27]
[16] [2003-02-24] [2003-03-24] [2003-04-21] [2003-05-19] [2003-06-16]
[21] [2003-07-14] [2003-08-11] [2003-09-08] [2003-10-06] [2003-11-03]
[26] [2003-12-01] [2003-12-29] [2004-01-26] [2004-02-23] [2004-03-22]
[31] [2004-04-19] [2004-05-17] [2004-06-14] [2004-07-12] [2004-08-09]
[36] [2004-09-06] [2004-10-04] [2004-11-01] [2004-11-29] [2004-12-27]
[41] [2005-01-24] [2005-02-21] [2005-03-21] [2005-04-18] [2005-05-16]
[46] [2005-06-13] [2005-07-11] [2005-08-08] [2005-09-05] [2005-10-03]
[51] [2005-10-31] [2005-11-28] [2005-12-26] [2006-01-23] [2006-02-20]
[56] [2006-03-20] [2006-04-17] [2006-05-15] [2006-06-12] [2006-07-10]
[61] [2006-08-07] [2006-09-04] [2006-10-02] [2006-10-30] [2006-11-27]
[66] [2006-12-25] [2007-01-22] [2007-02-19] [2007-03-19] [2007-04-16]
[71] [2007-05-14] [2007-06-11] [2007-07-09] [2007-08-06] [2007-09-03]
[76] [2007-10-01] [2007-10-29] [2007-11-26] [2007-12-24] [2008-01-21]
[81] [2008-02-18] [2008-03-17] [2008-04-14] [2008-05-12] [2008-06-09]
[86] [2008-07-07] [2008-08-04] [2008-09-01] [2008-09-29] [2008-10-27]
[91] [2008-11-24] [2008-12-22]

attr(,"control")
GMT
       start          end
[2001-01-01] [2009-01-01]
```

## 8.10 `timeLastDayInMonth()` - COMPUTING FIRST AND LAST DAYS

The dates describing the first or last day for a given month or quarter are also often needed. Such dates can be generated by the functions `timeLastDayInMonth()`, `timeFirstDayInMonth()`, `timeLastDayInQuarter()`, and `timeFirstDayInQuarter()`.

### *Last day in a given month*

The function `timeLastDayInMonth()` computes the last day in a given month and year

```
> args(timeLastDayInMonth)

function (charvec, format = "%Y-%m-%d", zone = "", FinCenter = "")
NULL
```

For example this function can be used to create a end-of-month time stamps, first creating a monthly calendar series and then shifting the dates to the end of each month

```
> tC <- timeCalendar()
> tC

GMT
 [1] [2014-01-01] [2014-02-01] [2014-03-01] [2014-04-01] [2014-05-01]
 [6] [2014-06-01] [2014-07-01] [2014-08-01] [2014-09-01] [2014-10-01]
[11] [2014-11-01] [2014-12-01]

> timeLastDayInMonth(tC)

GMT
 [1] [2014-01-31] [2014-02-28] [2014-03-31] [2014-04-30] [2014-05-31]
 [6] [2014-06-30] [2014-07-31] [2014-08-31] [2014-09-30] [2014-10-31]
[11] [2014-11-30] [2014-12-31]
```

### *First day in a given month*

The function `timeFirstDayInMonth()` computes the first day in a given month and year

```
> args(timeFirstDayInMonth)

function (charvec, format = "%Y-%m-%d", zone = "", FinCenter = "")
NULL
```

This function can be used to shift time stamps to the beginning of each month. for example, first generate a random day vector and then shift back:

```
> tR <- timeCalendar(d = round(runif(12, 1, 28)))
> tR
```

```
GMT
 [1] [2014-01-09] [2014-02-14] [2014-03-23] [2014-04-07] [2014-05-08]
 [6] [2014-06-21] [2014-07-11] [2014-08-25] [2014-09-21] [2014-10-21]
[11] [2014-11-06] [2014-12-12]
```

```
> timeFirstDayInMonth(tR)
```

```
GMT
 [1] [2014-01-01] [2014-02-01] [2014-03-01] [2014-04-01] [2014-05-01]
 [6] [2014-06-01] [2014-07-01] [2014-08-01] [2014-09-01] [2014-10-01]
[11] [2014-11-01] [2014-12-01]
```

### *Last day in a given quarter*

The next two functions do the same job on a quarterly base. The function `timeLastDayInQuarter()` computes the last day in a given quarter and year

```
> args(timeLastDayInQuarter)
```

```
function (charvec, format = "%Y-%m-%d", zone = "", FinCenter = "")
NULL
```

### *First day in a given quarter*

The function `timeFirstDayInQuarter()` computes the first day in a given quarter and year

```
> args(timeFirstDayInQuarter)
```

```
function (charvec, format = "%Y-%m-%d", zone = "", FinCenter = "")
NULL
```

### *Simple function wrappers*

For those who are these function names too long can create and use the wrappers

```
> firstMonthly <- timeFirstDayInMonth
> lastMonthly <- timeLastDayInMonth
> firstQuarterly <- timeFirstDayInQuarter
> lastQuarterly <- timeLastDayInQuarter
```

## 8.11 `timeNdayOnOrAfter()` - COMPUTING ON-OR-AFTER DATES

*On-Or-After* and *On-Or-Before* are further important building blocks to work with time and date objects.

*Date in month that is a n-day on or after a given date*

The function `timeNdayOnOrAfter()` computes the date in month that is a n-day on or after a given date

```
> args(timeNdayOnOrAfter)

function (charvec, nday = 1, format = "%Y-%m-%d", zone = "",
    FinCenter = "")
NULL
```

e.g. what date has the first Monday on or after March 15, 2008 ?

```
> timeNdayOnOrAfter("2008-03-15", 1)

GMT
[1] [2008-03-17]
```

*Date in month that is a n-day on or before a given date*

Similarly, the function `timeNdayOnOrBefore()` computes the date in month that is a n-day on or before a given date

```
> args(timeNdayOnOrBefore)

function (charvec, nday = 1, format = "%Y-%m-%d", zone = "",
    FinCenter = "")
NULL
```

e.g. what date has Friday on or before April 22, 2008 ?

```
> timeNdayOnOrBefore("2002-04-22", 5)

GMT
[1] [2002-04-19]
```

*simple function wrappers*

For those who are these function names too long, you can define your own functions

```
> after <- timeNdayOnOrAfter
> before <- timeNdayOnOrBefore
```

and try

```
> c(after("2008-03-15", 1), before("2002-04-22", 5))

GMT
[1] [2008-03-17] [2002-04-19]
```

## 8.12 timeNthNdayInMonth() - COMPUTING THE N-TH N-DAY

*The n-th occurrence of a n-day*

The function timeNthNdayInMonth() computes the n-th occurrence of a n-day in a given year/month. The arguments are as follows

```
> args(timeNthNdayInMonth)
function (charvec, nday = 1, nth = 1, format = "%Y-%m-%d", zone = "",
    FinCenter = "")
NULL
```

here, charvec is the character vector of dates and times, nday an integer vector with entries ranging from 0 (Sunday) to 6 (Saturday), nth an integer vector numbering the n-th occurrence, format the format specification of the input character vector, zone the time zone or financial center where the data were recorded, FinCenter a character with the the location of the financial center named as "continent/city".
For example, let us answer the question *What date was the second Monday in April 2008?*.

```
> timeNthNdayInMonth("2008-04-01", 1, 2)
GMT
[1] [2008-04-14]
```

*The last n-day*

The function timeLastNdayInMonth() computes the last n-day in a given year/month. The arguments are as follows,

```
> args(timeLastNdayInMonth)
function (charvec, nday = 1, format = "%Y-%m-%d", zone = "",
    FinCenter = "")
NULL
```

the meaning of the arguments is the same as listed above.
As a furher example, let us answer the question, *What date was the last Tuesday in May, 2008?*

```
> timeLastNdayInMonth("2008-05-01", 2)
GMT
[1] [2008-06-03]
```

# Chapter 9

# Ecclesiastical and Public Holidays

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

## 9.1 Ecclesiastical Holidays

It is non-trivial to implement function for business days, weekends and holidays. It is not difficult in an algorithmic sense, but it can become tedious to implement the rules of the calendar themselves, for example the date of Easter.
In this chapter we briefly summarize the functions that can calculate dates of ecclesiastical and public holidays. With the help of these functions we can also create business and holiday calendars.
Holidays may have two origins, ecclesiastical and public/federal. The *ecclesiastical calendars* of Christian churches are based on cycles of moveable and immovable feasts. *Christmas*, December 25, is the principal immovable feast. *Easter* is the principal moveable feast, and dates of most other moveable feasts are determined with respect to Easter. However, the moveable feasts of the Advent and Epiphany seasons are Sundays recognized from Christmas and the Feast of the Epiphany, respectively.

### *The Date of Easter*

The date of *Easter* is evaluated by a complex procedure whose detailed explanation is outside the scope of this book. The reason for the calculation being so complicated is that the date of Easter is linked to (an inaccurate version of) the Hebrew calendar. But nevertheless, a short answer to the

question "When is Easter?" is the following: *Easter Sunday is the first Sunday after the first full moon after vernal equinox.* For the long answer we refer to Toendering (1998).
The algorithm we use in Rmetrics computes the date of Easter based on the algorithm of Oudin [1940]. It is valid for any Gregorian Calendar year.

```
# Calculating the ISO-8601 Date for Easter:
C <- year%/%100
N <- year - 19*(year%/%19)
K <- (C-17)%/%25
I <- C - C%/%4 - (C-K)%/%3 + 19*N + 15
I <- I - 30*(I%/%30)
I <- I - (I%/%28)*(1-(I%/%28)*(29%/%(I+1))*((21-N)/11))
J <- year + year%/%4 + I + 2 - C + C%/%4
J <- J - 7*(J%/%7)
L <- I - J
month <- 3 + (L+40)%/%44
day <- L + 28 - 31*(month%/%4)
EASTER <- as.character(year*10000 + month*100 + day)
```

All variables are integers and the remainders of all divisions are dropped. The final date is given by the ISO-8601 date formatted variable EASTER. The Rmetrics function `easter()` implements this algorithm and computes the date(s) of Easter for an integer value or vector:

```
> Easter(getRmetricsOptions("currentYear"))
GMT
[1] [2014-04-20]
> Easter(2004:2011)
GMT
[1] [2004-04-11] [2005-03-27] [2006-04-16] [2007-04-08] [2008-03-23]
[6] [2009-04-12] [2010-04-04] [2011-04-24]
```

### *Dates of Easter Related Feasts*

Many feasts are related to Easter and thus it becomes very simple to compute their dates when the date of Easter is known:

LISTING 9.1: DATES OF EASTER RELATED FEASTS.

```
Holiday:
Ash Wednesday                                   46 days before Easter
Palm Sunday                                      7 days before Easter
Good Friday                                      2 days before Easter
Rogation Sunday                                 35 days after  Easter
Ascension                                       39 days after  Easter
Pentecost                                       49 days after  Easter
Trinity Sunday                                  56 days after  Easter
Corpus Christi                                  60 days after  Easter
Easter Monday                                    1 day after   Easter
```

```
PentecostMonday                                 50 dayw after  Easter
```

The function `Easter` also allows us to compute the dates of feasts related to Easter. The second argument of the function named `shift` allows us to compute the date(s) shifted by the given number of days.

```
> Pentecost <- Easter(2005, 49)
> Pentecost
GMT
[1] [2005-05-15]
```

### *Dates of Sundays in Advent*

Sundays in Advent are determined in the following straightforward method:

LISTING 9.2: DATES OF SUNDAYS IN ADVENT.

```
Holiday:
First Sunday of Advent             the Sunday on or after 27 November
Second Sunday of Advent            the Sunday on or after  4 December
3rd Sunday of Advent               the Sunday on or after 11 December
4th Sunday of Advent               the Sunday on or after 18 December
```

### *Ecclesiastical Feasts with Fix Dates*

Further feasts that are listed by the ecclesiastical calendar are:

LISTING 9.3: ECCLESIASTICAL FEASTS WITH FIX DATES.

```
Holiday:
Epiphany                                                  on 6 January
Presentation of the Lord                                 on 2 February
Annunciation usually                                       on 25 March
Transfiguration of the Lord                                on 6 August
Assumption of Mary                                        on 15 August
Birth of Virgin Mary                                    on 8 September
Celebration of the Holy Cross                          on 14 September
Mass of the Archangels                                 on 29 September
All Saints'                                              on 1 November
All Souls'                                               on 2 November
Boxing Day                                              on 25 December
```

## 9.2 PUBLIC AND FEDERAL HOLIDAYS

*Public and federal holidays* include some of the ecclesiastical holidays, e.g. Easter and Christmas, and usually national holidays, e.g. Labour Day,

Independance Day. It is also difficult to specify a holiday calendar for a country, since almost in every country rules on local holidays in cities and states exist. Therefore, we concentrate on holidays celebrated in the major financial market centers in Switzerland and the G7 countries; these include: In Europe Zurich, London, Frankfurt, Paris, Milano, in North America New York, Chicago, Toronto, Montreal, and in the Far East Tokyo and Osaka.
The following table gives a summary in which countries New Year's Day, Good Friday, Easter, Easter Monday, Labor Day on May 1, Pentecost, Pentecost Monday, Christmas Day and Boxing Day are celebrated as public or federal holidays:

LISTING 9.4: FEDERAL HOLIDAYS.

```
Feasts:                          Date    CH/DE  GB  FR  IT  US/CA  JP
New Year's Day                  1 Jan      X     X   X   X    X     X
Good Friday                                X     X
Easter Sunday                              X     X   X   X
Easter Monday                              X     X   X   X
Labor Day                       1 May      X         X   X
Pentecost Sunday                           X         X
Pentecost Monday                           X         X
Christmas Day                  25 Dec      X     X   X   X    X
Boxing Day                     26 Dec      X     X       X    X
```

The next tables give city/country specific information on additional feasts. Rules are also provided for when a public or federal holiday falls on a Saturday or Sunday.

## *Dates of Zurich Holidays*

For Zurich, Switzerland, we have the following public holidays:

LISTING 9.5: DATES OF ZURICH HOLIDAYS.

```
Additional Feasts:                                               Date
Berchtold's Day                                                 2 Jan
Sechselaeuten                                   3rd Monday in April *
Ascension                                        39 days after Easter
Confederation Day                                               1 Aug
Knabenschiessen                         2nd Saturday to Monday in Sep
* 1 week later if it coincides with Easter Monday
```

## *Dates of London Holidays*

For London we have the following public holidays:

LISTING 9.6: DATES OF LONDON HOLIDAYS.

```
Additional Feasts:                                              Date
May Day Bank Holiday                                 1st Monday in May
Bank Holiday                                        Last Monday in May
Summer Bank Holiday                              Last Monday in August
New Year's Eve, 31 December 1999 will be a public holiday.
  Holidays falling on a weekend are celebrated on the Monday
  following.
```

### *Dates of New York and Chicago Holidays*

In New York and Chicago we have the following public holidays:

LISTING 9.7: DATES OF NEW YORK AND CHICAGO HOLIDAYS.

```
Additional Feasts:                                              Date
New Year's Day                                                 1 Jan
Inauguration Day *                                            20 Jan
Martin Luther King Jr Day                        3rd Monday in January
Lincoln's Birthday                                            12 Feb
Washington's Birthday                           3rd Monday in February
Memorial Day                                        Last Monday in May
Independence Day                                              4 July
Labor Day                                      1st Monday in September
Columbus Day                                     2nd Monday in October
Election Day                             Tuesday on or after 2 November
Veterans' Day                                              11 November
Thanksgiving                                  4th Thursday in November
Christmas Day                                              25 December
Holidays on Saturdays are observed on the preceding
  Friday, those on Sundays on the Monday following.
```

LISTING 9.8: DATES OF ADDITIONAL CHICAGO HOLIDAYS.

```
Additional Feasts in Chicago/IL:                                Date
Casimir Pulaski's Birthday                         1st Monday in March
Good Friday                                       2 days before Easter
```

### *Dates of Frankfurt Holidays*

For Frankfurt, Germany, we have the following public holidays:

LISTING 9.9: DATES OF ADDITIONAL FRANKFURT HOLIDAYS.

```
Additional Feasts:                                              Date
Ascension                                         39 days after Easter
```

```
Corpus Christi                                60 days after Easter
Day of German Unity                                          3 Oct
Christmas Eve ∗                                             22 Dec
New Year's Eve ∗                                            31 Dec
∗ Government closed, half day for shops.
```

*Dates of Paris Holidays*

For Paris, France, we have the following public holidays:

LISTING 9.10: DATES OF ADDITIONAL PARIS HOLIDAYS.

```
Additional Feasts:                                            Date
Fete de la Victoire 1945                                     8 May
Ascension                                     39 days after Easter
Bastille Day                                                14 Jul
Assumption Virgin Mary                                      15 Aug
All Saints Day                                               1 Nov
Armistice Day                                               11 Nov
```

*Dates of Milan Holidays*

For Milan, Italy, we have the following public holidays:

LISTING 9.11: DATES OF ADDITIONAL MILANO HOLIDAYS.

```
Additional Feasts:                                            Date
Epiphany                                                     6 Jan
Liberation Day                                              25 Apr
Anniversary of the Republic Sunday nearest                   2 Jun
Assumption of Virgin Mary                                   15 Aug
All Saints Day                                               1 Nov
WWI Victory Anniversary    ∗ Sunday nearest                  4 Nov
St Amrose (Milan local)                                      7 Dec
Immaculate Conception                                        8 Dec
∗ Sunday is a holiday anyway, but holiday pay
  rules apply.
```

*Dates of Toronto and Montreal Holidays*

In Toronto and Montreal we have the following public holidays:

LISTING 9.12: DATES OF ADDITIONAL TORONTO AND MONTREAL HOLIDAYS.

```
Additional Feasts:                                            Date
```

```
Victoria Day Monday on or preceding                        24 May
Canada Day *                                                1 Jul
Civic or Provincial Holiday                      1st Monday in Aug
Labor Day                                        1st Monday in Sep
Thanksgiving Day                                 2nd Monday in Oct
Remembrance Day (Govt offices and banks only)              11 Nov
* When these days fall on a Sunday, the next
  working day is considered a holiday.
```

### *Dates of Tokyo and Osaka Holidays*

In Tokyo and Osaka we have the following public holidays.

LISTING 9.13: DATES OF TOKYO AND OSAKA HOLIDAYS.

```
Feasts:                                                      Date
New Year's Day (Gantan)                                     1 Jan
Bank Holiday                                                2 Jan
Bank Holiday                                                3 Jan
Coming of Age Day (Seijin-no-hi)                           15 Jan
Nat. Foundation Day (Kenkoku-kinen-no-hi)                  11 Feb
Vernal Equinox (Shunbun-no-hi)                                  *
Greenery Day (Midori-no-hi)                                29 Apr
Constitution Memorial Day (Kenpou-kinen-bi)                 3 May
Holiday for a Nation (Kokumin-no-kyujitu)                 4 May**
Children's Day (Kodomo-no-hi)                               5 May
Marine Day (Umi-no-hi)                                     20 Jul
Respect for the Aged Day (Keirou-no-hi)                    15 Sep
Autumnal Equinox (Shuubun-no-hi)                      23/24 Sep***
Health and Sports Day (Taiiku-no-hi)                       10 Oct
National Culture Day (Bunka-no-hi)                          3 Nov
Thanksgiving Day (Kinrou-kansha-no-hi)                     23 Nov
Emperor's Birthday (Tennou-tanjyou-bi)                     23 Nov
Bank Holiday                                               31 Dec
* 21 March in 1999, 20 March in 2000. Observed on a
  Monday if it falls on a Sunday. There are no moveable
  feasts other than the Equinoxes which obviously depend
  on the lunar ephemeris.
** If it falls between Monday and Friday.
*** 23 September in both 1999 and 2000. Holidays falling
  on a Sunday are observed on the Monday following except
  for the Bank Holidays associated with the New Year.
```

## 9.3 LISTING HOLIDAYS

### *The Complete Holiday List*

The complete list of holidays implemented in Rmetrics can be obtained calling the function `listHolidays()`

```
> listHolidays()
  [1] "Advent1st"               "Advent2nd"
  [3] "Advent3rd"               "Advent4th"
  [5] "AllSaints"               "AllSouls"
  [7] "Annunciation"            "Ascension"
  [9] "AshWednesday"            "AssumptionOfMary"
 [11] "BirthOfVirginMary"       "BoxingDay"
 [13] "CACanadaDay"             "CACivicProvincialHoliday"
 [15] "CALabourDay"             "CaRemembranceDay"
 [17] "CAThanksgivingDay"       "CAVictoriaDay"
 [19] "CelebrationOfHolyCross"  "CHAscension"
 [21] "CHBerchtoldsDay"         "CHConfederationDay"
 [23] "CHKnabenschiessen"       "ChristmasDay"
 [25] "ChristmasEve"            "ChristTheKing"
 [27] "CHSechselaeuten"         "CorpusChristi"
 [29] "DEAscension"             "DEChristmasEve"
 [31] "DECorpusChristi"         "DEGermanUnity"
 [33] "DENewYearsEve"           "Easter"
 [35] "EasterMonday"            "EasterSunday"
 [37] "Epiphany"                "FRAllSaints"
 [39] "FRArmisticeDay"          "FRAscension"
 [41] "FRAssumptionVirginMary"  "FRBastilleDay"
 [43] "FRFetDeLaVictoire1945"   "GBBankHoliday"
 [45] "GBMayDay"                "GBMilleniumDay"
 [47] "GBSummerBankHoliday"     "GoodFriday"
 [49] "ITAllSaints"             "ITAssumptionOfVirginMary"
 [51] "ITEpiphany"              "ITImmaculateConception"
 [53] "ITLiberationDay"         "ITStAmrose"
 [55] "JPAutumnalEquinox"       "JPBankHolidayDec31"
 [57] "JPBankHolidayJan2"       "JPBankHolidayJan3"
 [59] "JPBunkaNoHi"             "JPChildrensDay"
 [61] "JPComingOfAgeDay"        "JPConstitutionDay"
 [63] "JPEmperorsBirthday"      "JPGantan"
 [65] "JPGreeneryDay"           "JPHealthandSportsDay"
 [67] "JPKeirouNOhi"            "JPKenkokuKinenNoHi"
 [69] "JPKenpouKinenBi"         "JPKinrouKanshaNoHi"
 [71] "JPKodomoNoHi"            "JPKokuminNoKyujitu"
 [73] "JPMarineDay"             "JPMidoriNoHi"
 [75] "JPNatFoundationDay"      "JPNationalCultureDay"
 [77] "JPNationHoliday"         "JPNewYearsDay"
 [79] "JPRespectForTheAgedDay"  "JPSeijinNoHi"
 [81] "JPShuubunNoHi"           "JPTaiikuNoHi"
 [83] "JPTennouTanjyouBi"       "JPThanksgivingDay"
 [85] "JPUmiNoHi"               "LaborDay"
 [87] "MassOfArchangels"        "NewYearsDay"
 [89] "PalmSunday"              "Pentecost"
 [91] "PentecostMonday"         "PresentationOfLord"
 [93] "Quinquagesima"           "RogationSunday"
 [95] "Septuagesima"            "SolemnityOfMary"
 [97] "TransfigurationOfLord"   "TrinitySunday"
 [99] "USChristmasDay"          "USColumbusDay"
[101] "USCPulaskisBirthday"     "USDecorationMemorialDay"
[103] "USElectionDay"           "USGoodFriday"
[105] "USInaugurationDay"       "USIndependenceDay"
[107] "USLaborDay"              "USLincolnsBirthday"
```

```
[109] "USMemorialDay"            "USMLKingsBirthday"
[111] "USNewYearsDay"            "USPresidentsDay"
[113] "USThanksgivingDay"        "USVeteransDay"
[115] "USWashingtonsBirthday"
```

# CHAPTER 10

# HOLIDAY CALENDARS

```
> library(timeDate)
```

## 10.1 PUBLIC HOLIDAY CALENDARS

The Zurich holiday calendar is encoded in the function `holidayZURICH()`.

To get the calendar for the current year, type

```
> holidayZURICH()
Zurich
 [1] [2014-01-01] [2014-01-02] [2014-04-18] [2014-04-21] [2014-04-21]
 [6] [2014-05-01] [2014-05-29] [2014-06-09] [2014-08-01] [2014-09-08]
[11] [2014-12-25] [2014-12-26]
```

To print the calendar for the next year, use

```
> currentYear <- getRmetricsOptions("currentYear")
> nextYear <- currentYear + 1
> holidayZURICH(currentYear:nextYear)
Zurich
 [1] [2014-01-01] [2014-01-02] [2014-04-18] [2014-04-21] [2014-04-21]
 [6] [2014-05-01] [2014-05-29] [2014-06-09] [2014-08-01] [2014-09-08]
[11] [2014-12-25] [2014-12-26] [2015-01-01] [2015-01-02] [2015-04-03]
[16] [2015-04-06] [2015-04-20] [2015-05-01] [2015-05-14] [2015-05-25]
[21] [2015-09-14] [2015-12-25]
```

To find out which holidays are included in the calendar, have a look in the function

```
> holidayZURICH
```

```
function (year = getRmetricsOptions("currentYear"))
{
    holidays <- c(NewYearsDay(year), GoodFriday(year), EasterMonday(year),
        LaborDay(year), PentecostMonday(year), ChristmasDay(year),
        BoxingDay(year), CHBerchtoldsDay(year), CHSechselaeuten(year),
        CHAscension(year), CHConfederationDay(year), CHKnabenschiessen(year))
    holidays <- sort(holidays)
    holidays <- holidays[isWeekday(holidays)]
    holidays <- timeDate(format(holidays), zone = "Zurich", FinCenter = "Zurich")
    holidays
}
<environment: namespace:timeDate>
```

To see the encoding for the Swiss bank holiday Sechseläuten print the function

```
> CHSechselaeuten
```

```
function (year = getRmetricsOptions("currentYear"))
{
    ans = NULL
    for (y in year) {
        theDate = .nth.of.nday(y, 4, 1, 3)
        if (as.character(theDate) == as.character(Easter(y, +1))) {
            theDate = .nth.of.nday(y, 4, 1, 4)
        }
        ans = c(ans, theDate)
    }
    timeDate(as.character(ans))
}
<environment: namespace:timeDate>
```

## 10.2 EXCHANGE CALENDARS

In Rmetrics one can easily build holiday calenders. Currently, the following calendars are implemented: the NYSE stock exchange holiday `holidayNYSE`, and the TSX holiday calendar `holidayTSX`.
To get the holiday calendar for the current year type, proceed like in the case of the holiday calendar for Zurich

```
> holidayNYSE()
```

```
NewYork
[1] [2014-01-01] [2014-01-20] [2014-02-17] [2014-04-18] [2014-05-26]
[6] [2014-07-04] [2014-09-01] [2014-11-27] [2014-12-25]
```

and to view holidays only for the years 2008 to 2010 type:

```
> holidayNYSE(2008:2010)
```

```
NewYork
 [1] [2008-01-01] [2008-01-21] [2008-02-18] [2008-03-21] [2008-05-26]
 [6] [2008-07-04] [2008-09-01] [2008-11-27] [2008-12-25] [2009-01-01]
[11] [2009-01-19] [2009-02-16] [2009-04-10] [2009-05-25] [2009-07-03]
```

```
[16] [2009-09-07] [2009-11-26] [2009-12-25] [2010-01-01] [2010-01-18]
[21] [2010-02-15] [2010-04-02] [2010-05-31] [2010-07-05] [2010-09-06]
[26] [2010-11-25] [2010-12-24]
```

Notice that the construction of the NYSE exchange trading calendar is quite complex and has very long history. To inspect the code of the calendar function, just enter the function name without brackets:

```
> holidayNYSE
```

```
function (year = getRmetricsOptions("currentYear"))
{
    holidays <- NULL
    for (y in year) {
        if (y >= 1885)
            holidays <- c(holidays, as.character(USNewYearsDay(y)))
        if (y >= 1885)
            holidays <- c(holidays, as.character(USIndependenceDay(y)))
        if (y >= 1885)
            holidays <- c(holidays, as.character(USThanksgivingDay(y)))
        if (y >= 1885)
            holidays <- c(holidays, as.character(USChristmasDay(y)))
        if (y >= 1887)
            holidays <- c(holidays, as.character(USLaborDay(y)))
        if (y != 1898 & y != 1906 & y != 1907)
            holidays <- c(holidays, as.character(USGoodFriday(y)))
        if (y >= 1909 & y <= 1953)
            holidays <- c(holidays, as.character(USColumbusDay(y)))
        if (y >= 1998)
            holidays <- c(holidays, as.character(USMLKingsBirthday(y)))
        if (y >= 1896 & y <= 1953)
            holidays <- c(holidays, as.character(USLincolnsBirthday(y)))
        if (y <= 1970)
            holidays <- c(holidays, as.character(USWashingtonsBirthday(y)))
        if (y > 1970)
            holidays <- c(holidays, as.character(USPresidentsDay(y)))
        if (y == 1918 | y == 1921 | (y >= 1934 & y <= 1953))
            holidays <- c(holidays, as.character(USVeteransDay(y)))
        if (y <= 1968 | y == 1972 | y == 1976 | y == 1980)
            holidays <- c(holidays, as.character(USElectionDay(y)))
        if (y <= 1970)
            holidays <- c(holidays, as.character(USDecorationMemorialDay(y)))
        if (y >= 1971)
            holidays <- c(holidays, as.character(USMemorialDay(y)))
    }
    holidays <- sort(holidays)
    ans <- timeDate(format(holidays), zone = "NewYork", FinCenter = "NewYork")
    posix1 <- as.POSIXlt(ans, tz = "GMT")
    ans <- ans + as.integer(posix1$wday == 0) * 24 * 3600
    posix2 <- as.POSIXlt(as.POSIXct(ans, tz = "GMT") - 24 * 3600)
    y <- posix2$year + 1900
    m <- posix2$mon + 1
    calendar <- timeCalendar(y = y + (m + 1)%/%13, m = m + 1 -
        (m + 1)%/%13 * 12, d = 1, zone = "GMT", FinCenter = "GMT")
    lastday <- as.POSIXlt(calendar - 24 * 3600, tz = "GMT")$mday
```

```
        lon <- .last.of.nday(year = y, month = m, lastday = lastday,
            nday = 5)
        ExceptOnLastFriday <- timeDate(format(lon), zone = "NewYork",
            FinCenter = "NewYork")
        ans <- ans - as.integer(ans >= timeDate("1959-07-03", zone = "GMT",
            FinCenter = "GMT") & as.POSIXlt(ans, tz = "GMT")$wday ==
            6 & (ans - 24 * 3600) != ExceptOnLastFriday) * 24 * 3600
        posix3 <- as.POSIXlt(ans, tz = "GMT")
        ans <- ans[!(posix3$wday == 0 | posix3$wday == 6)]
        ans
    }
    <environment: namespace:timeDate>
```

# PART III

# timeSeries OBJECTS

# CHAPTER 11

# RMETRICS 'TIMESERIES' OBJECTS

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

## 11.1 CLASS REPRESENTATION

An Rmetrics `timeSeries` object is represented by an S4 object of class `timeSeries`. The class has eight slots, the `@Data` slot which holds the time series data in numeric matrix form, the `@position` slot which holds the date/time stamps as a character vector, the `@format` and `@FinCenter` slots which specify the position slot, the `@units` slot which holds the column names of the data matrix, the record identification slot `@recordsIDs`, which allows to save additional information as a data.frame, and a `@title` and `@documentation` slot which hold descriptive character strings.

LISTING 11.1: TIMESERIES CLASS REPRESENTATION

```
setClass("timeSeries",
    representation(
    .Data="matrix",
    positions="character",
    format="character",
    FinCenter="character",
    units="character",
    recordIDs="data.frame",
    title="character",
    documentation="character")
    )
```

## 11.2 CREATING 'TIMESERIES' OBJECTS FROM SCRATCH

### *The data matrix*

To create a `timeSeries` object we have to generate a numeric matrix where each column represents the data records for an univariate time series or financial market instrument. Let us generate a matrix with two columns and 12 rows of random normal variates. This may represent two artificial financial return series with monthly records over a period of 12 months for example for 2007.

```
> data <- matrix(rnorm(24), 12)
> data
```

```
            [,1]     [,2]
 [1,] -0.087124 -1.15672
 [2,] -0.222029  0.47747
 [3,] -0.625133  0.82025
 [4,] -0.764903 -0.32956
 [5,]  0.429583  0.66769
 [6,]  0.757527  0.65073
 [7,] -0.308916  0.63757
 [8,] -0.344570  0.31350
 [9,]  0.298759  0.69472
[10,]  0.244152 -0.32122
[11,] -0.788608  0.44199
[12,] -0.721909 -0.81990
```

### *The Character Vector of Date Stamps*

Now we have to specify the time stamps in form of a character vector. Say, we have one data record at the beginning of each month in 2007, and we are not further interested in the time stamps, then we can define the character vector from the calendar atoms year, month, and day, as

```
> year <- "2007"
> month <- sub(" ", "0", format(1:12))
> day <- "01"
> charvec <- paste(year, month, day, sep = "-")
> charvec
```

```
 [1] "2007-01-01" "2007-02-01" "2007-03-01" "2007-04-01" "2007-05-01"
 [6] "2007-06-01" "2007-07-01" "2007-08-01" "2007-09-01" "2007-10-01"
[11] "2007-11-01" "2007-12-01"
```

Note, here we have formatted the dates in the standard ISO-8601 human readable date format. To make evident, that `charvec` is a character vector, the dates are displayed quoted.

*Putting the Data and Date/Time stamps together*

The data matrix and the character vector of date and time stamps can now be combined to construct a new `timeSeries` object using the function `timeSeries()`

```
> tM <- timeSeries(data, charvec, units = c("A", "B"))
> tM
```

```
GMT
                   A        B
2007-01-01 -0.087124 -1.15672
2007-02-01 -0.222029  0.47747
2007-03-01 -0.625133  0.82025
2007-04-01 -0.764903 -0.32956
2007-05-01  0.429583  0.66769
2007-06-01  0.757527  0.65073
2007-07-01 -0.308916  0.63757
2007-08-01 -0.344570  0.31350
2007-09-01  0.298759  0.69472
2007-10-01  0.244152 -0.32122
2007-11-01 -0.788608  0.44199
2007-12-01 -0.721909 -0.81990
```

The new created object is a S4 object of class `timeSeries` which contains the following slots

```
> class(tM)
```

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

```
> slotNames(tM)
```

```
[1] ".Data"         "units"         "positions"     "format"
[5] "FinCenter"     "recordIDs"     "title"         "documentation"
```

As we notice a `timeSeries` object does not only contain the data matrix named `Data` and the character vector of date (and time) stamps names `positions`, but also slots defining by default the `format`, the financial center `FinCenter` where the data were recorded, the `units` names of the individual instrument columns (by default "TS.1". "TS.2", ...), an data frame named `recordIDs` holding optional record IDs, and a title and documentation added by defaults.

*The structure of a 'timeSeries' object*

The structure of a `timeSeries` object can be shown in more detail calling the function `str()`

```
> str(tM)
```

```
Time Series:
 Name:               object
Data Matrix:
 Dimension:          12 2
 Column Names:       A B
 Row Names:          2007-01-01  ...  2007-12-01
Positions:
 Start:              2007-01-01
 End:                2007-12-01
With:
 Format:             %Y-%m-%d
 FinCenter:          GMT
 Units:              A B
 Title:              Time Series Object
 Documentation:      Mon Dec  8 11:46:52 2014
```

The call of the `str()` function lists the name of the time series, the dimension and column names of the data matrix, the start and end positions of the data records, and additionally the POSIX format string, the name of the financial center, the time series units, the title and the documentation string.
Alternatively we can have a look on the unclassed `timeSeries` object

```
> unclass(tM)

              A        B
 [1,] -0.087124 -1.15672
 [2,] -0.222029  0.47747
 [3,] -0.625133  0.82025
 [4,] -0.764903 -0.32956
 [5,]  0.429583  0.66769
 [6,]  0.757527  0.65073
 [7,] -0.308916  0.63757
 [8,] -0.344570  0.31350
 [9,]  0.298759  0.69472
[10,]  0.244152 -0.32122
[11,] -0.788608  0.44199
[12,] -0.721909 -0.81990
attr(,"units")
[1] "A" "B"
attr(,"positions")
 [1] 1167609600 1170288000 1172707200 1175385600 1177977600 1180656000
 [7] 1183248000 1185926400 1188604800 1191196800 1193875200 1196467200
attr(,"format")
[1] "%Y-%m-%d"
attr(,"FinCenter")
[1] "GMT"
attr(,"recordIDs")
data frame with 0 columns and 0 rows
attr(,"title")
[1] "Time Series Object"
attr(,"documentation")
[1] "Mon Dec  8 11:46:52 2014"
```

This displays the attributes of the `timeSeries` object including the `Data` matrix, the `position` vector the `format` string, the `FinCenter`, name, the `units`, names, the `recordIDs`, data frame, the title and documentation strings. Beside the `Data` and `position` attributes the remaining attributes are assigned to their default values.
It is worth to note that after loading the Rmetrics packages the default setting of the financial center is that specified by the value of the global variable `myFinCenter`

```
> getRmetricsOption("myFinCenter")

myFinCenter
      "GMT"
```

This setting can be changed anytime to your personal financial center without changing your personal R environment. To work with daily data, it is best to stay with "GMT", so the whole computations do not care about time zones (TZ) and daylight savings time (DST) rules which are irrelevant for this case.

### *Using the function 'timeCalendar' to create time stamps*

To create character time stamps it is in most cases much more convenient to use the 'timeDate' and the `format` functions from the `timeDate` package. Especially the functions `timeCalendar` and `timeSequence` are powerful options to create arbitrary character vectors of time stamps.
Using the first function we can create date/time sequences with equidistant time spans. For example, using the default settings

```
> tc <- format(timeCalendar(2009))
> print(tc)

 [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"

> class(tc)

[1] "character"
```

we get a a monthly calendar for the year 2009. Note, the function `timeCalendar` returned an object of class 'timeDate' which which we have easily transformed into a simple character string using the function `format`. The remaining arguments `zone` and `FinCenter`, we used their default settings, take the responsibility for an appropriate time zone and DST information where the data were recorded, and for the financial center where they will be finally used. For just creating character time stamps we use the empty default strings "" or alternatively set `zone="GMT"` and `FinCenter="GMT"` and format the resulting `timeDate` object as a character string. This produces the same time stamp strings as used in the previous example. Note, the calendar atoms have default settings, so it is not necessary to specify all calendar atoms explicitly.

*Using the function 'timeSequence' to create time stamps*

On the other hand the function `timeSequence` can be used to produce sequences of time stamp strings, e.g. when created `from` a start `to` an end date/time,

```
> charvec <- format(timeSequence(from = "2007-01-01", to = "2007-12-01",
     by = "month"))
> timeSeries(data, charvec)
GMT
                TS.1     TS.2
2007-01-01 -0.087124 -1.15672
2007-02-01 -0.222029  0.47747
2007-03-01 -0.625133  0.82025
2007-04-01 -0.764903 -0.32956
2007-05-01  0.429583  0.66769
2007-06-01  0.757527  0.65073
2007-07-01 -0.308916  0.63757
2007-08-01 -0.344570  0.31350
2007-09-01  0.298759  0.69472
2007-10-01  0.244152 -0.32122
2007-11-01 -0.788608  0.44199
2007-12-01 -0.721909 -0.81990
```

A sequence can also be created specifying a start date `from`, a `by`-sequence string, and a `length.out` value

```
> charvec <- format(timeSequence(from = "2007-01-01", by = "month",
     length.out = 12))
> timeSeries(data, charvec)
GMT
                TS.1     TS.2
2007-01-01 -0.087124 -1.15672
2007-02-01 -0.222029  0.47747
2007-03-01 -0.625133  0.82025
2007-04-01 -0.764903 -0.32956
2007-05-01  0.429583  0.66769
2007-06-01  0.757527  0.65073
2007-07-01 -0.308916  0.63757
2007-08-01 -0.344570  0.31350
2007-09-01  0.298759  0.69472
2007-10-01  0.244152 -0.32122
2007-11-01 -0.788608  0.44199
2007-12-01 -0.721909 -0.81990
```

The `by` argument can handle periods from "year", "quarter", "month", "week", "day", "hour", "min", down to "sec" periods.
The two approaches can be combined to create sequences over the last n-periods, since the `to` argument has as default setting the current date/time stamp, e.g. the last y days

```
> Sys.timeDate()

GMT
[1] [2014-12-08 10:46:52]

> charvec <- format(timeSequence(length.out = 12))
> timeSeries(data, charvec)

GMT
                         TS.1     TS.2
2014-11-09 10:46:52 -0.087124 -1.15672
2014-11-10 10:46:52 -0.222029  0.47747
2014-11-11 10:46:52 -0.625133  0.82025
2014-11-12 10:46:52 -0.764903 -0.32956
2014-11-13 10:46:52  0.429583  0.66769
2014-11-14 10:46:52  0.757527  0.65073
2014-11-15 10:46:52 -0.308916  0.63757
2014-11-16 10:46:52 -0.344570  0.31350
2014-11-17 10:46:52  0.298759  0.69472
2014-11-18 10:46:52  0.244152 -0.32122
2014-11-19 10:46:52 -0.788608  0.44199
2014-11-20 10:46:52 -0.721909 -0.81990
```

*Creating a series with special dates*

The `timeDate` package also offers functions to create special dates for `timeDate' objects. These include`

| | |
|---|---|
| timeLastDayInMonth | Computes the last day in a given month and year |
| timeFirstDayInMonth | Computes the first day in a given month and year |
| timeLastDayInQuarter | Computes the last day in a given quarter and year |
| timeFirstDayInQuarter | Computes the first day in a given quarter and year |
| timeNthNdayInMonth | Computes n-th ocurrance of a n-day in year/month |
| timeLastNdayInMonth | Computes the last n-day in year/month |
| timeNdayOnOrAfter | Computes date in month that is a n-day ON OR AFTER |
| timeNdayOnOrBefore | Computes date in month that is a n-day ON OR BEFORE |

The next two examples show how to create date stamps for a time series recorded at the last day in each month,

```
> Date <- timeCalendar(2007)
> lastDayInMonth <- timeLastDayInMonth(Date)
> charvec <- format(lastDayInMonth)
> timeSeries(data, charvec)

GMT
                TS.1     TS.2
2007-01-31 -0.087124 -1.15672
2007-02-28 -0.222029  0.47747
2007-03-31 -0.625133  0.82025
2007-04-30 -0.764903 -0.32956
```

```
2007-05-31  0.429583  0.66769
2007-06-30  0.757527  0.65073
2007-07-31 -0.308916  0.63757
2007-08-31 -0.344570  0.31350
2007-09-30  0.298759  0.69472
2007-10-31  0.244152 -0.32122
2007-11-30 -0.788608  0.44199
2007-12-31 -0.721909 -0.81990
```

or for a time series recorded every third Friday in a month

```
> thirdFridayInMonth <- timeNthNdayInMonth(Date, nday = 5, nth = 3)
> charvec <- format(thirdFridayInMonth)
> timeSeries(data, charvec)
GMT
                TS.1     TS.2
2007-01-19 -0.087124 -1.15672
2007-02-16 -0.222029  0.47747
2007-03-16 -0.625133  0.82025
2007-04-20 -0.764903 -0.32956
2007-05-18  0.429583  0.66769
2007-06-15  0.757527  0.65073
2007-07-20 -0.308916  0.63757
2007-08-17 -0.344570  0.31350
2007-09-21  0.298759  0.69472
2007-10-19  0.244152 -0.32122
2007-11-16 -0.788608  0.44199
2007-12-21 -0.721909 -0.81990
```

### *Univariate and multivariate time series*

The data slot of a time series is a numeric matrix. The only difference between an univariate and multivariate `timeSeries` object is that the data matrix in the first case has exactly one column, and in the multivariate case more than one column. Note, we do not drop indices and represent univariate time series as vector.
To find out if a time series is univariate or multivariate we can use the functions `isUnivariate()` and its negation `isMultivariate()`.

```
> tUni <- timeSeries(data[, 1], charvec)
> c(isUnivariate(tUni), isMultivariate(tUni))
[1]  TRUE FALSE
> tBiv <- timeSeries(data, charvec, units = c("A", "B"))
> c(isUnivariate(tBiv), isMultivariate(tBiv))
[1] FALSE  TRUE
```

Furthermore the functions `ncol()`, `NCOL()`, `nrow()`, `NROW()`, `dim()`, give us information about the size of a `timeSeries` object, i.e. the number of rows and columns. These functions behave in the same way as their matrix counterparts.

```
> dim(tUni)
[1] 12  1
> dim(tBiv)
[1] 12  2
```

*Retrieving data and positions from 'timeSeries' objects*

To retrieve the data and time stamps of a `timeSeries` object we can call the functions `series()` and `time()`, respectively

```
> series(tBiv)
                   A        B
2007-01-19 -0.087124 -1.15672
2007-02-16 -0.222029  0.47747
2007-03-16 -0.625133  0.82025
2007-04-20 -0.764903 -0.32956
2007-05-18  0.429583  0.66769
2007-06-15  0.757527  0.65073
2007-07-20 -0.308916  0.63757
2007-08-17 -0.344570  0.31350
2007-09-21  0.298759  0.69472
2007-10-19  0.244152 -0.32122
2007-11-16 -0.788608  0.44199
2007-12-21 -0.721909 -0.81990
```

and

```
> time(tBiv)
GMT
 [1] [2007-01-19] [2007-02-16] [2007-03-16] [2007-04-20] [2007-05-18]
 [6] [2007-06-15] [2007-07-20] [2007-08-17] [2007-09-21] [2007-10-19]
[11] [2007-11-16] [2007-12-21]
```

To assign new time stamps, we can use the assignment function `time<-` and `series<-`. The following example shows how to shift the positions of a `timeSeries` object one day back in time

```
> time(tBiv) <- time(tBiv) - 24 * 3600
> tBiv
GMT
                   A        B
2007-01-18 -0.087124 -1.15672
2007-02-15 -0.222029  0.47747
2007-03-15 -0.625133  0.82025
2007-04-19 -0.764903 -0.32956
2007-05-17  0.429583  0.66769
2007-06-14  0.757527  0.65073
2007-07-19 -0.308916  0.63757
2007-08-16 -0.344570  0.31350
2007-09-20  0.298759  0.69472
2007-10-18  0.244152 -0.32122
2007-11-15 -0.788608  0.44199
2007-12-20 -0.721909 -0.81990
```

To display and modify the names of a `timeSeries` we can use the functions `colnames()`

```
> colnames(tBiv)
[1] "A" "B"
> head(tBiv)
GMT
                   A        B
2007-01-18 -0.087124 -1.15672
2007-02-15 -0.222029  0.47747
2007-03-15 -0.625133  0.82025
2007-04-19 -0.764903 -0.32956
2007-05-17  0.429583  0.66769
2007-06-14  0.757527  0.65073
```

*Working with intraday 'timeSeries' objects*

So far we were working only under the conditions that the `timeSeries` were recorded on a daily basis and used with "GMT" time stamps neglecting time zones and daylight savings times. Now we want to consider `timeSeries` objects when the time stamps in addition to the dates are becoming important.
Assume our data were recorded at the end of the day, say "16:15" in New York City local time. So we have to set the time zone, `zone="NewYork"`, and the financial center, `FinCenter="NewYork"`

```
> charvec <- format(timeCalendar(2007, h = 16, min = 15))
> tNYC <- timeSeries(data, charvec, zone = "New_York", FinCenter = "New_York")
> tNYC
New_York
                          TS.1     TS.2
2007-01-01 16:15:00 -0.087124 -1.15672
2007-02-01 16:15:00 -0.222029  0.47747
2007-03-01 16:15:00 -0.625133  0.82025
2007-04-01 16:15:00 -0.764903 -0.32956
2007-05-01 16:15:00  0.429583  0.66769
2007-06-01 16:15:00  0.757527  0.65073
2007-07-01 16:15:00 -0.308916  0.63757
2007-08-01 16:15:00 -0.344570  0.31350
2007-09-01 16:15:00  0.298759  0.69472
2007-10-01 16:15:00  0.244152 -0.32122
2007-11-01 16:15:00 -0.788608  0.44199
2007-12-01 16:15:00 -0.721909 -0.81990
```

Note that the printout is in New York City local time, as remarked at the beginning of the printout. However, if our financial center is in Zurich, then we can use the New York City recorded 'timeSeries' under local Zurich time stamps

```
> tZRH <- timeSeries(data, charvec, zone = "New_York", FinCenter = "Zurich")
> tZRH
Zurich
                         TS.1     TS.2
2007-01-01 22:15:00 -0.087124 -1.15672
2007-02-01 22:15:00 -0.222029  0.47747
2007-03-01 22:15:00 -0.625133  0.82025
2007-04-01 22:15:00 -0.764903 -0.32956
2007-05-01 22:15:00  0.429583  0.66769
2007-06-01 22:15:00  0.757527  0.65073
2007-07-01 22:15:00 -0.308916  0.63757
2007-08-01 22:15:00 -0.344570  0.31350
2007-09-01 22:15:00  0.298759  0.69472
2007-10-01 22:15:00  0.244152 -0.32122
2007-11-01 21:15:00 -0.788608  0.44199
2007-12-01 22:15:00 -0.721909 -0.81990
```

Note, the time stamp in November when the data was becoming one hour earlier available in Zurich due to the circumstance that USA and Switzerland switched at different dates from summer time to winter time.

*Changing the financial center of a timeSeries object*

The function `finCenter<-()` allows to change the financial center of a timeSeries object. Create an artificial monthly time series for the financial center Zurich with closing afternoon time stamps at 16:00

```
> tS <- timeSeries(data = rnorm(12), charvec = timeCalendar(y = 2010,
     h = 16, zone = "Zurich", FinCenter = "Zurich"))
> tS
Zurich
                         TS.1
2010-01-01 16:00:00  0.856252
2010-02-01 16:00:00  0.522510
2010-03-01 16:00:00  1.042153
2010-04-01 16:00:00 -1.088653
2010-05-01 16:00:00 -0.661440
2010-06-01 16:00:00  0.782724
2010-07-01 16:00:00 -0.620497
2010-08-01 16:00:00 -1.873180
2010-09-01 16:00:00  0.060007
2010-10-01 16:00:00  0.194489
2010-11-01 16:00:00 -1.069811
2010-12-01 16:00:00 -0.975770
```

and then change the center to New York

```
> finCenter(tS) <- "New_York"
> tS
New_York
                         TS.1
2010-01-01 10:00:00  0.856252
2010-02-01 10:00:00  0.522510
```

```
2010-03-01 10:00:00  1.042153
2010-04-01 10:00:00 -1.088653
2010-05-01 10:00:00 -0.661440
2010-06-01 10:00:00  0.782724
2010-07-01 10:00:00 -0.620497
2010-08-01 10:00:00 -1.873180
2010-09-01 10:00:00  0.060007
2010-10-01 10:00:00  0.194489
2010-11-01 11:00:00 -1.069811
2010-12-01 10:00:00 -0.975770
```

Note, the November record which seems not to fit in the series of regular time stamps. Have a look on the DST rules for Zurich and New York

```
> Zurich()[65, ]

                Zurich offSet isdst TimeZone    numeric
65 2010-10-31 01:00:00   3600     0      CET 1288486800

> New_York()[181, ]

               New_York offSet isdst TimeZone    numeric
181 2010-11-07 06:00:00 -18000     0      EST 1289109600
```

and we see that in Zurich winter time started on the last day of October and in New York one week later, and the end of the first week in November.

## 11.3 Importing 'timeSeries' Objects

Rmetrics comes with many time series demo files, allows downloads of economic and financial market data from the Internet and from csv Excel ASCII files.

### *data: Loading example data files*

The demo files coming with Rmetrics are semicolon separated csv ASCII files, where the first column holds the time stamps and the remaining columns the numeric data records. The first row holds the POSIX "format" string and "units" names one for each time series column. As for data frames we can load example files using the function `data` and convert the result after loading into an object ov class 'timeSeries'

```
> data(msft.dat)
> class(msft.dat)

[1] "data.frame"

> MSFT <- as.timeSeries(msft.dat)
```

Have a look on the first few records of the MSFT time series

```
> head(MSFT)
```

```
GMT
              Open   High    Low  Close   Volume
2000-09-27 63.438 63.562 59.812 60.625 53077800
2000-09-28 60.812 61.875 60.625 61.312 26180200
2000-09-29 61.000 61.312 58.625 60.312 37026800
2000-10-02 60.500 60.812 58.250 59.125 29281200
2000-10-03 59.562 59.812 56.500 56.562 42687000
2000-10-04 56.375 56.562 54.500 55.438 68226700
```

*Reading CSV files*

The function `readSeries()` can be used to load data from a csv or ASCII file.

*Importing time series data from the Internet*

Rmetrics also offers functions to download economic and financial market data from the Internet. Three of them we will briefly describe, they are part of the Rmetrics package `fImport`.
The function `yahooSeries()` imports financial market data from the Yahoo Finance server.
The function `fredSeries()` imports economic time Series data from the Federal Reserve database FRED2.
The function `oandaSeries()`: imports foreign exchange time Series from the Oanda Web Server.

## 11.4 EXTRACTING AND MODIFYING THE SLOTS OF A TIME SERIES

*The '.Data' slot*

The function `getDataPart()` extracts the data slot.

```
> series.tS <- timeSeries(data = rnorm(12), charvec = timeCalendar(),
     units = "Series")
> data <- getDataPart(series.tS)
> class(data)

[1] "matrix"

> data

         Series
 [1,] -0.666625
 [2,] -0.682131
 [3,]  1.201481
 [4,]  0.072575
 [5,]  0.597136
 [6,] -1.385716
 [7,] -1.254512
 [8,]  2.185639
 [9,] -1.856955
[10,] -2.063412
```

```
[11,] -0.363998
[12,] -0.360139
```

The function `series()` provides an alternative option to extract the data part, it keeps the date/time stamps

```
> data <- series(series.tS)
> class(data)
[1] "matrix"
> data
              Series
2014-01-01 -0.666625
2014-02-01 -0.682131
2014-03-01  1.201481
2014-04-01  0.072575
2014-05-01  0.597136
2014-06-01 -1.385716
2014-07-01 -1.254512
2014-08-01  2.185639
2014-09-01 -1.856955
2014-10-01 -2.063412
2014-11-01 -0.363998
2014-12-01 -0.360139
```

The function `as.matrix()` extracts the data part as a matrix it returns the same result as the function `series()`.

```
> data <- as.matrix(series.tS)
> class(data)
[1] "matrix"
> data
              Series
2014-01-01 -0.666625
2014-02-01 -0.682131
2014-03-01  1.201481
2014-04-01  0.072575
2014-05-01  0.597136
2014-06-01 -1.385716
2014-07-01 -1.254512
2014-08-01  2.185639
2014-09-01 -1.856955
2014-10-01 -2.063412
2014-11-01 -0.363998
2014-12-01 -0.360139
```

### *The 'units' slot*

The functions `colnames()` returns the column or unit names of a `timeSeries` object.

```
> colnames(series.tS)
[1] "Series"
```

*The 'position' slot*

The function `time()` extracts the vector of time stamps at which a time series was sampled, or in other words they extract the date/time position values from a `timeSeries` object as an object of class `timeDate`.

```
> positions <- time(series.tS)
> class(positions)

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

> positions

GMT
 [1] [2014-01-01] [2014-02-01] [2014-03-01] [2014-04-01] [2014-05-01]
 [6] [2014-06-01] [2014-07-01] [2014-08-01] [2014-09-01] [2014-10-01]
[11] [2014-11-01] [2014-12-01]
```

Now you can use the functions `format()`

```
> charvec <- format(positions)
> class(charvec)

[1] "character"

> charvec

 [1] "2014-01-01" "2014-02-01" "2014-03-01" "2014-04-01" "2014-05-01"
 [6] "2014-06-01" "2014-07-01" "2014-08-01" "2014-09-01" "2014-10-01"
[11] "2014-11-01" "2014-12-01"
```

or `as.character()`

```
> charvec <- as.character(positions)
> class(charvec)

[1] "character"

> charvec

 [1] "2014-01-01" "2014-02-01" "2014-03-01" "2014-04-01" "2014-05-01"
 [6] "2014-06-01" "2014-07-01" "2014-08-01" "2014-09-01" "2014-10-01"
[11] "2014-11-01" "2014-12-01"
```

to transform the `timeDate` positions into an ISO-8601 formatted character vector.

*More about 'timeSeries' slots*

The `.Data`, `units`, and `positions` slots are the three most important slots of a `timeSeries` object. But there are 5 more slots.

```
> slotNames(series.tS)

[1] ".Data"         "units"         "positions"     "format"
[5] "FinCenter"     "recordIDs"     "title"         "documentation"
```

to retrieve the content of these slots, use the function `slot()` from the `R` package `methods`

```
> args(slot)
function (object, name)
NULL
```

For example to get the financial center, type

```
> slot(series.tS, "FinCenter")
[1] "GMT"
```

or to get the format of the date/time position stamps, type

```
> slot(series.tS, "format")
[1] "%Y-%m-%d"
```

## 11.5 PRINTING 'TIMESERIES' OBJECTS

The `print()` function offers a view on a `timeSeries` object. Such a view can be tailored in several aspects. This may concern the form how we display the series, for example in vertical or horizontal printing style, to which time zone and financial center we relate the time series, in which format we like to present the date and time stamps.

### *Vertical printing style*

Printing by default a `timeSeries` object under Rmetrics is done displaying the series column-wise, even if the series is univariate. We call this output style "vertical" printing of a `timeSeries` object. The default `print` function, and in addition the functions `head` and `tail` produce all the same style of printouts.

```
> print(tUni)
GMT
                TS.1
2007-01-19 -0.087124
2007-02-16 -0.222029
2007-03-16 -0.625133
2007-04-20 -0.764903
2007-05-18  0.429583
2007-06-15  0.757527
2007-07-20 -0.308916
2007-08-17 -0.344570
2007-09-21  0.298759
2007-10-19  0.244152
2007-11-16 -0.788608
2007-12-21 -0.721909
```

### *Horizontal printing style*

If you prefer "horizontal" printouts for univariate ’timeSeries’ objects, then you can call the print function with an additional argument, `style="h"`.

```
> print(tUni, style = "h")
2007-01-19 2007-02-16 2007-03-16 2007-04-20 2007-05-18 2007-06-15 2007-07-20
 -0.087124  -0.222029  -0.625133  -0.764903   0.429583   0.757527  -0.308916
2007-08-17 2007-09-21 2007-10-19 2007-11-16 2007-12-21
 -0.344570   0.298759   0.244152  -0.788608  -0.721909
```

### *Printing with Respect to Another Financial Center*

It is also possible to display a `timeSeries` with respect to another financial center. In this case we have just to specify the desired financial center

```
> print(tNYC, FinCenter = "London")
London
                         TS.1     TS.2
2007-01-01 21:15:00 -0.087124 -1.15672
2007-02-01 21:15:00 -0.222029  0.47747
2007-03-01 21:15:00 -0.625133  0.82025
2007-04-01 21:15:00 -0.764903 -0.32956
2007-05-01 21:15:00  0.429583  0.66769
2007-06-01 21:15:00  0.757527  0.65073
2007-07-01 21:15:00 -0.308916  0.63757
2007-08-01 21:15:00 -0.344570  0.31350
2007-09-01 21:15:00  0.298759  0.69472
2007-10-01 21:15:00  0.244152 -0.32122
2007-11-01 20:15:00 -0.788608  0.44199
2007-12-01 21:15:00 -0.721909 -0.81990
```

### *Printing with tailored POSIX formats*

Sometimes it may be desired, for example in business reports, to replace ISO-8601 formatted time stamps with American data formats, or to abbreviate time stamps. As an example we consider the `tNYC` time series and print it in US date format and suppress the obsolete seconds in the time stamps

```
> print(tNYC, format = "%m/%d/%y %H:%M")
New_York
                    TS.1     TS.2
01/01/07 11:15 -0.087124 -1.15672
02/01/07 11:15 -0.222029  0.47747
03/01/07 11:15 -0.625133  0.82025
04/01/07 12:15 -0.764903 -0.32956
05/01/07 12:15  0.429583  0.66769
06/01/07 12:15  0.757527  0.65073
07/01/07 12:15 -0.308916  0.63757
08/01/07 12:15 -0.344570  0.31350
```

```
09/01/07 12:15  0.298759  0.69472
10/01/07 12:15  0.244152 -0.32122
11/01/07 12:15 -0.788608  0.44199
12/01/07 11:15 -0.721909 -0.81990
```

*Printing regular 'timeSeries' objects*

For the user there is **no** difference in Rmetrics between regular and irregular `timeSeries` objects. As we have seen, the function `timeSequence` is ideally suited for generating `timeSeries` objects with time stamps arranged in an regular way.

```
> tMonth <- timeSeries(data[, 1], timeSequence(from = "2008-01-01",
     by = "month", length.out = 12))
> tQuarter <- timeSeries(data[, 1], timeSequence(from = "2008-01-01",
     by = "quarter", length.out = 12))
```

Such regular `timeSeries` objects can be printed in in "ts-like" style, e.g. for a regular monthly series as

```
> print(tMonth, style = "ts")
           Jan       Feb       Mar       Apr       May       Jun       Jul
2008 -0.666625 -0.682131  1.201481  0.072575  0.597136 -1.385716 -1.254512
           Aug       Sep       Oct       Nov       Dec
2008  2.185639 -1.856955 -2.063412 -0.363998 -0.360139
```

and for a quarterly series as

```
> print(tQuarter, style = "ts", by = "quarter")
          Qtr1      Qtr2      Qtr3      Qtr4
2008 -0.666625 -0.682131  1.201481  0.072575
2009  0.597136 -1.385716 -1.254512  2.185639
2010 -1.856955 -2.063412 -0.363998 -0.360139
2011 -0.666625 -0.682131  1.201481  0.072575
2012  0.597136 -1.385716
```

## 11.6 PLOTTING 'TIMESERIES' OBJECTS

Plotting by default a 'timeSeries' under Rmetrics is done by calling the generic function `plot`. The function is implemented in the same spirit as the function `plot.ts` for regular time series objects `ts` in R's base package. The function comes with the same arguments and some additional arguments, for user specified "axis" labelling, and for modifying the plot "layout".
Like for `ts`, three different types of plots can be displayed, a multiple plot, where up to 10 subplots can be produced on one sheet of a paper, a single plot, where all curves are drawn in one plot on the same sheet, and a scatter plot of two univariate `timeSeries` objects. The type of the plot can be selected by specifying the function argument `plot.type`

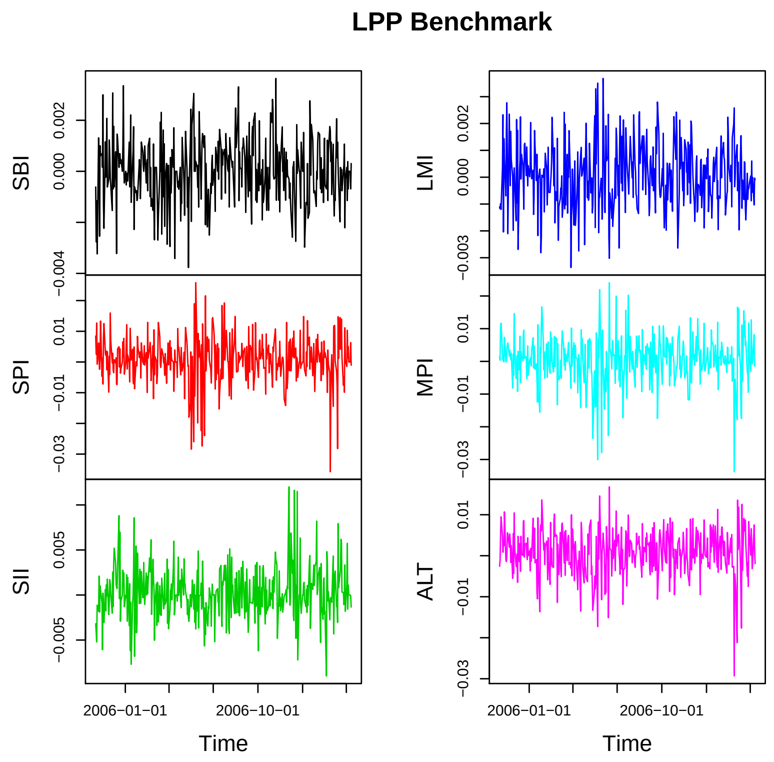

FIGURE 11.1: Multiple time series plot of the Swiss pension fund assets.

### *Multiple plots*

As an example we load the Swiss pension fund asset set, and transform the data frame into an object of class `timeSeries`

```
> LPP2005REC <- as.timeSeries(data(LPP2005REC))[, 1:6]
> colnames(LPP2005REC)

[1] "SBI" "SPI" "SII" "LMI" "MPI" "ALT"
```

and plot the six time series

```
> plot(LPP2005REC, main = "LPP Benchmark")
```

### *Single plots*

To create a single plot, we have to set the argument `plot.type` to `"single"`

```
> plot(LPP2005REC, plot.type = "single", col = 1:6)
```

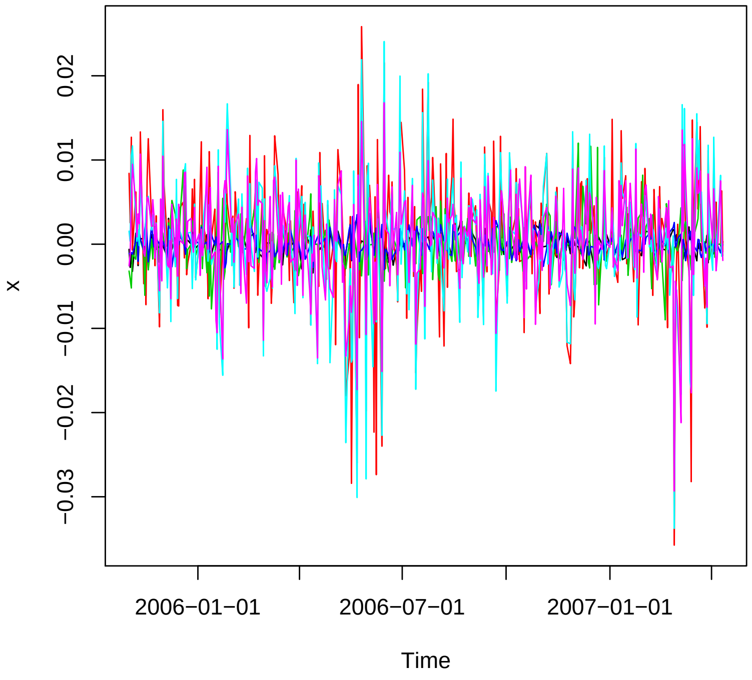

FIGURE 11.2: Single plot of the swiss pension fund asset returns.

### *Scatter plots*

To create a scatterplot of the Bond versus Equities returns we type

```
> plot(LPP2005REC[, "SBI"], LPP2005REC[, "SPI"], xlab = "SBI",
     ylab = "SPI", pch = 19)
> grid()
```

### *Selecting plot symbols*

If we like to change the symbols on a plot and we like to see an overview of the availaible plotting symbols, we can display them calling he function `symbolTable()` from the package `fBasics`.

```
> symbolTable()
```

### *Selecting plot characters and fonts*

The function `characterTable` shows a table of characters for the specified font.

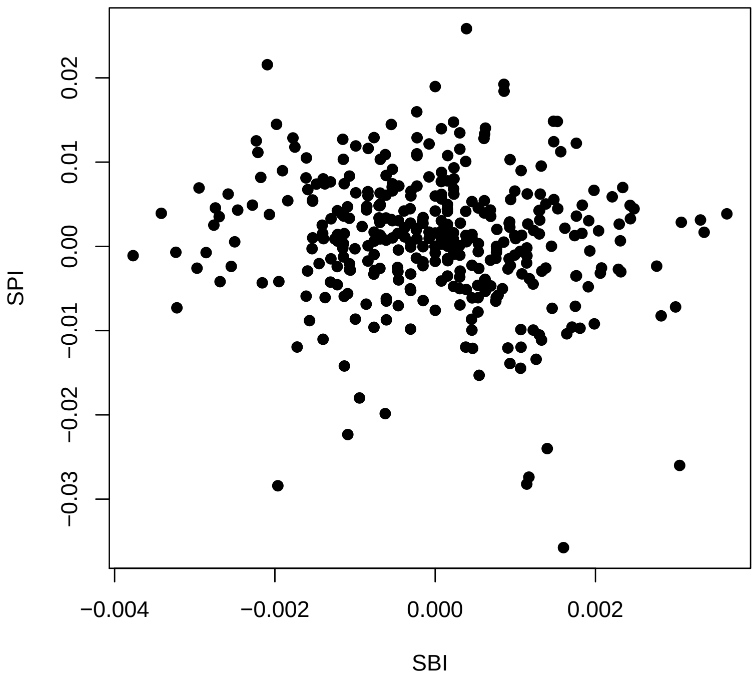

FIGURE 11.3: Scatter plot of Swiss bond versus equities returns.

```
> characterTable(font = 1, cex = 0.7)
```

*Selecting colors*

To display the color set call the function `colorTable()`

```
> colorTable()
```

## 11.7 SIGNAL SERIES: TIME SERIES WITHOUT TIME STAMPS

A signal series is a time series which has no time stamps. In this case time is considered as an event or a tick value and is measured in counts. The format of these, we call them signal series, is described by the value `format="counts"`. Nothing else is different from a time series with time stamps beside the fact the time is measured in counts.

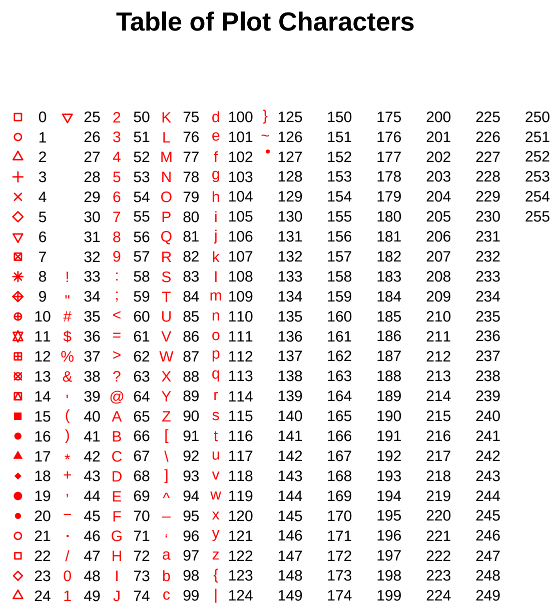

FIGURE 11.4: Display of available plotting symbols.

Signal series can be created in the same way as time Series objects with time stamps, just setting `format="counts"` and omitting the `charvec` argument

```
> signal <- timeSeries(data = matrix(rnorm(12), ncol = 2), format = "counts")
> signal

         SS.1       SS.2
[1,]  0.74372  0.1540124
[2,] -2.29390 -0.8317458
[3,]  1.87913  0.3559079
[4,] -1.30957 -0.0568090
[5,] -1.00029  0.0063327
[6,]  1.36818  0.5641710

> class(signal)

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

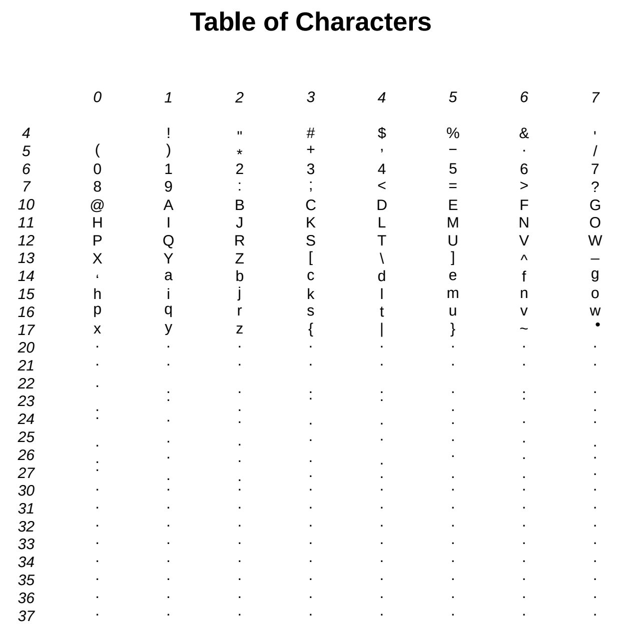

**Table of Characters**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 4 | | ! | " | # | $ | % | & | ' |
| 5 | ( | ) | * | + | , | − | . | / |
| 6 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 7 | 8 | 9 | : | ; | < | = | > | ? |
| 10 | @ | A | B | C | D | E | F | G |
| 11 | H | I | J | K | L | M | N | O |
| 12 | P | Q | R | S | T | U | V | W |
| 13 | X | Y | Z | [ | \ | ] | ^ | _ |
| 14 | ‘ | a | b | c | d | e | f | g |
| 15 | h | i | j | k | l | m | n | o |
| 16 | p | q | r | s | t | u | v | w |
| 17 | x | y | z | { | \| | } | ~ | • |
| 20 | . | . | . | . | . | . | . | . |
| 21 | . | . | . | . | . | . | . | . |
| 22 | . | | . | | | . | | . |
| 23 | | : | . | : | : | . | : | . |
| 24 | : | . | : | . | . | . | . | . |
| 25 | . | . | . | . | . | . | . | . |
| 26 | : | . | . | . | . | . | . | . |
| 27 | | . | : | : | : | . | . | . |
| 30 | . | . | . | . | . | . | . | . |
| 31 | . | . | . | . | . | . | . | . |
| 32 | . | . | . | . | . | . | . | . |
| 33 | . | . | . | . | . | . | . | . |
| 34 | . | . | . | . | . | . | . | . |
| 35 | . | . | . | . | . | . | . | . |
| 36 | . | . | . | . | . | . | . | . |
| 37 | . | . | . | . | . | . | . | . |

FIGURE 11.5: Display of available characters.

Since a signal series is an object of the same class as a 'timeSeries' object with time stamps all operations on the signal series can be handled in the same way. This includes printing, subsetting, merging and binding, math operations, ordering, and handling of missing values.

## 11.8 REGULAR TIME SERIES

A time series is a sequence of time varying observations. Usually this observation are recorded in equidistant time intervals. Here equidistant means for example every day, or every hour, 15 minutes,or 30 seconds. Evenmore these intervals must not necessarily equidistant in (physical) time, the may also equidistant on calendar intervals, e.g. every quarter (3 months), or months.

We want to go one step further and also consider a time series as a regular time series if we have a rule how to construct the time intervals. This may

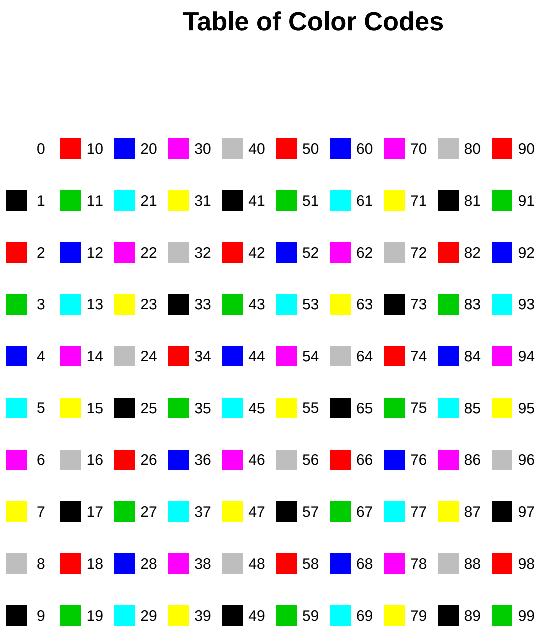

FIGURE 11.6: Display of available colors.

be for example records with time stamps, say every last business day in a month, or the last trading day of a quarter.

*The function ’ts’*

The function `ts()`

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

is used to create time series objects in the basic `R` environment. The created objects are vectors or matrices with class of `ts` and additional attributes which represent data which has been sampled at equidistant spaced points in time. In the matrix case, each column of the matrix data is assumed to contain a single (univariate) time series. Time series must

have at least one observation, and although they need not be numeric there is very limited support for non-numeric series.
Let us create an univariate and a bivariate monthly time series of random data starting January 2010

```
> ts(data = rnorm(12), start = 2010, frequency = 1)

Time Series:
Start = 2010
End = 2021
Frequency = 1
 [1] -1.575795 -0.047126  2.147334 -1.653221 -1.325466  1.457115  0.358089
 [8]  1.915577 -1.519728  0.345085  0.891109  0.247309
```

```
> ts(data = matrix(rnorm(24), ncol = 2), start = 2010, frequency = 1)

Time Series:
Start = 2010
End = 2021
Frequency = 1
     Series 1 Series 2
2010  0.74971  1.92408
2011 -0.12573  0.46931
2012  0.71983  0.84278
2013 -1.65857 -0.18119
2014  0.58234  1.42461
2015 -0.47709 -1.60968
2016  1.11040 -1.03675
2017  1.28125 -0.51656
2018 -0.47747  0.18430
2019 -0.92231 -1.04344
2020  0.40536 -1.51041
2021 -0.56451  0.99528
```

To create quarterly data proceed in the following way

```
> tUni.ts <- ts(data = rnorm(12), start = 2010, frequency = 4)
> tUni.ts

          Qtr1      Qtr2      Qtr3      Qtr4
2010 -1.340341  0.259384  0.195881 -1.090700
2011 -1.392591 -0.091133  1.377784 -0.120940
2012  0.387746  0.123078 -1.680027  0.030659

> class(tUni.ts)

[1] "ts"
```

```
> tBiv.ts <- ts(data = matrix(rnorm(24), ncol = 2), start = 2010,
     frequency = 4)
> tBiv.ts

         Series 1  Series 2
2010 Q1 -0.077533 -0.605248
2010 Q2 -0.324401  0.376807
2010 Q3 -1.414077 -1.255779
2010 Q4  0.650082  1.433665
```

```
2011 Q1 -1.041054 -0.459613
2011 Q2  0.962520  1.086854
2011 Q3 -1.939346  0.612105
2011 Q4  0.583909 -0.760253
2012 Q1 -1.546535 -0.038751
2012 Q2  1.281426  1.035139
2012 Q3  0.219714  1.402232
2012 Q4 -0.490338 -0.796047
```

```
> class(tBiv.ts)
```

```
[1] "mts"    "ts"     "matrix"
```

*Creating 'timeSeries' Objects from Regular Time Series*

To create `timeSeries` objects from a regular time series we can use the function `as.timeSeries()`. This works for univariate `timeSeries`

```
> tUniQ <- as.timeSeries(tUni.ts)
> tUniQ
```

```
GMT
                TS.1
2010-03-31 -1.340341
2010-06-30  0.259384
2010-09-30  0.195881
2010-12-31 -1.090700
2011-03-31 -1.392591
2011-06-30 -0.091133
2011-09-30  1.377784
2011-12-31 -0.120940
2012-03-31  0.387746
2012-06-30  0.123078
2012-09-30 -1.680027
2012-12-31  0.030659
```

but also for bivariate and multivariate time series objects

```
> tBivQ <- as.timeSeries(tBiv.ts)
> tBivQ
```

```
GMT
            Series 1  Series 2
2010-03-31 -0.077533 -0.605248
2010-06-30 -0.324401  0.376807
2010-09-30 -1.414077 -1.255779
2010-12-31  0.650082  1.433665
2011-03-31 -1.041054 -0.459613
2011-06-30  0.962520  1.086854
2011-09-30 -1.939346  0.612105
2011-12-31  0.583909 -0.760253
2012-03-31 -1.546535 -0.038751
2012-06-30  1.281426  1.035139
2012-09-30  0.219714  1.402232
2012-12-31 -0.490338 -0.796047
```

*Checking if a 'timeSeries' is regular*

To test if a time series is regular or not, we have to check if the time and date stamps are regular. Thus we have to extract the time and date stamps using the function `time()` and to test the vector. The rules will be the same as for `timeDate` objects

- A `timeSeries` is defined as daily if the vector has not more than one date/time stamp per day.
- A `timeSeries` is defined as monthly if the vector has not more than one date/time stamp per month.
- A `timeSeries` is defined as quarterly if the vector has not more than one date/time stamp per quarter.
- A monthly `timeSeries` is also a daily `timeSeries`, a quarterly `timeSeries` is also a monthly vector.
- A regular `timeSeries` is either a monthly or a quarterly `timeSeries`.

Note, there exist also generic functions for `timeSeries` objects, `isRegular()`, `isDaily()`, `isMonthly()`, and `isQuarterly()`.

```
> showMethods(isRegular)

Function: isRegular (package timeDate)
x="timeDate"
x="timeSeries"

> showMethods(isDaily)

Function: isDaily (package timeDate)
x="timeDate"
x="timeSeries"

> showMethods(isMonthly)

Function: isMonthly (package timeDate)
x="timeDate"
x="timeSeries"

> showMethods(isQuarterly)

Function: isQuarterly (package timeDate)
x="timeDate"
x="timeSeries"
```

Let us create a daily, a monthly and a quarterly `timeSeries` with random records

```
> tD <- timeSeries(data = matrix(rnorm(365 * 2), ncol = 2), charvec = timeSequence(from = "2010-01-01",
     length.out = 365))
> tM <- timeSeries(data = matrix(rnorm(24), ncol = 2), charvec = timeCalendar(2010))
> tQ <- tM[c(3, 6, 9, 12), ]
```

The daily time series is irregular, the monthly and quarterly time series are regular

```
> isRegular(time(tD))
[1] FALSE
> isRegular(time(tM))
[1] TRUE
> isRegular(tQ)
[1] TRUE
```

Now check which of the time series are daily series

```
> isDaily(tD)
[1] TRUE
> isDaily(tM)
[1] FALSE
> isDaily(tQ)
[1] FALSE
```

Note, that `tD` is no longer a daily series if date records are missing. Let us remove as an example record number 99.

```
> isDaily(tD[-99, ])
[1] FALSE
```

Next check which of the time series are monthly series

```
> isMonthly(tD)
[1] FALSE
> isMonthly(tM)
[1] TRUE
> isMonthly(tQ)
[1] FALSE
```

and which of the time series are quarterly series

```
> isQuarterly(tD)
[1] FALSE
> isQuarterly(tM)
[1] FALSE
> isQuarterly(tQ)
[1] TRUE
> isQuarterly(tM[c(2, 5, 8, 11), ])
[1] TRUE
```

Note the time series `tM[c(2, 5, 8, 11), ]` is also considered as a quarterly classtimeSeries object.
Finally the frequency of a daily `timeSeries` object is 1, of a monthly `timeSeries` 12, and of of a quarterly `timeSeries` 4

```
> frequency(tD)
[1] 1
> frequency(tM)
[1] 12
> frequency(tQ)
[1] 4
```

# CHAPTER 12

# GROUP GENERIC OPERATIONS

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

In R methods can be defined for groups of functions known as group generic functions. These exist in both S3 and S4 flavors, with different groups. Methods are defined for the group of functions as a whole. A method defined for an individual member of the group takes precedence over a method defined for the group as a whole.
Ops(e1, e2)
Math(e1, e2)
Math2(x, digits)
Summary(x, ..., na.rm=FALSE)
All the functions in these groups (other than the group generics themselves) are basic functions in R. They are not by default S4 generic functions, and many of them are defined as primitives, meaning that they do not have formal arguments. However, we can still define formal methods for them. The effect of doing so is to create an S4 generic function with the appropriate arguments, in the environment where the method definition is to be stored[1]

## 12.1 OPS: ARITMETIC, COMPARISON AND LOGICAL OPERATIONS

First let us show some arithmetic, compare, and logic operations.

[1]Note that two members of the Math group, `log` and `trunc`, have more than one argument: S4 group dispatch will always pass only one argument to the method so if you want to handle base in log, set a specific method as well.

### *Aritmetic Operations*

Adding and subtracting a constant and/or multiplying and dividing a `timeSeries` object is done in the usual way, e.g.

```
> tBiv <- dummySeries()
> tAB <- tBiv
> 2.5 * (tBiv - 0.5) + 1.5
GMT
              TS.1    TS.2
2014-01-01 1.12700 1.03404
2014-02-01 0.32468 2.61945
2014-03-01 0.68207 2.53835
2014-04-01 1.72181 1.29924
2014-05-01 1.26611 2.73600
2014-06-01 1.45397 1.30219
2014-07-01 0.62144 0.32399
2014-08-01 0.73180 1.14258
2014-09-01 0.83757 2.13659
2014-10-01 1.61599 2.58825
2014-11-01 0.83559 0.87031
2014-12-01 0.44650 2.28816
> tBiv^2
GMT
                 TS.1       TS.2
2014-01-01 0.12306177 0.09835451
2014-02-01 0.00089235 0.89828559
2014-03-01 0.02986886 0.83784620
2014-04-01 0.34659825 0.17614356
2014-05-01 0.16519538 0.98882739
2014-06-01 0.23192814 0.17713757
2014-07-01 0.02207512 0.00087583
2014-08-01 0.03714150 0.12747268
2014-09-01 0.05523769 0.56947586
2014-10-01 0.29854789 0.87478869
2014-11-01 0.05486715 0.06156574
2014-12-01 0.00617816 0.66465292
```

### *Comparisons*

```
> tBiv^2 == tBiv * tBiv
GMT
           TS.1 TS.2
2014-01-01 TRUE TRUE
2014-02-01 TRUE TRUE
2014-03-01 TRUE TRUE
2014-04-01 TRUE TRUE
2014-05-01 TRUE TRUE
2014-06-01 TRUE TRUE
2014-07-01 TRUE TRUE
2014-08-01 TRUE TRUE
2014-09-01 TRUE TRUE
```

```
2014-10-01 TRUE TRUE
2014-11-01 TRUE TRUE
2014-12-01 TRUE TRUE
```

*Logical Operations*

```
> tBiv > 0.2 & tBiv < 0.4
GMT
            TS.1  TS.2
2014-01-01  TRUE  TRUE
2014-02-01 FALSE FALSE
2014-03-01 FALSE FALSE
2014-04-01 FALSE FALSE
2014-05-01 FALSE FALSE
2014-06-01 FALSE FALSE
2014-07-01 FALSE FALSE
2014-08-01 FALSE  TRUE
2014-09-01  TRUE FALSE
2014-10-01 FALSE FALSE
2014-11-01  TRUE  TRUE
2014-12-01 FALSE FALSE
```

## 12.2 MATH: MATHEMATICAL OPERATIONS

### Math S4 group generics

```
> log(abs(tAB))
GMT
               TS.1       TS.2
2014-01-01 -1.04753 -1.1595884
2014-02-01 -3.51082 -0.0536336
2014-03-01 -1.75547 -0.0884604
2014-04-01 -0.52979 -0.8682280
2014-05-01 -0.90031 -0.0056177
2014-06-01 -0.73066 -0.8654143
2014-07-01 -1.90665 -3.5201689
2014-08-01 -1.64651 -1.0299266
2014-09-01 -1.44805 -0.2815194
2014-10-01 -0.60441 -0.0668865
2014-11-01 -1.45142 -1.3938249
2014-12-01 -2.54337 -0.2042452
```

### Math2 S4 group generics

```
> round(tAB, 2)
GMT
           TS.1 TS.2
2014-01-01 0.35 0.31
2014-02-01 0.03 0.95
2014-03-01 0.17 0.92
2014-04-01 0.59 0.42
```

```
2014-05-01 0.41 0.99
2014-06-01 0.48 0.42
2014-07-01 0.15 0.03
2014-08-01 0.19 0.36
2014-09-01 0.24 0.75
2014-10-01 0.55 0.94
2014-11-01 0.23 0.25
2014-12-01 0.08 0.82
```

## 12.3 SUMMARY: SUMMARY OPERATIONS

```
> max(tAB)
[1] 0.9944
> range(tAB)
[1] 0.029594 0.994398
```

# CHAPTER 13

# SUBSETTING OPERATIONS

indextimeSeries!subsetting

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

## 13.1 GET THE FIRST AND LAST RECORDS OF TIME SERIES OBJECTS

Subsetting a `timeSeries` is a very important issue in the management of financial time seeries. Rmetrics offers several functions which are useful in this context: These include among others `"["` which extracts or replaces subsets from `timeSeries` objects, and `window` which extracts a piece from a 'timeDate' object. In this context it is also important to know the functions `start` and the `end` which extract the first and last time stamp. Rmetrics has several functions to extract the first and last records of a `timeSeries` object. These include the functions `start()`, `end()`, and `range()`, as well as the functions `head()` and `tail()`.

### *Using the functions start and end*

To extract the first and the last record of a `timeSeries` object we can use the functions `start` and `end`. The function `start()` sorts the `timeSeries` object in increasing time order and returns the first element of the sorted vector

```
> tBiv <- timeSeries(data = matrix(round(rnorm(24), 3), ncol = 2),
     charvec = timeCalendar(2009))
> colnames(tBiv) <- c("BIV1", "BIV2")
> tBiv
```

```
GMT
              BIV1   BIV2
2009-01-01  0.569 -1.096
2009-02-01 -0.060 -0.158
2009-03-01  2.503 -0.062
2009-04-01 -1.865 -1.031
2009-05-01  1.316  1.019
2009-06-01 -0.423 -2.116
2009-07-01  0.235 -0.188
2009-08-01  1.055 -0.013
2009-09-01  0.191 -0.439
2009-10-01  1.168  0.179
2009-11-01  1.121  0.619
2009-12-01  0.451  0.693
```

```
> tUni <- tBiv[, 1]
> colnames(tUni) <- "UNI"
> tUni
```

```
GMT
               UNI
2009-01-01  0.569
2009-02-01 -0.060
2009-03-01  2.503
2009-04-01 -1.865
2009-05-01  1.316
2009-06-01 -0.423
2009-07-01  0.235
2009-08-01  1.055
2009-09-01  0.191
2009-10-01  1.168
2009-11-01  1.121
2009-12-01  0.451
```

```
> start(tUni)
```

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

```
> start(rev(time(tUni)))
```

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

To demonstrate this we have reverted the time sequence `tUni` and calculated the starting value again. Similarly, the function `end` returns the latest record in the `timeSeries` vector, i.e.

```
> tUni[end(tUni), ]
```

```
GMT
              UNI
2009-12-01 0.451
```

Here and in the following tUni is an univariate and tBiv a bivariate (dummy) time series object.

### *The function range*

Finally, we can express the time interval as[1]

```
> c(start(tUni), end(tUni))

GMT
[1] [2009-01-01] [2009-12-01]

> range(time(tUni))

GMT
[1] [2009-01-01] [2009-12-01]
```

And we get the timeSeries records from

```
> tUni[c(start(tUni), end(tUni)), ]

GMT
             UNI
2009-01-01 0.569
2009-12-01 0.451
```

or alternative from

```
> tUni[range(time(tUni)), ]

GMT
             UNI
2009-01-01 0.569
2009-12-01 0.451
```

Note, the following command only returns the time series values without time stamps

```
> range(tUni)

[1] -1.865  2.503
```

### *head: Extracting the first few lines of a Time Series*

We can use the functions `head()` and `tail()` to print the first or last few records of a `timeSeries` object. Note the printed default length are 6 records.

```
> head(tBiv)

GMT
             BIV1   BIV2
2009-01-01  0.569 -1.096
2009-02-01 -0.060 -0.158
2009-03-01  2.503 -0.062
2009-04-01 -1.865 -1.031
2009-05-01  1.316  1.019
2009-06-01 -0.423 -2.116
```

[1] Note, the function `range` operates on the data part and not on the time stamp positions. `range` returns the smallest and largest entries in the data matrix of the time series.

```
> tail(tBiv, 3)

GMT
             BIV1  BIV2
2009-10-01 1.168 0.179
2009-11-01 1.121 0.619
2009-12-01 0.451 0.693
```

## 13.2 EXTRACT ELEMENTS BY INTEGER SUBSETTING

Subsetting by counts allows to extract desired records from the rows, and desired instruments from the columns of the data series matrix. The first example shows how to subset a univariate or multivariate `timeSeries()` by row, here the second to the fifth row

```
> tUni[2:5, ]

GMT
              UNI
2009-02-01 -0.060
2009-03-01  2.503
2009-04-01 -1.865
2009-05-01  1.316

> tBiv[2:5, ]

GMT
             BIV1   BIV2
2009-02-01 -0.060 -0.158
2009-03-01  2.503 -0.062
2009-04-01 -1.865 -1.031
2009-05-01  1.316  1.019
```

Note, single indexing of the expressions

```
> tUni[2:5]

[1] -0.060  2.503 -1.865  1.316

> class(tUni[2:5])

[1] "numeric"

> tBiv[2:5]

[1] -0.060  2.503 -1.865  1.316

> class(tBiv[2:5])

[1] "numeric"
```

returns a numeric vector and not a `timeSeries` object!
If we like to operate only an the second column, we have to index it explicitely. For all rows type

```
> tBiv[, 2]
```

```
GMT
             BIV2
2009-01-01 -1.096
2009-02-01 -0.158
2009-03-01 -0.062
2009-04-01 -1.031
2009-05-01  1.019
2009-06-01 -2.116
2009-07-01 -0.188
2009-08-01 -0.013
2009-09-01 -0.439
2009-10-01  0.179
2009-11-01  0.619
2009-12-01  0.693
```

and for rows number 2 to 5 type.

```
> tBiv[2:5, 2]
GMT
             BIV2
2009-02-01 -0.158
2009-03-01 -0.062
2009-04-01 -1.031
2009-05-01  1.019
```

Note the index can be in any order

```
> tBiv[c(5, 2, 4), 2]
GMT
             BIV2
2009-05-01  1.019
2009-02-01 -0.158
2009-04-01 -1.031
```

Negative indexes remove the records or columns

```
> tBiv[-c(3, 5), ]
GMT
             BIV1   BIV2
2009-01-01  0.569 -1.096
2009-02-01 -0.060 -0.158
2009-04-01 -1.865 -1.031
2009-06-01 -0.423 -2.116
2009-07-01  0.235 -0.188
2009-08-01  1.055 -0.013
2009-09-01  0.191 -0.439
2009-10-01  1.168  0.179
2009-11-01  1.121  0.619
2009-12-01  0.451  0.693
```

```
> tBiv[, -2]
GMT
             BIV1
2009-01-01  0.569
2009-02-01 -0.060
```

```
2009-03-01  2.503
2009-04-01 -1.865
2009-05-01  1.316
2009-06-01 -0.423
2009-07-01  0.235
2009-08-01  1.055
2009-09-01  0.191
2009-10-01  1.168
2009-11-01  1.121
2009-12-01  0.451
```

## 13.3 EXTRACT ELEMENTS BY COLUMN NAMES

Instead of using counts, like for example the 4th-column, we can reference and extract columns by column names, which are usually the names of the financial instruments

```
> colnames(MSFT)
[1] "Open"   "High"   "Low"    "Close"  "Volume"
> MSFT[2:5, c("Open", "Close")]
GMT
             Open  Close
2000-09-28 60.812 61.312
2000-09-29 61.000 60.312
2000-10-02 60.500 59.125
2000-10-03 59.562 56.562
```

### *Extract Elements by Logical Predicates*

Logical subsetting can be used for example to subset time frames and windows according to a given time schedule.

```
> tBiv[time(tBiv) > time(tBiv)[6], ]
GMT
            BIV1   BIV2
2009-07-01 0.235 -0.188
2009-08-01 1.055 -0.013
2009-09-01 0.191 -0.439
2009-10-01 1.168  0.179
2009-11-01 1.121  0.619
2009-12-01 0.451  0.693
> tBiv[time(tBiv) > time(tBiv)[6] & time(tBiv) < time(tBiv)[10],
     ]
GMT
            BIV1   BIV2
2009-07-01 0.235 -0.188
2009-08-01 1.055 -0.013
2009-09-01 0.191 -0.439
```

### *Extract Elements by Python Like Indexing*

Subsetting via "[" methods offers also the ability to specify series by time ranges, if they are enclosed in quotes. The style borrows from python[2] by creating ranges with a double colon "::" operator. Each side of the operator may be left blank, which would then default to the beginning and end of the data, respectively. To specify a subset of times, it is only required that the time specified be in the human readible form of the standard ISO format. The time must be 'left-filled', that is to specify a full year one needs only to provide the year, a month would require the full year and the integer of the month requested - e.g. "1999-01". This format would extend all the way down to seconds - e.g. "1999-01-01 08:35:23". Leading zeros are not necessary.

```
> Index <- time(tUni)["2009-07::"]
> Index

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

> tUni[Index, ]

GMT
             UNI
2009-07-01 0.235
2009-08-01 1.055
2009-09-01 0.191
2009-10-01 1.168
2009-11-01 1.121
2009-12-01 0.451
```

### *Extract Elements by Span Indexes*

We can also subset defining time spans, the origin of the span, and the direction of the span.

```
> Index <- time(tUni)["last 3 months"]
> tUni[Index, ]

GMT
             UNI
2009-09-01 0.191
2009-10-01 1.168
2009-11-01 1.121
2009-12-01 0.451


> Index <- time(tUni)["last 5 days"]
> tUni[Index, ]
```

[2]This type of subsetting was introduced by Jeff Ryan and used in his "xts" and "quantmod" packages".

```
GMT
             UNI
2009-12-01 0.451
```

*Extract Elements by 'timeDate' Indexes*

Subsetting by date vectors, allows to extract desired records from the rows for a specified date or dates. We show first an example for the univariate case where we extract two specific dates,
Extract two specific dates:

```
> Index <- time(tUni)[c(3, 7)]
> Index

GMT
[1] [2009-03-01] [2009-07-01]

> tUni[Index, ]

GMT
             UNI
2009-03-01 2.503
2009-07-01 0.235
```

The same works for multivariate `timeSeries` objects. The last and first record in time can be subsetted using the functions `start()` and `end()`.

## 13.4 EXTRACT ELEMENTS USING THE 'WINDOW' FUNCTION

The function `window()` allows to extract all records in the window spanned by `from` and `end`. The example extracts all records from the second quarter

```
> window(tBiv, start = "2009-02-01", end = "2009-05-31")

GMT
             BIV1   BIV2
2009-02-01 -0.060 -0.158
2009-03-01  2.503 -0.062
2009-04-01 -1.865 -1.031
2009-05-01  1.316  1.019
```

The next example shows how to subset records starting at the beginning of the second quarter ranging to the end of the data set:

```
> window(tBiv, start = "2009-04-01", end = format(end(tBiv)))

GMT
             BIV1   BIV2
2009-04-01 -1.865 -1.031
2009-05-01  1.316  1.019
2009-06-01 -0.423 -2.116
2009-07-01  0.235 -0.188
2009-08-01  1.055 -0.013
2009-09-01  0.191 -0.439
2009-10-01  1.168  0.179
```

```
2009-11-01  1.121  0.619
2009-12-01  0.451  0.693
```

## 13.5 EXTRACT WEEKDAYS AND BUSINESS DAYS

To extract weekdays, weekends, business days and holidays from a time series we can subset such days by logical predication using the timeDate functions `isWeekday()`, `isWeekend()`, `isBizday()` and `isHoliday()`.
As an example let us consider a simulated series in April 2001 with random data records.

```
> aprilSeries <- timeSeries(data = rnorm(30), charvec = timeCalendar(2001,
     4, 1:30), finCenter = "Zurich")
> aprilSeries
```

```
GMT
                TS.1
2001-04-01 -0.502193
2001-04-02 -0.583467
2001-04-03  2.585839
2001-04-04  1.059601
2001-04-05 -1.286702
2001-04-06  1.308167
2001-04-07 -0.944777
2001-04-08  0.075931
2001-04-09 -1.033972
2001-04-10 -0.760692
2001-04-11  1.172259
2001-04-12 -0.457750
2001-04-13 -0.676208
2001-04-14  0.336187
2001-04-15 -1.152873
2001-04-16 -0.090942
2001-04-17 -0.869590
2001-04-18 -0.746242
2001-04-19 -1.470557
2001-04-20 -0.058197
2001-04-21  0.441804
2001-04-22  1.446972
2001-04-23  0.072939
2001-04-24 -0.565400
2001-04-25 -0.516987
2001-04-26  1.611400
2001-04-27 -0.526394
2001-04-28  0.214672
2001-04-29 -0.434008
2001-04-30  0.209457
```

```
> nrow(aprilSeries)
```

```
[1] 30
```

The date of Good Friday 2001 was

```
> GoodFriday(2001)

GMT
[1] [2001-04-13]
```

Then extract the weekdays

```
> aprilDates <- time(aprilSeries)
> aprilDates

GMT
 [1] [2001-04-01] [2001-04-02] [2001-04-03] [2001-04-04] [2001-04-05]
 [6] [2001-04-06] [2001-04-07] [2001-04-08] [2001-04-09] [2001-04-10]
[11] [2001-04-11] [2001-04-12] [2001-04-13] [2001-04-14] [2001-04-15]
[16] [2001-04-16] [2001-04-17] [2001-04-18] [2001-04-19] [2001-04-20]
[21] [2001-04-21] [2001-04-22] [2001-04-23] [2001-04-24] [2001-04-25]
[26] [2001-04-26] [2001-04-27] [2001-04-28] [2001-04-29] [2001-04-30]

> weekdaySeries <- aprilSeries[isWeekday(aprilDates), ]
> nrow(weekdaySeries)

[1] 21

> weekdaySeries

GMT
                TS.1
2001-04-02 -0.583467
2001-04-03  2.585839
2001-04-04  1.059601
2001-04-05 -1.286702
2001-04-06  1.308167
2001-04-09 -1.033972
2001-04-10 -0.760692
2001-04-11  1.172259
2001-04-12 -0.457750
2001-04-13 -0.676208
2001-04-16 -0.090942
2001-04-17 -0.869590
2001-04-18 -0.746242
2001-04-19 -1.470557
2001-04-20 -0.058197
2001-04-23  0.072939
2001-04-24 -0.565400
2001-04-25 -0.516987
2001-04-26  1.611400
2001-04-27 -0.526394
2001-04-30  0.209457
```

and the weekends

```
> weekendSeries <- aprilSeries[isWeekend(aprilDates), ]
> nrow(weekendSeries)

[1] 9

> weekendSeries
```

```
GMT
                TS.1
2001-04-01 -0.502193
2001-04-07 -0.944777
2001-04-08  0.075931
2001-04-14  0.336187
2001-04-15 -1.152873
2001-04-21  0.441804
2001-04-22  1.446972
2001-04-28  0.214672
2001-04-29 -0.434008
```

Since in 2001 Easter was celebrated in April, we have also working days falling on holidays

```
> Easter(2001)

GMT
[1] [2001-04-15]

> holidayZURICH(2001)

Zurich
 [1] [2001-01-01] [2001-01-02] [2001-04-13] [2001-04-16] [2001-04-16]
 [6] [2001-05-01] [2001-05-24] [2001-06-04] [2001-08-01] [2001-09-10]
[11] [2001-12-25] [2001-12-26]
```

To extract the series for business days (valid in Zurich) we type

```
> bizdaySeries <- aprilSeries[isBizday(aprilDates), ]
> nrow(bizdaySeries)

[1] 21

> bizdaySeries

GMT
                TS.1
2001-04-02 -0.583467
2001-04-03  2.585839
2001-04-04  1.059601
2001-04-05 -1.286702
2001-04-06  1.308167
2001-04-09 -1.033972
2001-04-10 -0.760692
2001-04-11  1.172259
2001-04-12 -0.457750
2001-04-13 -0.676208
2001-04-16 -0.090942
2001-04-17 -0.869590
2001-04-18 -0.746242
2001-04-19 -1.470557
2001-04-20 -0.058197
2001-04-23  0.072939
2001-04-24 -0.565400
2001-04-25 -0.516987
2001-04-26  1.611400
2001-04-27 -0.526394
2001-04-30  0.209457
```

and to retrieve the records for the holidays in April we type

```
> holidaySeries <- aprilSeries[isHoliday(aprilDates), ]
> nrow(holidaySeries)
```

```
[1] 9
```

```
> holidaySeries
```

```
GMT
                TS.1
2001-04-01 -0.502193
2001-04-07 -0.944777
2001-04-08  0.075931
2001-04-14  0.336187
2001-04-15 -1.152873
2001-04-21  0.441804
2001-04-22  1.446972
2001-04-28  0.214672
2001-04-29 -0.434008
```

Note, here the holidays include free days on Saturdays and Sundays.

## 13.6 EXTRACT DATA RECORDS WITH MISSING VALUES

The function `is.na` Checks if an univariate `timeSeries` object has NA entries. Let us create a dummy timeSeries with two NA entries in the second column

```
> tNA <- tBiv
> tNA[2:3, 2] <- NA
> tNA
```

```
GMT
             BIV1   BIV2
2009-01-01  0.569 -1.096
2009-02-01 -0.060     NA
2009-03-01  2.503     NA
2009-04-01 -1.865 -1.031
2009-05-01  1.316  1.019
2009-06-01 -0.423 -2.116
2009-07-01  0.235 -0.188
2009-08-01  1.055 -0.013
2009-09-01  0.191 -0.439
2009-10-01  1.168  0.179
2009-11-01  1.121  0.619
2009-12-01  0.451  0.693
```

*Searching for NA data values in a time series*

The function `is.na()` returns the present and missing values in a logical time series by TRUE and FALSE Values

```
> is.na(tNA)
```

```
GMT
            BIV1  BIV2
2009-01-01 FALSE FALSE
2009-02-01 FALSE  TRUE
2009-03-01 FALSE  TRUE
2009-04-01 FALSE FALSE
2009-05-01 FALSE FALSE
2009-06-01 FALSE FALSE
2009-07-01 FALSE FALSE
2009-08-01 FALSE FALSE
2009-09-01 FALSE FALSE
2009-10-01 FALSE FALSE
2009-11-01 FALSE FALSE
2009-12-01 FALSE FALSE
```

*Extracting NA records from a time series*

```
> tNA2 <- tNA[is.na(tNA[, 2]), 2]
> tNA2
GMT
           BIV2
2009-02-01   NA
2009-03-01   NA
```

The dates of the NA records are

```
> time(tNA2)
GMT
[1] [2009-02-01] [2009-03-01]
```

and the total number of NA records is

```
> nrow(tNA2)
[1] 2
```

*Omitting NA records from the series*

Rmetrics offers several possibilities to handle NAs in a `timeSeries` object. We can remove NAs from the series

```
> na.omit(tNA)
GMT
             BIV1   BIV2
2009-01-01  0.569 -1.096
2009-04-01 -1.865 -1.031
2009-05-01  1.316  1.019
2009-06-01 -0.423 -2.116
2009-07-01  0.235 -0.188
2009-08-01  1.055 -0.013
2009-09-01  0.191 -0.439
2009-10-01  1.168  0.179
```

```
2009-11-01  1.121  0.619
2009-12-01  0.451  0.693
```

or we can replace them by zeroes, which is a very useful option when we are dealing with financial time series returns,

```
> na.omit(tNA, method = "z")
GMT
             BIV1   BIV2
2009-01-01  0.569 -1.096
2009-02-01 -0.060  0.000
2009-03-01  2.503  0.000
2009-04-01 -1.865 -1.031
2009-05-01  1.316  1.019
2009-06-01 -0.423 -2.116
2009-07-01  0.235 -0.188
2009-08-01  1.055 -0.013
2009-09-01  0.191 -0.439
2009-10-01  1.168  0.179
2009-11-01  1.121  0.619
2009-12-01  0.451  0.693
```

and we can substitute them. Substitution of NAs is done by interpolation and/or extrapolation of data records having NAs. Note, interpolation fails for starting and ending NAs at the head and tail of the series. The input argument for the method, a two character value, allows to select then a proper choice. `method="ir"` interpolates and trims the series, i.e. it deletes starting and ending NAs, `method="iz"` interpolates and replaces the starting and ending NAs with zeros, `method="ie"` interpolates and extrapolates the starting NAs by carrying the first non NA value backward and extrapolates ending NAs by carrying the last nonNA value forward.

```
> na.omit(tNA, method = "ir")
GMT
             BIV1   BIV2
2009-01-01  0.569 -1.096
2009-02-01 -0.060 -1.096
2009-03-01  2.503 -1.096
2009-04-01 -1.865 -1.031
2009-05-01  1.316  1.019
2009-06-01 -0.423 -2.116
2009-07-01  0.235 -0.188
2009-08-01  1.055 -0.013
2009-09-01  0.191 -0.439
2009-10-01  1.168  0.179
2009-11-01  1.121  0.619
2009-12-01  0.451  0.693
```

# CHAPTER 14

# COERCIONS AND TRANSFORMATIONS

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

## 14.1 COERCION TO VECTOR, MATRIX, AND DATA FRAMES

Rmetrics offers furthermore many additional advanced `timeSeries` operations. These include functions for column statistics, for the alignment, aggregation and lagging of `timeSeries` objects, and furthermore for rolling window applications.
Common to most of these functions is the quite general function `fapply` which allows to apply functions to arbitrary date/time periods and subsets of `timeSeries` objects.
Inside Rmetrics all functions which require as input time series information work with "timeSeries" objects. However, many of the R functions in R's base, stats, and graphics packages which are also relevant for financial market analysis work with vector, matrix, or data.frame objects. If the time stamps are not relevant then we can simply coerce the "timeSeries" objects to a vector, matrix or dataframe. Here are some examples:
Compute daily percentual log returns and treat them as a numeric vector,

```
> x <- 100 * diff(log(as.vector(MSFT[, "Open"])))
> head(x)
[1] -4.22598  0.30785 -0.82305 -1.56172 -5.50004 -1.56428
```

then create an autocorrelation function plot, a histogram chart, a density plot, and a quantile-quantile plot using R's base functions with default settings

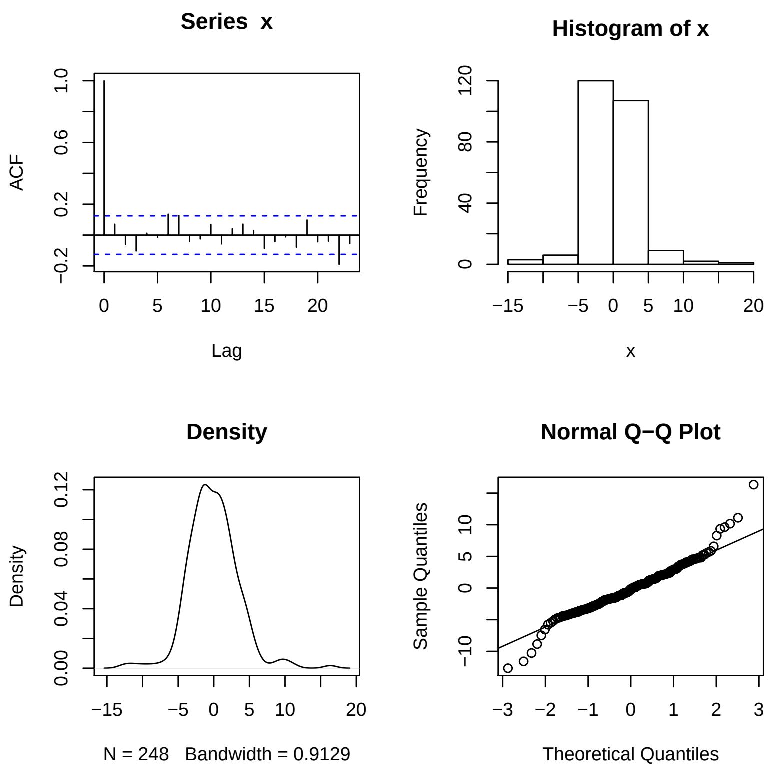

FIGURE 14.1: Statistical Plots.

```
> par(mfrow = c(2, 2))
> acf(x)
> hist(x)
> plot(density(x), main = "Density")
> qqnorm(x)
> qqline(x)
> par(mfrow = c(1, 1))
```

As an second example we consider a multivariate series, and compute the covariance matrix, and create a pairs plot

```
> X <- 100 * diff(log(as.matrix(MSFT[, c("High", "Low", "Volume")])))
> cov(X)
```

```
          High      Low  Volume
High    8.5662   8.0073   10.49
Low     8.0073  10.5431  -15.29
Volume 10.4895 -15.2905 1201.24
```

```
> pairs(X)
```

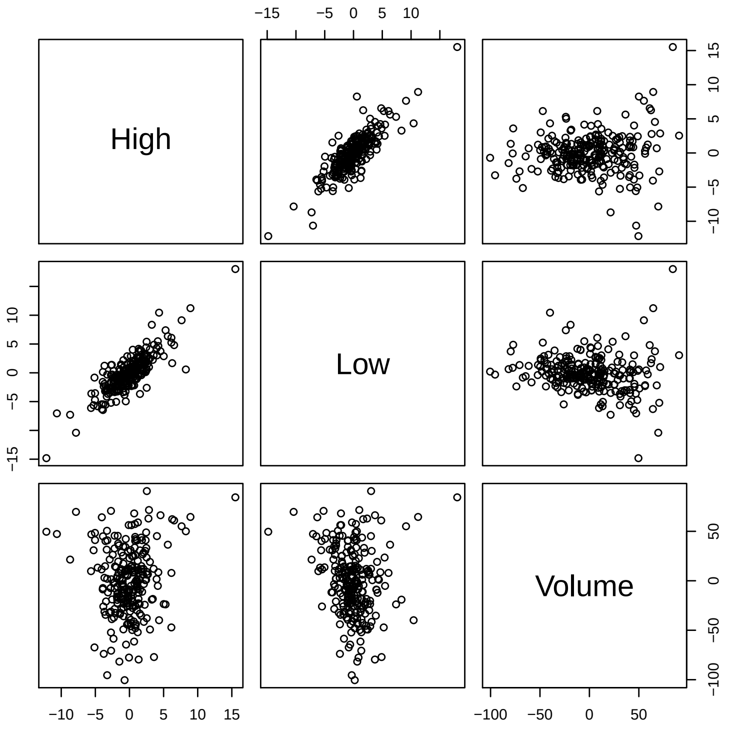

FIGURE 14.2: Pairs Plot.

## 14.2 COLUMN STATISTICS OF 'TIMESERIES' OBJECTS

Rmetrics has implemented several functions to compute column and row statistics of univariate and multivariate `timeSeries` objects. The functions return a numeric vector of the same length as the number of columns of the `timeSeries`. Functions include

```
colStats           calculates arbitrary column statistics,

colSums            returns column sums,
colMeans           returns column means,
colSds             returns column standard deviations,
colVars            returns column variances,
colSkewness        returns column skewness,
colKurtosis        returns column kurtosis,
colMaxs            returns maximum values in each column,
colMins            returns minimum values in each column,
colProds           returns product of all values in each column,
colQuantiles       returns quantiles of each column.
```

```
> print(head(MSFT))
GMT
             Open   High    Low  Close   Volume
2000-09-27 63.438 63.562 59.812 60.625 53077800
2000-09-28 60.812 61.875 60.625 61.312 26180200
2000-09-29 61.000 61.312 58.625 60.312 37026800
2000-10-02 60.500 60.812 58.250 59.125 29281200
2000-10-03 59.562 59.812 56.500 56.562 42687000
2000-10-04 56.375 56.562 54.500 55.438 68226700
```

```
> colMeans(MSFT)
      Open       High        Low      Close     Volume
6.1646e+01 6.2937e+01 6.0447e+01 6.1643e+01 4.3144e+07
```

```
> colQuantiles(MSFT)
      Open       High        Low      Close     Volume
4.8675e+01 5.0200e+01 4.7200e+01 4.9013e+01 2.2598e+07
```

You can also define your own stats functions and let it execute by the function colStats. If we like to know for example the column medians of the timeSeries, we can simply write

```
> colStats(MSFT, FUN = "median")
      Open       High        Low      Close     Volume
6.1900e+01 6.3360e+01 6.0688e+01 6.1880e+01 4.0638e+07
```

## 14.3 CUMULATED COLUMN STATISTICS OF `timeSeries` OBJECTS

Functions to compute cumulated column statistics are

| | |
|---|---|
| `colCumstats` | `Cumulated statistics` |
| `colCumsums` | `returns column cumulated sums` |
| `colCummaxs` | `returns column cumulated maximums` |
| `colCummins` | `returns column cumulated minimums` |
| `colCumprods` | `returns column cumulated products` |
| `colCumreturns` | `returns column cumulated returns` |

## 14.4 ROLLING STATISTICS OF 'TIMESERIES' OBJECTS

Rolling statistics is a very often applied method to look upon the movement and changement over time of a time series value. The rolling windows are determined by two time measures, the "period" which measures the length or size of the window, and a "by" shift, which moves the window in time. The function rollStats() only offers to define the period character (k).

```
> rS <- rollStats(MSFT, k = 12, by = "monthly", FUN = mean, na.pad = TRUE,
     align = "right")
> head(rS, 24)
```

```
GMT
           Open_RSTATS High_RSTATS Low_RSTATS Close_RSTATS Volume_RSTATS
2000-09-27          NA          NA         NA           NA            NA
2000-09-28          NA          NA         NA           NA            NA
2000-09-29          NA          NA         NA           NA            NA
2000-10-02          NA          NA         NA           NA            NA
2000-10-03          NA          NA         NA           NA            NA
2000-10-04          NA          NA         NA           NA            NA
2000-10-05          NA          NA         NA           NA            NA
2000-10-06          NA          NA         NA           NA            NA
2000-10-09          NA          NA         NA           NA            NA
2000-10-10          NA          NA         NA           NA            NA
2000-10-11          NA          NA         NA           NA            NA
2000-10-12      57.740      58.589     56.078       56.932      40319500
2000-10-13      56.943      57.865     55.438       56.359      40251400
2000-10-16      56.333      57.193     54.516       55.448      43059675
2000-10-17      55.573      56.453     53.818       54.625      43360633
2000-10-18      54.667      55.823     53.000       54.010      45526217
2000-10-19      54.573      56.021     53.125       54.453      52677017
2000-10-20      54.984      56.818     53.677       55.266      53673900
2000-10-23      55.745      57.568     54.130       55.828      58010192
2000-10-24      56.312      58.083     54.583       56.323      59369917
2000-10-25      56.839      58.724     55.203       56.911      63923258
2000-10-26      57.427      59.516     55.786       57.734      66121608
2000-10-27      58.318      60.536     56.672       58.729      67083550
2000-10-30      59.250      61.641     57.802       59.953      67910133
```

The function fapply() is much more general. The functions allows time varying windows, i.e. width window "periods" and "by" shifts over time. A window can be arbitrarily placed on a timeSeries object specifying its start and end postions in time by the arguments "from" and "to", which are two vectors of "timeDate" objects.

```
> oneDay <- 24 * 3600
> from <- unique(timeFirstDayInQuarter(time(MSFT))) - oneDay
> from <- timeLastNdayInMonth(from, nday = 5)
> to <- unique(timeLastDayInQuarter(time(MSFT)))
> to <- timeLastNdayInMonth(to, nday = 5)
> data.frame(from = as.character(from), to = as.character(to))
```

```
        from         to
1 2000-06-30 2000-10-06
2 2000-10-06 2001-01-05
3 2001-01-05 2001-04-06
4 2001-04-06 2001-07-06
5 2001-07-06 2001-10-05
```

```
> fapply(MSFT, from, to, FUN = colMeans)
```

```
GMT
             Open   High    Low  Close   Volume
2000-10-06 59.125 59.742 57.289 58.039 40990800
2001-01-05 58.896 60.520 57.452 58.808 52160668
2001-04-06 56.998 58.436 55.814 57.162 43649597
2001-07-06 68.485 69.619 67.521 68.619 40732749
2001-10-05 62.419 63.398 61.286 62.280 35029004
```

# Part IV

# Base, Stats and Financial Functions

# Chapter 15

# Base Functions and Methods

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

In this chapter we present functions for time series objects which we now from R's base package for vectors, matrices, and/or data.frames.

LISTING 15.1: TIMESERIES FUNCTIONS KNOWN FROM R'S BASE PACKAGE.

```
Function:
apply           applying functions over time series margins
attach          attaching a time series to the search path
cbind           binding time series together
diff            differencing and lagging a time series
dim             getting the size of a time series
length          returning the length of a time series
merge           merging tow time series together
min             getting the extremes from a time series
range           returning the range of a time series
rev             reversing the time order of a series
sample          sampling the time stamps of a time series
scale           scaling a time series
sort            sorting a time series
```

## 15.1 `apply()` - Applying Functions Over Time Series Margins

The function `apply()` returns a vector or array or list of values obtained by applying a function to margins of a `timeSeries` object.

*Operating row by row*

Now we apply the function on each row of an univariate series:

```
> tUni <- timeSeries(data = rnorm(12), charvec = timeCalendar(2008),
     units = "UNI")
> apply(tUni, MARGIN = 1, abs)
```

```
GMT
               TS.1
2008-01-01 2.275557
2008-02-01 0.200533
2008-03-01 0.390188
2008-04-01 0.781777
2008-05-01 0.036102
2008-06-01 1.513528
2008-07-01 0.545697
2008-08-01 0.443483
2008-09-01 1.297703
2008-10-01 1.115054
2008-11-01 0.609536
2008-12-01 0.599379
```

```
> apply(tUni, MARGIN = 1, sum)
```

```
GMT
                TS.1
2008-01-01 -2.275557
2008-02-01  0.200533
2008-03-01 -0.390188
2008-04-01 -0.781777
2008-05-01  0.036102
2008-06-01 -1.513528
2008-07-01  0.545697
2008-08-01 -0.443483
2008-09-01 -1.297703
2008-10-01 -1.115054
2008-11-01 -0.609536
2008-12-01 -0.599379
```

```
> apply(tUni, MARGIN = 1, cumsum)
```

```
GMT
                TS.1
2008-01-01 -2.275557
2008-02-01  0.200533
2008-03-01 -0.390188
2008-04-01 -0.781777
2008-05-01  0.036102
2008-06-01 -1.513528
2008-07-01  0.545697
2008-08-01 -0.443483
2008-09-01 -1.297703
2008-10-01 -1.115054
2008-11-01 -0.609536
2008-12-01 -0.599379
```

```
> tBiv <- timeSeries(data = matrix(rnorm(24), ncol = 2), charvec = timeCalendar(2008),
     units = c("BIV1", "BIV2"))
> apply(tBiv, MARGIN = 1, abs)
```

```
        [,1]    [,2]    [,3]    [,4]    [,5]     [,6]    [,7]    [,8]    [,9]
BIV1 0.31969 0.30025 0.94037 0.96073 0.88904 0.824891 0.51711 0.30291 0.27987
BIV2 1.06050 0.30477 0.59826 0.34123 0.82701 0.026822 0.15096 0.81056 0.72537
       [,10]  [,11]   [,12]
BIV1 2.39251 1.0387 0.88848
BIV2 0.87718 1.1139 0.92440
```

```
> apply(tBiv, MARGIN = 1, sum)
```

```
GMT
                 TS.1
2008-01-01 -1.3801954
2008-02-01  0.0045206
2008-03-01  1.5386302
2008-04-01 -0.6194929
2008-05-01 -1.7160498
2008-06-01  0.7980686
2008-07-01  0.3661586
2008-08-01 -0.5076487
2008-09-01 -1.0052386
2008-10-01 -3.2696965
2008-11-01  0.0752760
2008-12-01 -0.0359191
```

```
> apply(tBiv, MARGIN = 1, cumsum)
```

```
         [,1]       [,2]    [,3]     [,4]     [,5]    [,6]    [,7]     [,8]
BIV1 -0.31969 -0.3002520 0.94037 -0.96073 -0.88904 0.82489 0.51711  0.30291
BIV2 -1.38020  0.0045206 1.53863 -0.61949 -1.71605 0.79807 0.36616 -0.50765
         [,9]   [,10]     [,11]     [,12]
BIV1 -0.27987 -2.3925 -1.038664  0.888480
BIV2 -1.00524 -3.2697  0.075276 -0.035919
```

*Operating column by column*

To apply a function on each column of a multivariate time series we set `MARGIN=2`.

*Operating on both margins*

Note, we can also apply a function on each column and row, then we have set `MARGIN=c(1,2)`

## 15.2 attach() - ATTACHING A TIME SERIES TO THE SEARCH PATH

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.

```
> headMSFT <- head(MSFT)
> attach(headMSFT)
> Diff <- High - Low
> class(Diff)

[1] "numeric"

> Diff

[1] 3.7500 1.2500 2.6875 2.5625 3.3125 2.0625
```

To detach a the series, type

```
> detach(headMSFT)
```

## 15.3 `cbind()` - BINDING TIME SERIES TOGETHER

Rmetrics has four functions to bind time series together. These are with increasing complexity `c()`, `cbind()`, `rbind()`, and `merge()`. The latter function will be considered in its own section about merging a time series. Before we start to interpret the results of binding and merging several `timeSeries` let us consider the following time series examples

```
> set.seed(1953)
> charvec <- format(timeCalendar(2008, sample(12, 6)))
> data <- matrix(round(rnorm(6), 3))
> t1 <- sort(timeSeries(data, charvec, units = "A"))
> t1

GMT
                A
2008-02-01  0.236
2008-05-01  1.484
2008-06-01  0.231
2008-07-01  0.187
2008-10-01 -0.005
2008-11-01  1.099

> charvec <- format(timeCalendar(2008, sample(12, 9)))
> data <- matrix(round(rnorm(9), 3))
> t2 <- sort(timeSeries(data, charvec, units = "B"))
> t2

GMT
                B
2008-01-01 -1.097
2008-03-01 -0.890
2008-04-01 -1.472
2008-05-01 -1.009
2008-06-01  0.983
2008-07-01 -0.068
2008-10-01 -2.300
2008-11-01  1.023
2008-12-01  1.177
```

```
> charvec <- format(timeCalendar(2008, sample(12, 5)))
> data <- matrix(round(rnorm(10), 3), ncol = 2)
> t3 <- sort(timeSeries(data, charvec, units = c("A", "C")))
> t3

GMT
                A      C
2008-02-01  0.620 -0.109
2008-03-01 -1.490  0.796
2008-04-01  0.210 -0.649
2008-05-01  0.654  0.231
2008-06-01 -1.603  0.318
```

The first series t1 and second series t2 are univariate series with 6 and 9 random records and column names "A" and "B", respectively. The third t3 series is a bivariate series with 5 records per column and column names "A" and "C". Note, the first column "A" of the third time series t3 describes the same time series "A" as the first series "t1".

*Concatenation of time series object*

The function c() just concatenates all the data from a timeSeries in a (numeric) vector going along the first column and through all following columns if we have a multivariate time series

```
> c(t1, t2)

 [1]  0.236  1.484  0.231  0.187 -0.005  1.099 -1.097 -0.890 -1.472 -1.009
[11]  0.983 -0.068 -2.300  1.023  1.177
```

Note, this might not what you will expect by concatenation of time series objects.

*Binding column- and/or rowwise a time series*

The functions cbind() and rbind() allow to bind timeSeries objects either column or rowwise. Let us bind series t1 and series t2 columnwise

```
> cbind(t1, t2)

GMT
                A      B
2008-01-01     NA -1.097
2008-02-01  0.236     NA
2008-03-01     NA -0.890
2008-04-01     NA -1.472
2008-05-01  1.484 -1.009
2008-06-01  0.231  0.983
2008-07-01  0.187 -0.068
2008-10-01 -0.005 -2.300
2008-11-01  1.099  1.023
2008-12-01     NA  1.177
```

We obtain a bivariate time series with column names "A" and "B", where the gaps were filled with NAs. Binding series `t1` and `t3` column by column

```
> cbind(t1, t3)
GMT
              A.1    A.2      C
2008-02-01  0.236  0.620 -0.109
2008-03-01     NA -1.490  0.796
2008-04-01     NA  0.210 -0.649
2008-05-01  1.484  0.654  0.231
2008-06-01  0.231 -1.603  0.318
2008-07-01  0.187     NA     NA
2008-10-01 -0.005     NA     NA
2008-11-01  1.099     NA     NA
```

we obtain a new time series with three columns and the names of the two series with identical colmunn names "A" got suffixes ".1" and ".2" to distinguish them from each other.
The function `rbind()` behaves similar, but the number of rows must be the same in all time series to be binded by rows

```
> rbind(t1, t2)
GMT
              A_B
2008-02-01  0.236
2008-05-01  1.484
2008-06-01  0.231
2008-07-01  0.187
2008-10-01 -0.005
2008-11-01  1.099
2008-01-01 -1.097
2008-03-01 -0.890
2008-04-01 -1.472
2008-05-01 -1.009
2008-06-01  0.983
2008-07-01 -0.068
2008-10-01 -2.300
2008-11-01  1.023
2008-12-01  1.177
```

The columnname is now "A_B" to make evident that series named "A" and "B" were bindedt together. Note, binding the univariate series `t1` and the bivariate series `t3` would result in an error.

## 15.4 `diff()` - DIFFERENCING A TIME SERIES

This function `diff()` differences the data records of a `timeSeries` object. The following example extracts the closing records from MSFT stock prices and then combines the results for the prices, for the differences of the log prices and for the returns in a three-column time series.

```
> closeMSFT <- MSFT[, "Close"]
> diffMSFT <- cbind(closeMSFT, diff(log(closeMSFT)), returns(closeMSFT))
> colnames(diffMSFT) <- c("close", "difflog", "returns")
> tail(diffMSFT)
GMT
           close    difflog    returns
2001-09-20 50.76 -0.0594651 -0.0594651
2001-09-21 49.71 -0.0209025 -0.0209025
2001-09-24 52.01  0.0452299  0.0452299
2001-09-25 51.30 -0.0137453 -0.0137453
2001-09-26 50.27 -0.0202823 -0.0202823
2001-09-27 49.96 -0.0061858 -0.0061858
```

We see, that the function `returns()` just calculates the log returns (differences) of the stock prices.
Note, the function `diff()` has two additional arguments, `lag` and `differences`. The argument `lag` is an integer indicating which lag to use, and the argument `diffrences` is another integer indicating the order of the difference.

## 15.5 dim() - GETTING THE SIZE OF A TIME SERIES

*The dimensions*

The function `dim()` returns the dimension of the data matrix, i.e. the number of rows and columns.

```
> dim(MSFT)
[1] 249   5
```

*Column and row dimensions*

Furthermore, the functions `nrow()` and `ncol()` return the number of rows or columns, and the functions `NCOL()` and `NROW()` do the same treating an univariate time series (data vector) as a 1-column multivariate time series (data matrix).

```
> ncol(MSFT)
[1] 5
> nrow(MSFT)
[1] 249
```

## 15.6 length() - RETURNS THE LENGTH OF A TIME SERIES

The result of the function `length()` applied to a time series is the same as applied to its data part. In the case of an univariate time series we get the length of the series

```
> Open <- MSFT[, "Open"]
> length(Open)
[1] 249
> length(series(Open))
[1] 249
```

and in the case of a multivariate time series we get back the total number of time series values

```
> length(MSFT)
[1] 1245
> length(series(MSFT))
[1] 1245
> dim(MSFT)
[1] 249   5
```

## 15.7 `merge()` - MERGING TWO TIME SERIES TOGETHER

Merging two `timeSeries` objects is the most general case for binding two time series. `merge()` will take care of the names of the individual columns. The function combines the two series either univariate or multivariate column and rowwise, and in addition intersects columns with identical column names which is the most important point. To show this we merge the time series `t1`, `t2`, and `t3` created in section on binding time series objects together

```
> t1
GMT
                A
2008-02-01  0.236
2008-05-01  1.484
2008-06-01  0.231
2008-07-01  0.187
2008-10-01 -0.005
2008-11-01  1.099
> t2
GMT
                B
2008-01-01 -1.097
2008-03-01 -0.890
2008-04-01 -1.472
2008-05-01 -1.009
2008-06-01  0.983
2008-07-01 -0.068
2008-10-01 -2.300
2008-11-01  1.023
2008-12-01  1.177
```

```
> t3

GMT
                A      C
2008-02-01  0.620 -0.109
2008-03-01 -1.490  0.796
2008-04-01  0.210 -0.649
2008-05-01  0.654  0.231
2008-06-01 -1.603  0.318
```

We first merge `t1` and `t2`, and then we merge them together with `t3`

```
> tM <- merge(merge(t1, t2), t3)
> tM

GMT
                A      B      C
2008-01-01     NA -1.097     NA
2008-02-01  0.236     NA     NA
2008-02-01  0.620     NA -0.109
2008-03-01 -1.490     NA  0.796
2008-03-01     NA -0.890     NA
2008-04-01  0.210     NA -0.649
2008-04-01     NA -1.472     NA
2008-05-01  0.654     NA  0.231
2008-05-01  1.484 -1.009     NA
2008-06-01 -1.603     NA  0.318
2008-06-01  0.231  0.983     NA
2008-07-01  0.187 -0.068     NA
2008-10-01 -0.005 -2.300     NA
2008-11-01  1.099  1.023     NA
2008-12-01     NA  1.177     NA
```

What we get is a 3-column time series with names "A", "B", and "C". Note the records from time series `t1` and from the first column of time series `t3` both maned "A" were merged together in the same first columnn of the new time series.

## 15.8 min() - GETTING THE EXTREMES OF DATA FROM A TIME SERIES

The functions `min` and `max` return numeric values with the smallest and largest value of the data matrix. As an example let us compute the all time high and low of the MSFT closing prices, type

```
> Close <- MSFT[, "Close"]
> min(Close)

[1] 41.5

> max(Close)

[1] 73.68
```

To obtain the smallest and largest data values in a multivariate `timeSeries` object use the functions `colMins()` and `colMaxs()`, respectively. As an example we consider the dataset with the monthly pension fund benchmark and search the highest loss and biggest monthly gain

```
> colMins(MSFT)
      Open       High        Low      Close     Volume
4.0750e+01 4.4000e+01 4.0312e+01 4.1500e+01 1.3916e+07
> colMaxs(MSFT)
     Open      High       Low     Close    Volume
7.370e+01 7.615e+01 7.208e+01 7.368e+01 1.285e+08
```

These are the all time high and all time low values of the index period where the time series was recorded.

## 15.9 `range()` - Getting the Range of Data from a Time Series

The function `range` returns a numeric vector with two elements the smallest and largest value of the data matrix. To extract the time range of the `timeSeries` object, use the function `range()` together with `time()`

```
> range(MSFT[, "Volume"])/1e+06
[1]  13.916 128.497
```

This is the lowest and highest daily traded Volume measured in Millions.

## 15.10 `rev()` - Reverse Order of Time Stamps of a Time Series

The function `rev()` reverses the order of the times stamps of a time series.

```
> head(rev(MSFT))
GMT
            Open  High   Low Close   Volume
2001-09-27 50.10 50.68 48.00 49.96 40595600
2001-09-26 51.51 51.80 49.55 50.27 29262200
2001-09-25 52.27 53.00 50.16 51.30 42470300
2001-09-24 50.65 52.45 49.87 52.01 42790100
2001-09-21 47.92 50.60 47.50 49.71 92488300
2001-09-20 52.35 52.61 50.67 50.76 58991600
```

If the time stamps of the series increased in time, then after reversing the series, the time stamps are decreasing in time.

## 15.11 `sample()` - Sample the Time Stamps of a Time Series

The function `sample()` takes a sample of the specified size from the time stamps of a time series, either with or without replacement.

```
> sample(MSFT, 5)
```

```
GMT
             Open   High    Low  Close   Volume
2000-10-18 49.625 53.250 48.438 51.750 55268200
2000-10-04 56.375 56.562 54.500 55.438 68226700
2001-02-27 59.375 61.188 58.672 59.375 49574300
2001-02-06 62.062 63.812 61.688 62.562 48221000
2000-10-03 59.562 59.812 56.500 56.562 42687000
```

## 15.12 scale() - SCALES A CENTERS THE VALUES OF A TIME SERIES

The functionscale centers and/or scales the columns of a time series object

```
> Scaled <- scale(MSFT[, "Open"])
> head(Scaled)
GMT
                Open
2000-09-27  0.240197
2000-09-28 -0.111816
2000-09-29 -0.086672
2000-10-02 -0.153722
2000-10-03 -0.279441
2000-10-04 -0.706886
```

To get the scaling parameters, type

```
> attr(Scaled, "scaled:center")
  Open
61.646
> attr(Scaled, "scaled:scale")
  Open
7.4571
```

## 15.13 sort() - SORTING A TIME SERIES

The date and time stamps of `timeSeries` objects can sorted and ordered.

### *Sorting a time series object*

`sort()` or `order()` are functions to arrange a time series into ascending or descending order.

```
> tUni <- timeSeries(data = runif(5), charvec = timeCalendar(2010,
     1, runif(5, 1, 30)))
> Sorted <- sort(tUni)
> Sorted
GMT
              TS.1
2010-01-03 0.59651
2010-01-15 0.59713
```

```
2010-01-19 0.56322
2010-01-19 0.99550
2010-01-20 0.15849
```

```
> Index <- order(tUni)
> Index
[1] 4 1 5 2 3
> Ordered <- tUni[Index, ]
> Ordered
GMT
              TS.1
2010-01-20 0.15849
2010-01-19 0.56322
2010-01-03 0.59651
2010-01-15 0.59713
2010-01-19 0.99550
```

### *Computing the Order Statistics of a Time Series*

The function `orderStatistics()` sorts the record values in increasing order.

```
> orderStatistics(tUni)
$TS.1
GMT
              TS.1
2010-01-20 0.15849
2010-01-19 0.56322
2010-01-03 0.59651
2010-01-15 0.59713
2010-01-19 0.99550
```

# CHAPTER 16

# STATS FUNCTIONS AND METHODS

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

In this chapter we present functions for time series objects which we know from R's stats package for vectors, matrices, data.frames, and/or ts objects.

LISTING 16.1: FUNCTIONS KNOWN FROM R'S STATS PACKAGE.

```
Function:
acf                 Computes Autocorrelations
aggregate           Aggregates a Time Series
arima               Fits an ARIMA Model
cov                 Computes Covariance Matrix
dist                Computes Distance Matrix
dnorm               Computes Normal Density
filter              Applies Linear Filtering
fivenum             Returns Five Number Summary
hist                Computes a Histogram
lag                 Creates Lagged Versions of a timeSeries
lm                  Fits LinearModels
lowess              Smoothes by Polynomial Regression
mad                 Computes Median Absolute Deviation
median              Computes the Sample Median
model.frame         Extracts Environment of aModel Formula
na.omit             Handles NAs in Time Series
qqnorm              Produces a Normal QQ Plot
smooth              Performs Running Median Smoothing
spectrum            Estimates the Sprectral Density
window              Subsets by Windows
```

As an example of a `timeSeries` object we use in the following code snippets the Swiss pension fund benchmark from bank Pictet.

```
> data(LPP2005REC)
> LPP2005REC.TS <- as.timeSeries(LPP2005REC)
> class(LPP2005REC.TS)

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

## 16.1 `acf()` - COMPUTING AUTOCORRELATIONS

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.

```
> ACF <- acf(LPP2005REC.TS[, "ALT"])
> print(ACF)

Autocorrelations of series 'LPP2005REC.TS[, "ALT"]', by lag

     0      1      2      3      4      5      6      7      8      9     10
 1.000  0.028  0.017  0.028 -0.029 -0.036  0.007 -0.096 -0.055  0.037  0.078
    11     12     13     14     15     16     17     18     19     20     21
 0.070  0.025  0.001 -0.013  0.015 -0.034  0.050  0.005  0.009  0.014  0.027
    22     23     24     25
-0.014  0.006 -0.049 -0.003
```

## 16.2 `aggregate()` - AGGREGATING A TIME SERIES

Temporal aggregation of financial time series changes the sampling frequency of the data values. The data are generated in finer time intervals than the sampling interval we use under aggregation, as an example end-of-day data may be aggregated to end-of-week data or end-of-month data, and these data can be further aggregated e.g. to quarterly or semi-annually or annually data. Note, in many cases temporal aggregation implies a loss of information about the underlying data processes, but contrary, it generates a more smoothing process which may be desired and useful in many situations.
More general, the function `aggregate` is in R a generic function which splits the data into subsets, computes summary statistics for each, and returns the result in a convenient form.
Let us consider Pictet's family of Swiss Pension Fund Indices LPP25, LPP40, and LPP60 representing Benchmarks with increasing risk through reducing the weights of bonds and increasing the weights of equities.

```
> c(start(LPP2005REC.TS), end(LPP2005REC.TS))
```

```
GMT
[1] [2005-11-01] [2007-04-11]

> colnames(LPP2005REC.TS)

[1] "SBI"   "SPI"   "SII"   "LMI"   "MPI"   "ALT"   "LPP25" "LPP40" "LPP60"
```

The series is a series of daily log returns of six asset classes consisting of Swiss Bonds SBI, Swiss equities SPI, Swiss Reits SII, foreign Bonds LMI, foreign equities MPI, alternative investments ALT, and the three Benchmark indices LPP25, LPP40, and LPP60. We have multiplied the series with 100 to work with percentual returns. To aggregate the Swiss series to monthly returns we proceed as follows:

```
> dailyReturns <- LPP2005REC.TS[, c("SBI", "SPI", "SII")]
> byEndOfMonth <- timeSequence(start(dailyReturns), end(dailyReturns),
     by = "month")
> byEndOfMonth <- timeLastDayInMonth(byEndOfMonth)
> endOfMonthReturns <- round(aggregate(dailyReturns, by = byEndOfMonth,
     FUN = sum), 2)
> endOfMonthReturns

GMT
             SBI   SPI   SII
2005-11-30  0.00  0.05 -0.02
2005-12-31  0.01  0.02  0.04
2006-01-31  0.00  0.03 -0.01
2006-02-28  0.00  0.01  0.03
2006-03-31 -0.01  0.02 -0.01
2006-04-30 -0.01  0.01  0.01
2006-05-31  0.00 -0.05  0.00
2006-06-30  0.00  0.01 -0.03
2006-07-31  0.01  0.04  0.00
2006-08-31  0.01  0.03  0.00
2006-09-30  0.01  0.03 -0.01
2006-10-31  0.00  0.02  0.00
2006-11-30  0.01 -0.01  0.01
2006-12-31 -0.01  0.04  0.03
2007-01-31  0.00  0.04  0.03
2007-02-28  0.01 -0.04 -0.01
2007-03-31  0.00  0.03  0.02
2007-04-30  0.00  0.02  0.00
```

Note, the time stamps are (as desired) those of the end of each monthly period.
To compute for example for the LPP40 the Open, High, Low and Close prices for each month we first cumulate the returns

```
> dailyLPP40 <- round(100 * cumulated(LPP2005REC.TS[, "LPP40"]/100),
     2)
> head(dailyLPP40)

GMT
            LPP40
2005-11-01 100.00
2005-11-02 100.00
```

```
2005-11-03 100.00
2005-11-04 100.00
2005-11-07 100.01
2005-11-08 100.01
```

Note, here we have indexed the series to 100 at the introduction of the index in November 2005 and rounded the values to two digits. Next we aggregate the series:

```
> High <- aggregate(dailyLPP40, by = byEndOfMonth, FUN = colMaxs,
     units = "High")
> Low <- aggregate(dailyLPP40, by = byEndOfMonth, FUN = colMins,
     units = "Low")
> Open <- aggregate(dailyLPP40, by = byEndOfMonth, FUN = function(x) x[1],
     units = "Open")
> Close <- aggregate(dailyLPP40, by = byEndOfMonth, FUN = function(x) rev(x)[1],
     units = "Close")
> monthlyOHLC <- cbind(Open, High, Low, Close)
> monthlyOHLC
```

```
GMT
             Open   High    Low  Close
2005-11-30 100.00 100.03 100.00 100.02
2005-12-31 100.03 100.04 100.03 100.04
2006-01-31 100.04 100.05 100.04 100.05
2006-02-28 100.05 100.06 100.05 100.06
2006-03-31 100.06 100.06 100.05 100.06
2006-04-30 100.06 100.06 100.05 100.05
2006-05-31 100.05 100.06 100.02 100.03
2006-06-30 100.03 100.03 100.01 100.03
2006-07-31 100.03 100.04 100.02 100.04
2006-08-31 100.04 100.06 100.04 100.06
2006-09-30 100.06 100.08 100.06 100.08
2006-10-31 100.07 100.09 100.07 100.09
2006-11-30 100.09 100.10 100.09 100.09
2006-12-31 100.09 100.11 100.09 100.11
2007-01-31 100.11 100.13 100.11 100.12
2007-02-28 100.13 100.14 100.12 100.12
2007-03-31 100.12 100.13 100.11 100.13
2007-04-30 100.13 100.13 100.13 100.13
```

## 16.3 `arima()` - FITTING AN ARIMA MODEL

The function `arima()` fits an ARIMA model to a univariate time series.

```
> LPP40 <- LPP2005REC.TS[, "LPP40"]
> arima(LPP40)
```

```
Call:
arima(x = LPP40)

Coefficients:
      intercept
              0
```

```
s.e.          0

sigma^2 estimated as 7.88e-06:  log likelihood = 1680.1,  aic = -3356.3
```

## 16.4 cov() - COMPUTING THE COVARIANCE MATRIX

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 timeSeries objects these functions behave in the following way.

```
> var(LPP2005REC.TS[, "SPI"])
           SPI
SPI 5.8461e-05

> var(LPP2005REC.TS[, 1:6])
            SBI         SPI        SII         LMI         MPI         ALT
SBI  1.5900e-06 -1.2741e-06 1.7994e-07  9.8039e-07 -1.5888e-06 -1.3238e-06
SPI -1.2741e-06  5.8461e-05 3.0336e-06 -1.4075e-06  4.1160e-05  2.9840e-05
SII  1.7994e-07  3.0336e-06 8.5138e-06  9.2505e-08  2.4816e-06  1.5549e-06
LMI  9.8039e-07 -1.4075e-06 9.2505e-08  1.4951e-06 -2.3322e-06 -1.7247e-06
MPI -1.5888e-06  4.1160e-05 2.4816e-06 -2.3322e-06  5.3503e-05  3.6481e-05
ALT -1.3238e-06  2.9840e-05 1.5549e-06 -1.7247e-06  3.6481e-05  3.2312e-05

> cov(LPP2005REC.TS[, 1:6])
            SBI         SPI        SII         LMI         MPI         ALT
SBI  1.5900e-06 -1.2741e-06 1.7994e-07  9.8039e-07 -1.5888e-06 -1.3238e-06
SPI -1.2741e-06  5.8461e-05 3.0336e-06 -1.4075e-06  4.1160e-05  2.9840e-05
SII  1.7994e-07  3.0336e-06 8.5138e-06  9.2505e-08  2.4816e-06  1.5549e-06
LMI  9.8039e-07 -1.4075e-06 9.2505e-08  1.4951e-06 -2.3322e-06 -1.7247e-06
MPI -1.5888e-06  4.1160e-05 2.4816e-06 -2.3322e-06  5.3503e-05  3.6481e-05
ALT -1.3238e-06  2.9840e-05 1.5549e-06 -1.7247e-06  3.6481e-05  3.2312e-05

> cor(LPP2005REC.TS[, 1:6])
          SBI      SPI      SII       LMI      MPI       ALT
SBI  1.000000 -0.13216 0.048907  0.635870 -0.17226 -0.184690
SPI -0.132157  1.00000 0.135978 -0.150546  0.73595  0.686554
SII  0.048907  0.13598 1.000000  0.025928  0.11627  0.093746
LMI  0.635870 -0.15055 0.025928  1.000000 -0.26076 -0.248135
MPI -0.172265  0.73595 0.116274 -0.260761  1.00000  0.877377
ALT -0.184690  0.68655 0.093746 -0.248135  0.87738  1.000000
```

## 16.5 dist() - COMPUTING THE DISTANCE MATRIX

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.

```
> dist(t(LPP2005REC.TS[, 1:6]))
```

```
         SBI      SPI      SII      LMI      MPI
SPI 0.154286
SII 0.060705 0.151784
LMI 0.020588 0.154386 0.060881
MPI 0.148463 0.105689 0.146625 0.150137
ALT 0.118405 0.108128 0.119690 0.119379 0.069716
```

## 16.6 `dnorm()` - COMPUTING NORMAL DENSITY

The function `dnorm()` computes the density for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`.

```
> LMI <- LPP2005REC.TS[, "LMI"]
> head(dnorm(LMI, mean = mean(LMI), sd = sd(LMI)))
GMT
               LMI
2005-11-01 207.360
2005-11-02 196.514
2005-11-03 226.010
2005-11-04 192.859
2005-11-07 316.270
2005-11-08  58.081
```

## 16.7 `filter()` - APPLYING LINEAR FILTERING

The function `filter()` applies linear filtering to a univariate time series or to each series separately of a multivariate time series.

```
> LMI <- LPP2005REC.TS[, "LMI"]
> head(filter(LMI, rep(1, 3)))
GMT
                   LMI
2005-11-01          NA
2005-11-02 -0.00327728
2005-11-03 -0.00336692
2005-11-04 -0.00183062
2005-11-07  0.00148888
2005-11-08  0.00064781
```

## 16.8 `fivenum()` - RETURNING FIVE NUMBER SUMMARY

The function `fivenum()` returns Tukey's five number summary (minimum, lower-hinge, median, upper-hinge, maximum) for the input data.

```
> LMI <- LPP2005REC.TS[, "LMI"]
> fivenum(LMI)
[1] -0.00110888 -0.00068646  0.00058803  0.00209235 -0.00009090
```

## 16.9 hist() - Computing a Histogram

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.

```
> LMI <- LPP2005REC.TS[, "LMI"]
> hist(LMI)$density
```

```
[1]   5.3050  34.4828 156.4987 307.6923 281.1671 145.8886  61.0080   7.9576
```

## 16.10 lag() - Creating Lagged Versions of a timeSeries

The function `lag()`

```
> args(lag)
```

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

computes a lagged version of a time series, shifting the time base back or forward by a given number of observations.

```
> SPI <- 100 * LPP2005REC.TS[1:12, "SPI"]
> SWISS <- round(100 * LPP2005REC.TS[1:12, 1:3], 3)
```

### *Lagged Univariate Series*

```
> lag(SPI, k = -1:1)
```

```
GMT
             SPI[-1]    SPI[0]    SPI[1]
2005-11-01  0.251934  0.841460        NA
2005-11-02  1.270729  0.251934  0.841460
2005-11-03 -0.070276  1.270729  0.251934
2005-11-04  0.620523 -0.070276  1.270729
2005-11-07  0.032926  0.620523 -0.070276
2005-11-08 -0.237820  0.032926  0.620523
2005-11-09  0.092209 -0.237820  0.032926
2005-11-10  1.333491  0.092209 -0.237820
2005-11-11 -0.469306  1.333491  0.092209
2005-11-14  0.126687 -0.469306  1.333491
2005-11-15 -0.718750  0.126687 -0.469306
2005-11-16        NA -0.718750  0.126687
```

### *Lagged Multivariate Series*

```
> lag(SWISS, k = 0:1)
```

```
GMT
           SBI[0] SBI[1] SPI[0] SPI[1] SII[0] SII[1]
2005-11-01 -0.061     NA  0.841     NA -0.319     NA
2005-11-02 -0.276 -0.061  0.252  0.841 -0.412 -0.319
2005-11-03 -0.115 -0.276  1.271  0.252 -0.521 -0.412
2005-11-04 -0.324 -0.115 -0.070  1.271 -0.113 -0.521
2005-11-07  0.131 -0.324  0.621 -0.070 -0.180 -0.113
2005-11-08  0.054  0.131  0.033  0.621  0.210 -0.180
2005-11-09 -0.255  0.054 -0.238  0.033 -0.190  0.210
2005-11-10  0.100 -0.255  0.092 -0.238  0.103 -0.190
2005-11-11  0.062  0.100  1.333  0.092  0.046  0.103
2005-11-14  0.069  0.062 -0.469  1.333 -0.087  0.046
2005-11-15  0.015  0.069  0.127 -0.469 -0.607 -0.087
2005-11-16  0.300  0.015 -0.719  0.127  0.021 -0.607
```

## 16.11 `lm()` - FITTING LINEAR MODELS

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).

```
> fit <- lm(LPP40 ~ SPI + SBI + SII, data = LPP2005REC.TS)
> fit

Call:
lm(formula = LPP40 ~ SPI + SBI + SII, data = LPP2005REC.TS)

Coefficients:
(Intercept)          SPI          SBI          SII
   7.69e-05     3.11e-01     2.66e-01     6.33e-02

> head(resid(fit))

 2005-11-01  2005-11-02  2005-11-03  2005-11-04  2005-11-07  2005-11-08
-2.1297e-03 -9.8493e-04 -7.5978e-05  3.4959e-03  3.7618e-06  5.0650e-04
```

Convert the residuals into a `timeSeries` object:

```
> residData <- resid(fit)
> resid.tS <- timeSeries(data = residData, charvec = names(residData),
     units = "RESID")
> head(resid.tS)

GMT
                 RESID
2005-11-01 -2.1297e-03
2005-11-02 -9.8493e-04
2005-11-03 -7.5978e-05
2005-11-04  3.4959e-03
2005-11-07  3.7618e-06
2005-11-08  5.0650e-04
```

## 16.12 lowess() - SMOOTHING BY POLYNOMIAL REGRESSION

The function `lowess()` performs the computations for the LOWESS smoother which uses locally-weighted polynomial regression.

```
> smoothedData <- lowess(LPP2005REC.TS[, "LPP40"])
> class(smoothedData)

[1] "list"
```

The function `lowess()` returns a list where the second element `y` is a numeric vector of the smoothed data. So we have to convert it into an object of class `timeSeries`.

```
> smoothedData.tS <- timeSeries(data = smoothedData$y, charvec = time(LPP2005REC.TS),
     units = "SMOOTHED")
> head(smoothedData.tS)

GMT
                SMOOTHED
2005-11-01 0.00091292
2005-11-02 0.00090673
2005-11-03 0.00090055
2005-11-04 0.00089436
2005-11-07 0.00088821
2005-11-08 0.00088207
```

## 16.13 mad() - COMPUTING MEDIAN ABSOLUTE DEVIATION

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.

```
> SPI <- 100 * LPP2005REC.TS[1:12, "SPI"]
> mad(SPI)

[1] 0.2007
```

The function `mad()` return the value for the median absolute deviation, a numeric value.

## 16.14 median() - COMPUTING THE SAMPLE MEDIAN

The function `median()` computes the sample median.

```
> median(LPP2005REC.TS[, "SII"])

[1] 0.0010624
```

and returns its numeric value.

## 16.15 `na.omit()` - HANDLING NAS IN TIME SERIES

### Create an artificial timeSeries with NAs

```
> X <- matrix(round(rnorm(100), 3), ncol = 5)
> X[1, 3] <- NA
> X[6, 2] <- NA
> X[17, 2:4] <- c(NA, NA, NA)
> X[13:15, 4] <- c(NA, NA, NA)
> X[11:12, 5] <- c(NA, NA)
> X[20, 1] <- NA
> colnames(X) <- LETTERS[1:5]
> Y <- timeSeries(X)
> class(Y)
```

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

```
> Y
```

```
          A      B      C      D      E
 [1,] -0.194 -0.350     NA  0.740 -0.926
 [2,]  0.580  1.782 -1.879  0.699 -1.479
 [3,] -0.889  0.559 -1.854  0.162 -2.218
 [4,] -0.506 -1.194 -0.405  1.014 -0.399
 [5,] -0.636 -2.423  0.309 -1.280  0.456
 [6,]  0.462     NA -0.652 -1.123  0.170
 [7,] -1.550  0.061 -0.764 -1.490  0.647
 [8,] -0.300  0.656 -0.009 -0.979  0.773
 [9,]  0.774 -2.082  0.998 -1.069 -0.608
[10,] -0.338  0.741 -1.619 -1.403 -1.760
[11,] -0.139 -0.371 -0.952  0.020     NA
[12,]  0.248 -1.384  0.871 -0.865     NA
[13,]  0.243 -1.356  1.443     NA -1.121
[14,]  0.092 -1.804  0.099     NA -0.182
[15,]  1.170  0.685  0.026     NA -0.291
[16,] -0.676  0.944 -1.482 -1.077 -0.581
[17,]  0.972     NA     NA     NA  0.032
[18,] -0.193  0.350  0.458 -1.370 -0.761
[19,]  0.960 -0.195 -0.922  0.187 -0.465
[20,]     NA -0.515 -1.005  1.361  0.125
```

### Skip, do nothing

```
> na.omit(Y, method = "s")
          A      B      C      D      E
 [1,] -0.194 -0.350     NA  0.740 -0.926
 [2,]  0.580  1.782 -1.879  0.699 -1.479
 [3,] -0.889  0.559 -1.854  0.162 -2.218
 [4,] -0.506 -1.194 -0.405  1.014 -0.399
 [5,] -0.636 -2.423  0.309 -1.280  0.456
 [6,]  0.462     NA -0.652 -1.123  0.170
 [7,] -1.550  0.061 -0.764 -1.490  0.647
 [8,] -0.300  0.656 -0.009 -0.979  0.773
 [9,]  0.774 -2.082  0.998 -1.069 -0.608
[10,] -0.338  0.741 -1.619 -1.403 -1.760
```

```
[11,] -0.139 -0.371 -0.952  0.020     NA
[12,]  0.248 -1.384  0.871 -0.865     NA
[13,]  0.243 -1.356  1.443     NA -1.121
[14,]  0.092 -1.804  0.099     NA -0.182
[15,]  1.170  0.685  0.026     NA -0.291
[16,] -0.676  0.944 -1.482 -1.077 -0.581
[17,]  0.972     NA     NA     NA  0.032
[18,] -0.193  0.350  0.458 -1.370 -0.761
[19,]  0.960 -0.195 -0.922  0.187 -0.465
[20,]     NA -0.515 -1.005  1.361  0.125
```

Remove rows

```
> na.omit(Y, method = "r")
           A      B      C      D      E
 [1,]  0.580  1.782 -1.879  0.699 -1.479
 [2,] -0.889  0.559 -1.854  0.162 -2.218
 [3,] -0.506 -1.194 -0.405  1.014 -0.399
 [4,] -0.636 -2.423  0.309 -1.280  0.456
 [5,] -1.550  0.061 -0.764 -1.490  0.647
 [6,] -0.300  0.656 -0.009 -0.979  0.773
 [7,]  0.774 -2.082  0.998 -1.069 -0.608
 [8,] -0.338  0.741 -1.619 -1.403 -1.760
 [9,] -0.676  0.944 -1.482 -1.077 -0.581
[10,] -0.193  0.350  0.458 -1.370 -0.761
[11,]  0.960 -0.195 -0.922  0.187 -0.465
```

Substitute NAs with zeros

```
> na.omit(Y, method = "z")
           A      B      C      D      E
 [1,] -0.194 -0.350  0.000  0.740 -0.926
 [2,]  0.580  1.782 -1.879  0.699 -1.479
 [3,] -0.889  0.559 -1.854  0.162 -2.218
 [4,] -0.506 -1.194 -0.405  1.014 -0.399
 [5,] -0.636 -2.423  0.309 -1.280  0.456
 [6,]  0.462  0.000 -0.652 -1.123  0.170
 [7,] -1.550  0.061 -0.764 -1.490  0.647
 [8,] -0.300  0.656 -0.009 -0.979  0.773
 [9,]  0.774 -2.082  0.998 -1.069 -0.608
[10,] -0.338  0.741 -1.619 -1.403 -1.760
[11,] -0.139 -0.371 -0.952  0.020  0.000
[12,]  0.248 -1.384  0.871 -0.865  0.000
[13,]  0.243 -1.356  1.443  0.000 -1.121
[14,]  0.092 -1.804  0.099  0.000 -0.182
[15,]  1.170  0.685  0.026  0.000 -0.291
[16,] -0.676  0.944 -1.482 -1.077 -0.581
[17,]  0.972  0.000  0.000  0.000  0.032
[18,] -0.193  0.350  0.458 -1.370 -0.761
[19,]  0.960 -0.195 -0.922  0.187 -0.465
[20,]  0.000 -0.515 -1.005  1.361  0.125
```

Interpolate NAs and remove NAs at the beginning and end of the series

```
> na.omit(Y, method = "ir")
```

```
           A      B      C      D      E
 [1,]  0.580  1.782 -1.879  0.699 -1.479
 [2,] -0.889  0.559 -1.854  0.162 -2.218
 [3,] -0.506 -1.194 -0.405  1.014 -0.399
 [4,] -0.636 -2.423  0.309 -1.280  0.456
 [5,]  0.462 -2.423 -0.652 -1.123  0.170
 [6,] -1.550  0.061 -0.764 -1.490  0.647
 [7,] -0.300  0.656 -0.009 -0.979  0.773
 [8,]  0.774 -2.082  0.998 -1.069 -0.608
 [9,] -0.338  0.741 -1.619 -1.403 -1.760
[10,] -0.139 -0.371 -0.952  0.020 -1.760
[11,]  0.248 -1.384  0.871 -0.865 -1.760
[12,]  0.243 -1.356  1.443 -0.865 -1.121
[13,]  0.092 -1.804  0.099 -0.865 -0.182
[14,]  1.170  0.685  0.026 -0.865 -0.291
[15,] -0.676  0.944 -1.482 -1.077 -0.581
[16,]  0.972  0.944 -1.482 -1.077  0.032
[17,] -0.193  0.350  0.458 -1.370 -0.761
[18,]  0.960 -0.195 -0.922  0.187 -0.465
```

Interpolate NAs and substitute NAs at the beginning and end of the series

```
> na.omit(Y, method = "iz")
           A      B      C      D      E
 [1,] -0.194 -0.350  0.000  0.740 -0.926
 [2,]  0.580  1.782 -1.879  0.699 -1.479
 [3,] -0.889  0.559 -1.854  0.162 -2.218
 [4,] -0.506 -1.194 -0.405  1.014 -0.399
 [5,] -0.636 -2.423  0.309 -1.280  0.456
 [6,]  0.462 -2.423 -0.652 -1.123  0.170
 [7,] -1.550  0.061 -0.764 -1.490  0.647
 [8,] -0.300  0.656 -0.009 -0.979  0.773
 [9,]  0.774 -2.082  0.998 -1.069 -0.608
[10,] -0.338  0.741 -1.619 -1.403 -1.760
[11,] -0.139 -0.371 -0.952  0.020 -1.760
[12,]  0.248 -1.384  0.871 -0.865 -1.760
[13,]  0.243 -1.356  1.443 -0.865 -1.121
[14,]  0.092 -1.804  0.099 -0.865 -0.182
[15,]  1.170  0.685  0.026 -0.865 -0.291
[16,] -0.676  0.944 -1.482 -1.077 -0.581
[17,]  0.972  0.944 -1.482 -1.077  0.032
[18,] -0.193  0.350  0.458 -1.370 -0.761
[19,]  0.960 -0.195 -0.922  0.187 -0.465
[20,]  0.000 -0.515 -1.005  1.361  0.125
```

Interpolate NAs and extrapolate NAs at the beginning and end of the series

```
> na.omit(Y, method = "ie")
           A      B      C      D      E
 [1,] -0.194 -0.350 -1.879  0.740 -0.926
 [2,]  0.580  1.782 -1.879  0.699 -1.479
 [3,] -0.889  0.559 -1.854  0.162 -2.218
 [4,] -0.506 -1.194 -0.405  1.014 -0.399
 [5,] -0.636 -2.423  0.309 -1.280  0.456
 [6,]  0.462 -2.423 -0.652 -1.123  0.170
 [7,] -1.550  0.061 -0.764 -1.490  0.647
 [8,] -0.300  0.656 -0.009 -0.979  0.773
 [9,]  0.774 -2.082  0.998 -1.069 -0.608
[10,] -0.338  0.741 -1.619 -1.403 -1.760
[11,] -0.139 -0.371 -0.952  0.020 -1.760
[12,]  0.248 -1.384  0.871 -0.865 -1.760
[13,]  0.243 -1.356  1.443 -0.865 -1.121
[14,]  0.092 -1.804  0.099 -0.865 -0.182
[15,]  1.170  0.685  0.026 -0.865 -0.291
[16,] -0.676  0.944 -1.482 -1.077 -0.581
[17,]  0.972  0.944 -1.482 -1.077  0.032
[18,] -0.193  0.350  0.458 -1.370 -0.761
[19,]  0.960 -0.195 -0.922  0.187 -0.465
[20,]  0.960 -0.515 -1.005  1.361  0.125
```

Different interpolation types
Before

```
> na.omit(Y, method = "ie", interp = "before")
```

```
           A      B      C      D      E
 [1,] -0.194 -0.350 -1.879  0.740 -0.926
 [2,]  0.580  1.782 -1.879  0.699 -1.479
 [3,] -0.889  0.559 -1.854  0.162 -2.218
 [4,] -0.506 -1.194 -0.405  1.014 -0.399
 [5,] -0.636 -2.423  0.309 -1.280  0.456
 [6,]  0.462 -2.423 -0.652 -1.123  0.170
 [7,] -1.550  0.061 -0.764 -1.490  0.647
 [8,] -0.300  0.656 -0.009 -0.979  0.773
 [9,]  0.774 -2.082  0.998 -1.069 -0.608
[10,] -0.338  0.741 -1.619 -1.403 -1.760
[11,] -0.139 -0.371 -0.952  0.020 -1.760
[12,]  0.248 -1.384  0.871 -0.865 -1.760
[13,]  0.243 -1.356  1.443 -0.865 -1.121
[14,]  0.092 -1.804  0.099 -0.865 -0.182
[15,]  1.170  0.685  0.026 -0.865 -0.291
[16,] -0.676  0.944 -1.482 -1.077 -0.581
[17,]  0.972  0.944 -1.482 -1.077  0.032
[18,] -0.193  0.350  0.458 -1.370 -0.761
[19,]  0.960 -0.195 -0.922  0.187 -0.465
[20,]  0.960 -0.515 -1.005  1.361  0.125
```

Linear

```
> na.omit(Y, method = "ie", interp = "linear")
```

```
           A      B      C       D      E
 [1,] -0.194 -0.350 -1.879  0.7400 -0.926
 [2,]  0.580  1.782 -1.879  0.6990 -1.479
 [3,] -0.889  0.559 -1.854  0.1620 -2.218
 [4,] -0.506 -1.194 -0.405  1.0140 -0.399
 [5,] -0.636 -2.423  0.309 -1.2800  0.456
 [6,]  0.462 -1.181 -0.652 -1.1230  0.170
 [7,] -1.550  0.061 -0.764 -1.4900  0.647
 [8,] -0.300  0.656 -0.009 -0.9790  0.773
 [9,]  0.774 -2.082  0.998 -1.0690 -0.608
```

```
[10,] -0.338  0.741 -1.619 -1.4030 -1.760
[11,] -0.139 -0.371 -0.952  0.0200 -1.547
[12,]  0.248 -1.384  0.871 -0.8650 -1.334
[13,]  0.243 -1.356  1.443 -0.9180 -1.121
[14,]  0.092 -1.804  0.099 -0.9710 -0.182
[15,]  1.170  0.685  0.026 -1.0240 -0.291
[16,] -0.676  0.944 -1.482 -1.0770 -0.581
[17,]  0.972  0.647 -0.512 -1.2235  0.032
[18,] -0.193  0.350  0.458 -1.3700 -0.761
[19,]  0.960 -0.195 -0.922  0.1870 -0.465
[20,]  0.960 -0.515 -1.005  1.3610  0.125
```

After

```
> na.omit(Y, method = "ie", interp = "after")
           A      B      C      D      E
 [1,] -0.194 -0.350 -1.879  0.740 -0.926
 [2,]  0.580  1.782 -1.879  0.699 -1.479
 [3,] -0.889  0.559 -1.854  0.162 -2.218
 [4,] -0.506 -1.194 -0.405  1.014 -0.399
 [5,] -0.636 -2.423  0.309 -1.280  0.456
 [6,]  0.462  0.061 -0.652 -1.123  0.170
 [7,] -1.550  0.061 -0.764 -1.490  0.647
 [8,] -0.300  0.656 -0.009 -0.979  0.773
 [9,]  0.774 -2.082  0.998 -1.069 -0.608
[10,] -0.338  0.741 -1.619 -1.403 -1.760
[11,] -0.139 -0.371 -0.952  0.020 -1.121
[12,]  0.248 -1.384  0.871 -0.865 -1.121
[13,]  0.243 -1.356  1.443 -1.077 -1.121
[14,]  0.092 -1.804  0.099 -1.077 -0.182
[15,]  1.170  0.685  0.026 -1.077 -0.291
[16,] -0.676  0.944 -1.482 -1.077 -0.581
[17,]  0.972  0.350  0.458 -1.370  0.032
[18,] -0.193  0.350  0.458 -1.370 -0.761
[19,]  0.960 -0.195 -0.922  0.187 -0.465
[20,]  0.960 -0.515 -1.005  1.361  0.125
```

## 16.16 `qqnorm()` - PRODUCING A NORMAL QQ PLOT

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.

```
> qqnorm(LPP2005REC.TS[, "ALT"])
```

## 16.17 `smooth()` - PERFORMING RUNNING MEDIAN SMOOTHING

The functions `smooth()` performs running median smoothing. The function implements Tukey's smoothers, 3RS3R, 3RSS, 3R, etc.

```
> ALT <- LPP2005REC.TS[, "ALT"]
> smoothedValues <- smooth(ALT)
> class(smoothedValues)
[1] "tukeysmooth"
```

Note the returned value has to be converted into a `timeSeries` object

```
> smoothed.tS <- timeSeries(data = smoothedValues, charvec = time(ALT),
     units = "SMOOTED.ALT")
> head(smoothed.tS)
GMT
           SMOOTED.ALT
2005-11-01  -0.0025730
2005-11-02  -0.0011416
2005-11-03   0.0017211
2005-11-04   0.0047240
2005-11-07   0.0047240
2005-11-08   0.0036029
```

## 16.18 spectrum() - ESTIMATING THE SPRECTRAL DENSITY

The function `spectrum()` estimates the spectral density of a time series.

```
> ALT <- LPP2005REC.TS[, "ALT"]
> print(spectrum(ALT)[1:2])
$freq
  [1] 0.0026042 0.0052083 0.0078125 0.0104167 0.0130208 0.0156250 0.0182292
  [8] 0.0208333 0.0234375 0.0260417 0.0286458 0.0312500 0.0338542 0.0364583
 [15] 0.0390625 0.0416667 0.0442708 0.0468750 0.0494792 0.0520833 0.0546875
 [22] 0.0572917 0.0598958 0.0625000 0.0651042 0.0677083 0.0703125 0.0729167
 [29] 0.0755208 0.0781250 0.0807292 0.0833333 0.0859375 0.0885417 0.0911458
 [36] 0.0937500 0.0963542 0.0989583 0.1015625 0.1041667 0.1067708 0.1093750
 [43] 0.1119792 0.1145833 0.1171875 0.1197917 0.1223958 0.1250000 0.1276042
 [50] 0.1302083 0.1328125 0.1354167 0.1380208 0.1406250 0.1432292 0.1458333
 [57] 0.1484375 0.1510417 0.1536458 0.1562500 0.1588542 0.1614583 0.1640625
 [64] 0.1666667 0.1692708 0.1718750 0.1744792 0.1770833 0.1796875 0.1822917
 [71] 0.1848958 0.1875000 0.1901042 0.1927083 0.1953125 0.1979167 0.2005208
 [78] 0.2031250 0.2057292 0.2083333 0.2109375 0.2135417 0.2161458 0.2187500
 [85] 0.2213542 0.2239583 0.2265625 0.2291667 0.2317708 0.2343750 0.2369792
 [92] 0.2395833 0.2421875 0.2447917 0.2473958 0.2500000 0.2526042 0.2552083
 [99] 0.2578125 0.2604167 0.2630208 0.2656250 0.2682292 0.2708333 0.2734375
[106] 0.2760417 0.2786458 0.2812500 0.2838542 0.2864583 0.2890625 0.2916667
[113] 0.2942708 0.2968750 0.2994792 0.3020833 0.3046875 0.3072917 0.3098958
[120] 0.3125000 0.3151042 0.3177083 0.3203125 0.3229167 0.3255208 0.3281250
[127] 0.3307292 0.3333333 0.3359375 0.3385417 0.3411458 0.3437500 0.3463542
[134] 0.3489583 0.3515625 0.3541667 0.3567708 0.3593750 0.3619792 0.3645833
[141] 0.3671875 0.3697917 0.3723958 0.3750000 0.3776042 0.3802083 0.3828125
[148] 0.3854167 0.3880208 0.3906250 0.3932292 0.3958333 0.3984375 0.4010417
[155] 0.4036458 0.4062500 0.4088542 0.4114583 0.4140625 0.4166667 0.4192708
[162] 0.4218750 0.4244792 0.4270833 0.4296875 0.4322917 0.4348958 0.4375000
[169] 0.4401042 0.4427083 0.4453125 0.4479167 0.4505208 0.4531250 0.4557292
[176] 0.4583333 0.4609375 0.4635417 0.4661458 0.4687500 0.4713542 0.4739583
[183] 0.4765625 0.4791667 0.4817708 0.4843750 0.4869792 0.4895833 0.4921875
```

```
[190] 0.4947917 0.4973958 0.5000000

$spec
  [1] 3.6647e-05 6.9767e-05 5.1309e-05 2.8434e-05 5.5068e-05 2.9109e-05
  [7] 1.1693e-05 4.3646e-06 3.4298e-05 2.9808e-05 5.4924e-07 2.4879e-05
 [13] 3.8772e-05 3.5752e-06 2.0750e-06 2.8281e-05 9.8413e-05 4.4120e-05
 [19] 2.0280e-05 1.2315e-05 3.1932e-05 3.4188e-05 2.8183e-05 6.9791e-05
 [25] 1.6419e-05 1.4450e-05 4.4812e-05 2.1920e-04 7.1563e-06 6.2808e-05
 [31] 7.1840e-06 6.7136e-05 4.2707e-05 1.1973e-05 1.2749e-05 5.3856e-06
 [37] 1.2290e-04 3.1736e-05 3.8557e-05 1.2030e-04 2.2297e-05 6.4306e-06
 [43] 6.9150e-06 2.0537e-05 7.3274e-06 5.0405e-06 2.8282e-05 1.8429e-06
 [49] 3.0464e-05 1.4307e-06 3.3224e-05 3.6581e-06 7.8195e-06 1.3745e-05
 [55] 4.3434e-06 1.8456e-05 9.6339e-06 3.5364e-05 5.5090e-05 4.2584e-05
 [61] 4.2215e-07 2.5241e-05 2.2186e-06 3.5113e-06 6.9775e-05 8.7521e-06
 [67] 4.3435e-05 2.0728e-05 1.1619e-04 1.0213e-05 2.0514e-05 2.6299e-05
 [73] 6.1933e-05 1.5180e-06 8.5905e-05 7.7336e-06 9.8840e-05 1.5846e-05
 [79] 7.0308e-06 2.6261e-06 2.1854e-05 2.1403e-05 8.7372e-05 4.4001e-05
 [85] 8.7875e-05 6.2369e-06 4.1192e-06 3.4354e-05 1.3957e-05 2.9808e-05
 [91] 1.4303e-05 3.4215e-05 6.3997e-06 8.1105e-05 2.3131e-05 1.4712e-05
 [97] 2.1753e-06 2.7383e-05 3.3614e-06 2.2428e-05 2.8009e-05 3.2551e-05
[103] 7.6379e-06 7.7152e-06 4.2875e-05 7.1196e-05 5.3246e-06 2.4691e-05
[109] 1.8666e-05 7.1875e-05 1.3076e-05 3.7610e-05 3.1320e-06 4.2016e-06
[115] 8.8496e-06 9.3560e-05 2.1255e-05 1.0373e-05 1.2002e-05 1.5838e-05
[121] 1.1971e-04 5.6298e-05 3.7835e-06 3.6736e-05 8.1358e-05 5.1174e-07
[127] 6.6478e-05 4.0274e-06 5.0737e-07 5.0004e-05 1.2758e-04 1.7874e-05
[133] 5.6211e-05 2.7870e-05 4.9705e-05 1.5926e-05 1.0539e-04 5.8640e-05
[139] 2.8704e-05 4.7235e-06 4.6709e-05 9.6925e-06 5.5474e-05 1.8023e-05
[145] 7.3100e-06 1.7918e-05 3.7372e-05 4.5979e-05 4.1000e-05 3.3199e-05
[151] 6.3405e-05 7.7540e-08 2.5305e-05 2.0555e-06 2.5882e-06 2.3432e-05
[157] 6.1866e-05 5.3509e-05 1.7556e-05 2.1672e-06 2.0150e-05 2.6091e-05
[163] 5.9727e-05 1.1915e-05 7.7370e-06 7.7044e-06 3.2009e-05 2.6313e-05
[169] 1.4957e-05 6.5149e-05 1.8595e-06 7.6299e-05 6.6492e-06 2.3140e-05
[175] 2.3054e-07 9.2174e-06 8.1372e-06 3.3856e-05 9.5630e-06 3.8631e-06
[181] 1.4073e-04 6.5389e-05 1.7051e-06 3.2884e-05 3.3838e-06 8.1191e-05
[187] 5.7136e-06 3.4705e-05 4.6381e-06 1.1658e-05 4.0027e-05 3.5888e-05
```

## 16.19 `window()` - SUBSETTING BY WINDOWS

The function `window()` is a generic function which extracts the subset of the object `x` observed between the times `start` and `end`.
There is no method for `timeSeries` objects, only for `timeDate` objects.

# CHAPTER 17

# FINANCIAL FUNCTIONS AND METHODS

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

## 17.1 `align()` - ALIGNING A 'TIMESERIES' OBJECTS

The alignment of `timeSeries` objects is another important aspect. For daily data sets due to holidays we expect missing data records in any financial time series. For example around Easter we may miss a data record for Good Friday in most countries of thew world, and possibly on Easter Monday the markets will be also closed. Then to analyze such data sets we want to align the series to a regular weekly calendar series. Missing records can then be coded as NAs, the values can be interpolated or extrapolated, and handled according to any other rule. For intra day data it becomes often desirable to to generate series equidistant in time, for example hourly data records or on a short scale 5 minutes data.

As an example let us consider stock market data of the MSFT stock starting one week before and ending one week after Easter. So we would expect that Good Friday, and likely also Easter Monday are missing in the time series. The goal is now to make the series regular and to align the series to weekdays from Monday to Friday.

Before we start to align the daily series, have a look on the arguments of the generic function `align`

```
> args(.align.timeSeries)
function (x, by = "1d", offset = "0s", method = c("before", "after",
    "interp", "fillNA", "fmm", "periodic", "natural", "monoH.FC"),
    include.weekends = FALSE, ...)
```

```
NULL
```

The function `align` takes a `timeSeries` object `x`, and creates regular time stamps equidistent in time defined by the argument `by`. These time stamps may have an optional `offset` measures with respect to the first time stamp of the series `x`. The argument ...
The function aligns by default on calendar dates, i.e. on every day of the week, or on the weekdays from Monday to Friday. The last is the default setting with `include.weekends=FALSE`. How the alignment is done is determined by the argument `method`. The following options ar available: `"before"`, `"after"`, `"interp"`, `"fillNA"`. Note, the function can also align intraday data, the startup time stamp can then be cosen by setting an appropriate offset, see ...
Let us align the series of MSFT stock prices to daily dates including weekends and Easter holidays and express the missing values with NAs, this helps us to locate easily the missing data records. To make the outputs not too long we consider the week befor and after Easter:

```
> oneWeek <- 5 * 24 * 3600
> c(GoodFriday(2001), Easter(2001))
GMT
[1] [2001-04-13] [2001-04-15]
> tEASTER <- MSFT[(time(MSFT) > Easter(2001) - oneWeek) & (time(MSFT) <
      Easter(2001) + oneWeek), ]
> tEASTER
GMT
            Open  High   Low Close   Volume
2001-04-11 60.65 61.50 59.70 60.04 54939800
2001-04-12 59.56 62.31 59.35 62.18 43760000
2001-04-16 61.40 61.58 60.12 60.79 32928700
2001-04-17 60.52 62.11 60.04 61.48 42574600
2001-04-18 63.39 66.31 63.00 65.43 78348200
2001-04-19 65.81 69.00 65.75 68.04 79687800


> .align.timeSeries(tEASTER, method = "fillNA", include.weekends = TRUE)
GMT
            Open  High   Low Close   Volume
2001-04-11 60.65 61.50 59.70 60.04 54939800
2001-04-12 59.56 62.31 59.35 62.18 43760000
2001-04-13    NA    NA    NA    NA       NA
2001-04-14    NA    NA    NA    NA       NA
2001-04-15    NA    NA    NA    NA       NA
2001-04-16 61.40 61.58 60.12 60.79 32928700
2001-04-17 60.52 62.11 60.04 61.48 42574600
2001-04-18 63.39 66.31 63.00 65.43 78348200
2001-04-19 65.81 69.00 65.75 68.04 79687800
```

Finally we align the series to the Easter holiday(s) (excluding weekends) and replace the missing value rolling forward with the previous value:

```
> .align.timeSeries(tEASTER, method = "before")
GMT
            Open  High   Low Close   Volume
2001-04-11 60.65 61.50 59.70 60.04 54939800
2001-04-12 59.56 62.31 59.35 62.18 43760000
2001-04-13 59.56 62.31 59.35 62.18 43760000
2001-04-16 61.40 61.58 60.12 60.79 32928700
2001-04-17 60.52 62.11 60.04 61.48 42574600
2001-04-18 63.39 66.31 63.00 65.43 78348200
2001-04-19 65.81 69.00 65.75 68.04 79687800
```

Note, the function can also align intraday data. Consider the USDTHB Bid prices. We like to align these prices to equidistant time stamps of 3 hours, starting on midnight to the full hour.

```
> data(usdthb)
> USDTHB <- timeSeries(data = as.matrix(usdthb[, c("BID", "ASK")]),
     charvec = as.character(usdthb[, 1]))
> start(USDTHB)
GMT
[1] [1997-06-01 19:28:00]
> USDTHB3H <- .align.timeSeries(USDTHB, by = "3h", offset = "92m")
> head(USDTHB3H)
GMT
                       BID   ASK
1997-06-02 00:00:00 24.770 24.82
1997-06-02 03:00:00 24.800 24.85
1997-06-02 06:00:00 24.800 24.85
1997-06-02 09:00:00 24.800 24.90
1997-06-02 12:00:00 24.845 24.92
1997-06-02 15:00:00 24.800 24.90
```

## 17.2 cumulated() - CUMULATING A FINANCIAL SERIES

The function `cumulated()` cumulates financial series, to obtain for example prices or indexes, from financial returns.

*Compute discrete return series*

```
> data(MSFT)
> MSFT <- 100 * MSFT/as.numeric(MSFT[1, 1])
> MSFT.RET <- returns(MSFT, method = "discrete")
> head(MSFT.RET)
GMT
                Open       High        Low      Close   Volume
2000-09-28 -0.0413793 -0.0265487  0.0135841  0.0113402 -0.50676
2000-09-29  0.0030832 -0.0090909 -0.0329897 -0.0163099  0.41431
2000-10-02 -0.0081967 -0.0081549 -0.0063966 -0.0196891 -0.20919
2000-10-03 -0.0154959 -0.0164440 -0.0300429 -0.0433404  0.45783
2000-10-04 -0.0535152 -0.0543365 -0.0353982 -0.0198895  0.59830
2000-10-05 -0.0155211  0.0121547  0.0137615 -0.0011274 -0.40566
```

*Obtain cumulative series, indexed to 100*

```
> MSFT.CUM <- cumulated(MSFT.RET, method = "discrete")
> head(MSFT.CUM)

GMT
              Open    High     Low   Close  Volume
2000-09-28 0.95862 0.97345 1.01358 1.01134 0.49324
2000-09-29 0.96158 0.96460 0.98015 0.99485 0.69759
2000-10-02 0.95369 0.95674 0.97388 0.97526 0.55167
2000-10-03 0.93892 0.94100 0.94462 0.93299 0.80423
2000-10-04 0.88867 0.88987 0.91118 0.91443 1.28541
2000-10-05 0.87488 0.90069 0.92372 0.91340 0.76397
```

## 17.3 `daily()` - SPECIFIC DAILY TIME SERIES HANDLING

*Align a 'timeSeries' object to new positions*

```
> args(alignDailySeries)

function (x, method = c("before", "after", "interp", "fillNA",
    "fmm", "periodic", "natural", "monoH.FC"), include.weekends = FALSE,
    units = NULL, zone = "", FinCenter = "", ...)
NULL
```

```
> data(MSFT)
> dim(MSFT)

[1] 249   5

> MSFT.ALGD <- alignDailySeries(MSFT)
> dim(MSFT.ALGD)

[1] 262   5
```

*Roll daily a 'timeSeries' on a given period*

```
> args(rollDailySeries)

function (x, period = "7d", FUN, ...)
NULL
```

## 17.4 `drawdowns()` - CALCULATING DRAWDOWNS

The function `drawdowns()` compute series of drawdowns from financial returns and calculates drawdown statisitcs.

### *Series of drawdowns*

```
> data(LPP2005REC)
> SPI <- LPP2005REC[, "SPI"]
```

```
> dd <- drawdowns(SPI)
> head(dd, 10)
```

```
GMT
                  SPI
2005-11-01  0.00000000
2005-11-02  0.00000000
2005-11-03  0.00000000
2005-11-04 -0.00070276
2005-11-07  0.00000000
2005-11-08  0.00000000
2005-11-09 -0.00237820
2005-11-10 -0.00145831
2005-11-11  0.00000000
2005-11-14 -0.00469306
```

### *Drawdown statistics*

```
> drawdownsStats(SPI)
```

```
         From     Trough         To       Depth Length ToTrough Recovery
1  2006-05-10 2006-06-13 2006-09-01 -0.12359254     83       25       58
2  2007-02-20 2007-03-14 2007-04-11 -0.07712377     38       17       NA
3  2006-11-08 2006-12-01 2006-12-15 -0.04042220     28       18       10
4  2006-01-17 2006-01-25 2006-02-01 -0.01902306     12        7        5
5  2005-12-07 2005-12-15 2005-12-29 -0.01877532     17        7       10
6  2006-02-23 2006-03-08 2006-03-13 -0.01815480     13       10        3
7  2006-09-05 2006-09-11 2006-09-15 -0.01598341      9        5        4
8  2006-10-13 2006-10-17 2006-10-23 -0.01435509      7        3        4
9  2006-04-11 2006-04-18 2006-04-20 -0.01363434      8        6        2
10 2006-10-30 2006-10-31 2006-11-03 -0.01298943      5        2        3
11 2006-04-24 2006-05-03 2006-05-05 -0.01201538     10        8        2
12 2005-11-28 2005-11-30 2005-12-01 -0.01195378      4        3        1
13 2006-09-22 2006-09-25 2006-09-26 -0.01188161      3        2        1
14 2007-01-25 2007-01-26 2007-01-30 -0.01130967      4        2        2
15 2005-11-14 2005-11-16 2005-11-18 -0.01059497      5        3        2
16 2006-02-14 2006-02-15 2006-02-16 -0.01044179      3        2        1
17 2006-03-23 2006-03-28 2006-04-03 -0.00921396      8        4        4
18 2006-12-19 2006-12-22 2006-12-27 -0.00871083      7        4        3
19 2007-01-05 2007-01-08 2007-01-11 -0.00711091      5        2        3
20 2006-01-05 2006-01-10 2006-01-11 -0.00672046      5        4        1
21 2007-02-12 2007-02-13 2007-02-14 -0.00647794      3        2        1
22 2007-02-08 2007-02-08 2007-02-09 -0.00608619      2        1        1
23 2006-02-02 2006-02-02 2006-02-03 -0.00523154      2        1        1
24 2006-10-02 2006-10-02 2006-10-04 -0.00500829      3        1        2
25 2007-01-22 2007-01-23 2007-01-24 -0.00496833      3        2        1
26 2005-12-30 2005-12-30 2006-01-03 -0.00397647      3        1        2
27 2006-10-24 2006-10-24 2006-10-27 -0.00351258      4        1        3
28 2006-12-28 2006-12-29 2007-01-03 -0.00313462      5        2        3
```

```
29 2006-04-04 2006-04-04 2006-04-05 -0.00261849      2        1        1
30 2006-09-19 2006-09-19 2006-09-20 -0.00256865      2        1        1
31 2005-11-24 2005-11-24 2005-11-25 -0.00255954      2        1        1
32 2005-11-09 2005-11-09 2005-11-11 -0.00237820      3        1        2
33 2007-01-16 2007-01-16 2007-01-17 -0.00233852      2        1        1
34 2006-03-14 2006-03-15 2006-03-17 -0.00229763      4        2        2
35 2006-02-07 2006-02-08 2006-02-09 -0.00114567      3        2        1
36 2006-02-10 2006-02-10 2006-02-13 -0.00102901      2        1        1
37 2005-11-04 2005-11-04 2005-11-07 -0.00070276      2        1        1
38 2006-03-21 2006-03-21 2006-03-22 -0.00027532      2        1        1
39 2005-12-05 2005-12-05 2005-12-06 -0.00026869      2        1        1
40 2007-01-31 2007-01-31 2007-02-01 -0.00006490      2        1        1
41 2007-01-18 2007-01-18 2007-01-19 -0.00003060      2        1        1
```

## 17.5 `durations()` - MEASURING DURATIONS

The function `durations()` computes durations between consecutive records.

```
> data(MSFT)
> Close <- MSFT[, "Close"]
```

```
> oneDay <- 24 * 3600
> MSFT.DUR <- na.omit(durations(Close)/oneDay)
> head(MSFT.DUR, 12)
```

```
GMT
           Duration
2000-09-28        1
2000-09-29        1
2000-10-02        3
2000-10-03        1
2000-10-04        1
2000-10-05        1
2000-10-06        1
2000-10-09        3
2000-10-10        1
2000-10-11        1
2000-10-12        1
2000-10-13        1
```

## 17.6 `monthly()` - HANDLING MONTHLY SERIES

*Returns a series with monthly counts of records*

The function `countMonthlyRecords()`

```
> args(countMonthlyRecords)
```

```
function (x)
NULL
```

returns the number of records in each month of a time series. Let us count the number of monthly records for the MSFT time series

```
> counts.MSFT <- countMonthlyRecords(MSFT)
> counts.MSFT
GMT
           Counts
2000-09-30      3
2000-10-31     22
2000-11-30     21
2000-12-31     20
2001-01-31     21
2001-02-28     19
2001-03-31     22
2001-04-30     20
2001-05-31     22
2001-06-30     21
2001-07-31     21
2001-08-31     23
```

*Decides if the series consists of monthly records*

If we like to decide if a time series is a monthly series, then we can use the function `isMonthly()`

```
> args(isMonthly)
function (x)
NULL
```

The time series is a monthly series if for each month there exist none or one record.

```
> isMonthly(MSFT)
[1] FALSE
> isMonthly(counts.MSFT)
[1] TRUE
```

*Returns start/end dates for rolling time windows*

```
> args(rollMonthlyWindows)
function (x, period = "12m", by = "1m")
NULL
```

*Rolls Monthly a 'timeSeries' on a given period*

```
> args(rollMonthlySeries)
function (x, period = "12m", by = "1m", FUN, ...)
NULL
```

## 17.7 `orderColnames()` - ORDERING COLUMN NAMES

*Returns ordered column names of a time Series*

```
> args(orderColnames)

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

*Returns sorted column names of a time Series*

```
> args(sortColnames)

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

*Returns sampled column names of a time Series*

```
> args(sampleColnames)

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

*Returns statistically rearranged column names*

```
> args(statsColnames)

function (x, FUN = colMeans, ...)
NULL
```

*Returns PCA correlation ordered column names*

```
> args(pcaColnames)

function (x, robust = FALSE, ...)
NULL
```

*Returns hierarchical clustered column names*

```
> args(hclustColnames)

function (x, method = c("euclidean", "complete"), ...)
NULL
```

## 17.8 orderStatistics() - COMPUTING ORDER STATISTICS

The function `orderStatistics()` computes the order statistics of a `timeSeries` object.

```
> orderStats <- orderStatistics(LPP2005REC[1:10, 1:2])
> orderStats
```

```
$SBI
GMT
                   SBI
2005-11-04 -0.00323575
2005-11-02 -0.00276201
2005-11-09 -0.00254502
2005-11-03 -0.00115309
2005-11-01 -0.00061275
2005-11-08  0.00053931
2005-11-11  0.00061695
2005-11-14  0.00069361
2005-11-10  0.00100336
2005-11-07  0.00131097

$SPI
GMT
                   SPI
2005-11-14 -0.00469306
2005-11-09 -0.00237820
2005-11-04 -0.00070276
2005-11-08  0.00032926
2005-11-10  0.00092209
2005-11-02  0.00251934
2005-11-07  0.00620523
2005-11-01  0.00841460
2005-11-03  0.01270729
2005-11-11  0.01333491
```

The returned value is a list with all ordered time series, the names are those of the columns of the original time series.

```
> class(orderStats)
```

```
[1] "list"
```

## 17.9 periodical() - COMPUTING PERIODICAL PERFORMANCE

There are several functions to compute periodical performance

*Returns series back to a given period*

```
> args(endOfPeriodSeries)
```

```
function (x, nYearsBack = c("1y", "2y", "3y", "5y", "10y", "YTD"))
NULL
```

*Returns statistics back to a given period*

```
> args(endOfPeriodStats)

function (x, nYearsBack = c("1y", "2y", "3y", "5y", "10y", "YTD"))
NULL
```

*Returns benchmarks back to a given period*

```
> args(endOfPeriodBenchmarks)

function (x, benchmark = ncol(x), nYearsBack = c("1y", "2y",
    "3y", "5y", "10y", "YTD"))
NULL
```

## 17.10 `returns()` - COMPUTING FINANCIAL RETURNS

The function `returns()` computes financial returns from a timeSeries of prices or indexes.
«args.returns» args(returns)

*Continuous returns*

```
> data(MSFT)
> MSFT.CRET <- returns(MSFT[, 1:4])
> head(MSFT.CRET, 10)

GMT
                 Open       High        Low      Close
2000-09-28 -0.0422598 -0.0269075  0.0134927  0.0112764
2000-09-29  0.0030785 -0.0091325 -0.0335461 -0.0164444
2000-10-02 -0.0082305 -0.0081884 -0.0064171 -0.0198855
2000-10-03 -0.0156172 -0.0165807 -0.0305035 -0.0443076
2000-10-04 -0.0550004 -0.0558684 -0.0360399 -0.0200900
2000-10-05 -0.0156428  0.0120814  0.0136676 -0.0011280
2000-10-06  0.0056148 -0.0087720 -0.0090910  0.0033803
2000-10-09 -0.0033651 -0.0177782 -0.0324855 -0.0250583
2000-10-10 -0.0308068 -0.0033689  0.0152139  0.0068966
2000-10-11  0.0011581  0.0244457  0.0034783  0.0215306
```

*Discrete returns*

```
> MSFT.DRET <- returns(MSFT[, 1:4], type = "discrete")
> head(MSFT.DRET, 10)

GMT
                 Open       High        Low      Close
2000-09-28 -0.0422598 -0.0269075  0.0134927  0.0112764
2000-09-29  0.0030785 -0.0091325 -0.0335461 -0.0164444
2000-10-02 -0.0082305 -0.0081884 -0.0064171 -0.0198855
2000-10-03 -0.0156172 -0.0165807 -0.0305035 -0.0443076
```

```
2000-10-04 -0.0550004 -0.0558684 -0.0360399 -0.0200900
2000-10-05 -0.0156428  0.0120814  0.0136676 -0.0011280
2000-10-06  0.0056148 -0.0087720 -0.0090910  0.0033803
2000-10-09 -0.0033651 -0.0177782 -0.0324855 -0.0250583
2000-10-10 -0.0308068 -0.0033689  0.0152139  0.0068966
2000-10-11  0.0011581  0.0244457  0.0034783  0.0215306
```

*Percentage returns*

```
> MSFT.PRET <- returns(MSFT[, 1:4], percentage = TRUE)
> head(MSFT.PRET, 10)
GMT
               Open     High      Low    Close
2000-09-28 -4.22598 -2.69075  1.34927  1.12764
2000-09-29  0.30785 -0.91325 -3.35461 -1.64444
2000-10-02 -0.82305 -0.81884 -0.64171 -1.98855
2000-10-03 -1.56172 -1.65807 -3.05035 -4.43076
2000-10-04 -5.50004 -5.58684 -3.60399 -2.00900
2000-10-05 -1.56428  1.20814  1.36676 -0.11280
2000-10-06  0.56148 -0.87720 -0.90910  0.33803
2000-10-09 -0.33651 -1.77782 -3.24855 -2.50583
2000-10-10 -3.08068 -0.33689  1.52139  0.68966
2000-10-11  0.11581  2.44457  0.34783  2.15306
```

*Untrimmed returns*

If we compute returns, the the series will be trimmed, that means that the first record will be dropped from the series. To keep this record set the argument trim=FALSE

```
> MSFT.TRET <- returns(MSFT[, 1:4], trim = FALSE)
> head(MSFT.TRET, 10)
GMT
                 Open       High        Low      Close
2000-09-27         NA         NA         NA         NA
2000-09-28 -0.0422598 -0.0269075  0.0134927  0.0112764
2000-09-29  0.0030785 -0.0091325 -0.0335461 -0.0164444
2000-10-02 -0.0082305 -0.0081884 -0.0064171 -0.0198855
2000-10-03 -0.0156172 -0.0165807 -0.0305035 -0.0443076
2000-10-04 -0.0550004 -0.0558684 -0.0360399 -0.0200900
2000-10-05 -0.0156428  0.0120814  0.0136676 -0.0011280
2000-10-06  0.0056148 -0.0087720 -0.0090910  0.0033803
2000-10-09 -0.0033651 -0.0177782 -0.0324855 -0.0250583
2000-10-10 -0.0308068 -0.0033689  0.0152139  0.0068966
```

Note, the non-trimmed time series has the same length as the index series

```
> nrow(MSFT)
[1] 249
> nrow(MSFT.TRET)
[1] 249
```

## 17.11 `smooth()` - SMOOTHING A TIME SERIES

Rmetrics has three functions implemented to smooth a financial time series. These are Friedman's SuperSmoother `supsmu()`, the scatter plot smoother `lowess()`, and the cubic spline smoother `smooth.spline()`. The (current) function names are `smoothSupsmu()`, `smoothLowess()`, and `smoothSpline()`.

### *Friedman's Super Smoother*

```
> s <- smoothSupsmu(MSFT[, "Close"], bass = 0.1)
> head(s)
```

```
GMT
            Close supsmu
2000-09-27 60.625 59.884
2000-09-28 61.312 59.401
2000-09-29 60.312 58.919
2000-10-02 59.125 58.436
2000-10-03 56.562 57.953
2000-10-04 55.438 57.471
```

### *Lowess Scatter Plot Smoother*

```
> s <- smoothLowess(MSFT[, "Close"], f = 0.05)
> head(s)
```

```
GMT
            Close lowess
2000-09-27 60.625 61.441
2000-09-28 61.312 60.476
2000-09-29 60.312 59.511
2000-10-02 59.125 58.595
2000-10-03 56.562 57.679
2000-10-04 55.438 56.854
```

### *Cubic Spline Smoother*

```
> s <- smoothSpline(MSFT[, 4], spar = 0.4)
> head(s)
```

```
GMT
            Close spline
2000-09-27 60.625 61.412
2000-09-28 61.312 60.443
2000-09-29 60.312 59.444
2000-10-02 59.125 58.408
2000-10-03 56.562 57.402
2000-10-04 55.438 56.495
```

## 17.12 splits() - REMOVING OUTLIERS FROM SPLITS

Sometimes there may exist huge jumps in a time series due to splits in equity indexes or share prices.

```
> args(outlier)

function (x, sd = 5, complement = TRUE, ...)
NULL
```

## 17.13 spreads() - COMPUTING SPREADS AND MIDQUITES

*Compute spreads from a 'timeSeries' object*

```
> args(spreads)

function (x, which = c("Bid", "Ask"), tickSize = NULL)
NULL
```

*Compute mid quotes from a 'timeSeries' object*

```
> args(midquotes)

function (x, which = c("Bid", "Ask"))
NULL
```

## 17.14 turnpoints() - SEARCHING FOR PEAKS AND PITS

*Return peaks and pits from a timeSeries*

The function turnpoints detects and returns turnpoints in a financial time series.

```
> args(turns)

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

```
> S <- smoothLowess(MSFT[, "Close"], f = 0.05)[, "lowess"]
> S.tp <- turns(S)
> head(S.tp, 36)

GMT
           lowess TP
2000-09-27 61.441  0
2000-09-28 60.476  0
2000-09-29 59.511  0
2000-10-02 58.595  0
2000-10-03 57.679  0
2000-10-04 56.854  0
2000-10-05 56.028  0
2000-10-06 55.550  0
```

```
2000-10-09 55.071  0
2000-10-10 54.830  0
2000-10-11 54.590 -1
2000-10-12 55.116  0
2000-10-13 55.643  0
2000-10-16 56.826  0
2000-10-17 58.009  0
2000-10-18 58.905  0
2000-10-19 59.800  0
2000-10-20 60.933  0
2000-10-23 62.065  0
2000-10-24 62.965  0
2000-10-25 63.864  0
2000-10-26 65.002  0
2000-10-27 66.140  0
2000-10-30 67.141  0
2000-10-31 68.142  0
2000-11-01 68.717  0
2000-11-02 69.292  0
2000-11-03 69.414  0
2000-11-06 69.535  1
2000-11-07 69.365  0
2000-11-08 69.194  0
2000-11-09 69.118  0
2000-11-10 69.042  0
2000-11-13 68.893  0
2000-11-14 68.744  0
2000-11-15 68.617  0
```

*Compute turnpoints statistics*

```
> args(turnsStats)
function (x, doplot = TRUE)
NULL
```

```
> turnsStats(S)
Turning points for: x

nbr observations  : 249
nbr ex-aequos     : 0
nbr turning points: 18 (first point is a pit)
E(p) = 164.67 Var(p) = 43.944 (theoretical)

   point type      proba    info
1     11  pit 2.9684e-24  78.157
2     29 peak 5.7390e-22  70.562
3     37  pit 6.0125e-06  17.344
4     39 peak 5.9694e-26  83.793
5     63  pit 6.3671e-47 153.460
6     85 peak 2.8220e-36 118.093
7    101  pit 1.3194e-17  56.073
8    107 peak 3.0522e-13  41.575
```

```
9    119  pit 5.2323e-13  40.798
10   125 peak 1.2626e-05  16.273
11   129  pit 4.6314e-27  87.481
12   153 peak 6.4604e-30  96.966
13   161  pit 2.4738e-17  55.166
14   175 peak 3.8482e-17  54.529
15   183  pit 5.2490e-09  27.505
16   189 peak 2.9677e-07  21.684
17   195  pit 2.9677e-07  21.684
18   201 peak 4.0684e-66 217.223
```

PART V

# APPENDIX

# APPENDIX A

# FAQ: TIME SERIES BASICS

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

## A.1 CREATING TIME SERIES OBJECTS

*What is my current time zone environment?*

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

```
> Sys.timezone()

[1] "Europe/Zurich"
```

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

Create example data and time stamps

```
> 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"
```

create the time series

```
> 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 10 days?*

Create example data and time stamps

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

create the series

```
> 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
```

## A.2 REGULAR TIME SERIES OBJECTS

*How can I create a regular monthly time series object?*

Create monthly example data as an object of class `ts`

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

then convert it to a time series object

```
> sm <- as.timeSeries(tm)
> sm
GMT
              TS.1
2008-03-31  0.4941
2008-04-30 -0.3688
```

```
2008-05-31  0.2365
2008-06-30  0.6201
2008-07-31  0.2098
2008-08-31 -1.4904
2008-09-30 -1.6032
2008-10-31  0.6542
2008-11-30 -0.1091
2008-12-31 -0.6485
2009-01-31  0.7964
2009-02-28  0.3177
2009-03-31  0.2305
2009-04-30 -1.5229
2009-05-31 -0.1986
2009-06-30 -2.0670
2009-07-31  0.2459
2009-08-31 -1.0007
2009-09-30  0.3347
2009-10-31  0.3607
2009-11-30 -1.3589
2009-12-31 -0.4298
2010-01-31 -0.3458
2010-02-28  1.0028
```

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 2008-09-30
    0.4941    -0.3688     0.2365     0.6201     0.2098    -1.4904    -1.6032
2008-10-31 2008-11-30 2008-12-31 2009-01-31 2009-02-28 2009-03-31 2009-04-30
    0.6542    -0.1091    -0.6485     0.7964     0.3177     0.2305    -1.5229
2009-05-31 2009-06-30 2009-07-31 2009-08-31 2009-09-30 2009-10-31 2009-11-30
   -0.1986    -2.0670     0.2459    -1.0007     0.3347     0.3607    -1.3589
2009-12-31 2010-01-31 2010-02-28
   -0.4298    -0.3458     1.0028
```

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
  0.4941  -0.3688   0.2365   0.6201   0.2098  -1.4904  -1.6032   0.6542
2008 Nov 2008 Dec 2009 Jan 2009 Feb 2009 Mar 2009 Apr 2009 May 2009 Jun
 -0.1091  -0.6485   0.7964   0.3177   0.2305  -1.5229  -0.1986  -2.0670
2009 Jul 2009 Aug 2009 Sep 2009 Oct 2009 Nov 2009 Dec 2010 Jan 2010 Feb
  0.2459  -1.0007   0.3347   0.3607  -1.3589  -0.4298  -0.3458   1.0028
```

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

```
2008(03) 2008(04) 2008(05) 2008(06) 2008(07) 2008(08) 2008(09) 2008(10)
  0.4941  -0.3688   0.2365   0.6201   0.2098  -1.4904  -1.6032   0.6542
2008(11) 2008(12) 2009(01) 2009(02) 2009(03) 2009(04) 2009(05) 2009(06)
 -0.1091  -0.6485   0.7964   0.3177   0.2305  -1.5229  -0.1986  -2.0670
2009(07) 2009(08) 2009(09) 2009(10) 2009(11) 2009(12) 2010(01) 2010(02)
  0.2459  -1.0007   0.3347   0.3607  -1.3589  -0.4298  -0.3458   1.0028
```

*How can I create a regular quarterly time series object?*

Create quarterly example data as an object of class `ts`

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

```
        Qtr1    Qtr2    Qtr3    Qtr4
2008                 -0.2183  1.1060
2009 -0.0444  0.4914 -1.4419  2.0744
2010 -2.5492 -0.8575  0.4469 -0.2412
2011 -0.3665  1.2000  0.5114  0.4695
2012  0.2093  1.2756 -2.1948 -0.7763
2013  0.3733 -0.6908  1.1456 -1.5655
2014 -1.5795  0.2059
```

then convert it to a time series object

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

```
GMT
              TS.1
2008-09-30 -0.2183
2008-12-31  1.1060
2009-03-31 -0.0444
2009-06-30  0.4914
2009-09-30 -1.4419
2009-12-31  2.0744
```

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 2010-03-31
   -0.2183     1.1060    -0.0444     0.4914    -1.4419     2.0744    -2.5492
2010-06-30 2010-09-30 2010-12-31 2011-03-31 2011-06-30 2011-09-30 2011-12-31
   -0.8575     0.4469    -0.2412    -0.3665     1.2000     0.5114     0.4695
2012-03-31 2012-06-30 2012-09-30 2012-12-31 2013-03-31 2013-06-30 2013-09-30
    0.2093     1.2756    -2.1948    -0.7763     0.3733    -0.6908     1.1456
2013-12-31 2014-03-31 2014-06-30
   -1.5655    -1.5795     0.2059
```

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
 -0.2183   1.1060  -0.0444   0.4914  -1.4419   2.0744  -2.5492  -0.8575
2010 Sep 2010 Dec 2011 Mar 2011 Jun 2011 Sep 2011 Dec 2012 Mar 2012 Jun
  0.4469  -0.2412  -0.3665   1.2000   0.5114   0.4695   0.2093   1.2756
2012 Sep 2012 Dec 2013 Mar 2013 Jun 2013 Sep 2013 Dec 2014 Mar 2014 Jun
 -2.1948  -0.7763   0.3733  -0.6908   1.1456  -1.5655  -1.5795   0.2059
```

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

```
 2008 Q3 2008 Q4 2009 Q1 2009 Q2 2009 Q3 2009 Q4 2010 Q1 2010 Q2 2010 Q3 2010 Q4
-0.2183  1.1060 -0.0444  0.4914 -1.4419  2.0744 -2.5492 -0.8575  0.4469 -0.2412
 2011 Q1 2011 Q2 2011 Q3 2011 Q4 2012 Q1 2012 Q2 2012 Q3 2012 Q4 2013 Q1 2013 Q2
-0.3665  1.2000  0.5114  0.4695  0.2093  1.2756 -2.1948 -0.7763  0.3733 -0.6908
 2013 Q3 2013 Q4 2014 Q1 2014 Q2
  1.1456 -1.5655 -1.5795  0.2059
```

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

Create the quarterly example data and time stamps

```
> data <- round(rnorm(24), 4)
> tq <- ts(data, start = c(2008, 3), frequency = 4)
> sq <- as.timeSeries(tq)
```

and check if the series is regular

```
> isRegular(sq)
```

```
[1] TRUE
```

*How I can find out if a time series is a regular monthly or quarterly time series?*

Create the monthly example data and time stamps

```
> data <- round(rnorm(24), 4)
> tm <- ts(data, start = c(2008, 3), frequency = 12)
> sm <- as.timeSeries(tm)
```

and a quarterly time series

```
> data <- round(rnorm(24), 4)
> tq <- ts(data, start = c(2008, 3), frequency = 4)
> sq <- as.timeSeries(tq)
```

Then check if they are regular monthly or quarterly series

```
> isMonthly(sm)
```

```
[1] TRUE
```

```
> isQuarterly(sm)
```

```
[1] FALSE
```

```
> isMonthly(sq)
```

```
[1] FALSE
```

```
> isQuarterly(sq)
```

```
[1] TRUE
```

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

Create the daily example data and time stamps

```
> data <- round(rnorm(12), 4)
> charvec <- timeCalendar(2010, rep(1, 12), 1:12)
> sd <- timeSeries(data, charvec)
```

then a monthly time series

```
> data <- round(rnorm(24), 4)
> tm <- ts(data, start = c(2008, 3), frequency = 12)
> sm <- as.timeSeries(tm)
```

and finally a quarterly time series

```
> data <- round(rnorm(24), 4)
> tq <- ts(data, start = c(2008, 3), frequency = 4)
> sq <- as.timeSeries(tq)
```

Then check if they are regular dauily, monthly or quarterly series

```
> isDaily(sd)

[1] TRUE

> isMonthly(sd)

[1] FALSE

> isQuarterly(sd)

[1] FALSE


> isDaily(sm)

[1] FALSE

> isMonthly(sm)

[1] TRUE

> isQuarterly(sm)

[1] FALSE


> isDaily(sq)

[1] FALSE

> isMonthly(sq)

[1] FALSE

> isQuarterly(sq)

[1] TRUE
```

## A.3 TIME ZONE AND DAYLIGHT SAVING TIME

*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. First create the data and time stamps

```
> 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"
```

then create the time series

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

Zurich
           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
```

Note that the RfuntimeSeries 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 center the `charvec` of time stamps belongs, and the second argument `FinCenter` tells us in which time zone or at which financial center 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?*

Create example data and time stamps

```
> 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"
```

and then the time series

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

```
Zurich
           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
```

Note that `timeSeries` objects show on top to which time zone (or more specific financial center) 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 function `time()` for `timeSeries()` objects to display the zone information
Create example data and time stamps

```
> 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"
```

and then the time series

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

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?*

For `timeSeries` objects we can print the list of supported financial centers using the function `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 centers 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 centers. 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
2014-01-01 0.434582
```

```
2014-02-01 0.410611
2014-03-01 0.441023
2014-04-01 0.508886
2014-05-01 0.221342
2014-06-01 0.075440
2014-07-01 0.081973
2014-08-01 0.060637
2014-09-01 0.156579
2014-10-01 0.445384
2014-11-01 0.818251
2014-12-01 0.615232
```

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?*

In `timeSeries()` we use the assignment function `finCenter<-()` to assign a new financial center 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 -1.0108869
2009-08-01 01:00:00 -0.0010012
2009-09-01 01:00:00  0.1179064
2009-10-01 01:00:00  0.6749883
2009-11-01 01:00:00  1.1936528
2009-12-01 01:00:00 -0.3568767

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

New_York
                          TS.1
2009-06-30 19:00:00 -1.0108869
2009-07-31 19:00:00 -0.0010012
2009-08-31 19:00:00  0.1179064
2009-09-30 19:00:00  0.6749883
2009-10-31 20:00:00  1.1936528
2009-11-30 19:00:00 -0.3568767
```

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?*

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    0.11055
1980-03-15 16:00:00    0.57496
1980-04-15 17:00:00    0.66548
1980-05-15 17:00:00    0.40895
```

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.

## A.4 ORDERING AND OVERLAPPING OF TIME SERIES

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

Create example data and time stamps

```
> 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. `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  0.964348
2009-05-01  0.739852
2009-04-01  1.152593
2009-03-01  0.206231
2009-02-01 -0.795679
2009-01-01  0.006977
```

Now reverse the time order

```
> rev(tS)
GMT
                TS.1
2009-01-01  0.006977
2009-02-01 -0.795679
2009-03-01  0.206231
2009-04-01  1.152593
2009-05-01  0.739852
2009-06-01  0.964348
```

Sample the time series in arbitrary time order

```
> sample(tS)
GMT
                TS.1
2009-03-01  0.206231
2009-05-01  0.739852
2009-01-01  0.006977
2009-02-01 -0.795679
2009-06-01  0.964348
2009-04-01  1.152593
```

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

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

```
> 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"
```

`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  0.54     6
2009-02-01  0.67     5
2009-03-01 -0.18     2
2009-04-01 -0.35     1
2009-04-01 -1.25     7
2009-05-01 -0.68     3
2009-06-01  0.47     4
```

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

```
> tS@recordIDs
  X1.7
6    6
5    5
2    2
1    1
7    7
3    3
4    4
```

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  0.54    6
2009-02-01  0.67    5
2009-03-01 -0.18    2
2009-04-01 -0.35    1
2009-04-01 -1.25    7
2009-05-01 -0.68    3
2009-06-01  0.47    4
> data3
```

```
[1] -0.35 -0.18 -0.68  0.47  0.67  0.54 -1.25
```

*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
```

Now create the time series

```
> tS <- timeSeries(offeredPrice, dateOfOffer, recordIDs = priceInfo)
> tS
GMT
           TS.1 providerCompany* ratingOfOffer*
2009-01-01  340         UISA Ltd            AAA
2009-02-01  320         UISA Ltd            AAA
2009-03-01  400         UISA Ltd            AAA
2009-03-01  400       HK Company            BBB
2009-04-01  410         UISA Ltd            AAB
2009-05-01  350         UISA Ltd            AAB
2009-06-01  290         UISA Ltd            AAB
```

```
> 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.

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

GMT
           TS.1 providerCompany* ratingOfOffer*
2009-01-01  340         UISA Ltd            AAA
2009-02-01  320         UISA Ltd            AAA
2009-03-01  400         UISA Ltd            AAA
2009-04-01  410         UISA Ltd            AAB
2009-05-01  350         UISA Ltd            AAB
2009-06-01  290         UISA Ltd            AAB

> 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 take care of attributes that are saved in the `@recordIDs` slot.

## A.5 BINDING AND MERGING OF TIME SERIES

*How can I bind two time series objects by row?*

To bind a time series row by row use the function `rbind()`. First create the data and time stamps

```
> 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"
```

then the time series

```
> 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?*

Create example data and time stamps

```
> 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"
```

then the time series

```
> s1 <- timeSeries(data1, as.Date(charvec1))
> s2 <- timeSeries(data2, as.Date(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
2009-03-01         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

> rbind(s2, s1)
```

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

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?*

Create example data and time stamps

```
> 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"
```

then create the time series

```
> 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
```

Note, 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?*

Create example data and time stamps

```
> 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"
```

then create the time series

```
> 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.

*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?*

Create example data and time stamps

```
> 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"
```

then create the time series

```
> 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?*

Create example data and time stamps

```
> 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"
```

then create the time series

```
> 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?*

Create example data and time stamps

```
> 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"
```

then create the time series

```
> 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?*

Create example data and time stamps

```
> 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
```

then create the time series

```
> 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?*

Create example data and time stamps

```
> 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
```

then create the time series

```
> 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?*

Create example data and time stamps

```
> 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
```

then create the time series

```
> 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
```

## A.6 SUBSETTING TIME SERIES OBJECTS

*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] -0.39674 -0.79743  0.78461  0.40683 -1.22138  0.62980
```

```
> vec[3:4]
```

```
[1] 0.78461 0.40683
```

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,] -0.44491 -0.36335  0.66098
[2,] -1.03386  0.25993 -0.74574
[3,]  0.46802 -1.52355 -1.51726
[4,] -1.07640  1.83432 -0.65494
[5,] -0.59736  0.19809  0.23294
[6,] -0.17823 -0.89844 -2.61328
```

```
> mat[3:4, ]
```

```
         [,1]    [,2]     [,3]
[1,]  0.46802 -1.5236 -1.51726
[2,] -1.07640  1.8343 -0.65494
```

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

```
         [,1]     [,2]
[1,] -0.36335  0.66098
[2,]  0.25993 -0.74574
[3,] -1.52355 -1.51726
[4,]  1.83432 -0.65494
[5,]  0.19809  0.23294
[6,] -0.89844 -2.61328
```

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

```
        [,1]     [,2]
[1,] -1.5236 -1.51726
[2,]  1.8343 -0.65494
```

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]  0.46802 -1.52355 -1.51726
> mat[, 2]
[1] -0.36335  0.25993 -1.52355  1.83432  0.19809 -0.89844
```

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?*

Create example data and time stamps

```
> 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"
```

then create the time series, first a univariate time series

```
> s <- timeSeries(data1, charvec)
> s[3:4]
[1] -0.70395 -1.67623
> s[3:4, ]
GMT
               TS.1
2009-03-01 -0.70395
2009-04-01 -1.67623
```

then a multivariate time series

```
> S <- timeSeries(data2, charvec)
> S[3:4, ]
GMT
                  A        B        C
2009-03-01 -0.42120 -0.93301  0.53239
2009-04-01 -0.61949  0.65967 -0.70406
> S[, 2:3]
GMT
                  B        C
2009-01-01  0.37339  0.26236
2009-02-01  0.87548 -1.42209
2009-03-01 -0.93301  0.53239
2009-04-01  0.65967 -0.70406
2009-05-01 -1.38376  1.23525
2009-06-01  0.95942 -0.79267
> S[3:4, 2:3]
GMT
                  B        C
2009-03-01 -0.93301  0.53239
2009-04-01  0.65967 -0.70406
```

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  0.37339  0.26236
2009-02-01  0.87548 -1.42209
2009-03-01 -0.93301  0.53239
2009-04-01  0.65967 -0.70406
2009-05-01 -1.38376  1.23525
2009-06-01  0.95942 -0.79267
> S[3:4, 2:3]
GMT
                  B        C
2009-03-01 -0.93301  0.53239
2009-04-01  0.65967 -0.70406
```

`timeSeries` objects are rectangular objects.

*Can I subset a multivariate time series by column names?*

Create example data and time stamps

```
> 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"
```

then create the time series

```
> S <- timeSeries(data, charvec)
> S[, "A"]
GMT
                   A
2009-01-01 -0.828935
2009-02-01  0.082349
2009-03-01 -0.343858
2009-04-01  1.307897
2009-05-01 -0.204417
2009-06-01 -0.213058
```

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

Create example data and time stamps

```
> 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"
```

then create the time series

```
> S <- timeSeries(data, charvec)
> S$B
[1]  0.60090  0.50337 -0.10377  0.20449  0.54399  1.14339
```

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

Create example data and time stamps

```
> 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"
```

then create the ime series

```
> 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  0.27546 1.9762 -0.84115
2009-04-01 -0.90749 1.6124  1.30982
```

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

Create example data and time stamps

```
> 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"
```

then create the time series

```
> 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 -2.016550 0.053983 -0.80077
2009-04-01 -0.051557 0.230375 -0.58837
```

*Can I subset a multivariate time series by logical predicates?*

Create example data and time stamps

```
> 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"
```

then create the time series

```
> 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 0.44876 1.421444 -0.20257
2009-05-01 1.59157 0.082545  0.71184
2009-06-01 0.19230 0.269123  0.26942
```

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

Create example data and time stamps

```
> 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"
```

then create the time series

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

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

Since `timeSeries()` also supports 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] [2009-05-01] [2009-02-01]
```

## A.7 MISSING DATA HANDLING

*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.
Create example data and time stamps

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

[1]        NA -0.034117 -0.647320  0.671190  0.020970  0.357884        NA
[8] -1.464489 -0.755933

> 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"
```

then the time series

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

GMT
                TS.1
2009-02-01 -0.034117
2009-03-01 -0.647320
2009-04-01  0.671190
2009-05-01  0.020970
2009-06-01  0.357884
2009-08-01 -1.464489
2009-09-01 -0.755933
```

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

Create example data and time stamps

```
> 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 -1.915669  1.08871
[2,] -1.96850  0.149223  0.68885
[3,]       NA        NA -0.44576
[4,]  0.64341        NA  0.54427
[5,]  0.79167 -0.728209  0.34092
[6,] -0.16961  0.046504 -0.59887
[7,] -0.42911  1.009777  1.20828

> 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"
```

then the time series data

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

GMT
               TS.1      TS.2     TS.3
2009-02-01 -1.96850  0.149223  0.68885
2009-05-01  0.79167 -0.728209  0.34092
2009-06-01 -0.16961  0.046504 -0.59887
2009-07-01 -0.42911  1.009777  1.20828
```

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

Create example data and time stamps

```
> 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.
Then let us create the time series data.
Replace for example missing values in a series of financial returns by zero values

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

```
GMT
               TS.1
2009-01-01  0.00000
2009-02-01  0.03468
2009-03-01  0.10769
2009-04-01  0.15969
2009-05-01 -2.27392
2009-06-01  0.49775
2009-07-01  0.00000
2009-08-01  1.42359
2009-09-01 -1.06590
```

by their mean values

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

```
GMT
               TS.1
2009-01-01 -0.15949
2009-02-01  0.03468
2009-03-01  0.10769
2009-04-01  0.15969
2009-05-01 -2.27392
2009-06-01  0.49775
2009-07-01 -0.15949
2009-08-01  1.42359
2009-09-01 -1.06590
```

or by their median

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

GMT
               TS.1
2009-01-01  0.10769
2009-02-01  0.03468
2009-03-01  0.10769
2009-04-01  0.15969
2009-05-01 -2.27392
2009-06-01  0.49775
2009-07-01  0.10769
2009-08-01  1.42359
2009-09-01 -1.06590
```

*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.
Create example data and time stamps

```
> 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 -0.35157 -0.60808
[2,]  1.970099  1.43206 -1.34172
[3,]        NA       NA -0.66738
[4,]  0.008771       NA  0.97534
[5,] -1.624621  1.31433  0.82137
[6,]  0.295023  0.43150  0.16356
[7,] -0.415434 -0.27018 -1.57619

> 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"
```

and then create the time series

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

GMT
                TS.1     TS.2     TS.3
2009-01-01        NA -0.35157 -0.60808
2009-02-01  1.970099  1.43206 -1.34172
2009-03-01        NA       NA -0.66738
2009-04-01  0.008771       NA  0.97534
2009-05-01 -1.624621  1.31433  0.82137
```

```
2009-06-01  0.295023  0.43150  0.16356
2009-07-01 -0.415434 -0.27018 -1.57619
```

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

```
GMT
                TS.1     TS.2     TS.3
2009-01-01  0.000000 -0.35157 -0.60808
2009-02-01  1.970099  1.43206 -1.34172
2009-03-01  0.000000  0.00000 -0.66738
2009-04-01  0.008771  0.00000  0.97534
2009-05-01 -1.624621  1.31433  0.82137
2009-06-01  0.295023  0.43150  0.16356
2009-07-01 -0.415434 -0.27018 -1.57619
```

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

Rmetrics has the function `na.omit()` with the argument `method` for selecting interpolation.
Create example data and time stamps

```
> 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"
```

and then create the time series examples
Interpolation of time series is done by calling internally the function `approx` whoch provides linear interpolation.
Interpolate and remove (trim) `NA`s 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 `NA`s 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, 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.
Create example data and time stamps

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

 [1]  0.091483  1.472572        NA  0.385343  0.347471  0.121550 -0.554277
 [8]        NA  1.073946  0.434426 -0.872392 -1.617038

> 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"
```

then create the time series

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

GMT
                TS.1
2009-01-01  0.091483
2009-02-01  1.472572
2009-03-01        NA
2009-04-01  0.385343
2009-05-01  0.347471
2009-06-01  0.121550
2009-07-01 -0.554277
2009-08-01        NA
2009-09-01  1.073946
2009-10-01  0.434426
2009-11-01 -0.872392
2009-12-01 -1.617038

> na.contiguous(s)

GMT
               TS.1
2009-04-01  0.38534
2009-05-01  0.34747
2009-06-01  0.12155
2009-07-01 -0.55428
```

# APPENDIX B

# FAQ: BASE, STATS, AND UTILITY FUNCTIONS

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

## B.1 FUNCTIONS AND METHODS FROM BASE R

*How can I calculate the column means*

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?
Create a multivariate example time Series

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

and use the apply function to calculate the mean

```
> apply(S, 1, mean)
GMT
              TS.1
2009-06-01 0.67128
2009-05-01 0.48323
2009-04-01 0.54613
2009-03-01 0.67537
2009-02-01 0.58386
2009-01-01 0.22103
```

*How can I attach a time series to the search path*

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.
Create a multivariate example time series

```
> 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"
> S <- timeSeries(data, charvec, units = c("A", "B", "C"))
```

and attach it to the search path

```
> attach(S)
> colnames(S)
[1] "A" "B" "C"
> B
[1] 0.57402 0.57365 0.59127 0.96524 0.59316 0.23193
```

*How can I create a multivariate time series and difference it*

The function `diff()` returns suitably lagged and iterated differences.
Create an univariate and multivariate example time series

```
> 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"
> S <- timeSeries(data, charvec)
> s <- S[, 1]
```

and differnece the series

```
> diff(S)
GMT
                TS.1        TS.2     TS.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
                TS.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
```

*How can I compute the dimensions of a time series*

The function `dim()` retrieves or sets the dimension of an object.
Create an univariate and multivariate example time series

```
> 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"
> S <- timeSeries(data, charvec)
> s <- S[, 1]
```

and then compute the dimension of the series

```
> dim(S)
[1] 6 3
> dim(s)
[1] 6 1
```

*How can I compute the sample ranks of a time series*

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.
Create an univariate and multivariate example time series

```
> 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"
> S <- timeSeries(data, charvec)
> s <- S[, 1]
```

and then compute the ranks

```
> rank(S)
```

```
GMT
           TS.1 TS.2 TS.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
           TS.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
```

*How can I reverse a time series*

The function `rev()` provides a reversed version of its argument.
Create an univariate and multivariate example time series

```
> 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"
> S <- timeSeries(data, charvec)
> s <- S[, 1]
```

and then reverse the time order of the series

```
> rev(S)
GMT
              TS.1    TS.2    TS.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
              TS.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
```

*How can I sample randomly the time stamps of a series*

The function `sample()` takes a sample of the specified size from the elements of `x` using either with or without replacement.
Create an univariate and multivariate example time series

```
> 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"
> S <- timeSeries(data, charvec)
> s <- S[, 1]
```

and then sample randomly the time order of the series

```
> sample(S)
GMT
              TS.1    TS.2    TS.3
2009-06-01 0.50875 0.57402 0.93106
2009-02-01 0.66028 0.59316 0.49813
2009-05-01 0.47702 0.57365 0.39901
2009-03-01 0.44730 0.96524 0.61359
2009-04-01 0.18291 0.59127 0.86422
2009-01-01 0.22283 0.23193 0.20833
> sample(s)
GMT
              TS.1
2009-04-01 0.18291
2009-01-01 0.22283
2009-06-01 0.50875
2009-03-01 0.44730
2009-02-01 0.66028
2009-05-01 0.47702
```

*How can I scale a time series*

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 series sobjects?
Create an univariate and multivariate example time series

```
> 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"
> S <- timeSeries(data, charvec)
> s <- S[, 1]
```

and then scale the series

```
> scale(S)
GMT
               TS.1      TS.2     TS.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
               TS.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
```

*How can I sort a time series*

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.
Create a randomly sampled univariate and multivariate example time series

```
> set.seed(1953)
> data <- matrix(runif(18), ncol = 3)
> charvec <- rev(paste("2009-0", sample(1:6), "-01", sep = ""))
> charvec
[1] "2009-06-01" "2009-03-01" "2009-04-01" "2009-02-01" "2009-05-01"
[6] "2009-01-01"
> S <- timeSeries(data, charvec)
> s <- S[, 1]
```

and then sort the time order of the series

```
> sort(S)
GMT
              TS.1    TS.2    TS.3
2009-01-01 0.22283 0.23193 0.20833
2009-02-01 0.44730 0.96524 0.61359
2009-03-01 0.47702 0.57365 0.39901
2009-04-01 0.18291 0.59127 0.86422
2009-05-01 0.66028 0.59316 0.49813
2009-06-01 0.50875 0.57402 0.93106
> sort(s)
```

```
GMT
              TS.1
2009-01-01 0.22283
2009-02-01 0.44730
2009-03-01 0.47702
2009-04-01 0.18291
2009-05-01 0.66028
2009-06-01 0.50875
```

### *How can I extract the start and end dates of a series*

The functions `start()` and `end()` extract and encode the times the first and last observations were taken.

```
> 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"
> S <- timeSeries(data, charvec)
> s <- S[, 1]


> start(S)
GMT
[1] [2009-01-01]
> end(S)
GMT
[1] [2009-06-01]
> start(s)
GMT
[1] [2009-01-01]
> end(s)
GMT
[1] [2009-06-01]
```

## B.2 FUNCTIONS AND METHODS FROM STATS R

### *How can I model an univariate time series by an ARIMA process*

The function `arima()` fits an ARIMA model to a univariate time series. Create ARMA(2,1) process

```
> s <- as.timeSeries(arima.sim(n = 300, list(ar = c(0.8, -0.4),
     ma = -0.2)))
> head(s)
```

```
         SS.1
[1,]  0.61336
[2,] -1.31480
[3,] -2.60232
[4,] -0.58109
[5,]  0.33612
[6,] -0.12538
```

Estimate the parameters

```
> arima(series(s))

Call:
arima(x = series(s))

Coefficients:
      intercept
         -0.066
s.e.      0.063

sigma^2 estimated as 1.19:  log likelihood = -451.25,  aic = 906.5
```

*How can I compute the autocorrelation function of a time series*

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.
Create ARMA(2,1) process

```
> s <- as.timeSeries(arima.sim(n = 60, list(ar = c(0.8, -0.4),
     ma = -0.2)))
> head(s)

         SS.1
[1,] -0.11637
[2,] -0.36539
[3,] -2.59765
[4,]  0.71924
[5,] -0.51497
[6,]  0.58280
```

Compute the autocorrelations

```
> print(acf(series(s)))

Autocorrelations of series 'series(s)', by lag

     0      1      2      3      4      5      6      7      8      9     10 
 1.000  0.341 -0.091 -0.412 -0.252  0.094  0.248  0.148 -0.137 -0.295 -0.225 
    11     12     13     14     15     16     17 
 0.066  0.138  0.110 -0.070 -0.032  0.034  0.129
```

*How can I compute the covariance of a time series*

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.
Create example multivariate time series

```
> 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"

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

and compute variances and covariances

```
> var(s)

       SS.1
SS.1 1.3397

> var(S)

          TS.1     TS.2      TS.3
TS.1 0.0329245 0.015783 0.0016191
TS.2 0.0157826 0.053906 0.0286396
TS.3 0.0016191 0.028640 0.0765103

> cov(S)

          TS.1     TS.2      TS.3
TS.1 0.0329245 0.015783 0.0016191
TS.2 0.0157826 0.053906 0.0286396
TS.3 0.0016191 0.028640 0.0765103

> cor(S)

         TS.1    TS.2     TS.3
TS.1 1.000000 0.37463 0.032259
TS.2 0.374627 1.00000 0.445951
TS.3 0.032259 0.44595 1.000000
```

*How can I compute the distance matrix of a time series records?*

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.
Create example multivariate time series

```
> 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"
```

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

and create distance matrix from columns

```
> dist(t(S))
        TS.1    TS.2
TS.2 0.67321        
TS.3 0.83831 0.60475
```

*How can I xompute the normal distribution from time series returns*

The function `dnorm()` computes the density for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`.
Create example time series points

```
> set.seed(1953)
> data <- seq(-4, 4, length = 11)
> s <- timeSeries(data)
```

and derive the normal density

```
> dnorm(s, mean = 0, sd = 1)
            SS.1
 [1,] 0.00013383
 [2,] 0.00238409
 [3,] 0.02239453
 [4,] 0.11092083
 [5,] 0.28969155
 [6,] 0.39894228
 [7,] 0.28969155
 [8,] 0.11092083
 [9,] 0.02239453
[10,] 0.00238409
[11,] 0.00013383
```

*How can I apply the filter function to generate a series*

The function `filter()` applies linear filtering to a univariate time series or to each series separately of a multivariate time series.

```
> filter(s, rep(1, 3))
             SS.1
 [1,]          NA
 [2,] -9.6000e+00
 [3,] -7.2000e+00
 [4,] -4.8000e+00
 [5,] -2.4000e+00
 [6,]  8.8818e-16
 [7,]  2.4000e+00
 [8,]  4.8000e+00
 [9,]  7.2000e+00
[10,]  9.6000e+00
[11,]          NA
```

*How can I compute 'fivenum' statistics*

The function `fivenum` returns Tukey's five number summary (minimum, lower-hinge, median, upper-hinge, maximum) for the input data.
Create example univariate (signal) time series

```
> set.seed(1953)
> data <- rnorm(100)
> s <- timeSeries(data)
```

and compute statistics

```
> fivenum(s)

[1]  0.021925  0.428334 -0.475240 -1.029249 -0.440123
```

*How can I compute a histogram from a time series*

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.

```
> hist(s)$density

 [1] 0.02 0.06 0.14 0.16 0.16 0.42 0.54 0.28 0.18 0.00 0.02 0.02
```

*How can I create a lagged time series*

The function `lag()` computes a lagged version of a time series, shifting the time base back by a given number of observations.

```
> lag(S)

GMT
           TS.1[1] TS.2[1] TS.3[1]
2009-06-01      NA      NA      NA
2009-05-01 0.50875 0.57402 0.93106
2009-04-01 0.47702 0.57365 0.39901
2009-03-01 0.18291 0.59127 0.86422
2009-02-01 0.44730 0.96524 0.61359
2009-01-01 0.66028 0.59316 0.49813
```

*Ho can I fit a linear model*

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).
Create example multivariate time series

```
> 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"

> S <- timeSeries(data, charvec, units = c("Y", "X1", "X2"))


> fit <- lm(Y ~ X1 + X2, data = S)
> fit

Call:
lm(formula = Y ~ X1 + X2, data = S)

Coefficients:
(Intercept)           X1           X2
      0.274        0.351       -0.110

> resid(fit)

2009-06-01 2009-05-01 2009-04-01 2009-03-01 2009-02-01 2009-01-01
  0.135337   0.045015  -0.203938  -0.098638   0.232361  -0.110136
```

*How can I smooth a series using the 'lowess' scatter plot smoother*

The function `lowess()` performs the computations for the LOWESS smoother which uses locally-weighted polynomial regression.

```
> lowess(s)

$x
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
 [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 [91]  91  92  93  94  95  96  97  98  99 100

$y
  [1]  0.1179250  0.1118441  0.1057657  0.0997007  0.0936600  0.0876530
  [7]  0.0816881  0.0757710  0.0698986  0.0640595  0.0582426  0.0524428
 [13]  0.0466578  0.0408846  0.0351216  0.0293720  0.0236454  0.0179591
 [19]  0.0123284  0.0067649  0.0012768 -0.0041260 -0.0094351 -0.0146469
 [25] -0.0197635 -0.0247880 -0.0297297 -0.0346106 -0.0394613 -0.0442999
 [31] -0.0491108 -0.0538220 -0.0582231 -0.0584108 -0.0579602 -0.0569444
 [37] -0.0556203 -0.0543164 -0.0532951 -0.0527357 -0.0527901 -0.0534225
 [43] -0.0544473 -0.0557336 -0.0572059 -0.0587451 -0.0602446 -0.0615808
 [49] -0.0625505 -0.0630069 -0.0630306 -0.0627935 -0.0624715 -0.0622272
 [55] -0.0621862 -0.0623679 -0.0627269 -0.0631599 -0.0636146 -0.0640057
 [61] -0.0641757 -0.0639412 -0.0633209 -0.0624852 -0.0615202 -0.0604721
 [67] -0.0594581 -0.0585005 -0.0554490 -0.0515963 -0.0473922 -0.0430809
 [73] -0.0387755 -0.0345164 -0.0303187 -0.0261863 -0.0221079 -0.0180664
 [79] -0.0140465 -0.0100283 -0.0059911 -0.0019229  0.0021844  0.0063451
 [85]  0.0105793  0.0149116  0.0193675  0.0239693  0.0287322  0.0336673
 [91]  0.0387822  0.0440849  0.0495784  0.0552621  0.0611323  0.0671909
 [97]  0.0734420  0.0798849  0.0865144  0.0933243
```

*How can I compute the mean absolute deviations*

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.

```
> mad(s)
[1] 1.4331
```

*How can I compute the median*

The function `median()` computes the sample median.

```
> median(s)
[1] -0.47524
```

*How can I create a quantile-quantile plot*

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.

```
> print(qqnorm(s))
$x
  [1] -0.037608 -0.859617  0.426148  0.037608  0.189118  0.214702  1.959964
  [8]  1.310579 -0.062707 -1.103063 -0.189118  0.012533 -0.138304 -2.170090
 [15] -0.658838 -1.253565  1.103063  0.789192  2.575829 -0.345126  0.062707
 [22]  0.597760  0.658838 -0.426148  0.240426  0.859617  0.138304 -1.200359
 [29] -1.514102  0.896473 -0.163658 -0.690309  1.150349  0.318639  0.163658
 [36] -1.310579 -0.266311 -1.811911  0.266311 -0.896473  0.345126  0.371856
 [43] -1.058122 -0.538836 -0.371856  1.253565 -0.292375  1.372204 -0.113039
 [50]  0.628006 -1.150349  2.170090 -2.575829 -0.823894  0.481727 -0.318639
 [57] -0.398855  1.514102  0.722479  0.568051  0.113039  1.695398 -1.959964
 [64] -0.755415  0.398855 -0.722479  1.439531 -1.372204 -1.439531  0.087845
 [71]  0.690309 -0.510073  0.974114  0.823894 -1.598193 -0.597760  0.453762
 [78] -0.934589 -0.628006  1.811911  0.538836 -1.015222 -0.789192 -1.695398
 [85] -0.453762 -0.087845  1.200359 -0.012533  0.510073 -0.481727  1.058122
 [92]  0.292375  1.015222 -0.974114 -0.214702  0.755415  0.934589 -0.240426
 [99]  1.598193 -0.568051

$y
             SS.1
  [1,]  0.0219246
  [2,] -0.9043255
  [3,]  0.4132378
  [4,]  0.1866212
  [5,]  0.2308177
  [6,]  0.2356802
  [7,]  1.4837389
  [8,]  1.0994627
  [9,] -0.0046855
 [10,] -1.4211978
 [11,] -0.1326078
 [12,]  0.1647963
 [13,] -0.0755139
 [14,] -2.3306995
 [15,] -0.5232119
 [16,] -1.5177505
 [17,]  0.7917815
 [18,]  0.5702386
 [19,]  2.5049592
 [20,] -0.2599761
 [21,]  0.2015900
 [22,]  0.4736224
 [23,]  0.4941460
 [24,] -0.3687956
 [25,]  0.2365270
 [26,]  0.6201417
 [27,]  0.2098003
 [28,] -1.4903779
 [29,] -1.6032132
 [30,]  0.6542039
 [31,] -0.1090904
 [32,] -0.6485312
 [33,]  0.7964054
 [34,]  0.3176671
 [35,]  0.2305061
 [36,] -1.5228711
 [37,] -0.1986330
 [38,] -2.0670249
 [39,]  0.2459147
 [40,] -1.0006743
 [41,]  0.3346520
 [42,]  0.3607380
 [43,] -1.3588743
 [44,] -0.4297557
 [45,] -0.3457670
 [46,]  1.0027721
 [47,] -0.2182967
 [48,]  1.1060465
 [49,] -0.0444201
 [50,]  0.4914047
```

```
 [51,] -1.4418857
 [52,]  2.0744036
 [53,] -2.5492092
 [54,] -0.8574600
 [55,]  0.4468640
 [56,] -0.2411536
 [57,] -0.3664824
 [58,]  1.1999795
 [59,]  0.5113627
 [60,]  0.4694576
 [61,]  0.2093195
 [62,]  1.2756392
 [63,] -2.1947838
 [64,] -0.7763209
 [65,]  0.3732846
 [66,] -0.6907615
 [67,]  1.1455887
 [68,] -1.5655148
 [69,] -1.5795134
 [70,]  0.2059036
 [71,]  0.5029462
 [72,] -0.4230215
 [73,]  0.6697981
 [74,]  0.5941339
 [75,] -1.6125955
 [76,] -0.4459029
 [77,]  0.4150787
 [78,] -1.0811459
 [79,] -0.5060688
 [80,]  1.3544389
 [81,]  0.4680967
 [82,] -1.2928523
 [83,] -0.8286872
 [84,] -1.9706835
 [85,] -0.3976714
 [86,] -0.0201497
 [87,]  0.8832659
 [88,]  0.1466988
 [89,]  0.4610524
 [90,] -0.4184609
 [91,]  0.7092965
 [92,]  0.2486696
 [93,]  0.6717223
 [94,] -1.2478298
 [95,] -0.1579903
 [96,]  0.5480112
 [97,]  0.6680581
 [98,] -0.1785927
 [99,]  1.2580994
[100,] -0.4401231
```

*How can I smooth an univariate time series*

The functions `smooth()` performs running median smoothing. The function implements Tukey's smoothers, 3RS3R, 3RSS, 3R, etc.

```
> smooth(s)
3RS3R Tukey smoother resulting from  smooth(x = s)
 used 7 iterations
  [1]  0.0219246  0.0219246  0.1866212  0.2308177  0.2308177  0.2356802
  [7]  0.2454052  0.2454052 -0.0046855 -0.0755139 -0.1326078 -0.1326078
 [13] -0.5232119 -0.5232119 -0.5232119 -0.5232119  0.5702386  0.5702386
 [19]  0.5702386  0.5702386  0.4736224  0.2365270  0.2365270  0.2365270
 [25]  0.2365270  0.2365270  0.2098003  0.1563468 -0.1090904 -0.1090904
 [31] -0.1090904 -0.1090904 -0.1090904  0.2305061  0.2305061 -0.1986330
 [37] -0.1986330 -0.1986330 -0.1986330  0.2459147  0.2459147  0.2459147
 [43] -0.3457670 -0.3457670 -0.3457670 -0.2182967 -0.0444201 -0.0444201
 [49] -0.0444201 -0.0444201 -0.0444201 -0.8574600 -0.8574600 -0.8574600
 [55] -0.2411536 -0.2411536 -0.2411536 -0.2411536  0.4694576  0.4694576
 [61]  0.4694576  0.2093195 -0.3109567 -0.6907615 -0.6907615 -0.6907615
 [67] -0.6907615 -0.6907615  0.2059036  0.2059036  0.2059036  0.5029462
 [73]  0.5029462 -0.4459029 -0.4459029 -0.4459029 -0.4459029 -0.4459029
 [79] -0.4459029 -0.5060688 -0.8286872 -0.8286872 -0.8286872 -0.8286872
 [85] -0.3976714 -0.0201497  0.1466988  0.1466988  0.2486696  0.2486696
 [91]  0.2486696  0.2486696  0.2486696  0.2486696  0.5480112  0.5480112
 [97]  0.5480112  0.5480112 -0.1785927 -0.4401231
```

*How can I compute the spectrum of an univariate time series*

The function `spectrum()` estimates the spectral density of a time series.

```
> Spectrum <- spectrum(series(s))
> print(Spectrum[1:2])
$freq
 [1] 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10 0.11 0.12 0.13 0.14 0.15
[16] 0.16 0.17 0.18 0.19 0.20 0.21 0.22 0.23 0.24 0.25 0.26 0.27 0.28 0.29 0.30
[31] 0.31 0.32 0.33 0.34 0.35 0.36 0.37 0.38 0.39 0.40 0.41 0.42 0.43 0.44 0.45
[46] 0.46 0.47 0.48 0.49 0.50

$spec
 [1] 0.165450 0.299767 0.753888 0.517609 0.172019 0.788075 3.250752 1.683149
 [9] 0.077400 1.480422 0.413807 0.480775 2.623808 0.124969 3.175454 0.339911
[17] 0.209422 0.449005 1.614554 1.845092 0.290587 0.385798 0.301470 0.146897
[25] 0.102949 1.370152 1.602608 2.346038 0.087254 0.046988 1.402371 3.904061
[33] 1.902128 0.192613 0.266849 0.104446 0.130845 0.430708 0.949801 1.272472
[41] 0.565025 0.233372 0.247837 0.769138 0.522743 3.302462 0.256360 1.805771
[49] 2.957782 0.706080
```

# APPENDIX C

# FAQ: PRINTING AND PLOTTING

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

## C.1 PRINTING TIME SERIES OBJECTS

*How can I print a time series object?*

Time series objects are displayed in the usual way as other data objects in
R
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"
```

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
```

Alternatively we can use explicitly 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"
```

Print the time series

```
> 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?*

Generate the 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"
```

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 ?*

Generate the 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"
```

The generic `print()` function for the `timeSeries` class allows to customize the printing format. Here comes an example

```
> 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?*

The print method for `timeSeries` objects has a `style="ts"` argument. 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?*

The print method for `timeSeries` objects has a `FinCenter=NULL` argument. 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  0.078507
2009-02-01 16:00:00  0.856417
2009-03-01 16:00:00 -0.221869
2009-04-01 16:00:00  0.079857
2009-05-01 16:00:00 -0.521955
2009-06-01 16:00:00  0.427480

> print(s, FinCenter = "Zurich")

Zurich
                         TS.1
2009-01-01 23:00:00  0.078507
2009-02-01 23:00:00  0.856417
2009-03-01 23:00:00 -0.221869
2009-04-01 23:00:00  0.079857
2009-05-01 23:00:00 -0.521955
2009-06-01 23:00:00  0.427480

> print(s, FinCenter = "Tokyo")

Tokyo
                         TS.1
2009-01-02 07:00:00  0.078507
2009-02-02 07:00:00  0.856417
2009-03-02 07:00:00 -0.221869
2009-04-02 06:00:00  0.079857
```

```
2009-05-02 06:00:00 -0.521955
2009-06-02 06:00:00  0.427480
```

## C.2 PLOTTING TIME SERIES OBJECTS

*How can I plot a univariate time series using the generic plot function?*

Generate the 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"
```

and plot them
`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?*

Generate the 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"
```

and plot them

```
> S <- timeSeries(data, charvec)
> plot(S)
```

*How can I plot a multivariate time series in a single chart using the generic plot function?*

Generate the 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"
```

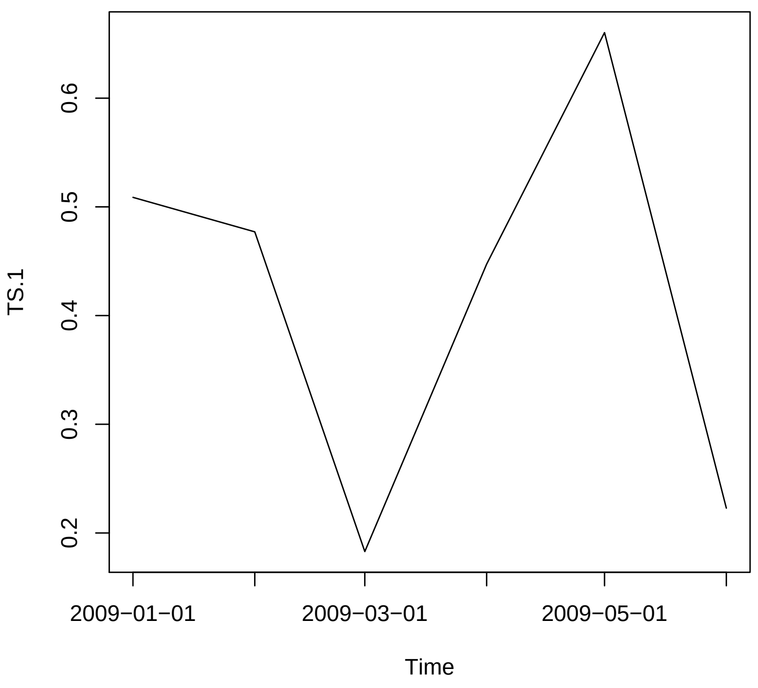

FIGURE C.1: Plot 1 - Example of a single panel plot

and plot them

```
> S <- timeSeries(data, as.POSIXct(charvec, FinCenter = "GMT"))
> plot(S, plot.type = "single")
```

*How can I add a line to an existing plot?*

Generate the data

```
> set.seed(1953)
```

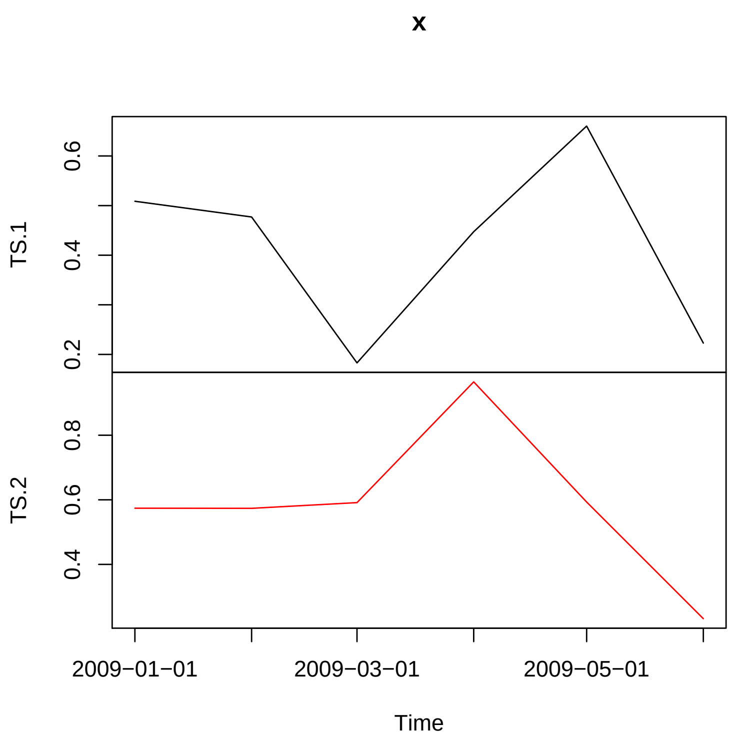

FIGURE C.2: Plot 2b - Multivariate Time Series plots in multiple graphs

```
> 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"
```

and plot them

```
> s1 <- timeSeries(data1, charvec, FinCenter = "GMT")
> s2 <- timeSeries(data2, charvec, FinCenter = "GMT")
```

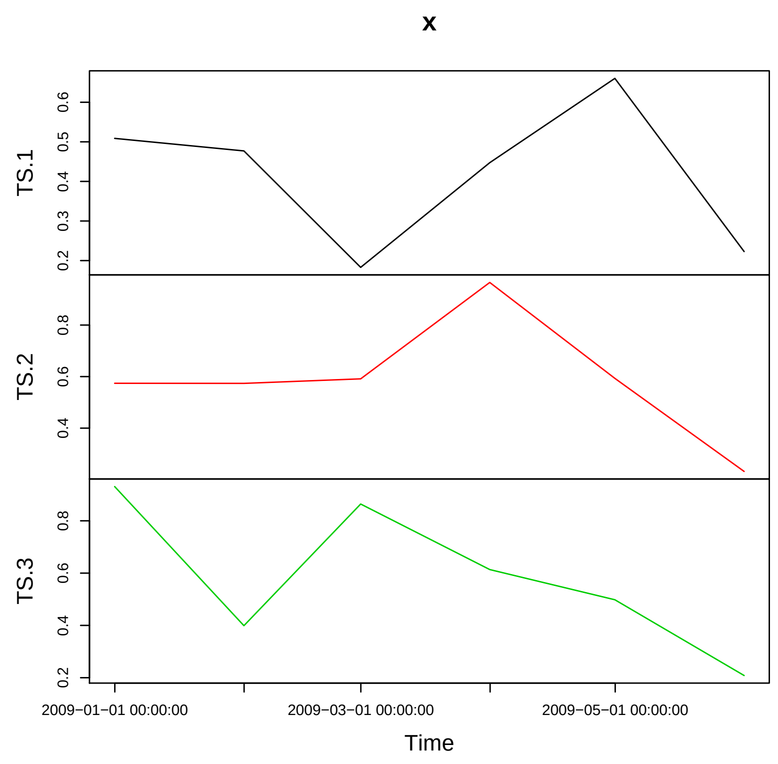

FIGURE C.3: Plot 3b - Example of a single panel plot from timeSeries

```
> plot(s1)
> lines(s2, col = 2)
```

*How can I add points to an existing plot?*

generate the 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"
```

and plot them

```
> s <- timeSeries(data, charvec, FinCenter = "GMT")
> plot(s)
> points(s, col = 2, pch = 19)
```

*Can I use 'abline' for time series plots?*

Generate the 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"
```

and plot them

```
> s <- timeSeries(data, charvec, FinCenter = "GMT")
> plot(s)
> abline(v = time(s), col = "darkgrey", lty = 3)
```

# APPENDIX D

# PACKAGES REQUIRED FOR THIS EBOOK

```
> library(timeDate)
> library(timeSeries)
> library(fBasics)
```

In the following we briefly describe the packages required for this ebook.

## D.1 RMETRICS PACKAGE: `timeDate`

`timeDate` (Würtz & Chalabi, 2009a) contains `R` functions to handle time, date and calender aspects. The S4 `timeDate` class is used in Rmetrics for financial data and time management together with the management of public and ecclesiastical holidays. The class fulfils the conventions of the ISO 8601 standard as well as of the ANSI C and POSIX standards. Beyond these standards, Rmetrics has added the 'Financial Center' concept, which allows you to handle data records collected in different time zones and combine them with the proper time stamps of your personal financial center, or, alternatively, to the GMT reference time. The S4 class can also handle time stamps from historical data records from the same time zone, even if the financial centers changed daylight saving times at different calendar dates. Moreover, `timeDate` is almost compatible with Insightful's SPlus `timeDate` class. If you move between the two worlds of `R` and SPlus, you will not have to rewrite your code. This is important for many business applications. The class offers not only date and time functionality, but also sophisticated calendar manipulations for business days, weekends, public and ecclesiastical holidays. `timeSeries` can be downloaded from the CRAN server. Development versions are also available from the R-forge repository.

```
> listDescription(timeDate)

Package: timeDate
Title: Rmetrics - Chronological and Calendar Objects
Date: 2013-10-30
Version: 3011.99
Author: Rmetrics Core Team, Diethelm Wuertz [aut], Tobias Setz
       [cre], Yohan Chalabi [ctb], Martin Maechler [ctb], W. Byers
       [ctb]
Maintainer: Tobias Setz <tobias.setz@rmetrics.org>
Description: Environment for teaching "Financial Engineering and
       Computational Finance".
Depends: R (>= 2.15.1), graphics, utils, stats, methods
Suggests: date, RUnit
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.
LazyData: yes
License: GPL (>= 2)
URL: https://www.rmetrics.org
Built: R 3.1.1; ; 2014-10-24 19:22:09 UTC; unix
```

## D.2 RMETRICS PACKAGE: `timeSeries`

`timeSeries` (Würtz & Chalabi, 2009b) is the Rmetrics package that allows us to work very efficiently with S4 `timeSeries` objects. Let us briefly summarize the major functions available in this package. You can create `timeSeries` objects in several different ways, i.e. you can create them from scratch or you can read them from a file. You can print and plot these objects, and modify them in many different ways. Rmetrics provides functions that compute financial returns from price/index series or the cumulated series from returns. Further modifications deal with drawdowns, durations, spreads, midquotes and may other special series. `timeSeries` objects can be subset in several ways. You can extract time windows, or you can extract start and end data records, and you can aggregate the records on different time scale resolutions. Time series can be ordered and resampled, and can be grouped according to statistical approaches. You can apply dozens of math operations on time series. `timeSeries` can also handle missing values.

```
> listDescription(timeSeries)

Package: timeSeries
Title: Rmetrics - Financial Time Series Objects
Date: 2014-10-22
Version: 3011.98
Author: Rmetrics Core Team, Diethelm Wuertz [aut], Tobias Setz
       [cre], Yohan Chalabi [ctb]
Maintainer: Tobias Setz <tobias.setz@rmetrics.org>
Description: Environment for teaching "Financial Engineering and
       Computational Finance".
```

```
Depends: R (>= 2.10), graphics, grDevices, stats, methods, utils,
       timeDate (>= 2150.95)
Suggests: robustbase, RUnit, xts, PerformanceAnalytics, fTrading
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.
LazyData: yes
License: GPL (>= 2)
URL: http://www.rmetrics.org
Built: R 3.1.1; ; 2014-10-24 19:22:18 UTC; unix
```

## D.3 RMETRICS PACKAGE: fBasics

fBasics (Würtz, 2009) provides basic functions to analyze and to model data sets of financial time series. The topics from this package include distribution functions for the generalized hyperbolic distribution, the stable distribution, and the generalized lambda distribution. Beside the functions to compute density, probabilities, and quantiles, you can find there also random number generators, functions to compute moments and to fit the distributional parameters. Matrix functions, functions for hypothesis testing, general utility functions and plotting functions are further important topics of the package.

```
> listDescription(fBasics)
Package: fBasics
Title: Rmetrics - Markets and Basic Statistics
Date: 2014-10-29
Version: 3011.87
Author: Rmetrics Core Team, Diethelm Wuertz [aut], Tobias Setz
       [cre], Yohan Chalabi [ctb]
Maintainer: Tobias Setz <tobias.setz@rmetrics.org>
Description: Environment for teaching "Financial Engineering and
       Computational Finance".
Depends: R (>= 2.15.1), timeDate, timeSeries
Imports: gss, stabledist, MASS
Suggests: methods, spatial, RUnit, tcltk, akima
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.
LazyData: yes
License: GPL (>= 2)
URL: https://www.rmetrics.org
Packaged: 2014-10-29 17:34:48 UTC; Tobi
NeedsCompilation: yes
Repository: CRAN
Date/Publication: 2014-10-29 20:07:26
Built: R 3.1.2; x86_64-apple-darwin13.4.0; 2014-10-31 05:04:06
       UTC; unix
```

APPENDIX E

# R MANUALS ON CRAN

The R core team creates several manuals for working with R[1]. The platform dependent versions of these manuals are part of the respective R installations. They can be downloaded as PDF files from the URL given below or directly browsed as HTML.

```
http://cran.r-project.org/manuals.html
```

The following manuals are available:

- An Introduction to R is based on the former "Notes on R", gives an introduction to the language and how to use R for doing statistical analysis and graphics.
- R Data Import/Export describes the import and export facilities available either in R itself or via packages which are available from CRAN.
- R Installation and Administration.
- Writing R Extensions covers how to create your own packages, write R help files, and the foreign language (C, C++, Fortran, ...) interfaces.
- A draft of The R language definition documents the language per se. That is, the objects that it works on, and the details of the expression evaluation process, which are useful to know when programming R functions.
- R Internals: a guide to the internal structures of R and coding standards for the core team working on R itself.
- The R Reference Index: contains all help files of the R standard and recommended packages in printable form.

[1]The manuals are created on Debian Linux and may differ from the manuals for Mac or Windows on platform-specific pages, but most parts will be identical for all platforms.

The LaTeX or texinfo sources of the latest version of these documents are contained in every R source distribution. Have a look in the subdirectory doc/manual of the extracted archive.
The HTML versions of the manuals are also part of most R installations. They are accessible using function `help.start()`.

# APPENDIX F

# RMETRICS ASSOCIATION

Rmetrics is a non-profit taking association founded in 2007 in Zurich working in the public interest. Regional bodies include the Rmetrics Asia Chapter. Rmetrics provides support for innovations in statistical computing. Starting with the Rmetrics Open Source code libraries which have become a valuable tool for education and business Rmetrics has developed a wide variety of activities.

- **Rmetrics Research:** supporting research activities done by the Econophysics group at the Institute for Theoretical Physics at ETH Zurich.
- **Rmetrics Software:** maintaining high quality open source code libraries.
- **Rmetrics Publishing:** publication of various Rmetrics books as well as from contributed authors.
- **Rmetrics Events:** organizing lectures, trainings and workshops on various topics.
- **Rmetrics Juniors:** helping companies to find students for interim jobs such as reviewing and checking code for higher quality or statistical analyses of various problems.
- **Rmetrics Stability:** licensing of stability signals and indicators to describe changing environments.

## RMETRICS RESEARCH

The Rmetrics Association is mainly run by the researchers working at the Econophysics group at the Institute for Theoretical Physics at ETH Zurich. Research activities include:

- PhD, Master, Bachelor and Semester Theses

- Papers and Articles
- Presentations on international conferences
- Sponsored and tailored theses for companies
- Paid student internships at ETH Zurich

## RMETRICS SOFTWARE

Without the Rmetrics Open Source Software Project it wouldn't be possible to realize all the research projects done in the Econophysics Group at ETH in such a short time . But it is not only the Econophysics group who has profited from the Open Source Rmetrics Software, there are worldwide many other research institutes and companies that are using Rmetrics Software.

The Rmetrics Software environment provides currently more than 40 R packages authored and maintained by 22 developers from all over the world. Amongst others it includes topics about basic statistics, portfolio optimization, option pricing as well as ARMA and GARCH processes.

An "ohloh" evaluation in 2012 of the Rmetrics Software concluded with the following results:

- Mature, well-established codebase
- Large, active development team
- Extremly well-documented source
- Cocomo project cost estimation
  - Codebase: 367'477 Lines
  - Effort: 97 Person Years
  - Estimated Cost: USD 5'354'262

This powerful software environment is freely available for scientific research and even for commerical applications.

## RMETRICS PUBLISHING

Rmetrics Publishing is an electronic publishing project with a bookstore [1] offering textbooks, handbooks, conference proceedings, software user guides and manuals related to R in finance and insurance. Most of the

[1] http://finance.e-bookshelf.ch/

books can be ordered and downloaded for free. The bookstore is sponsored by the Rmetrics Association and the ETH spin-off company Finance Online. For contributed authors our bookstore offers a peer-reviewing process and a free platform to publish and to distribute books without transfering their copyright to the publisher. You can find a list of our books on page ii.

### RMETRICS EVENTS

Trainings and Seminars are offered by Rmetrics for the most recent developments in R. Topics include all levels of knowledge:

- Basic R programming
- Advanced R project management
- Efficiently debugging code
- Speeding up code by e.g. byte compiling or using foreign language interfaces
- Managing big data
- Professional reporting by e.g. using R Sweave, knitr, Markdown or interactive R Shiny web applications and presentations

There also exists an Rmetrics Asia Chapter for teaching and training R with its home in Mumbai, India.
Besides that Rmetrics organizes a yearly international summer school together with a workshop for users and developers.

### RMETRICS JUNIORS

The Rmetrics Juniors initiative helps companies to find students for interim jobs. This ranges from reviewing and checking code for higher quality, to building R projects from scratch, to statistical analyses of inhouse problems and questions. The work is done by experienced Rmetrics Juniors members, usually Master or PhD thesis students. This is an advisory concept quite similar to that offered by ETH Juniors.

### RMETRICS STABILITY

Analyzing and enhancing the research results from the Econophysics Group at ETH Zurich and other research institutions worldwide, the Rmetrics Association implements stability and thresholding indicators. These indicators can then be licensed by industry.

In this context it is important to keep in mind that Rmetrics is an independent non-profit taking association. With the money we earn from the stability project, we support open source software projects, student internships, summer schools, and PhD student projects.

# APPENDIX G

# RMETRICS TERMS OF LEGAL USE

*Grant of License*

*Rmetrics Association* (Zurich) and *Finance Online* (Zurich) have authorized you to download one copy of this electronic book (eBook). The service includes free updates for the period of one year. *Rmetrics Association* (Zurich) and *Finance Online* (Zurich) grant you a nonexclusive, nontransferable license to use this eBook according to the terms and conditions specified in the following. This License Agreement permits you to install and use the eBook for your personal use only.

*Restrictions*

You shall not resell, rent, assign, timeshare, distribute, or transfer all or any part of this eBook (including code snippets and functions) or any rights granted hereunder to any other person.
You shall not duplicate this eBook, except for a single backup or archival copy. You shall not remove any proprietary notices, labels, or marks from this eBook and transfer to any other party.
The code snippets and functions provided in this book are for teaching and educational research, i.e. for non commercial use. It is not allowed to use the provided code snippets and functions for any commercial use. This includes workshops, seminars, courses, lectures, or any other events.
The unauthorized use or distribution of copyrighted or other proprietary content from this eBook is illegal.

*Intellectual Property Protection*

This eBook is owned by the *Rmetrics Association* (Zurich) and *Finance Online* (Zurich) and is protected by international copyright and other intellectual property laws.

*Rmetrics Association* (Zurich) and *Finance Online* (Zurich) reserve all rights in this eBook not expressly granted herein. This license and your right to use this eBook terminates automatically if you violate any part of this agreement. In the event of termination, you must destroy the original and all copies of this eBook.

*General*

This agreement constitutes the entire agreement between you and *Rmetrics Association* (Zurich) and *Finance Online* (Zurich) and supersedes any prior agreement concerning this eBook. This Agreement is governed by the laws of Switzerland.

# BIBLIOGRAPHY

Bateman, R. (2000). Time functionality in the standard c library. *Novell AppNotes*, 73–84.

ISO-8601 (1988). *Data Elements and Interchange Formats - Information Interchange, Representation of Dates and Time*, international organization for standardization, reference number iso 8601, 14 pages.

Würtz, D. (2009). *The fBasics Package*. cran.r-project.org.

Würtz, D. & Chalabi, Y. (2009a). *The timeDate Package*. cran.r-project.org.

Würtz, D. & Chalabi, Y. (2009b). *The timeSeries Package*. cran.r-project.org.

# INDEX

ANSI, 4

base functions, 45

calendar
  Easter, 79
  easter related feasts, 80
  ecclesiastical feasts with fixed dates, 81
  ecclesiastical holidays, 79
  federal holidays, 81
  public holidays, 81
  sundays in advent, 81
calendar atoms, 13
calendars, 61, 89
  alignments, 61
  atoms, 63
  business days, 65
  Chicago holidays, 83
  daily series, 67
  day of the week, 63
  day of the year, 64
  Easter, 65
  first Days, 74
  Frankfurt holidays, 83
  functions, 61
  holiday list, 85
  holidays, 65
  last Days, 74
  London holidays, 82
  Milano holidays, 84
  monthly series, 67
  Montreal holidays, 84
  New York holidays, 83
  NYSE, 90
  Osaka holidays, 85
  Paris holidays, 84
  period end points, 65
  public holidays, 89
  quarterly series, 67
  regular series, 67
  Tokyo holidays, 85
  Toronto holidays, 84
  TSX, 90
  Zurich, 89
  Zurich holidays, 82
class representation, 17

DST, 5, 7

extractor functions, 12

financial centers, 18, 29
  list, 30
  setting, 29

GMT, 4
group generic options, 33
  adding, 34
  comparing, 35
  retrieving, 35
  rounding, 34
  substracting, 34
  truncating, 34

holidays, 79
  ecclesiastical, 79
  federal, 81

    public, 81

intervals, 15
iso8601, 3

Julian, 6

keywords
    aggregate, 163
    all, 33, 145
    apply, 84, 151
    as, 8, 25
    attach, 151
    by, 37, 163
    c, 45
    cbind, 151
    character, 8, 25
    cov, 163
    cut, 8
    D, 25
    date, 25, 101
    de, 84
    diff, 45, 151
    dim, 151
    dnorm, 163
    duplicated, 45
    fivenum, 163
    for, 8, 85
    format, 25, 95
    frame, 163
    function, 37
    hist, 163
    integer, 37
    is, 25, 84, 85, 101
    length, 45, 151
    lm, 163
    local, 84
    logical, 37
    lowess, 163
    mad, 163
    Math, 33
    median, 163
    merge, 151
    methods, 8
    min, 45, 151
    model, 163
    na, 163
    next, 85
    objects, 8
    omit, 163
    on, 81, 83, 85
    Ops, 33
    or, 45, 81, 85
    order, 151
    path, 151
    qqnorm, 163
    range, 8, 33, 37, 45, 151
    rep, 45
    rev, 151
    round, 8, 33
    sample, 45, 151
    scale, 151
    search, 151
    seq, 8
    sequence, 8
    sort, 45, 151
    start, 45
    time, 8, 45, 151
    title, 95
    trunc, 8, 33
    unique, 45
    window, 37, 45, 163

local time, 4
logical operations, 12

midnight standard, 26

origin, 13

plot timeSeries, 112
    characters, 114
    color palettes, 115
    colors, 115
    fonts, 114
    multiple plots, 113
    scatter plots, 114
    single plots, 113
    symbols, 114

POSIX, 4, 7
POSIX objects, 8
print timeSeries, 110
  change FinCenter, 111
  horizontal style, 111
  regular series, 112
  tailored POSIX format, 111
  vertical style, 110

R classes
  data.frame, 63, 219
  Date, 11, 12, 46, 55, 57
  difftime, 47, 48
  histogram, 245
  POSIX, 19
  POSIXct, 7, 8, 10, 55, 56
  POSIXlt, 8, 9, 57
  POSIXt, 8, 12, 46
  timDate, 53, 54
  timeDate, 1, 17–22, 24–26, 28, 33–35, 37, 40–42, 46–52, 55–57, 61–64, 66, 67, 69, 99, 109, 121, 178, 205, 261
  timeDate' objects. These include, 101
  timeSeries, 93, 95–99, 102–104, 108–113, 120, 121, 123, 126, 129–132, 136, 140, 141, 143, 145, 146, 151, 154–156, 158, 160, 161, 164, 167, 170, 171, 177–180, 187, 199, 200, 205–207, 209–212, 214, 217–219, 225, 252–255, 262
  timesSeries, 252
  ts, 46, 118, 198, 200
R functions
  %in%, 14
  acf, 164, 242
  aggregate, 164
  align, 61, 62, 179
  all, 13
  any, 13
  apply, 151, 235
  arima, 166, 241
  as.character, 8, 41, 57, 109
  as.double, 48
  as.integer, 47
  as.matrix, 108
  as.POSIXct, 56
  as.timeSeries, 120
  as.vector, 64
  atoms, 63
  attach, 153
  c, 45, 62, 154, 155
  cbind, 154, 155, 218
  ccf, 164, 242
  class, 9
  colMaxs, 160
  colMins, 160
  colnames, 104, 108
  colorTable, 115
  cor, 167, 243
  countMonthlyRecords, 184
  cov, 167, 243
  cumulated, 181
  daily, 182
  dayOfWeek, 42, 63
  dayOfYear, 64
  diff, 46, 47, 156, 157, 236
  difftimDate, 47
  difftime, 11, 47
  difftimeDate, 47
  dim, 102, 157, 224, 237
  dist, 167, 243
  dnorm, 168, 244
  drawdowns, 182
  duplicated, 48, 52
  durations, 184
  Easter, 65
  easter, 80
  end, 49, 51, 54, 129, 136, 241
  endpoints, 65
  filter, 168, 244
  finCenter, 205
  finCenter<-, 105, 207
  findInterval, 15

findIntervals, 15
fivenum, 168
format, 8, 26, 57, 109
fredSeries, 107
frequency, 67
getArgs, 61
getDataPart, 107
head, 49, 129, 131
help.start, 266
hist, 169, 245
holidayZURICH, 43, 89
is.na, 140
isBizday, 42, 65, 137
isDaily, 121
isHoliday, 42, 65, 137
isMonthly, 121, 185
isMultivariate, 102
ISOdate, 8, 10
ISOdatetime, 8, 10
isQuarterly, 121
isRegular, 67, 121
isUnivariate, 102
isWeekday, 42, 137
isWeekend, 42, 137
julian, 13, 52, 54
lag, 169, 245
length, 49, 157, 224
listFinCenter, 29, 30, 206
listHolidays, 85
lm, 170, 246
lowess, 171, 190, 246
mad, 171, 247
match, 13
max, 49, 51, 52
median, 171, 247
merge, 154, 158
min, 49, 51, 52, 159
monthly, 184
monthlyRolling, 71
months, 13, 52, 53
na.contiguous, 234
na.omit, 172, 232
NCOL, 102, 157
ncol, 102, 157
NROW, 102, 157
nrow, 102, 157
oandaSeries, 107
order, 51, 161, 240
orderColnames, 186
orderStatistics, 162, 187
pacf, 164, 242
periodical, 187
periodicallyRolling, 71
periods, 69, 71
plot.histogram, 245
pmatch, 14
print, 110, 251, 253
qqline, 176, 247
qqnorm, 176, 247
quarters, 13, 52, 54
range, 40, 50, 129, 160
rank, 237
rbind, 154–156, 214, 215
readSeries, 107
rep, 50
returns, 157, 188
rev, 160, 238
sample, 50, 160, 239
scale, 161, 239
seq, 10, 22
series, 103, 108
slot, 110
smooth, 176, 190, 250
smooth.spline, 190
smoothLowess, 190
smoothSpline, 190
smoothSupsmu, 190
sort, 49, 51, 161, 211, 240
spectrum, 177, 250
splits, 191
spreads, 191
start, 49, 51, 54, 129, 136, 241
str, 97, 98
strftime, 8
strptime, 8, 10, 21
strptimeDate, 21
supsmu, 190
symbolTable, 114

Sys.Date, 11
Sys.time, 10
Sys.timezone, 10, 197
tail, 49, 129, 131
time, 103, 109, 121, 160, 205
timeCalendar, 18, 207
timeDate, 17–19, 27
timeFirstDayInMonth, 74
timeFirstDayInQuarter, 74, 75
timeLastDayInMonth, 74
timeLastDayInQuarter, 74, 75
timeLastNdayInMonth, 77
timeNdayOnOrAfter, 75, 76
timeNdayOnOrBefore, 76
timeNthNdayInMonth, 77
timeSequence, 18, 22
timeSeries, 97, 132, 205, 207, 214, 221, 228
ts, 118
turnpoints, 191
tzDiff, 19
unclass, 9
unique, 52
var, 167, 243
weekdays, 12, 52
window, 41, 54, 136, 178
yahooSeries, 107
Zurich(), 31

R packages
chron, 6, 7
fBasics, vii, 114, 263
fImport, 107
methods, 110
timeDate, 99, 101, 207, 261
timeSeries, 261, 262
xts, 65

regular series, 117

signal series, 115
special dates
n-th n-day, 77
on-or-After, 75
on-or-Before, 75

subsetting timeSeries, 129
with window(), 136
by column names, 134
by integers, 132
by logical predicates, 134
end, 129
head, 131
missing values, 140
Python like indexing, 135
range, 131
span indexes, 135
start, 129
tail, 131
timeDate indexes, 136
with isBizday(), 137
with isHoliday(), 137
with isWeekday(), 137
with isWeekend(), 137

time data
subsetting, 37
time date
adding, 34
comparing, 35
retrieving, 35
rounding, 34
substracting, 34
tuncating, 34
time zones, 5, 6
timeDate
auto formats, 26
c, 45
calendar atoms, 24
calendar dates, 50
coercion, 55
diff, 46
duplicated, 48
end time, 23
extracting, 37
extracting business days, 42
extracting first, 40
extracting last, 40
extracting weekdays, 42
extracting windows, 41

formatting, 26, 27
head, 49
holidays, 43
ISO 8601 formats, 25
length, 23, 49
logical predicates, 38
max, 49
midnight standard, 26
min, 49
ordering, 51
printing, 26, 27
Python like indexing, 39
range, 50
remove duplicates, 52
rep, 50
sampling, 50
sequences, 22
sorting, 51
span indexes, 40
start time, 23
tail, 49
terminal dates, 51
timeDate Indexes, 41
timeDate objects, 3
timeSerie
special dates, 101
timeSeries, 95
.Data slot, 107
assign time stamps, 103
class representation, 95
coercing date/time, 97
coercions, 143
creation from scratch, 96
data matrix, 96
extract slots, 107
group generics, 125
import series, 107
intraday data, 104
load series, 106
modify slots, 107
multivariate, 102
position slot, 109
read series, 107
retrieving data, 103
retriieving time stamps, 103
structure, 97
time stamps, 96
units slot, 108
univariate, 102
timeSeries coercions
column statistics, 145
cumulated column statistics, 146
rolling statistics, 146
to data frame, 143
to matrix, 143
to vector, 143
timeSeries group generics
Math, 127
Ops, 125
Summary, 128
timeSeries objects, 95

UTC, 6

value matching, 13

weekdays, 12

zone information, 5
zoneinfo, 6

# 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.

**Tobias Setz** has a Bachelor and Master degree in Computational Science and Engineering from ETH in Zurich and has contributed with his thesis projects on wavelet and Bayesian change point analytics to this ebook. He is now a PhD student in the Econophysics group at ETH Zurich at the Institute for Theoretical Physics and maintainer of the Rmetrics packages.

**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 worked for Finance Online and did an internship in the Econophysics group at ETH Zurich at the Institute for Theoretical Physics. Andrew co-authored this ebook about basic R.