Portfolio Optimization with R/Rmetrics

Diethelm Würtz
Yohan Chalabi
William Chen
Andrew Ellis

Rmetrics Association & Finance Online

Diethelm Würtz
Yohan Chalabi
William Chen
Andrew Ellis

Portfolio Optimization with R/Rmetrics

May 11, 2009

Rmetrics Association & Finance Online

Zurich

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

Dr. Martin Hanf
Finance Online GmbH
Weinbergstrasse 41
8006 Zurich

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

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

Authors:
Yohan Chalabi, ETH Zurich
William Chen, University of Auckland
Andrew Ellis, Finance Online GmbH Zurich
Diethelm Würtz, ETH Zurich

ISBN:
eISBN:
DOI:

Revision History

May 2009: 1st edition
Version control information : Revision 1142

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 portfolio optimization from the perspective of computational finance and financial engineering. Thus the main emphasis is to briefly introduce the concepts and to give the reader a set of powerful tools to solve the problems in the field of portfolio optimization.

This book divides roughly into five parts. The first part, Chapters 1-10, is dedicated to the exploratory data analysis of financial assets, the second part, Chapters 11-14, to the framework of portfolio design, selection and optimization, the third part, Chapters 15-19, to the mean-variance portfolio approach, the fourth part, Chapters 20-23, to the mean-conditional value-at-risk portfolio approach, and the fifth part, Chapters 24-26, to portfolio backtesting and benchmarking.

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 (central repository), cran.r-project.org[1] and is distributed under the GNU Public Licence. The R

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

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 S 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 also be downloaded from CRAN and the development branch of Rmetrics can be downloaded from the R-forge repository, r-forge.r-project.org[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 in the finance industry an introduction to using the statistical environment R and the Rmetrics packages for modelling and optimizing portfolios.

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 financial data sets and in designing and optimizing portfolios.

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.

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

Getting Started

When this ebook was written, the most recent copy of R was version 2.9.0. It can be downloaded from the CRAN[3] (Comprehensive R Archive 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[4] 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 bundle package `ebookPortfolio`. This can be done with the following commands in the R environment. If there is no binary package for your operating system, you can install the package from source with the argument `type = "source"`. The R Installation and Administration manual has detailed instructions regarding the required tools to compile packages from source for different platforms.

```
> install.packages("ebookPortfolio",
                   repos = c("http://pkg.rmetrics.org",
                             "http://cran.r-project.org"),
                   type = getOption("pkgType"))
```

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

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

In addition to the R environment and the portfolio ebook packages, we recommend consulting the R manual and the list of frequently asked questions (FAQ) available on the CRAN server. The FAQ document ranges from basic syntax questions to help on obtaining R and downloading and installing R packages. We also suggest having a look at the mailing lists for R and reading the general instructions on *http://www.R-project.org/mail.html*. If you need help for any kind of R and/or Rmetrics problems, we recommend consulting r-help, which is R's main mailing list. R-help has become quite an active list with often dozens of messages per day. r-devel is a public

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

[4] http://cran.r-project.org/web/views/Finance.html

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 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.

There are a number of R manuals in pdf format available on the CRAN server. These manuals include an installation and administration guide, an introduction to the R language, a manual on import and export facilities, a tutorial that describes how you can create your own R packages, an R reference index that contains printable versions all of the R help files for standard and recommended packages. In addition, there exists a number of introductory texts and more advanced tutorials in many languages that can help you to learn R. For an incomplete list of these documents and references visit the CRAN server.

How to read this book

This ebook serves as a tutorial presenting and summarizing algorithms in data analysis of financial assets and portfolio optimization, as a programming guide and as a reference guide. Figure 0.1 shows how the ebook is structured.

The first four chapters explain the S4 objects for portfolio optimization, including assets data, model and portfolio specification, as well as constraints definition and settings. The next ten chapters (5 to 14) deal with analysis of financial assets. Chapters 15 to 21 are concerned with Markowitz' mean-variance portfolio approach, and chapters 22 to 26 with the mean-Conditional Value-at-Risk portfolio approach. Chapters 34 to 36 show how to backtest portfolios.

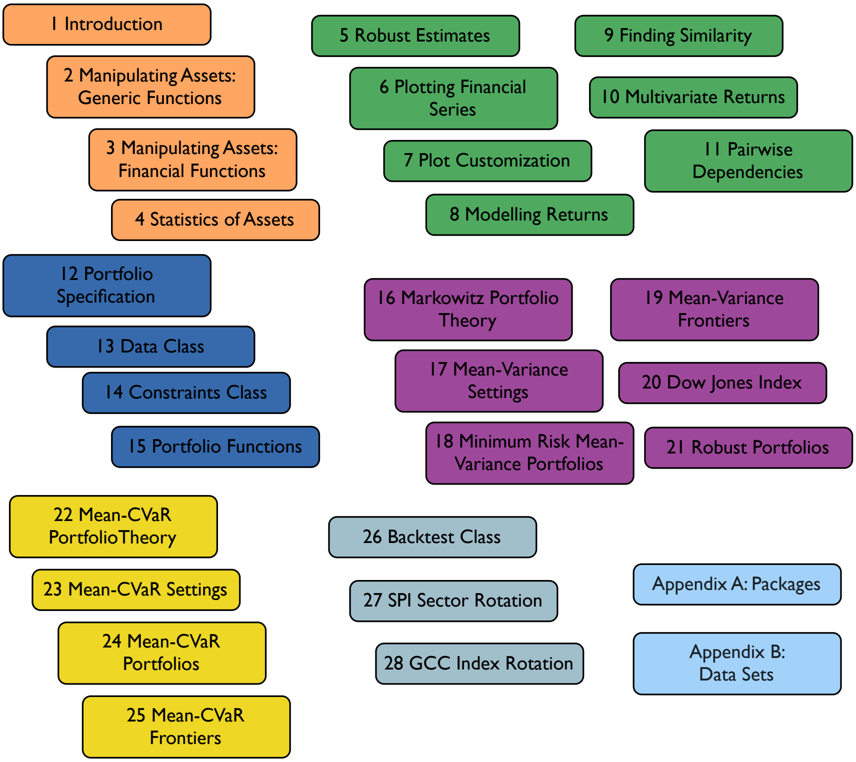


Figure 0.1 Structure of the book

Acknowledgements

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

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 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 to thank Enrique Bengoechea, Dirk Eddelbuettel, Alexios Ghalanos, Francisco Gochez, Oliver Greshake, Martin Hanf, Elmar Heeb, Kurt Hornik, Stephan Joeri, Dominik Locher, David Lüthi, Dominik Lutz, Martin Mächler, Mahendra Mehta, Brian Petersen, Bernhard Pfaff, Jeff Ryan, David Scott, Stefan Theussl, Achim Zeileis, the Institute for Theoretical Physics at ETH Zurich, and the participants and sponsors of the R/Rmetrics Meielisalp Workshops.

This book is the first in a series of Rmetrics ebooks. These ebooks will cover the whole spectrum of Rmetrics packages, from managing chronological objects to dealing with risk. In this ebook we introduce the Rmetrics packages and the implemented functions, which are concerned with the whole spectrum of portfolio analysis, selection, and optimization.

Enjoy it!

Diethelm Würtz
Zurich, May 2009

Contents

List of Figures

List of R Code

List of Abbreviations

ALT	Alternative instruments used in the LPP2005 data set
BAGGED	Bootstrap aggregated
CDaR	Conditional drawdown at risk
CML	Capital market line
CSV	Comma separated values files
CVaR	Conditional value at risk
DaR	Drawdown at risk
df	Degrees of freedom
EDA	Exploratory data analysis
EMA	Exponential moving average
GCC	Gulf Cooperation Council
GHT	Generalized hyperbolic Student's t distribution
GLPK	GNU linear programming kit
LMI	Lehman bond index used in LPP2005 data set
LP25	Low Risk Swiss Pension Fund Benchmark
LP40	Low Risk Swiss Pension Fund Benchmark
LP60	Low Risk Swiss Pension Fund Benchmark
LPP2005	2005 Swiss Pension Fund from Pictet
LP	Linear programming
MCD	Minimum covariance determinant estimator

MPI	Foreign equity index in LPP2005
MV	Mean-variance
MVE	Minimum volume ellipsoid estimator
NA	Not available
NIG	Normal inverse Gaussian
NNVE	Nearest-Neighbor variance estimator
OGK	Orthogonalized Gnanadesikan-Kettenring estimator
qqPlot	Quantile-Quantile plot
QP	Quadratic programming
PCA	Principal component analysis
.RET	Return series
RGB	Red-green-blue colour space
SBI	Swiss Bond Index
SPI	Swiss Performance Index
SWX	Swiss Exchange data set
SII	Swiss Immofunds Index
VaR	Value at risk

Introduction

Portfolio analysis, selection and optimization is the practise of dividing resources among different investments. These may be for example between stocks in an equity portfolio or between asset classes, such as stocks, bonds, mutual funds, real estate, cash equivalents, and private equity in a broader sense.

Portfolio design and optimization with the Rmetrics `fPortfolio` package relies on four pillars:

- *Definition* of the portfolio input, writing specifications, loading the data of the assets, and setting up the constraints.
- *Optimization* of the portfolio, including the computation of single portfolios such as feasible, efficient, tangency (max reward/risk) or minimum variance (global minimum risk) portfolios, and the evaluation of the entire efficient frontier.
- *Generation* of portfolio reports: printing, plotting and summarizing the results.
- *Analysis* of portfolio performance, including rolling analysis, backtesting and benchmarking.

The book is divided into six parts.

In Part III we describe the Rmetrics framework used for portfolio selection, optimization and backtesting. This includes the specification of the three S4 portfolio classes dealing with the *specification*, the *data*, and the portfolio *constraints*.

In Part II we present several aspects of the process of *exploratory data analysis* of the financial asset returns. This includes a brief summary which describes how to modify data sets of financial assets, a description of how to measure their statistical properties, and how to plot the time series and display related properties. We present several examples of how these plots and graphs can be customized by the user. Furthermore, we show how to model the multivariate distribution of the returns, how to group and compare the time series returns included in the data set of the assets, and how investigate and explore pairwise correlations and dependencies.

Part IV is dedicated to Markowitz mean-variance portfolio optimization. We give a brief introduction to the theoretical aspects of the model. We show how to optimize efficient portfolios, including the global minimum variance and the tangency portfolio. Furthermore, we show how to explore the portfolio's feasible set and the whole efficient frontier. Special emphasis is also given in this part to the aspects of robust covariance estimation.

In Part V we go through the same program for mean-CVaR portfolios. Again, we consider individual efficient portfolios, the feasible set, and the efficient frontier. CVaR is an alternative risk measure to the covariance which is also known as mean excess loss, mean shortfall or tail value at risk, VaR. We discuss the portfolio optimization as a convex optimization problem proposed in Rockafellar & Uryasev (2000). We briefly describe the mathematical formulation of mean-CVaR optimization problems which can be formulated as an equivalent linear programming problem and can be solved using standard linear programming solvers.

Part VI is dedicated to portfolio backtesting. We introduce the portfolio backtest class and show how to define rolling windows, how to define portfolio strategies, and how to re-balance the portfolios over time using for example a smoothing approach for the portfolio weights. To show how backtesting for portfolio explicitly works we present two detailed case studies: A Swiss Sector Rotation Portfolio, and a Gulf Country Index portfolio.

Part I
Managing Data Sets of Assets

Introduction

In Chapter 1 we start with the manipulation of data sets of financial assets. These are usually financial return series represented by an S4 `timeSeries` object. We show how to sort a time series by ascending or descending time, how to provide a time-reversed version of a time series and how to re-sample a time series either with or without replacement. Further R functions allow us to bind two ore more time series by columns or rows, and to merge two time series objects by common columns and/or row names. Another kind of manipulation which is often required is to align a time series to unique date and time stamps.

Financial time series analysis is concerned with data from financial markets, which mainly consist of prices, indexes and derived values, such as returns, cumulated returns, volatilities, drawdowns and durations. In Chapter 2 we list and describe functions provided by Rmetrics to compute such derived series.

In Chapter 3 we discuss basic statistics of time series. This includes summary and basic statistics reports, as well as the computation of measures such as mean, standard deviation, covariance, quantiles, or risk estimates.

In Chapter 4 we discuss robust estimates for mean and covariance of assets.

Chapter 1
Generic Functions to Manipulate Assets

Required R package(s):

```
> library(fPortfolio)
```

Portfolio optimization with R/Rmetrics and the acquisition and selection of financial data as input go hand in hand. Rmetrics has a very intuitive way of working with financial time series. A financial time series consists of the data themselves and date/time stamps, which tell us when the data were recorded. In the generic case, when we consider a multivariate data set of financial assets, the data, usually prices or index values, are represented by a numeric matrix, where each column belongs to the data of an individual asset and each row belongs to a specific time/date stamp. This is most easily represented by a position vector of character strings. Combining the string vector of positions and the numeric matrix of data records, we can generate `timeSeries` objects.

In Rmetrics, date/time stamps are used to create `timeDate` objects, which are composed of a position vector of character strings, and the information of the name of the financial centre where the data were recorded. The financial centre is related to a time zone and appropriate daylight saving time rules, so that we can use the data worldwide without any loss of information.

1.1 timeDate and timeSeries Objects

The chronological objects implemented by Rmetrics and used in portfolio optimization are described in detail in the ebook *Chronological Objects in R/Rmetrics*. We highly recommend consulting this ebook if you have any questions concerning creating, modifying, and qualifying financial data sets.

Several financial datasets, which are used throughout this book, are provided with the `fPortfolio` package. They are listed in Listing 1.1. Datasets are available as price/index series and as financial (log)-returns. They are stored as S4 `timeSeries` objects and do not need to be explicitly loaded. If you want to use data sets in the form of CSV files, you can load them as data frames using the `data()` function, and then convert them into S4 `timeSeries` objects using the function `as.timeSeries()`[1]. Listing 1.2 gives a brief summary of time series functions in Rmetrics, with a short description of their function.

```
Data Sets:
SWX                 Daily Swiss equities, bonds, and reits series
LPP2005             Daily Pictet Swiss pension fund benchmarks
SPISECTOR           Swiss Performance sector indexes
GCCINDEX            Gulf Cooperation Council equity indexes
SMALLCAP            Monthly selected US small capitalized equities
MSFT                Daily Microsoft open, high, low, close, volume
```

Listing 1.1 Rmetrics example data sets used in this ebook. The data sets are described in detail in Appendix B.

```
Function:
timeDate            creates timeDate objects from scratch
timeSeq, seq        creates regularly spaced timeDate objects
timeCalendar        creates timeDate objects from calendar atoms
as.timeDate         coerces and transforms timeDate objects
timeSeries          creates a timeSeries object from scratch
readSeries          reads a timeSeries from a spreadsheet file.
```

[1] You can also work with other time series objects, such as `ts` or `zoo` objects. Note that if `zoo` is required, you must load the package before the Rmetrics packages are loaded. In this case you have to coerce these objects into a `'timeSeries'` object using functions `as.timeSeries.foo()`, where foo is a placeholder for the alternative time series class.

```
as.timeSeries        coerces and transforms timeSeries objects
print, plot          generic timeSeries functions
+, -, *, ...         math operations on timeSeries objects
>, < == ...          logical operations on timeSeries objects
diff, log, ...       function operations on timeSeries objects
```

Listing 1.2 Rmetrics basic functions to work with 'timeDate' and 'timeSeries' objects. For further information we refer to the help pages and to the Rmetrics ebook 'Chronological Objects in R/Rmetrics'

The functions in Listing 1.2 can be used to create time series objects from scratch, to convert to and from different representations, to read the time series data from files, or to download data from the Internet. We assume that the reader is familiar with the basics of the `timeDate` and `timeSeries` classes in Rmetrics.

Often data sets of assets are not in the form required for portfolio design, analysis and optimization. If this is the case, we have to compose and modify the data sets. In the following we briefly present the most important functions for managing `timeSeries` objects.

1.2 Loading timeSeries Data Sets

How to load a demo file

The Rmetrics software environment comes with selected demo data sets, which can be used to execute and test examples. Demo data sets are provided as S4 `timeSeries` objects. Below, we show the returns from the daily SWX market indices:

```
> class(SWX.RET)

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

> colnames(SWX.RET)

[1] "SBI"  "SPI"  "SII"  "LP25" "LP40" "LP60"

> head(SWX.RET[, 1:3])
```

```
GMT
                   SBI        SPI         SII
2000-01-04 -0.00208812 -0.0343901  1.3674e-05
2000-01-05 -0.00010452 -0.0104083 -4.9553e-03
2000-01-06 -0.00135976  0.0121191  3.8129e-03
2000-01-07  0.00041859  0.0224617 -6.1621e-04
2000-01-10  0.00000000  0.0021077  2.3806e-03
2000-01-11 -0.00104679 -0.0027737 -2.9385e-04
```

In the third line, we have restricted the output to the first 6 lines of the Swiss Bond Index, the Swiss Performance Index and the Swiss Immofunds Index.

How to read data from CSV text files

`timeSeries` files can also be written to and read from CSV files; in the example given below, we first create a small data set using just the first 6 lines of the SBI, SPI and SII. Then, we can use the `write.csv()` function to write the data set to a CSV file[2]. The file 'myData.csv' will be created in the current working directory.

```
> # create small data set
> data <- head(SWX.RET[, 1:3])
> # write data to a CSV file in the current directory
> write.csv(data, file = "myData.csv")
```

We can now read the data from our CSV file, using the function `readSeries()`. Note that we have to specify the separator, `sep = ","` because the default separator is `sep = ";"`.

```
> # write CSV file in current directory, specifying the separators
> data2 <- readSeries(file = "myData.csv", header = TRUE, sep = ",")
```

The arguments of the `readSeries()` function are:

```
> args(readSeries)

function (file, header = TRUE, sep = ";", zone = "", FinCenter = "",
    ...)
NULL
```

For details we refer to the ebook *Chronological objects with R/Rmetrics* and the `timeSeries` help files.

[2] For help on this function, see `?write.csv`

How to download data from the Internet

The Rmetrics `fImport` (Würtz, 2009c) provides several functions to download time series data from the Internet, for example from

```
Download Functions:
fredSeries          imports market data from the US Federal Reserve
oandaSeries         imports FX market data from OANDA
yahooSeries         imports market data from Yahoo Finance
```

Listing 1.3 Functions for downloading data from the Internet

These functions are able to download CSV files and HTML files and then format the data and make the records available as an S4 `timeSeries` object. For further information, please consult the user and reference guide of the package `fImport` (Würtz, 2009c).

1.3 Sorting and Reverting Assets

In this chapter we use for our examples the daily data sets `SWX` and `SWX.RET`. The `SWX` data set contains six financial time series. The first three are Swiss indexes from the Swiss Exchange in Zurich, the *Swiss Performance Index*, `SPI`, the *Swiss Bond Index*, `SBI`, and the *Swiss Immofund Index* (reits), `SII`. The remaining three time series, named `LP25`, `LP40`, `LP60`, are *Swiss Pension Fund Benchmarks* provided by Pictet, a Swiss private bank in Geneva. The data set starts on January 3rd, 2000, and ends on May 5th, 2007. The data set contains 1917 time series records. The second data set, `SWX.RET`, contains daily log-returns derived from the `SWX` data set.

```
> head(SWX)

GMT
             SBI    SPI    SII  LP25  LP40  LP60
2000-01-03 95.88 5022.9 146.26 99.81 99.71 99.55
2000-01-04 95.68 4853.1 146.27 98.62 97.93 96.98
2000-01-05 95.67 4802.8 145.54 98.26 97.36 96.11
2000-01-06 95.54 4861.4 146.10 98.13 97.20 95.88
2000-01-07 95.58 4971.8 146.01 98.89 98.34 97.53
2000-01-10 95.58 4982.3 146.36 99.19 98.79 98.21
```

```
> end(SWX)

GMT
[1] [2007-05-08]

> class(SWX)

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

Loading the example data set `SWX` returns an object of class `timeSeries` as required for portfolio optimization.

Sometimes the records in a data set of assets are not ordered in time, or they are in reverse order. In this case the time stamps can be rearranged so that the series of assets becomes ordered in the desired way. Rmetrics has generic functions to sort, `sort()`, and revert, `rev()`, the time stamps of time series so that they appear in ascending or descending order. The function `sample()` samples a series in random order.

```
Functions:
sort                sorts a 'timeSeries' in ascending or descending order
rev                 provides a time-reversed version of a 'timeSeries'
sample              generates a sample either with or without replacement

Arguments:
x                   an object of class 'timeSeries'
```

Listing 1.4 Functions for sorting, reverting and sampling data sets of assets[3]

How to sample a time series randomly

The generic function `sample()` takes a random sample either with or without replacement.

In this example, we randomly take ten rows (without replacement) from the `SWX` data set:

```
> SAMPLE <- sample(SWX[1:10, ])
> SAMPLE
```

[3] Note that not all arguments are necessarily listed in a function listing. For a complete listing and a detailed description of all arguments of a function please see its help page.

```
GMT
             SBI    SPI    SII  LP25  LP40  LP60
2000-01-11 95.48 4968.5 146.31 98.95 98.48 97.80
2000-01-06 95.54 4861.4 146.10 98.13 97.20 95.88
2000-01-03 95.88 5022.9 146.26 99.81 99.71 99.55
2000-01-14 95.65 5042.2 146.94 99.79 99.68 99.52
2000-01-05 95.67 4802.8 145.54 98.26 97.36 96.11
2000-01-04 95.68 4853.1 146.27 98.62 97.93 96.98
2000-01-07 95.58 4971.8 146.01 98.89 98.34 97.53
2000-01-13 95.51 4985.2 147.09 99.20 98.81 98.24
2000-01-10 95.58 4982.3 146.36 99.19 98.79 98.21
2000-01-12 95.47 4977.8 146.28 98.91 98.42 97.71
```

Notice that the records of the sampled time series are no longer ordered in time, and thus follow each other in a completely irregular fashion.

How to sort a series in ascending order

The generic function `sort()` sorts the records of a time series in ascending or descending order. This is shown in the following example:

```
> sort(SAMPLE)

GMT
             SBI    SPI    SII  LP25  LP40  LP60
2000-01-03 95.88 5022.9 146.26 99.81 99.71 99.55
2000-01-04 95.68 4853.1 146.27 98.62 97.93 96.98
2000-01-05 95.67 4802.8 145.54 98.26 97.36 96.11
2000-01-06 95.54 4861.4 146.10 98.13 97.20 95.88
2000-01-07 95.58 4971.8 146.01 98.89 98.34 97.53
2000-01-10 95.58 4982.3 146.36 99.19 98.79 98.21
2000-01-11 95.48 4968.5 146.31 98.95 98.48 97.80
2000-01-12 95.47 4977.8 146.28 98.91 98.42 97.71
2000-01-13 95.51 4985.2 147.09 99.20 98.81 98.24
2000-01-14 95.65 5042.2 146.94 99.79 99.68 99.52
```

How to revert a series in time

A sorted `timeSeries` object is given either in an ascending or descending order. The time ordering of the records of a data set can be reversed using the generic function `rev()`. Alternatively, we can also use the function `sort(x,decreasing=FALSE)`, setting the argument `decreasing` either to `TRUE` or `FALSE`.

```
> rev(sort(SAMPLE))
```

and

```
> sort(SAMPLE, decreasing = TRUE)
```

```
GMT
             SBI    SPI    SII  LP25  LP40  LP60
2000-01-14 95.65 5042.2 146.94 99.79 99.68 99.52
2000-01-13 95.51 4985.2 147.09 99.20 98.81 98.24
2000-01-12 95.47 4977.8 146.28 98.91 98.42 97.71
2000-01-11 95.48 4968.5 146.31 98.95 98.48 97.80
2000-01-10 95.58 4982.3 146.36 99.19 98.79 98.21
2000-01-07 95.58 4971.8 146.01 98.89 98.34 97.53
2000-01-06 95.54 4861.4 146.10 98.13 97.20 95.88
2000-01-05 95.67 4802.8 145.54 98.26 97.36 96.11
2000-01-04 95.68 4853.1 146.27 98.62 97.93 96.98
2000-01-03 95.88 5022.9 146.26 99.81 99.71 99.55
```

produce the same output.

1.4 Alignment of Assets

The alignment of `timeSeries` objects is an important aspect in managing assets. Due to holidays, we must expect missing data records for daily data sets. For example, around Easter, data records for Good Friday may be missing in most countries of the world, and it is likely that the markets are also closed on Easter Monday. Even so, we still want to align the series to a regular weekly calendar series. Missing records can then be coded in several ways; a straightforward way is to use the price of the previous day for the subsequent day. The function `align()` aligns the asset series on calendar dates by default, i.e. on every day of the week, or, more naturally, on the weekdays from Monday to Friday.

Let us align the `SWX` series of indices to daily dates, including holidays, and replace the missing values with the indices from the previous days:

```
> nrow(SWX)
```

```
[1] 1917
```

```
> ALIGNED <- align(x = SWX, by = "1d", method = "before", include.weekends =
    FALSE)
> nrow(SWX)
```

```
[1] 1917
```

The returned number of rows shows that the original series does not have any missing data records due to holidays. The alignment function can be used not only to align daily series, but also to align a series to other time horizons. For example, we can align daily data sets by weekly time horizons starting on any arbitrary day of the week using the `offset` argument of the function.

```
Function:
align               aligns a 'timeSeries' object to calendar objects.

Arguments:
x                   an object of class 'timeSeries'
by                  a character string formed from an integer length and
                    a period identifier. Valid values are "w", "d", "h",
                    "m", "s", for weeks, days, hours, minutes and seconds
                    For example, a bi-weekly period is expressed as "2w"
offset              a character string formed from an integer length and
                    a period identifier in the same way as for 'by'
method              a character string, defining the alignment. Substitutes
                    a missing record with the value of the previous
                    ("before") record, of the following ("after") record,
                    interpolates ("interp") or fills with NAs ("NA")
include.weekends    should the weekend days (Saturdays and Sundays) be
                    included?
```

Listing 1.5 Function to align a time series records to time and calendar atoms.

1.5 Binding and Merging Assets

In many cases we have to compose the desired assets from several univariate and/or multivariate time series. Then we have to bind different time series together. The functions available in Rmetrics are shown in Listing 1.6, in order of increasing complexity:

```
Function:
c                   concatenates a 'timeSeries' object.
cbind               combines a 'timeSeries' by columns.
rbind               combines a 'timeSeries' by rows.
merge               merges two 'timeSeries' by common columns and/or rows.

Arguments:
x, y                objects of class 'timeSeries'.
```

Listing 1.6 Functions to concatenate data sets of assets

Before we start to interpret the results of binding and merging several time series objects, let us consider the following three time series examples to better understand how binding and merging works.

```
> 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"`. Notice that the first column `"A"` of the third time series `t3` describes the same time series `"A"` as the first series `"t1"`.

How to bind time series column- and row-wise

The functions `cbind()` and `rbind()` allow us to bind time series objects together either column or row-wise. Let us bind series `t1` and series `t2` column-wise

```
> 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 together 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 column names "A", but they receive the suffixes ".1" and ".2" to distinguish them.

The function rbind() behaves similarly, but the number of rows must be the same in all time series to be bound by rows

```
> rbind(t1, t2)

GMT
              A_B
2008-01-01 -1.097
2008-02-01  0.236
2008-03-01 -0.890
2008-04-01 -1.472
2008-05-01  1.484
2008-05-01 -1.009
2008-06-01  0.231
2008-06-01  0.983
2008-07-01  0.187
2008-07-01 -0.068
2008-10-01 -0.005
2008-10-01 -2.300
2008-11-01  1.099
2008-11-01  1.023
2008-12-01  1.177
```

The column name is now "A_B" to illustrate that series named "A" and "B" were bound together. Note that binding the univariate series t1 and the bivariate series `t3` would result in an error because they do not have the same number of columns.

How to merge time series column-wise and row-wise

Merging two data sets of assets is the most general case and will take the names of the individual columns. `merge()` combines the two series, which can be either univariate or multivariate, by column and by row, and, additionally, intersects columns with identical column names. This is the most important point. To show this, let us merge the time series `t1` and `t3`, and then merge them 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
```

This gives us a 3-column time series with names `"A"`, `"B"`, and `"C"`. Note that the records from time series `t1` and from the first column of time series `t3`, both named `"A"`, were merged into the same first column of the new time series.

1.6 Subsetting Assets

Subsetting a data set of assets and replacing parts of a data set by other records is a very important issue in the management of financial time series. There are several functions that are useful in this context. These include the `"["` operator, which extracts or replaces subsets, the `window()` function, which cuts out a piece from a data set between two 'timeDate' objects, `start` and `end`, and the functions `start()` and `end()` themselves, which return the first and last record of a data set.

Subsetting by using the `"["` operator can be done by simple counts, by date/-time stamps, by instrument (column) names, or even by logical predicates, e.g. extracting all records before or after a given date.

```
Function:
[                   extracts or replaces subsets by indexes, column
                    names, date/time stamps, logical predicates, etc
subset              returns subsets that meet specified conditions
window              extracts a piece between two 'timeDate' objects
start               extracts the first record
end                 extracts the last record

Arguments:
x                   an object of class 'timeSeries'
```

Listing 1.7 Functions for subsetting data sets of assets

How to subset by counts

Subsetting by counts allows us to extract desired records from the rows, and desired instruments from the columns of the data series matrix. The first example demonstrates how to subset a univariate or multivariate `timeSeries` by row, here the second to the fifth rows

```
> SWX[2:5, ]

GMT
             SBI    SPI    SII  LP25  LP40  LP60
2000-01-04 95.68 4853.1 146.27 98.62 97.93 96.98
```

```
2000-01-05 95.67 4802.8 145.54 98.26 97.36 96.11
2000-01-06 95.54 4861.4 146.10 98.13 97.20 95.88
2000-01-07 95.58 4971.8 146.01 98.89 98.34 97.53
```

```
> SWX[2:5, 2]
```

```
GMT
              SPI
2000-01-04 4853.1
2000-01-05 4802.8
2000-01-06 4861.4
2000-01-07 4971.8
```

Note that in the first example we have to explicitly write `SWX[2:5, ]` instead of `SWX[2:5]` since the data part is a two dimensional rectangular object.

How to find the first and last records

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 assets in increasing time order and returns the first element of the time positions

```
> SWX[start(SWX), ]
```

```
GMT
             SBI    SPI    SII  LP25  LP40  LP60
2000-01-03 95.88 5022.9 146.26 99.81 99.71 99.55
```

```
> SWX[start(sample(SWX)), ]
```

```
GMT
             SBI    SPI    SII  LP25  LP40  LP60
2000-01-03 95.88 5022.9 146.26 99.81 99.71 99.55
```

`end()` behaves in the same way, but in the opposite order.

How to subset by column names

Instead of using counts, e.g. the 4th column, we can reference and extract columns by column names, which are usually the names of the financial instruments

```
> tail(SWX[, "SPI"])
```

```
GMT
                SPI
2007-05-01 7620.1
2007-05-02 7634.1
2007-05-03 7594.9
2007-05-04 7644.8
2007-05-07 7647.6
2007-05-08 7587.9
```

How to subset by date/time stamps

Subsetting by date vectors allows you to extract desired records from the rows for a specified date or dates. We first show an example for the univariate case where we extract a specific date:

```
> # Extract a specific date:
> SWX["2007-04-24", ]

GMT
             SBI    SPI    SII   LP25  LP40   LP60
2007-04-24 97.05 7546.5 214.91 129.65 128.1 124.34

> # Subset all records from the first and second quarter:
> round(window(SWX, start = "2006-01-15", end = "2006-01-21"), 1)

GMT
             SBI    SPI   SII  LP25  LP40  LP60
2006-01-16 101.3 5939.4 196.7 123.1 118.2 110.5
2006-01-17 101.4 5903.5 196.7 123.0 117.9 110.2
2006-01-18 101.5 5861.7 197.2 122.8 117.5 109.5
2006-01-19 101.3 5890.5 198.9 122.9 117.8 110.0
2006-01-20 101.2 5839.4 197.6 122.5 117.3 109.4
```

Here we have rounded the results to one digit in order to shorten the output using the generic function `round()`. Note that there are additional functions to round numbers in R. These include: `ceiling()`, `floor()`, `truncate()`, and `signif()`. For details we refer to the help pages.

1.7 Aggregating Assets

In finance we often want to aggregate time series, that is we want to go from a fine-grained resolution to a coarse-grained resolution. For example, we have collected data on a daily basis and now we want to display them on a weekly, monthly, or quarterly basis.

We can use the generic function `aggregate()` from R's base package `stats` to do this. The function splits the data set into individual subsets, and then computes summary statistics for each subset. Finally, the result is returned in a convenient form. Rmetrics provides a method for aggregating `timeSeries` objects. The function requires three input arguments, the time series itself, a sequence of date/time stamps defining the grouping, and the function that is to be applied.

```
Function:
aggregate   aggregates a 'timeSeries' object.

Arguments:
x                   is a uni- or multivariate 'timeSeries' object
by                  is a 'timeDate' sequence of grouping dates
FUN                 a scalar function to compute the summary statistics
                    to be applied to all data subsets
```

Listing 1.8 Function for aggregating a data set of assets

To be more specific, let us define an artificial monthly `timeSeries` that we want to aggregate on a quarterly base

```
> charvec <- timeCalendar()
> data <- matrix(round(runif(24, 0, 10)), 12)
> tS <- timeSeries(data, charvec)
> tS

GMT
           TS.1 TS.2
2009-01-01    1    6
2009-02-01    4    4
2009-03-01    2   10
2009-04-01   10    9
2009-05-01    6    6
2009-06-01    1    6
```

```
2009-07-01    1    2
2009-08-01   10    0
2009-09-01    7    4
2009-10-01    0    2
2009-11-01    9    8
2009-12-01    0   10
```

Next, we create the quarterly breakpoints from the `charvec` vector searching for the last day in a quarter for each date. To suppress double dates we make the breakpoints unique

```
> by <- unique(timeLastDayInQuarter(charvec))
> by

GMT
[1] [2009-03-31] [2009-06-30] [2009-09-30] [2009-12-31]
```

and finally we create the quarterly series with the aggregated monthly sums and new units passed in by the dots argument.

```
> aggregate(tS, by, FUN = sum, units = c("TSQ.1", "TSQ.2"))

GMT
           TSQ.1 TSQ.2
2009-03-31     7    20
2009-06-30    17    21
2009-09-30    18     6
2009-12-31     9    20
```

Rmetrics also has many utility functions to manage special dates. These are shown in Listing 1.9.

```
Function:
timeLastDayInMonth       last day in a given month/year
timeFirstDayInMonth      first day in a given month/ year
timeLastDayInQuarter     last day in a given quarter/year
timeFirstDayInQuarter    first day in a given quarter/year
timeNdayOnOrAfter        date month that is a n-day ON OR AFTER
timeNdayOnOrBefore       date in month that is a n-day ON OR BEFORE
timeNthNdayInMonth       n-th occurrence of a n-day in year/month
timeLastNdayInMonth      last n-day in year/month
```

Listing 1.9 Utility functions for managing special dates to determine date breakpoints, e.g. when the accounting is quarterly on the first Monday or working day of the following quarter. More examples are provided in the Rmetrics ebook 'Chronological Objects with R/Rmetrics'.

Now let us demonstrate a real-world example. We will aggregate the daily returns of the SPI index on monthly periods:

```
> tS <- 100 * LPP2005.RET[, "SPI"]
> by <- timeLastDayInMonth(time(tS))
> aggregate(tS, by, sum)
```

```
GMT
                SPI
2005-11-30  4.81406
2005-12-31  2.48124
2006-01-31  3.19682
2006-02-28  1.39536
2006-03-31  2.48262
2006-04-30  1.41992
2006-05-31 -5.37181
2006-06-30  0.52306
2006-07-31  3.61945
2006-08-31  2.91602
2006-09-30  3.24490
2006-10-31  2.00248
2006-11-30 -0.54849
2006-12-31  3.90591
2007-01-31  4.41255
2007-02-28 -3.76150
2007-03-31  2.95389
2007-04-30  2.04765
```

1.8 Rolling Assets

Let us write a simple function named `rollapply()` that can compute rolling statistics using the function `applySeries()`, which can be found in the Rmetrics package `timeSeries`. The periods may be overlapping or not, and we even allow gaps between the periods.

```
> rollapply <- function(x, by, FUN, ...)
 {
     ans <- applySeries(x, from = by$from, to = by$to, by = NULL,
```

```
        FUN = FUN, format = x@format,
        zone = finCenter(x), FinCenter = finCenter(x),
        title = x@title, documentation = x@documentation, ...)
    attr(ans, "by") <- data.frame(from = format(by$from), to = format(by$to)
    )
    ans
}
```

Here we also want to focus on the `periods` function from the `timeDate` package. This allows us to compute periods from time spans. The following example demonstrates how to compute the returns on a multivariate data set of assets subset in annual windows and shifted monthly:

```
Function:
periods             constructs equidistantly sized and shifted windows

Arguments:
period              size (length) of the periods
by                  shift (interval) of the periods, "m" monthly, "w"
                    weekly, "d" daily, "H" by hours, "M" by minutes,
                    "S" by seconds.
```

Listing 1.10 Function to generate shifted time periods (windows)

```
> DATA <- 100 * SWX.RET[, c(1:2, 4:5)]
> by <- periods(time(DATA), "12m", "1m")
> SWX.ROLL <- rollapply(DATA, by, FUN = "colSums")
> SWX.ROLL

GMT
                 SBI        SPI     LP25      LP40
2000-12-31 -0.648741  11.253322  1.96435  0.809074
2001-01-31  0.671355  16.511381  4.19642  3.738940
2001-02-28  1.442358  11.328914  2.96835  1.269027
2001-03-31  3.710197  -2.463213  1.23250 -2.479424
2001-04-30  2.969093   0.027254  2.49592 -0.099612
2001-05-31  3.219551  -1.649499  3.59474  1.479009
2001-06-30  3.332580  -5.570437  3.00181  0.508096
2001-07-31  3.357122 -15.100256  1.25626 -2.234674
2001-08-31  3.225746 -22.418822 -2.03597 -7.394520
2001-09-30  3.480715 -28.331323 -3.63421 -9.924043
2001-10-31  4.993481 -29.770540 -2.73963 -9.160043
```

```
2001-11-30  3.024667 -26.143558 -1.10969  -5.396260
2001-12-31  0.481727 -24.881298 -1.51451  -4.684220
2002-01-31 -0.240750 -26.585610 -2.63821  -6.193178
2002-02-28 -0.083656 -20.539959 -1.29404  -4.012585
2002-03-31 -1.095530  -8.217245  0.11847  -1.134575
2002-04-30  0.814541 -11.385227 -1.22037  -4.359142
2002-05-31  1.230336 -13.121213 -2.10761  -5.936913
2002-06-30  1.419934 -18.837676 -3.71606  -8.662299
2002-07-31  2.153280 -26.706309 -4.51401 -10.290890
2002-08-31  2.215930 -21.770164 -2.27123  -6.858606
2002-09-30  3.633073 -20.261114 -1.14134  -5.868589
2002-10-31  1.793860 -18.400350 -1.66831  -5.610174
2002-11-30  3.445123 -17.909739 -0.57954  -4.843977
2002-12-31  6.374161 -30.045031 -2.17794  -8.776987
2003-01-31  6.727268 -32.240982 -2.81224  -9.970441
2003-02-28  6.833010 -39.906837 -3.78933 -11.455874
2003-03-31  6.281915 -45.904194 -4.92441 -13.212261
2003-04-30  4.831956 -33.420139 -1.62719  -7.521366
2003-05-31  5.944136 -31.715345  0.16828  -5.303379
2003-06-30  4.487675 -18.755693  3.40429   0.454104
2003-07-31  2.625509  -0.085590  5.78381   5.261781
2003-08-31  1.340408   0.608975  5.26646   5.219139
2003-09-30  0.159331   8.341326  6.61208   7.878437
2003-10-31 -0.209969   9.073305  6.56976   7.860532
2003-11-30 -1.021132   7.595050  4.61191   5.157326
2003-12-31 -1.780110  19.937351  7.50905  10.133467
2004-01-31 -1.619949  29.559969  9.66430  13.498650
2004-02-29 -1.139837  37.298131 11.73699  16.394898
2004-03-31 -0.236733  35.187756 11.91641  16.339255
2004-04-30 -0.805934  27.335878  8.64325  11.900439
2004-05-31 -2.953049  22.872705  5.75539   8.685517
2004-06-30 -3.057181  19.225778  4.33481   6.716452
2004-07-31 -1.621495  12.406890  4.25637   5.635294
2004-08-31  0.080056   8.888211  4.24793   4.779698
2004-09-30 -0.428779  11.227889  5.11336   6.368691
2004-10-31  0.956278   5.617209  3.84897   3.795212
2004-11-30  2.091085   5.321001  4.60066   4.556202
2004-12-31  0.982980   6.663649  4.77314   5.131901
2005-01-31  1.619949   3.627716  4.72099   4.704666
2005-02-28  0.177725   4.950409  3.22565   3.315760
2005-03-31  0.384407   8.390182  4.10097   4.490650
2005-04-30  2.632380   4.408484  4.97875   4.605340
2005-05-31  3.711807  11.004300  8.20661   8.901415
2005-06-30  4.454439  13.157849  9.48684  10.385646
2005-07-31  3.772538  20.377955 10.26315  12.135798
2005-08-31  3.005300  21.450785  9.53102  11.462192
```

```
2005-09-30  2.642891  26.169441 10.51858  13.249501
2005-10-31  0.464864  29.181418  9.26759  12.349521
2005-11-30 -0.573181  32.367089 10.01835  13.981393
2005-12-31  0.167827  30.459956  9.91385  13.556889
2006-01-31 -1.331578  31.990276  8.70780  12.619758
2006-02-28 -0.842048  30.415586  9.43640  13.342347
2006-03-31 -2.681981  32.421388  8.42184  12.812291
2006-04-30 -4.997799  34.125786  6.88055  11.764558
2006-05-31 -5.374963  24.323278  3.47413   6.653625
2006-06-30 -5.987898  22.569102  2.15225   5.017894
2006-07-31 -5.338725  20.610573  2.12615   4.534738
2006-08-31 -4.354767  24.483196  3.92950   6.790036
2006-09-30 -3.616941  21.992038  3.83319   6.196635
2006-10-31 -2.305837  22.724124  5.75166   8.502598
2006-11-30 -1.397228  17.361584  4.69972   6.401313
2006-12-31 -3.013995  18.786250  3.99333   6.154261
2007-01-31 -3.085544  20.001990  4.49084   6.852949
2007-02-28 -2.487916  14.845132  3.49016   5.017624
2007-03-31 -1.599219  15.316395  4.22788   5.660890
2007-04-30 -0.728919  19.796457  6.38363   8.401080
2007-05-31 -1.079535  24.744413  7.88086  11.077927
```

```
> attr(SWX.ROLL, "by")
```

```
         from         to
1  2000-01-01 2000-12-31
2  2000-02-01 2001-01-31
3  2000-03-01 2001-02-28
4  2000-04-01 2001-03-31
5  2000-05-01 2001-04-30
6  2000-06-01 2001-05-31
7  2000-07-01 2001-06-30
8  2000-08-01 2001-07-31
9  2000-09-01 2001-08-31
10 2000-10-01 2001-09-30
11 2000-11-01 2001-10-31
12 2000-12-01 2001-11-30
13 2001-01-01 2001-12-31
14 2001-02-01 2002-01-31
15 2001-03-01 2002-02-28
16 2001-04-01 2002-03-31
17 2001-05-01 2002-04-30
18 2001-06-01 2002-05-31
19 2001-07-01 2002-06-30
20 2001-08-01 2002-07-31
21 2001-09-01 2002-08-31
22 2001-10-01 2002-09-30
```

```
23 2001-11-01 2002-10-31
24 2001-12-01 2002-11-30
25 2002-01-01 2002-12-31
26 2002-02-01 2003-01-31
27 2002-03-01 2003-02-28
28 2002-04-01 2003-03-31
29 2002-05-01 2003-04-30
30 2002-06-01 2003-05-31
31 2002-07-01 2003-06-30
32 2002-08-01 2003-07-31
33 2002-09-01 2003-08-31
34 2002-10-01 2003-09-30
35 2002-11-01 2003-10-31
36 2002-12-01 2003-11-30
37 2003-01-01 2003-12-31
38 2003-02-01 2004-01-31
39 2003-03-01 2004-02-29
40 2003-04-01 2004-03-31
41 2003-05-01 2004-04-30
42 2003-06-01 2004-05-31
43 2003-07-01 2004-06-30
44 2003-08-01 2004-07-31
45 2003-09-01 2004-08-31
46 2003-10-01 2004-09-30
47 2003-11-01 2004-10-31
48 2003-12-01 2004-11-30
49 2004-01-01 2004-12-31
50 2004-02-01 2005-01-31
51 2004-03-01 2005-02-28
52 2004-04-01 2005-03-31
53 2004-05-01 2005-04-30
54 2004-06-01 2005-05-31
55 2004-07-01 2005-06-30
56 2004-08-01 2005-07-31
57 2004-09-01 2005-08-31
58 2004-10-01 2005-09-30
59 2004-11-01 2005-10-31
60 2004-12-01 2005-11-30
61 2005-01-01 2005-12-31
62 2005-02-01 2006-01-31
63 2005-03-01 2006-02-28
64 2005-04-01 2006-03-31
65 2005-05-01 2006-04-30
66 2005-06-01 2006-05-31
67 2005-07-01 2006-06-30
68 2005-08-01 2006-07-31
```

```
69 2005-09-01 2006-08-31
70 2005-10-01 2006-09-30
71 2005-11-01 2006-10-31
72 2005-12-01 2006-11-30
73 2006-01-01 2006-12-31
74 2006-02-01 2007-01-31
75 2006-03-01 2007-02-28
76 2006-04-01 2007-03-31
77 2006-05-01 2007-04-30
78 2006-06-01 2007-05-31
```

The values `"12m"` and `"1m"` in the function `periods()` are called time spans. Here are two more examples with regular periodic periods and by-shifts.

```
> by <- periods(time(SWX), period = "52w", by = "4w")
> by <- periods(time(SWX), period = "360d", by = "30d")
```

The first example rolls on a period of 52 weeks shifted by 4 weeks, the second rolls every 30 days on 360 calendar days.

Periods created by `"12m"` yield an annual rolling period for a given fixed shift, `"6m"` yields a semi-annual rolling period, `"3m"` a quarterly rolling period, `"2m"` a bi-monthly rolling period, and so on. With these unit identifiers we can create calendar-based rolling as well as regular periodical rolling, or even irregular rolling periods and by-shifts. The latter are useful for periods triggered by the volatility, for example, and a shift given by automated trading signals or by human decision-makers.

Chapter 2
Financial Functions to Manipulate Assets

Required R package(s):

```
> library(fPortfolio)
> library(fImport)
```

Financial time series analysis investigates and models data sets from financial markets. These are usually prices, indices and derived values such as returns, cumulated returns, volatilities, drawdowns and durations, amongst others.

In this chapter we describe functions provided by Rmetrics to compute derived financial time series and show several examples how to use them. Moreover we show examples how a user can add his own functions to the Rmetrics framework.

2.1 Price and Index Series

Price and index series can be downloaded either from free sources, such as Yahoo Finance[1], the Swiss Exchange[2] or the Federal Reserve Bank in St. Louis[3], or can be obtained from commercial providers such as Bloomberg or IBrokers. Rmetrics provides interfaces for downloading free data from

[1] http://finance.yahoo.com

[2] http://www.six-swiss-exchange.com

[3] http://www.stlouisfed.org

the Internet (Würtz, 2009c). R also has packages for downloading data from commercial sources; these include for example the `RBloomberg` (Sams, 2009) and `IBrokers` (Ryan, 2008) packages.

```
Function:
returns           generates returns from a price/index series
cumulated         generates indexed values from a returns series
drawdowns         computes drawdowns from financial returns
lowess            smooths a price/index series
turnpoints        finds turnpoints for a smoothed price/index series
```

Listing 2.1 Functions for computing and exploring financial returns

2.2 Return and Cumulated Return Series

To calculate compound or simple *returns* (Bacon, 2008), usually on daily or monthly records for portfolio analysis and optimization, we can call the function `returns()`.

```
Function:
returns           generates returns from a price/index series
cumulated         generates indexed values from a returns series

Arguments:
x                 a price/index for a uni or multivariate
                  series of class timeSeries
methods           the method of computing the returns
                  "continuous", "discrete", "compound", "simple"
percentage        a logical, should percentual returns be computed?
```

Listing 2.2 Functions to compute and convert Price/Index values and financial returns

The argument `methods` allows us to define how the returns are computed. The methods `"continuous"` and `"discrete"` are synonyms for the methods `"compound"` and `"simple"`, respectively.

In the following example we first compute the compound returns for the `LP25` benchmark from the `SWX` data set, and then we cumulate the returns to recover the price/index series. To do so, we need to index the series to 1 on the first day before we calculate the returns. By cumulating the returns, we can recover the indexed series:

```
> LP25 <- SWX[, "LP25"]/as.numeric(SWX[1, "LP25"])
> head(LP25, 5)

GMT
              LP25
2000-01-03 1.00000
2000-01-04 0.98808
2000-01-05 0.98447
2000-01-06 0.98317
2000-01-07 0.99078

> head(returns(LP25), 5)

GMT
                 LP25
2000-01-04 -0.0119943
2000-01-05 -0.0036571
2000-01-06 -0.0013239
2000-01-07  0.0077150
2000-01-10  0.0030291

> head(cumulated(returns(LP25)), 5)

GMT
              LP25
2000-01-04 0.98808
2000-01-05 0.98447
2000-01-06 0.98317
2000-01-07 0.99078
2000-01-10 0.99379

> head(returns(cumulated(returns(LP25))), 4)

GMT
                 LP25
2000-01-05 -0.0036571
2000-01-06 -0.0013239
2000-01-07  0.0077150
2000-01-10  0.0030291
```

2.3 Drawdowns Series

Drawdown measures describe the decline from a historical peak in some price or index variable. This is typically in cumulated return series of a financial trading strategy.

```
Function:
drawdowns           computes drawdowns from financial returns

Arguments:
x                   a 'timeSeries' of financial returns
```

Listing 2.3 Function to compute drawdowns from financial returns

The maximum drawdown up to a given time is the maximum of the drawdown over the history of the price or index variable and can be considered as an indicator of risk. A `drawdowns()` [4] series can be computed from a return series as follows:

```
> head(drawdowns(SWX.RET[, 1:4]))

GMT
                   SBI        SPI        SII       LP25
2000-01-04  0.00000000  0.0000000  0.00000000  0.0000000
2000-01-05 -0.00010452 -0.0104083 -0.00495531 -0.0036571
2000-01-06 -0.00146414  0.0000000 -0.00116130 -0.0049761
2000-01-07 -0.00104617  0.0000000 -0.00177680  0.0000000
2000-01-10 -0.00104617  0.0000000  0.00000000  0.0000000
2000-01-11 -0.00209186 -0.0027737 -0.00029385 -0.0024225
```

The function returns a univariate time series object of `timeSeries`. Multiplying the series by a factor of 100 gives us the returns in percentages.

[4] The functions `drawdowns()` and `drawdownStats()` are reimplemented from the contributed R package `PerformanceAnalytics`, based on code written by Carl & Peterson (2008).

2.4 Durations Series

The *interval* is the time period between time series records.

```
Function:
intervals           computes intervals from a financial series

Arguments:
x                   a 'timeSeries' of financial returns
```

Listing 2.4 Function to compute intervals from a financial series

Let us consider 10 randomly selected records in ascending order:

```
> SPI <- SWX[, "SPI"]
> SPI10 <- SPI[c(4, 9, 51, 89, 311, 513, 756, 919, 1235, 1648),
     ]
> SPI10

GMT
              SPI
2000-01-06 4861.4
2000-01-13 4985.2
2000-03-13 4693.7
2000-05-04 5133.6
2001-03-12 5116.3
2001-12-19 4231.5
2002-11-25 3579.1
2003-07-10 3466.8
2004-09-24 4067.7
2006-04-26 6264.4
```

Then we compute their intervals in units of days.

```
> intervals(SPI10)/(24 * 3600)

GMT
           Duration
2000-01-06       NA
2000-01-13        7
2000-03-13       60
2000-05-04       52
2001-03-12      312
2001-12-19      282
```

```
2002-11-25      341
2003-07-10      227
2004-09-24      442
2006-04-26      579
```

Here we have divided the returned value by the length of one day, i.e. $24 * 3600$ seconds so that the intervals are given in days.

Intervals are especially of interest when we consider irregular time series and we want to know the time period between consecutive records, or between consecutive events like turnpoints.

2.5 How to Add Your Own Functions

It is very easy to add new generic functions operating on `timeSeries` objects to Rmetrics. In this section we show how to add a function to smooth price and index series, and a function to find the turnpoints in a time series.

```
Function:
lowess              a locally-weighted polynomial regression smoother
turnpoints          finds turnpoints in a financial series

Arguments:
x                   a 'timeSeries' of financial returns
f                   the smoother span for lowess
iter                the number of robustifying iterations for lowess
```

Listing 2.5 User supplied functions to smooth a series and to find turning points

How to smooth a time series with `lowess()`

We will demonstrate this by writing a smoother for financial time series built on top of the `lowess()` function from the R `stats` package. `lowess()` is a smoother based on robust locally weighted regression (Cleveland, 1979, 1981). Using the function `setMethod()` from the R `methods` package we can create and save a formal method for `lowess()`.

```
> setMethod("lowess", "timeSeries", function(x, y = NULL, f = 2/3,
      iter = 3) {
      stopifnot(isUnivariate(x))
      ans <- stats::lowess(x = as.vector(x), y, f, iter)
      series(x) <- matrix(ans$y, ncol = 1)
      x
 })

[1] "lowess"
```

We first extract the `SPI` from the `SWX` data set and then we smooth the index. The argument `f` determines the smoother span. This gives the proportion of points which influence the smooth at each value. Larger values give more smoothness and smaller values result in less smoothness keeping more of the original structure of the curve. The graph in Figure 2.1 shows the result.

```
> SPI <- SWX[, "SPI"]
> SPI.LW <- lowess(SPI, f = 0.08)
> plot(SPI)
> lines(SPI.LW, col = "brown", lwd = 2)
```

How to find the turnpoints of a time series

If we are interested in the turnpoints of the smoothed `SPI` index, we can use the `turnpoints()` function from the contributed R package `pastecs` (Ibanez, Grosjean & Etienne, 2009). The function determines the number and the positions of extrema, i.e. the turning points, either peaks or pits, in a regular time series. Writing an S4 method is straightforward:

```
> library(pastecs)
> setMethod("turnpoints", "timeSeries",

      function(x)
      {
          stopifnot(isUnivariate(x))
          tp <- suppressWarnings(pastecs::turnpoints(as.ts(x)))
          recordIDs <- data.frame(tp$peaks, tp$pits)
          rownames(recordIDs) <- rownames(x)
          colnames(recordIDs) <- c("peaks", "pits")
          timeSeries(data = x, charvec = time(x),
                     units = colnames(x), zone = finCenter(x),
                     FinCenter = finCenter(x),
                     recordIDs = recordIDs, title = x@title,
```

```
                        documentation = x@documentation)
          }
  )

[1] "turnpoints"
```

Using the function `isUnivariate()`, we first check if the input time series is univariate, then we compute the turnpoints, converting the timeSeries into an object of class `ts` as expect by the underlying function. Then we extract the peaks and pits and save them as record identification codes in the `data.frame recordIDs`, which is represented by a slot in the S4 `timeSeries` object. The result is given back as a `timeSeries` object.

Now let us compute the turnpoints for the smoothed `SPI`. We plot the original index series and the smoothed series. On top we put points for the peaks and pits in green and red, respectively.

```
> SPI.TP <- turnpoints(SPI.LW)
> SPI.PEAKS <- SPI.TP[SPI.TP@recordIDs[, "peaks"] == TRUE, ]
> SPI.PITS <- SPI.TP[SPI.TP@recordIDs[, "pits"] == TRUE, ]
> plot(SPI)
> lines(SPI.LW, col = "brown", lwd = 2)
> points(SPI.PEAKS, col = "green3", pch = 24)
> points(SPI.PITS, col = "red", pch = 25)
```

The turnpoints are added to Figure 2.1.

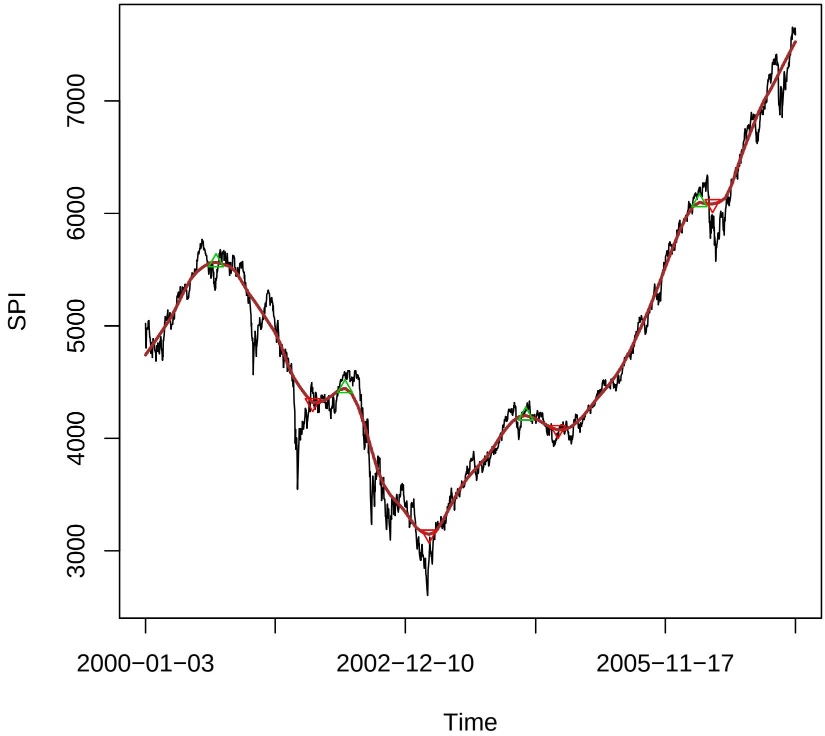


Figure 2.1 The graph shows the Swiss performance index `SPI` overlayed by a smoothed curve with turnpoints. For the smoother we used the R function `lowess()`, and for the turnpoints the contributed R function `turnpoints()` from the contributed R package `pastecs`.

Chapter 3
Basic Statistics of Financial Assets

Required R package(s):

```
> library(fPortfolio)
```

Rmetrics provides several functions and methods to compute basic statistics of financial time series from S4 `timeSeries` objects. These include summary and basic statistics, drawdown statistics, sample mean and covariance estimation, and quantile and risk estimation, amongst others.

Moreover, we have functions to compute column statistics and cumulated column statistics, which are very useful tools if we are interested in the statistical properties of each column of a data set of assets.

3.1 Summary Statistics

Three functions are available to compute basic statistics from a univariate or multivariate data set of assets, the generic `summary()` function and the functions `basicStats()` and `drawdownsStats()`.

Information on the size of a data set can be obtained from the functions `nrow`, `ncol`, `NROW`, `NCOL`, and `dim`. `nrow` and `ncol` return the number of rows or columns present in the `timeSeries` object `x`; `NCOL` and `NROW` do the same, but treat a univariate time series as 1-column multivariate time series.

```
Function:
summary             generates summary statistics of assets
basicStats          generates a basic statistics summary of assets
drawdownsStats      computes drawdown statistics from returns
mean, cov           computes sample mean and covariance of assets
skewness            computes sample skewness of assets
kurtosis            computes sample kurtosis of assets
quantile            computes quantiles of assets
colStats            computes column statistics of a data set of assets
colCumStats         computes cumulative column statistics of assets
covRisk             computes covariance portfolio risk
varRisk             computes value-at-risk for a portfolio
cvarRisk            computes conditional value-at-risk for a portfolio
```

Listing 3.1 Functions for computing basic statistics of financial returns

How to create summary statistics

The `summary()` function for `timeSeries` objects behaves in the same way as for numerical matrices. The function returns the minimum and maximum values for each series, the first and third quartiles, and the mean and median values. The following example computes summary statistics for the log-returns of the `SWX` data set.

```
> summary(SWX.RET)

      SBI                 SPI                 SII
 Min.   :-6.87e-03   Min.   :-0.069039   Min.   :-1.59e-02
 1st Qu.:-7.24e-04   1st Qu.:-0.004794   1st Qu.:-1.40e-03
 Median : 0.00e+00   Median : 0.000293   Median : 4.87e-05
 Mean   : 4.66e-06   Mean   : 0.000215   Mean   : 2.03e-04
 3rd Qu.: 7.85e-04   3rd Qu.: 0.005681   3rd Qu.: 1.85e-03
 Max.   : 5.76e-03   Max.   : 0.057860   Max.   : 1.54e-02
      LP25                LP40                LP60
 Min.   :-0.013154   Min.   :-0.019720   Min.   :-0.028106
 1st Qu.:-0.001248   1st Qu.:-0.001940   1st Qu.:-0.002916
 Median : 0.000247   Median : 0.000351   Median : 0.000430
 Mean   : 0.000139   Mean   : 0.000135   Mean   : 0.000123
 3rd Qu.: 0.001587   3rd Qu.: 0.002283   3rd Qu.: 0.003326
 Max.   : 0.013287   Max.   : 0.021178   Max.   : 0.032057
```

How to create a basic statistics report

The function `basicStats()` behaves similarly to `summary()` but returns a broader spectrum of statistical measures.

```
> args(basicStats)

function (x, ci = 0.95)
NULL
```

The argument `ci` specifies the confidence interval for calculating standard errors.

The following example computes daily basic statistics for the percentual log-returns of the three `SWX` indices, `SPI`, `SBI`, `SII`, and the `LP25` benchmark index from the `SWX` data set.

```
> basicStats(SWX.RET[, 1:4])

                    SBI         SPI         SII        LP25
nobs        1916.000000 1916.000000 1916.000000 1916.000000
NAs            0.000000    0.000000    0.000000    0.000000
Minimum       -0.006868   -0.069039   -0.015867   -0.013154
Maximum        0.005757    0.057860    0.015411    0.013287
1. Quartile   -0.000724   -0.004794   -0.001397   -0.001248
3. Quartile    0.000785    0.005681    0.001851    0.001587
Mean           0.000005    0.000215    0.000203    0.000139
Median         0.000000    0.000293    0.000049    0.000247
Sum            0.008930    0.412553    0.389689    0.266111
SE Mean        0.000030    0.000248    0.000069    0.000058
LCL Mean      -0.000054   -0.000270    0.000069    0.000025
UCL Mean       0.000063    0.000701    0.000338    0.000253
Variance       0.000002    0.000118    0.000009    0.000006
Stdev          0.001298    0.010843    0.003005    0.002542
Skewness      -0.313206   -0.221507    0.084294   -0.134810
Kurtosis       1.516963    5.213489    2.592051    2.893592
```

The `basicStats()` function returns a data frame with the following entries and row names: `nobs`, `NAs`, `Minimum`, `Maximum` , `1. Quartile`, `3. Quartile`, `Mean`, `Median`, `Sum`, `SE Mean`,`LCL Mean`, `UCL Mean`, `Variance`, `Stdev`, `Skewness`, `Kurtosis`.

How to compute drawdown statistics

To compute the drawdowns statistics for the `LPP25` benchmark index we use the `drawdownsStats()` function

```
> args(drawdownsStats)

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

which requires a univariate `timeSeries` object as input.

The example

```
> LP25 <- SWX.RET[, "LP25"]
> drawdownsStats(LP25)[1:10, ]

     drawdown       from     trough         to length peaktotrough
28  -0.084709 2001-05-23 2001-09-21 2003-08-22    588           88
118 -0.038749 2006-02-23 2006-06-13 2006-09-04    138           79
26  -0.031139 2001-02-07 2001-03-22 2001-05-17     72           32
54  -0.030972 2004-03-09 2004-06-14 2004-11-12    179           70
24  -0.021031 2000-09-06 2000-10-12 2001-02-06    110           27
104 -0.018214 2005-10-04 2005-10-28 2005-11-24     38           19
34  -0.017436 2003-09-19 2003-09-30 2003-11-03     32            8
8   -0.017321 2000-03-23 2000-05-22 2000-07-11     79           43
6   -0.016687 2000-01-18 2000-02-22 2000-03-17     44           26
152 -0.016227 2007-02-16 2007-03-14 2007-04-18     44           19
    recovery
28       500
118       59
26        40
54       109
24        83
104       19
34        24
8         36
6         18
152       25
```

returns the first ten drawdowns from the function value, which is a data.frame. The data frame lists the depth of the `drawdown`, the `from` (start) date, the `trough` period, the `to` (end) date, the `length` of the period, the `peaktotrough`, and the `recovery` periods. Note that lengths are measured in units of time series events.

3.2 Sample Mean and Covariance Estimates

How to compute the sample mean

A fundamental task in many statistical analyses is to estimate a location parameter for the distribution, that is to find a typical or central value that best describes the data.

```
Function:
mean                computes sample mean
var                 computes sample variance
cov                 computes sample covariance
skewness            computes sample skewness
kurtosis            computes sample kurtosis

Arguments:
x                   a 'timeSeries' object.
```

Listing 3.2 Functions to estimate moments and related quantities

Sample means can be computed using R's base functions `mean()`. Note that calling the function `mean()` on a multivariate time series will return the grand mean, as if the time series were a numeric matrix. To obtain the column means, which is what you usually require for your financial time series, you have to apply the function `colMeans()`.

```
> mean(100 * SWX.RET)

[1] 0.013664

> colMeans(100 * SWX.RET)

       SBI        SPI        SII       LP25       LP40       LP60
0.00046605 0.02153198 0.02033869 0.01388886 0.01349041 0.01226859
```

How to compute the sample variance and covariance

Sample variance and covariance can be computed using the R base functions `mean()` and `cov()`. Note that R's base function `cov()` operates in the same way on a `timeSeries` object as on a numeric `matrix`.

```
> Covariance <- round(cov(100 * SWX.RET), digits = 4)
> Covariance

        SBI     SPI    SII    LP25    LP40    LP60
SBI  0.0169 -0.0415 0.0014 -0.0011 -0.0094 -0.0206
```

```
SPI  -0.0415  1.1757 0.0066  0.2204  0.3617  0.5464
SII   0.0014  0.0066 0.0903  0.0027  0.0041  0.0062
LP25 -0.0011  0.2204 0.0027  0.0646  0.0993  0.1464
LP40 -0.0094  0.3617 0.0041  0.0993  0.1578  0.2372
LP60 -0.0206  0.5464 0.0062  0.1464  0.2372  0.3609
```

Here, we have rounded the output to four digits.

3.3 Estimates for Higher Moments

How to compute the sample skewness

Skewness is a measure of symmetry, or more precisely, the lack of symmetry. A distribution, or data set, is symmetric if it looks the same to the left and right of the centre point.

```
> args(skewness)

function (x, ...)
NULL

> SPI <- SWX[, "SPI"]
> skewness(SPI)

[1] 0.51945
attr(,"method")
[1] "moment"
```

How to compute the sample kurtosis

Kurtosis is a measure of whether the data are peaked or flat relative to a normal distribution. That is, data sets with high kurtosis tend to have a distinct peak near the mean, decline rather rapidly, and have heavier tails. Data sets with low kurtosis tend to have a flat top near the mean rather than a sharp peak[1].

```
> args(kurtosis)
```

[1] A distribution with high kurtosis is known as leptokurtic, whereas a distribution with low kurtosis is platykurtic.

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

> kurtosis(SPI)

[1] -0.31378
attr(,"method")
[1] "excess"
```

Note that R comes with the base functions `mean()` and `cov()`, but does not provide functions to compute skewness and kurtosis. The functions `skewness()` and `kurtosis()` are added by Rmetrics.

3.4 Quantiles and Related Risk Measures

How to compute quantiles

Quantiles of assets can be calculated using R's base generic function `quantile()`. This function produces sample quantiles corresponding to the given probabilities. The smallest observation corresponds to a probability of 0 and the largest to a probability of 1. Note that according to Hyndman & Fan (1996) there are different ways to compute quantiles. The method used in finance for calculating the (conditional) Value at Risk is `type=1`, which is not the default setting.

```
> quantile(SWX.RET, probs = seq(0, 1, 0.25), type = 1)

         0%         25%         50%         75%        100%
-2.0881e-03  1.2752e-03 -4.2049e-03 -1.0439e-04 -7.9406e-05
```

As you can see, the function concatenates the columns of all assets in the data set to one vector (the same as for the `mean()`) and then computes the quantiles. To compute the quantiles for each column, use the function `colQuantiles()`, and do not forget to specify the proper `type=1`.

```
> colQuantiles(SWX.RET, prob = 0.05, type = 1)

       SBI        SPI        SII       LP25       LP40       LP60
-0.0022108 -0.0175881 -0.0044958 -0.0041020 -0.0064546 -0.0098262
```

Portfolio risk measures

To compute the three major risk measures for portfolios Rmetrics provides the functions `covRisk`, `varRisk`, and `cvarRisk`.

```
Function:
covRisk             computes covariance portfolio risk
varRisk             computes Value at Risk for a portfolio
cvarRisk            computes conditional Value at Risk

Arguments:
x                   a 'timeSeries' object of asset returns
weights             vector of portfolio weights
alpha               the VaR and CVaR confidence level
```

Listing 3.3 Functions to compute portfolio risk measures

The example shows the three risk measures for an equally weighted portfolio composed of Swiss equities, `SPI`, Swiss bonds, `SBI`, and Swiss reits, `SII`. For the sample covariance risk we obtain

```
> SWX3 <- 100 * SWX.RET[, 1:3]
> covRisk(SWX3, weights = c(1, 1, 1)/3)

    Cov
0.36755
```

and for the sample VaR and Conditional VaR we obtain

```
> varRisk(SWX3, weights = c(1, 1, 1)/3, alpha = 0.05)

  VaR.5%
-0.56351

> cvarRisk(SWX3, weights = c(1, 1, 1)/3, alpha = 0.05)

 CVaR.5%
-0.87832
```

How to detect extreme values and outliers

The Rmetrics package `fExtremes` allows us to investigate univariate `timeSeries` objects from the point of view of extreme value theory. The package provides functions to investigate extreme values in a time series using peak over threshold and block methods. From this we can estimate *Value-at-Risk* and *Conditional-Value at-Risk* much more reliably than is possible using sample estimates.

For a detailed description of the statistical approaches and algorithms for the analysis of extreme values in financial time series we refer to the Rmetrics ebook *Managing Risk with R/Rmetrics*.

3.5 Computing Column Statistics

Rmetrics implements 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`.

Amongst the column statistics functions are

```
Functions:
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

Arguments:
x                   a 'timeSeries' object.
```

Listing 3.4 Column statistics functions

```
> 100 * colMeans(returns(SWX))

       SBI        SPI        SII       LP25       LP40       LP60
0.00046605 0.02153198 0.02033869 0.01388886 0.01349041 0.01226859

> 100 * colQuantiles(returns(SWX))
```

```
     SBI      SPI      SII     LP25     LP40     LP60
-0.21941 -1.74757 -0.44678 -0.40828 -0.64300 -0.98188
```

You can also define your own statistical functions and execute them with the function `colStats()`. If, for example, you want to know the column medians of the `timeSeries`, you can simply write

```
> round(colStats(returns(SWX, percentage = TRUE), FUN = "median"),
     digits = 4)
```

```
   SBI    SPI    SII   LP25   LP40   LP60
0.0000 0.0293 0.0049 0.0247 0.0351 0.0430
```

3.6 Computing Cumulated Column Statistics

Functions to compute cumulated column statistics are also available in Rmetrics. These are

```
Functions:
colCumstats          returns user-defined column 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

Arguments:
x                    a 'timeSeries' object.
```

Listing 3.5 Functions for cumulated column statistics

The function `colCumstats()` allows you to define your own functions to compute cumulated column statistics, in the same way as for the function `colStats()`.

Chapter 4

Robust Mean and Covariance Estimates of Assets

Required R package(s):

```
> library(fPortfolio)
```

Robust statistics provides an alternative approach to classical statistical methods. The idea behind robust statistics is to produce estimators that are not unduly affected by small departures from model assumptions. In Rmetrics we have included robust estimators for the mean and covariance of financial assets from several R packages.

Additionally, we have implemented a covariance ellipse plot, which visualizes the difference between two or more covariance matrices. It is intended to compare different methods of covariance estimation. We also show how to detect multivariate outliers.

Robust covariance estimators of a data set of asset returns are of great interest for the optimization of robust mean-covariance portfolios, where we replace the sample covariance estimate with a robust covariance estimate. From many investigations, we know that the use of robust covariances instead of the sample covariance achieves a much better diversification of the mean-variance portfolio weights.

4.1 Robust Covariance Estimators

The function `assetsMeanCov()` provides a collection of several robust estimators. The functions have their origin in several contributed R packages for robust estimation.

```
Function:
assetsMeanCov       returns robustified covariance estimates
getCenterRob        extracts the robust centre estimate
getCovRob           extracts the robust covariance estimate
covEllipsesPlot     creates a covariance ellipses plot
assetsOutliers      detects multivariate outliers in assets

Arguments:
x                   a univariate 'timeSeries' object
method              the method of robustification:
  "cov"             uses the sample covariance estimator from [base]
  "mve"             uses the "mve" estimator from [MASS]
  "mcd"             uses the "mcd" estimator from [MASS]
  "MCD"             uses the "MCD" estimator from [robustbase]
  "OGK"             uses the "OGK" estimator from [robustbase]
  "nnve"            uses the "nnve" estimator from [covRobust]
  "shrink"          uses "shrinkage" estimator from [corpcor]
  "bagged"          uses "bagging" estimator from [corpcor]
```

Listing 4.1 The robust function estimators functions

First, let us have a look at the argument list of the function `assetsMeanCov()` and the methods provided.

```
> args(assetsMeanCov)

function (x, method = c("cov", "mve", "mcd", "MCD", "OGK", "nnve",
    "shrink", "bagged"), check = TRUE, force = TRUE, baggedR = 100,
    sigmamu = scaleTau2, alpha = 1/2, ...)
NULL
```

Through the argument `method`, we can select the desired estimator. The function `assetsMeanCov()` returns a named list with four entries `center` (the estimated mean), `cov` (the estimated covariance matrix), and `mu` and `Sigma` which are just synonyms for `center` and `cov`. In addition the returned value of the function has a control attribute `attr(,"control")`, a character vector which holds the name of the `method` of the estimator, the `size` (number) of assets, and two flags. If the covariance matrix was positive definite then the `posdef` flag is set to `TRUE`, and if not, then the flag is set to `FALSE`.

```
> assetsMeanCov(100 * SWX.RET[, c(1:2, 4:5)], method = "cov")

$center
       SBI        SPI       LP25       LP40
0.00046605 0.02153198 0.01388886 0.01349041

$cov
            SBI       SPI       LP25      LP40
SBI   0.0168515 -0.041468 -0.0010760 -0.009442
SPI  -0.0414682  1.175681  0.2203762  0.361670
LP25 -0.0010760  0.220376  0.0646387  0.099278
LP40 -0.0094419  0.361670  0.0992780  0.157780

$mu
       SBI        SPI       LP25       LP40
0.00046605 0.02153198 0.01388886 0.01349041

$Sigma
            SBI       SPI       LP25      LP40
SBI   0.0168515 -0.041468 -0.0010760 -0.009442
SPI  -0.0414682  1.175681  0.2203762  0.361670
LP25 -0.0010760  0.220376  0.0646387  0.099278
LP40 -0.0094419  0.361670  0.0992780  0.157780

attr(,"control")
 method    size  posdef  forced  forced
  "cov"     "4"  "TRUE" "FALSE"  "TRUE"
```

The functions `getCenterRob()` and `getCovRob()` can be used to extract the robust mean, `center`, and the robust covariance, `cov`, from an object as returned by the function `assetsMeanCov()`.

4.2 Comparisons of Robust Covariances

The function covEllipsesPlot() visualizes the differences between two covariance matrices. This allows us to compare the sample estimate with robust estimates, or to compare robust estimators with each other.

How to display ellipses plots

```
> args(covEllipsesPlot)

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

The list argument has at least two covariance matrices as input, the dots argument allows us to pass optional arguments to the underlying plot, lines and text functions. Several examples how to use the covEllipsesPlot() function are shown in the following sections when we compare different robust covariance estimates.

4.3 Minimum Volume Ellipsoid Estimator

The method "mve" is the minimum volume ellipsoid estimator as implemented in R's recommended package MASS (Venables & Ripley, 2008). This is called internally with the argument method="mve"

```
> args(MASS::cov.rob)

function (x, cor = FALSE, quantile.used = floor((n + p + 1)/2),
    method = c("mve", "mcd", "classical"), nsamp = "best", seed)
NULL
```

The following example shows how to call the estimator from the function suite assetsMeanCov() and how to extract the robust $center and cov estimate. We investigate Swiss, SPI, and foreign equity indexes, MPI, and Swiss, SBI, and foreign bond indexes, LMI, which are stored in the columns 1, 2, 4, 5 of the LPP2005.RET data set

```
> lppData <- 100 * LPP2005.RET[, c(1:2, 4:5)]
> ans.mve <- assetsMeanCov(lppData, "mve")
> getCenterRob(ans.mve)

      SBI       SPI       LMI       MPI
0.0025143 0.1204896 0.0039215 0.0886053

> getCovRob(ans.mve)

           SBI       SPI        LMI       MPI
SBI  0.0121408 -0.017099  0.0078555 -0.016626
SPI -0.0170993  0.339045 -0.0153319  0.232042
LMI  0.0078555 -0.015332  0.0126992 -0.021367
MPI -0.0166264  0.232042 -0.0213668  0.368959

> attr(ans.mve, "control")

 method    size  posdef  forced  forced
  "mve"     "4"  "TRUE" "FALSE"  "TRUE"
```

Note that an attribute called `"control"` is returned, which allows us to extract additional information from the selected estimator.

With the help of the function `covEllipsesPlot()`, we can now compare the sample covariances with the `"mve"` robustified covariances

```
> covEllipsesPlot(list(cov(lppData), ans.mve$cov))
> title(main = "Sample vs. MVE Covariances")
```

The result is shown in Figure 4.1.

4.4 Minimum Covariance Determinant Estimator

Two methods, called `"mcd"` and `"MCD"`, are available to estimate a robust mean and covariance by the minimum covariance determinant estimator (Rousseeuw, 1985; Rousseeuw & Van Driessen, 1999). The first method uses the function `cov.rob()` from the `MASS` package (Venables & Ripley, 2008).

```
> args(MASS::cov.rob)
```

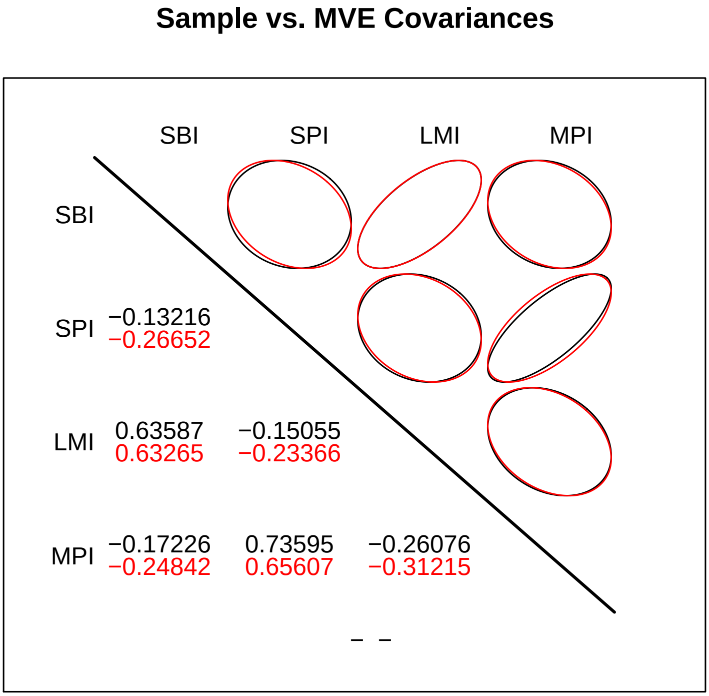


Figure 4.1 Comparison of sample and MVE robust Covariances.

```
function (x, cor = FALSE, quantile.used = floor((n + p + 1)/2),
    method = c("mve", "mcd", "classical"), nsamp = "best", seed)
NULL
```

and the second method uses the function `covMcd()` from the contributed package `robustbase` (Rousseeuw, Croux, Todorov, Ruckstuhl, Salibian-Barrera, Verbeke & Maechler, 2008). The second method requires loading the contributed package `robustbase` explicitly.

```
> library(robustbase)
> args(robustbase::covMcd)
```

```
function (x, cor = FALSE, alpha = 1/2, nsamp = 500, seed = NULL,
    trace = FALSE, use.correction = TRUE, control = rrcov.control())
NULL
```

You can call the estimators for the two methods as

```
> ans.mcd <- assetsMeanCov(lppData, "mcd")
> ans.MCD <- assetsMeanCov(lppData, "MCD")
```

and compare them

```
> getCovRob(ans.mcd)
```

```
           SBI       SPI        LMI       MPI
SBI  0.0133197 -0.012490  0.0083687 -0.013429
SPI -0.0124895  0.329241 -0.0100234  0.205545
LMI  0.0083687 -0.010023  0.0127320 -0.015668
MPI -0.0134294  0.205545 -0.0156683  0.324125
```

```
> getCovRob(ans.MCD)
```

```
           SBI       SPI        LMI       MPI
SBI  0.0159968 -0.011929  0.0097842 -0.016060
SPI -0.0119294  0.403244 -0.0101378  0.242938
LMI  0.0097842 -0.010138  0.0149387 -0.018262
MPI -0.0160600  0.242938 -0.0182619  0.370622
```

```
> covEllipsesPlot(list(ans.mcd$cov, ans.MCD$cov))
```

The result is shown in Figure 4.2.

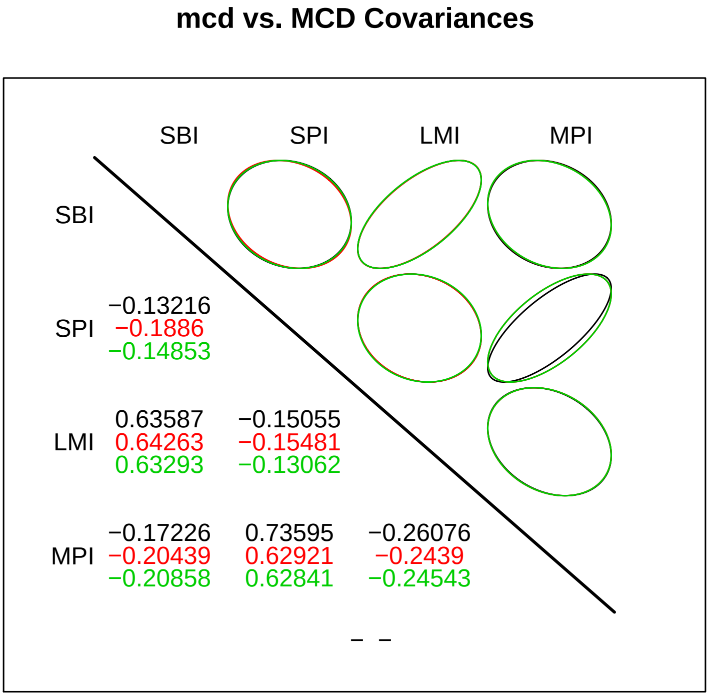


Figure 4.2 Comparison of sample and mcd/MCD robust covariances.

4.5 Orthogonalized Gnanadesikan-Kettenring Estimator

The "OGK" method[1] computes the orthogonalized pairwise covariance matrix estimate described in Maronna & Zamar (2002). The pairwise proposal goes back to Gnanadesikan & Kettenring (1972). The estimator is implemented in the contributed R package robustbase (Rousseeuw et al., 2008).

```
> library(robustbase)
> args(robustbase::covOGK)

function (X, n.iter = 2, sigmamu, rcov = covGK, weight.fn = hard.rejection,
    keep.data = FALSE, ...)
NULL
```

We can use the estimator in the same way as the others

```
> ans.ogk <- assetsMeanCov(lppData, "OGK")
> getCovRob(ans.ogk)

          SBI       SPI       LMI       MPI
SBI  0.016585 -0.010687  0.010368 -0.012045
SPI -0.010687  0.420980 -0.010904  0.275285
LMI  0.010368 -0.010904  0.015081 -0.017319
MPI -0.012045  0.275285 -0.017319  0.407193

> covEllipsesPlot(list(cov(lppData), ans.ogk$cov))
```

The result is shown in Figure 4.3.

4.6 Nearest-Neighbour Variance Estimator

The method "nnve" provides robust covariance estimation by the nearest neighbour variance estimation method of Wang & Raftery (2002). The function cov.nnve() is implemented in the contributed R package covRobust (Wang, Raftery & Fraley, 2008) and is available as an internal function built into Rmetrics' .cov.nnve()[2] function.

[1] The "OKG" method is very efficient for large covariance matrices.

[2] This means that the loading of the R package covRobust is not required.

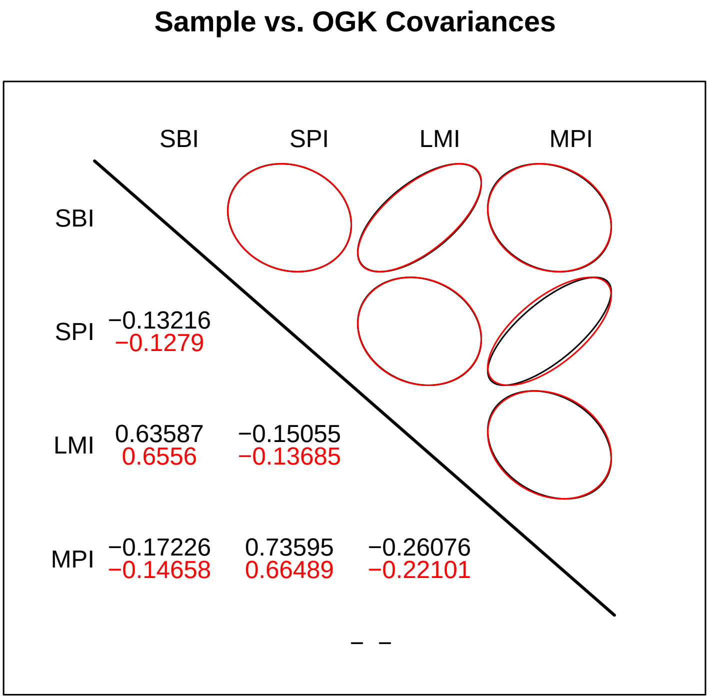


Figure 4.3 Comparison of sample and OGK robust covariances.

```
> args(.cov.nnve)

function (datamat, k = 12, pnoise = 0.05, emconv = 0.001, bound = 1.5,
    extension = TRUE, devsm = 0.01)
NULL
```

and again

```
> ans.nnve <- assetsMeanCov(lppData, "nnve")
> getCenterRob(ans.nnve)

       SBI        SPI        LMI        MPI
4.0663e-05 8.4175e-02 5.5315e-03 5.9052e-02

> getCovRob(ans.nnve)

           SBI        SPI        LMI       MPI
SBI  0.0115126 -0.0092162  0.0067991 -0.010614
SPI -0.0092162  0.2977014 -0.0080603  0.180524
LMI  0.0067991 -0.0080603  0.0111734 -0.014240
MPI -0.0106135  0.1805241 -0.0142400  0.270448

> attr(ans.nnve, "control")

 method    size  posdef  forced  forced
 "nnve"     "4"  "TRUE" "FALSE"  "TRUE"

> covEllipsesPlot(list(cov(lppData), ans.nnve$cov))
```

4.7 Shrinkage Estimator

The `"shrink"` method provides robust covariance estimation by the shrinkage method of ... .

The `shrinkage()` function is implemented in the contributed R package `corpcor` (Schaefer, Opgen-Rhein & Strimmer, 2008) and is available as an internal function built into the Rmetrics' `.cov.nnve()`[3] function.

```
> ans.shrink <- assetsMeanCov(lppData, "shrink")
> getCovRob(ans.shrink)
```

[3] This means that the loading of the Rpackage `covRobust` is not required.

```
           SBI       SPI        LMI       MPI
SBI  0.0158996 -0.012387  0.0095313 -0.015447
SPI -0.0123872  0.584612 -0.0136834  0.400155
LMI  0.0095313 -0.013683  0.0149511 -0.022674
MPI -0.0154466  0.400155 -0.0226738  0.535033
attr(,"lambda")
[1] 0.027802
```

```
> covEllipsesPlot(list(cov(lppData), ans.shrink$cov))
```

4.8 Bagging Estimator

The "bagged" method provides a variance-reduced estimator of the covariance matrix using *bootstrap aggregation*, hence the name "bagged".

"bagged" was implemented in a previous version of the contributed R package corpcor and is available as an internal function built into Rmetrics[4] [5].

```
> ans.bagged <- assetsMeanCov(lppData, "bagged")
> getCovRob(ans.bagged)
```

```
           SBI       SPI        LMI       MPI
SBI  0.0158323 -0.012814  0.0098123 -0.015934
SPI -0.0128145  0.586052 -0.0147162  0.414986
LMI  0.0098123 -0.014716  0.0149961 -0.023618
MPI -0.0159336  0.414986 -0.0236181  0.536907
```

```
> attr(ans.bagged, "control")
```

```
  method        R     size   posdef   forced    forced
"bagged"    "100"      "4"   "TRUE"  "FALSE"    "TRUE"
```

```
> covEllipsesPlot(list(cov(lppData), ans.bagged$cov))
```

[4] This means that the loading of the R package corpcor is not required.

[5] "bagged" is built into Rmetrics' .cov.bagged() function

4.9 How to Add a New Estimator to the Suite

It is very simple to add your own estimators to the suite. To do this, you have to define an estimator function, which takes as its first argument a `timeSeries` object, and has a dots argument which allows you to pass optional arguments to the underlying estimator. This function has to return a list, with at least two entries, called `$center` and `$cov`.

How to add a multivariate Student's t estimator

The recommended `MASS` citepVenables2008The-MAS package implements a covariance estimator for the multivariate Student's t Distribution. The assumption that the data come from a multivariate Student's t distribution provides some degree of robustness to outliers without giving a high breakdown point. The breakdown point of an estimator is intuitively the proportion of incorrect large observations an estimator can handle before giving an arbitrarily large result.

To add this estimator to the function `assetsMeanCov()`, proceed as follows:

```
> covt <- function(x, ...) MASS::cov.trob(x, ...)
> ans.covt <- assetsMeanCov(lppData, method = "covt")
> getCenterRob(ans.covt)

       SBI        SPI        LMI        MPI
-0.0012241  0.1187899  0.0017686  0.0920948

> getCovRob(ans.covt)

           SBI        SPI        LMI       MPI
SBI  0.0117503 -0.0091287  0.0073747 -0.011440
SPI -0.0091287  0.3581261 -0.0088391  0.239793
LMI  0.0073747 -0.0088391  0.0111483 -0.015084
MPI -0.0114398  0.2397926 -0.0150840  0.335681
```

How to write an adaptive re-weighted estimator

The contributed R package `mvoutlier` (Gschwandtner & Filzmoser, 2009) implements an adaptive re-weighted estimator for multivariate location and scatter with hard-rejection weights. The multivariate outliers are defined according to the supremum of the difference between the empirical distribution function of the robust Mahalanobis distance and the theoretical distribution function.

```
> library(mvoutlier)
> arw <- function(x, ...) {
     ans <- mvoutlier::arw(as.matrix(x), colMeans(x), cov(x), ...)
     list(center = ans$m, cov = ans$c)
 }
> ans.arw <- assetsMeanCov(lppData, method = "arw")
> getCenterRob(ans.arw)

      SBI       SPI       LMI       MPI
0.0019389 0.1503808 0.0060246 0.1121257

> getCovRob(ans.arw)

           SBI        SPI        LMI       MPI
SBI  0.0147643 -0.0092163  0.0090874 -0.013993
SPI -0.0092163  0.4045949 -0.0078659  0.267712
LMI  0.0090874 -0.0078659  0.0136485 -0.017386
MPI -0.0139928  0.2677117 -0.0173858  0.398549
```

4.10 How to Detect Outliers in a Set of Assets

The function `assetsOutliers()` allows us to detect outliers by analyzing the estimates for the mean (`center`) and for the covariance matrix (`cov`) as described by Filzmoser, Garrett & Reimann (2005).

```
Function:
assetsOutliers      detects multivariate outliers in assets.

Values:
center              mean vector as given by the input
cov                 covariance matrix as given by the input
cor                 correlation matrix computed from the covariances
...                 optional arguments to be passed in
quantile            quantile
outliers            vector of outliers
series              return series of outliers
```

Listing 4.2 Function to detect outliers in a multivariate data set of asset returns

```
> args(assetsOutliers)
```

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

The `assetsOutliers()` function expects as input a multivariate `timeSeries` object and returns a named list with the mean vector, the covariance and correlation matrices, the quantile, and the outliers. From the vector of outliers we can relate position numbers to dates, and from the series we obtain the corresponding outlier returns.

The following example shows outlier detection for the `mcd` estimator

```
> outliers <- assetsOutliers(lppData, ans.mcd$center, ans.mcd$cov)
> outliers

$center
        SBI         SPI         LMI         MPI
-0.00017813  0.12976675  0.00043613  0.09916002

$cov
           SBI        SPI        LMI       MPI
SBI  0.0139246 -0.0093237  0.0084616 -0.014067
SPI -0.0093237  0.3580961 -0.0085850  0.230081
LMI  0.0084616 -0.0085850  0.0126282 -0.017116
MPI -0.0140674  0.2300810 -0.0171158  0.335830

$cor
         SBI      SPI      LMI      MPI
SBI  1.00000 -0.13204  0.63810 -0.20571
SPI -0.13204  1.00000 -0.12766  0.66347
LMI  0.63810 -0.12766  1.00000 -0.26282
MPI -0.20571  0.66347 -0.26282  1.00000

$quantile
[1] 12.697

$outliers
2005-11-04 2005-11-16 2005-12-14 2006-01-23 2006-03-28 2006-04-18
         4         12         32         60        106        121
2006-05-11 2006-05-12 2006-05-17 2006-05-18 2006-05-22 2006-05-23
       138        139        142        143        145        146
2006-05-24 2006-05-26 2006-05-30 2006-06-02 2006-06-05 2006-06-06
       147        149        151        154        155        156
2006-06-08 2006-06-13 2006-06-15 2006-06-29 2006-06-30 2006-07-19
       158        161        163        173        174        187
2006-07-24 2006-08-22 2006-09-22 2006-12-11 2007-02-27 2007-03-14
       190        211        234        290        346        357
```

```
$series
GMT
                 SBI       SPI       LMI       MPI
2005-11-04 -0.323575 -0.070276 -0.119853  1.167956
2005-11-16  0.299966 -0.718750  0.277342  0.387121
2005-12-14 -0.322309 -0.728358  0.224813 -0.647012
2006-01-23 -0.083813  0.008050  0.030172 -1.557646
2006-03-28 -0.285880 -0.072786 -0.336518 -0.824240
2006-04-18 -0.031136 -0.503126  0.111080  0.966265
2006-05-11 -0.377003 -0.109661 -0.133194 -1.142449
2006-05-12 -0.094473 -1.798973 -0.106245 -2.357418
2006-05-17 -0.196379 -2.840692 -0.187847 -1.400989
2006-05-18  0.180683 -0.971142  0.329356 -1.354839
2006-05-22  0.305009 -2.599776  0.350507 -3.009080
2006-05-23  0.000000  1.897068  0.179534  0.691444
2006-05-24  0.132662 -1.111559 -0.206778  0.119183
2006-05-26  0.038985  2.584213 -0.074680  2.192910
2006-05-30 -0.062373 -1.984241 -0.021746 -2.788328
2006-06-02  0.233973  0.699306  0.367916 -0.080858
2006-06-05  0.000000  0.000000 -0.051341 -1.457172
2006-06-06 -0.109119 -2.232654 -0.131750 -0.478857
2006-06-08  0.116900 -2.737931  0.191942 -0.923718
2006-06-13  0.139762 -2.399230  0.234377 -2.270706
2006-06-15 -0.209522  2.156939 -0.302316  2.407531
2006-06-29  0.062671  1.404176  0.182883  1.997279
2006-06-30 -0.054835  1.447381  0.152170 -0.237578
2006-07-19  0.085907  1.841958  0.184291  1.563777
2006-07-24  0.085760  1.923981 -0.025710  2.024162
2006-08-22  0.330960  0.313653  0.153983  0.976584
2006-09-22  0.198504 -0.918960  0.279948 -1.745647
2006-12-11 -0.258418  0.621642  0.058515  0.759359
2007-02-27  0.160030 -3.574624  0.258065 -3.375456
2007-03-14  0.114203 -2.820549  0.065135 -1.747682
```

Part II
Exploratory Data Analysis of Assets

Introduction

In Chapter 5 we show how to create and display graphs and plots of financial time series and their properties. We show how to use the generic plot function to produce univariate and multivariate graphs of assets. Several hints and recipes are given to customize the plots to the user's needs. In addition to the time series plots, we show how to display box plots, histograms and density plots, and quantile-quantile plots.

In Chapter 6 we present hints and tricks to customize graphs. This concerns plot labels, axis labels, the use of optional plot function arguments and how to select colours, fonts and plot symbols.

In Chapter 7 we show how to estimate the parameters of a data set of assets to a normal or Student's t distribution and how to simulate artificial data sets with the same statistical properties. In order to test whether the empirical asset returns are multivariate normally distributed we can perform hypothesis tests.

Chapter 8 deals with portfolio selection. We try to find which assets in a portfolio are similar, and thus grouped together in clusters. We introduce approaches which are suggested from a statistical point of view. We consider two approaches which group the asset returns by hierarchical or, alternatively, by k-means clustering. In addition, we show how we can group similar assets by an eigenvalue decomposition of the asset returns series. As a visual approach to detect similarities or dissimilarities we discuss star plots.

In Chapter 9 we discuss star and segment plots.

In Chapter 10 we concentrate on the pairwise comparison of assets. To make dependencies among the assets visible we display the correlation between two assets as scatter plots. In addition, we present alternative views displaying min/max panels, histogram panels, pie (or pac man) panels, shaded square panels, coloured ellipse panel, correlation test panels, and lowess fit panels. As an alternative, image correlation plots and bivariate hexagonal binned histogram plots are available.

Chapter 5

Plotting Financial Time Series And Their Properties

Required R package(s):

```
> library(fPortfolio)
```

Rmetrics offers several kinds of plot functions for quick and efficient exploratory data analysis of financial assets. These include *financial time series plots* for prices/indices, returns and their cumulated values, and plots for displaying their distributional properties: *box plots*, *histogram and density plots*, and *quantile-quantile plots*.

5.1 Financial Time Series Plots

How to use the generic plot functions

The `plot()` function is a generic function to plot univariate and multivariate `timeSeries` objects. Furthermore, the two generic functions `lines()` and `points()` allow us to add lines and points to an already existing plot.

The `plot()` function is implemented in the same spirit as the function `plot.ts()` for regular time series objects, `ts`, in R's base package `stats`. The function comes with the same arguments and some additional arguments, for user-specified `"axis"` labelling, and for modifying the plot `"layout"`. As for `ts`, three different types of plots can be displayed: a multiple plot, a single plot, and a scatter plot.

```
Function:
plot              displays a plot of a timeSeries object.
  lines             adds lines to an already existing plot.
  points            adds lines to an already existing plot.
seriesPlot        displays a time series plot given by its input.
returnPlot        displays returns given the price or index series.
cumulatedPlot     displays a cumulated series given the returns.
```

Listing 5.1 Plot and related functions

How to generate multiple plots

If the input argument `x` is a multivariate `timeSeries` object then the generic plot function creates a graph for each individual series. Up to ten subplots can be produced on one page.

```
> colnames(LPP2005.RET)

[1] "SBI"   "SPI"   "SII"   "LMI"   "MPI"   "ALT"   "LPP25" "LPP40" "LPP60"

> plot(LPP2005.RET, main = "LPP Pension Fund", col = "steelblue")
```

How to generate single plots

If the input argument `x` is a multivariate `timeSeries` object and the argument `plot.type` is set to `"single"` then the generic plot function creates a plot, where all curves are drawn in one plot on the same page.

```
> plot(SWX[, 6:4], plot.type = "single", , col = 2:4, xlab = "Date",
     ylab = "LP Index Family")
> title(main = "LP25 - LP40 - LP60")
> hgrid()
```

How to generate scatter plots

If two arguments `x` and `y` are specified, the generic plot function generates a scatter plot of two univariate `timeSeries` objects.

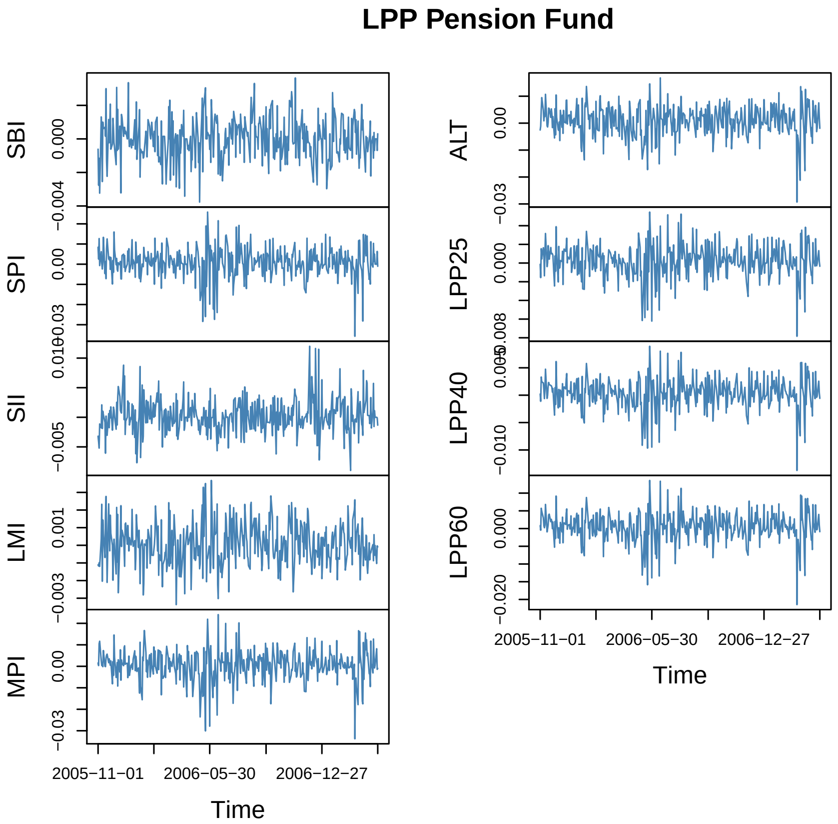


Figure 5.1 Time series plots of the Swiss pension fund benchmark: The generic plot function creates a graph for each individual series where up to 10 subplots can be produced on one sheet of paper. The series of graphs shows the logarithmic returns of six asset classes and the three benchmark series included in the LPP2005 benchmark index.

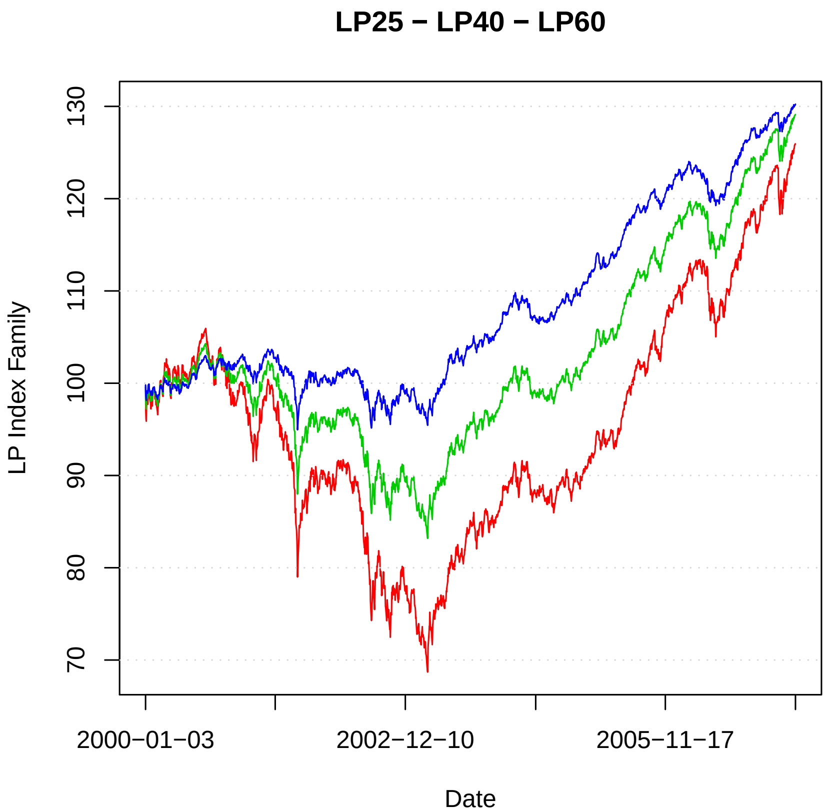


Figure 5.2 Time series plots of the LPP benchmark indices: The series of the three graphs show the logarithmic returns of the LPP benchmark indices, LPP25, LPP40, are part of the LPP2005 pension fund benchmark index family.

```
> SBI.RET <- 100 * SWX.RET[, "SBI"]
> SPI.RET <- 100 * SWX.RET[, "SPI"]
> plot(SBI.RET, SPI.RET, xlab = "SBI", ylab = "SPI", pch = 19,
     cex = 0.4, col = "brown")
> grid()
```

This plot is useful if we want to compare the daily returns of two time series day by day. It gives an impression of the strength of correlations.

How to use tailored plot functions

Rmetrics comes with three major types of tailored plots to display a financial time series. We can display the price or index series given either the series itself or the returns, and we can also display the financial returns given the returns themselves or the price or index series. A third option allows us to plot the cumulated series when financial returns are given.

```
Functions:
seriesPlot          generates an index plot
returnPlot          generates a financial returns plot
cumulatedPlot       generates a cumulative series plot

Arguments:
labels              a logical flag. Should the plot be returned with
                    default labels? By default TRUE
type                determines type of plot. By default we
                    use a line plot, type="l". An alternative
                    plot style which produces nice figures is for example
                    type="h"
col                 the colour for the series. In the univariate case use
                    just a colour name like the default, col="steelblue",
                    in the multivariate case we recommend selecting the
                    colours from a colour palette, e.g.
                    col=heat.colors(ncol(x))
title               a logical flag, by default TRUE. Should a
                    default title be added to the plot?
grid                a logical flag. Should a grid be added to the plot?
                    By default TRUE
box                 a logical flag. Should a box be added to the plot?
                    By default TRUE
rug                 a logical flag. By default TRUE. Should a
                    rug representation of the data added to the plot?
```

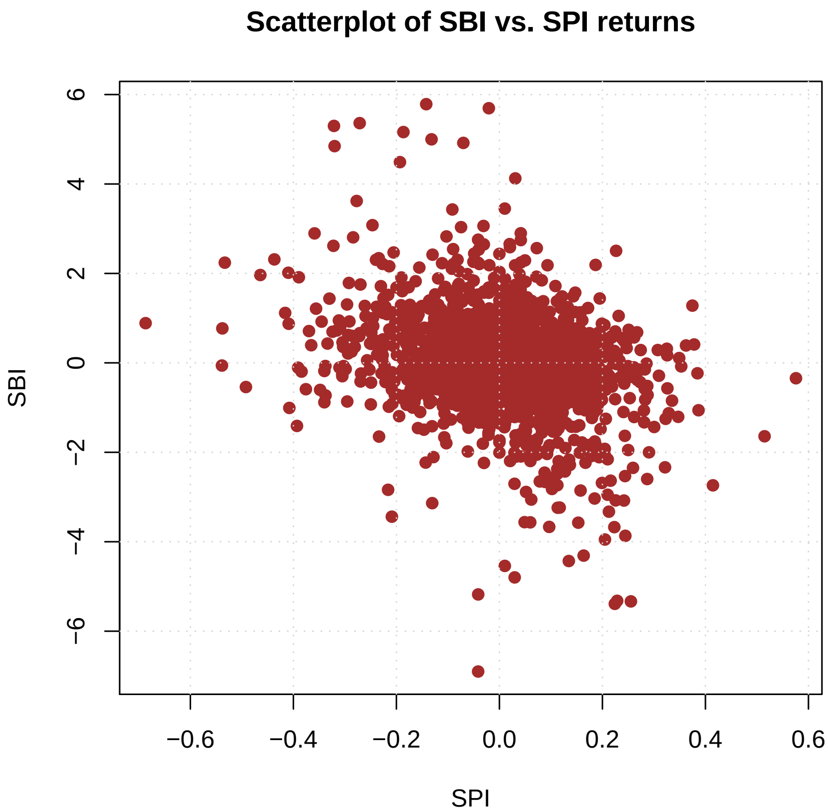


Figure 5.3 Scatter plot of the SPI versus the SBI: The generic plot function can also create scatter plots which show the values of the first versus the second time series. This figure shows the scatter plot for logarithmic returns of the SPI versus SBI indices in percentages.

Listing 5.2 Tailored plot functions and their arguments

`seriesPlot()` displays the financial time series as given by its input. In most cases this may be either a price or index series when the prices or index values are given as input, or a return series when the values are given as financial returns. If the input values represent returns and we want to plot their cumulated values over time, we use the function `cumulatedPlot()`, and, in the opposite case, if we have a cumulated series and want to display the returns, we use the function `returnPlot()`.

Let us consider some examples. The example data file `SWX` contains in its columns the index values for the *Swiss Bond Index*, for the *Swiss Performance Index*, and for the *Swiss Immofunds Index*, SII. In the following code snippet the first line loads the example data file and converts it into a time series object, the second line extracts the SPI column, and the last line computes logarithmic returns from the index.

```
> SPI <- SWX[, "SPI"]
> SPI.RET <- SWX.RET[, "SPI"]
```

To create default plots we just call the functions `seriesPlot()`, `returnPlot()` and `cumulatedPlot()`

```
> seriesPlot(SPI)
> returnPlot(SPI)
> cumulatedPlot(SPI.RET)
```

The three graphs for the Swiss Performance Index are shown in Figure 5.4.

The functions `seriesPlot()`, `returnPlot()` and `cumulatedPlot()` also allow for multivariate plots on one or more sheets. To create a two-column plot for the three SWX indices and the three LPP benchmarks on one sheet we proceed as follows:

```
> par(mfcol = c(3, 2))
> seriesPlot(SWX)
```

The indices for the SBI, SPI, SII, as well as for the three Pension Funds indices LPP25, LPP40, and LPP60 are shown in Figure 5.5.

Notice that the arguments of the three plot functions

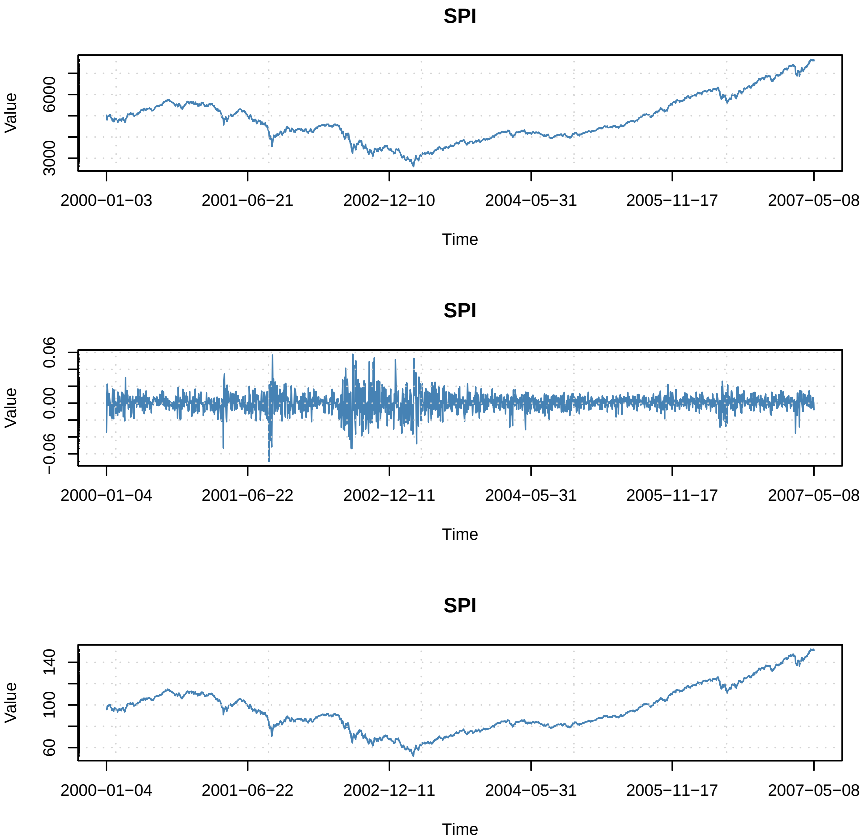


Figure 5.4 Plots of the SPI index and the returns: The three graphs show the index, the logarithmic returns, and the cumulated returns indexed to 100. The plot options used are the default options.

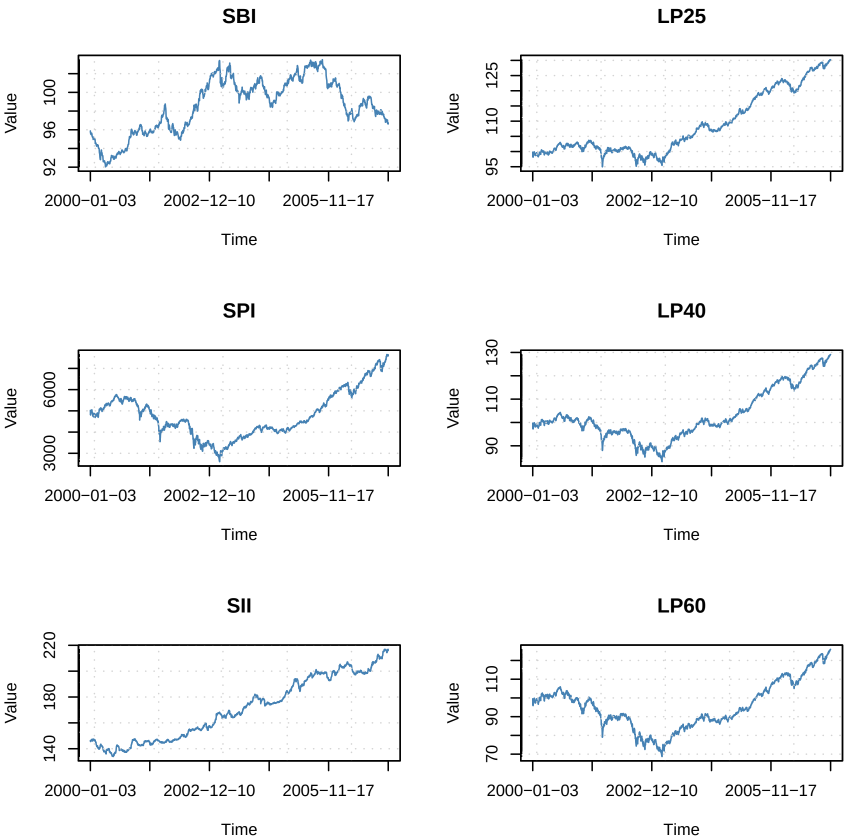


Figure 5.5 Plots of major Swiss indices and pension fund benchmark: The six graphs show to the left three SWX indices, the SBI, SPI and SII, as well as to the right the three Pictet Benchmark indices LLP25, LPP40 and LPP60 from Pictet's LPP2000 series.

```
> args(seriesPlot)

function (x, labels = TRUE, type = "l", col = "steelblue", title = TRUE,
    grid = TRUE, box = TRUE, rug = TRUE, ...)
NULL

> args(returnPlot)

function (x, labels = TRUE, type = "l", col = "steelblue", title = TRUE,
    grid = TRUE, box = TRUE, rug = TRUE, ...)
NULL

> args(cumulatedPlot)

function (x, index = 100, labels = TRUE, type = "l", col = "steelblue",
    title = TRUE, grid = TRUE, box = TRUE, rug = TRUE, ...)
NULL
```

allow you to adapt the plots according to your own requirements.

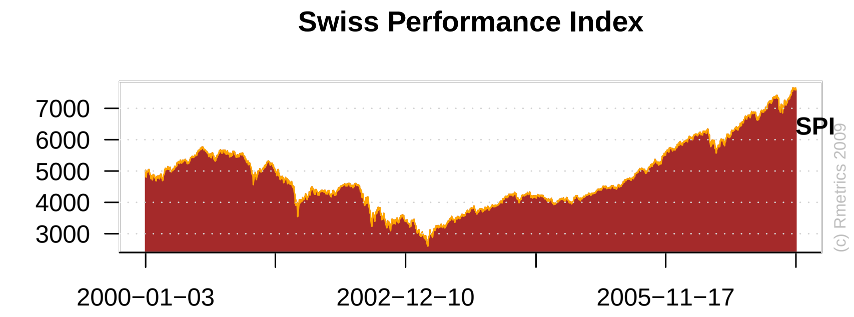
The following example shows a tailored graph:

```
> par(mfrow = c(1, 1))
> seriesPlot(SPI, labels = FALSE, type = "h", col = "brown",
     title = FALSE, grid = FALSE, rug = FALSE)
> lines(SPI, col = "orange")
> title(main = "Swiss Performance Index")
> hgrid()
> box_()
> copyright()
> mtext("SPI", side = 3, line = -2, adj = 1.02, font = 2)
```

In the `seriesPlot()` we suppress the `labels`, the `title`, the `grid`, and the `rug`, and change the `type` of the plot to histogram-like vertical lines. Finally, for `col` we choose a brown colour. Then we add an orange line on top of the plot. Then the main title is added calling the function `title()`. Horizontal grid lines are created by calling the Rmetrics function `hgrid()`, and a bottom lined box is created by calling the Rmetrics function `box_()`. Rmetrics also provides functions to easily add vertical grid lines, `vgrid()`, and L-shaped box frames, `boxL()`. The Rmetrics copyright is added by the function `copyright()`.

Derived Series Plots

For the future we plan to add several plots for derived series. Currently one such plot is available in Rmetrics for displaying the drawdowns of a



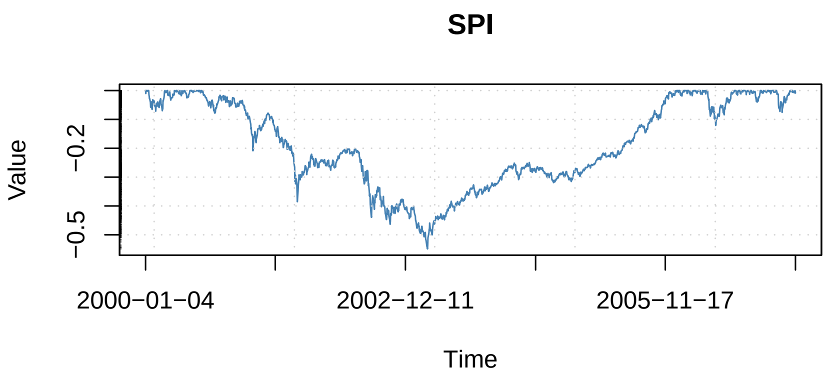


Figure 5.6 Tailored graphs for the SPI: The upper plot shows a tailored graph for the SPI, and the lower plot shows the drawdowns of the SPI returns.

financial time series. The function `drawdownsPlot()` takes as input a financial time series of returns and plots the drawdowns. For price or index series, we first have to compute the returns.

```
> drawdownPlot(returns(SPI, method = "discrete"))
```

User Generated Series

You can also plot any other derived series, or you can add your own plot functions using the function `seriesPlot()`. For example, let us smooth the SPI series using the `lowess()` function. This function performs the smoothing using locally-weighted polynomial regression.

```
> x <- SPI
> series(x) <- lowess(x = 1:nrow(SPI), y = as.vector(SPI),
      f = 0.1)$y
> seriesPlot(x, rug = FALSE, col = "red", ylim = c(2500, 8000),
      lwd = 2)
> lines(SPI)
```

Here is a recipe of how to create a user-generated plot function for the lowess smoother function:

```
lowessSeriesPlot <-
function (x, labels = TRUE, type = "l", col = "steelblue", title =
    TRUE, grid = TRUE, box = TRUE, rug = TRUE, add = TRUE, ...)
{
    stopifnot(isUnivariate(x))
    series(x) <- lowess(x = 1:nrow(x), y = as.vector(x), ...)$y
    seriesPlot(x = x, labels = labels, type = type, col = col,
        title = title, grid = grid, box = box, rug = rug, ...)
    invisible(x)
}
```

Listing 5.3 Example plot function for lowess smoother

And now, let us run it:

```
> lowessSeriesPlot(SPI, rug = FALSE, col = "red", f = 0.1)
```

Bear in mind that you can easily generalize your tailored series plot. For instance, you can include the multivariate case (inspect the function `series-`

`Plot()`) and, optionally, add the unsmoothed series. Another possible use case is a volatility plot.

How to tailor axis labelling

The graphs for the series and related plot functions use by default ISO8601 date/time formatted labels for the x-axis labelling. This can be modified by specifying the arguments `format` and `at` in the generic `plot()` function.

```
> plot(SPI, xlab = "", col = "steelblue")
> plot(SPI, format = "%b-%Y", xlab = "", col = "steelblue")
> plot(SPI, format = "%b-%Y", at = paste(2000:2007, 1, 1, sep = "-"),
     xlab = "", col = "steelblue")
```

How to display data from different time zones

The Rmetrics generic plot function can also display `timeSeries` objects recorded in different time zones in the same plot. For details we refer to the ebook *Chronological Objects with R/Rmetrics*.

5.2 Box Plots

Box plots are an excellent tool for conveying location and variation information in data sets, particularly for detecting and illustrating location and variation changes between different groups of data (Chambers, Cleveland, Kleiner & Tukey, 1983).

The R base package `graphics` provides the `boxplot()` function, which takes as input a numeric vector. Rmetrics has added the functions `boxPlot()` and `boxPercentilePlot()` for `timeSeries` objects of financial returns. These allow two different views on distributional data summaries. Both functions are built on top of R's `boxplot()` function.

```
Function:
boxPlot             creates a side-by-side standard box plot
boxPercentilePlot   creates a side-by-side box-percentile plot

Arguments:
```

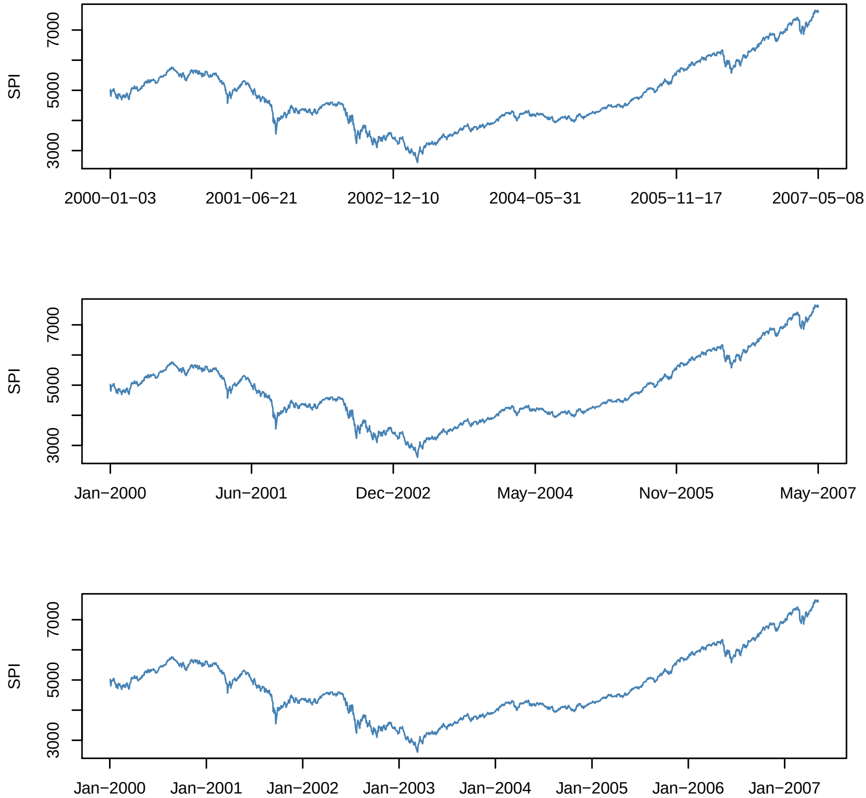


Figure 5.7 Plots with tailored axis labelling for the SPI: The upper plot shows the axis labelling using default settings. The plot in the middle shows "month-year" formatted axis labels. The lower plot labels the ticks from the beginning of the year.

```
x                    a 'timeSeries' object
col                  colours specified by a colour palette
```

Listing 5.4 Box and box percentile plot functions

How to display a box plot

Tukey (1977) introduced box plots as an efficient method for displaying a five-number data summary. The graph summarizes the following statistical measures: The median, upper and lower quartiles, and minimum and maximum data values. The box plot is interpreted as follows: The box itself contains the middle 50% of the data. The upper edge (hinge) of the box indicates the 75$^{\text{th}}$ percentile of the data set, and the lower hinge indicates the 25$^{\text{th}}$ percentile. The range of the middle two quartiles is known as the inter-quartile range. The line in the box indicates the median value of the data. If the median line within the box is not equidistant from the hinges, then the data is skewed. The ends of the vertical lines, the so called whiskers, indicate the minimum and maximum data values, unless outliers are present, in which case the whiskers extend to a maximum of 1.5 times the inter-quartile range. The points outside the ends of the whiskers are outliers or suspected outliers.

```
> args(boxPlot)

function (x, col = "steelblue", title = TRUE, ...)
NULL
```

The dot argument ... allows us to pass optional parameters to the underlying `boxplot()` function from the `graphics` package[1].

```
> args(boxplot)

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

```
> boxPlot(returns(SWX))
```

[1] `boxPlot()` is provided by `fBasics`, while `boxplot()` is from the `graphics` package

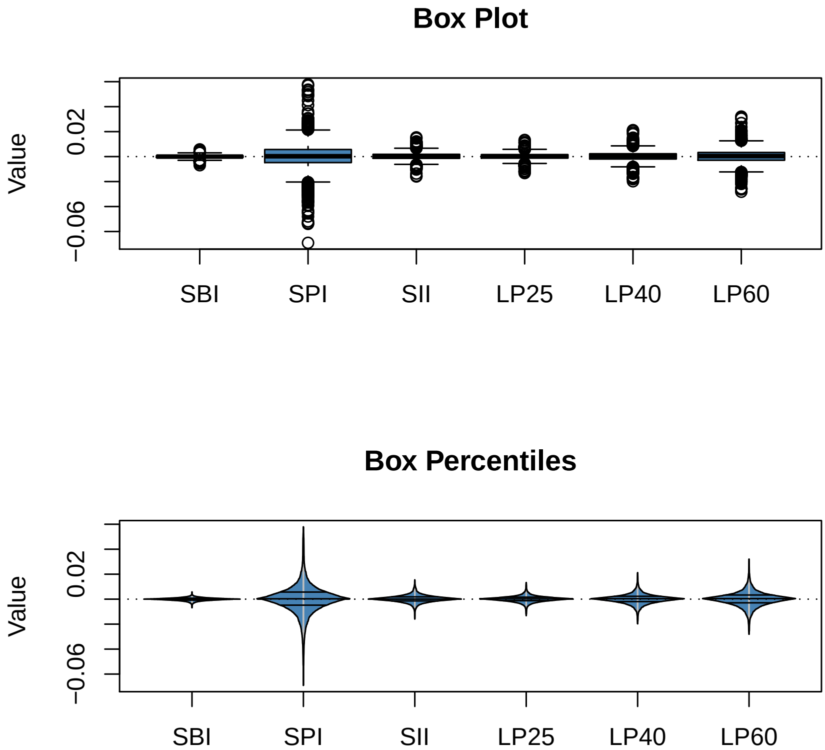


Figure 5.8 Box and box percentile plots of Swiss pension fund assets: The upper graph shows a box plot and the lower graph a box percentile plot. The presented data are the three Swiss assets classes SPI, SBI, SII, and Pictet's pension fund benchmark indices from the LPP2000 benchmark series.

How to display a box percentile plot

Unlike the box plot, which uses width only to emphasize the middle 50% of the data, the box-percentile plot uses width to encode information about the distribution of the data over the entire range of data values. Box-percentile plots convey the same graphical information as box plots. In addition, they also contain information about the shape of the distributions.

```
> args(boxPercentilePlot)

function (x, col = "steelblue", title = TRUE, ...)
NULL

> boxPercentilePlot(returns(SWX))
```

5.3 Histogram and Density Plots

To display a histogram or density plot for a univariate `timeSeries` object we can use R's base functions `hist()` and `density()`. In addition to these plots, Rmetrics offers three tailored plots, `histPlot()` `densityPlot()` and `logDensityPlot()`, which allow different views on density functions[2].

```
Function:
histPlot            returns a tailored histogram plot
densityPlot         returns a kernel density estimate plot
logDensityPlot      returns a log kernel density estimate plot
.hist               creates histograms with a fixed bin size

Arguments:
x                   a 'timeSeries' object
```

Listing 5.5 Histogram and density plot functions. Note that the internal Rmetricsfunction `.hist()` allows for developers to create histograms with controllable fixed bin sizes.

[2] `help()` and `density()` are from R's base package, `fooPlot()` and the internal utility function `.help()` are from Rmetrics.

How to display a histogram plot

The histogram is presumably the most pervasive of all graphical plots of financial returns. A histogram can be viewed as a graphical summary of distributional properties. On the other hand, we can consider it as a nonparametric estimator of a density function. The histogram is constructed by grouping the (return) data into equidistant bins or intervals and plotting the relative frequencies (or probabilities) falling in each interval.

The `histPlot()` plots a tailored histogram. By default, the probability is shown on the y-axis. Furthermore, the mean is added as an orange vertical line. For a comparison with a normal distribution with the same mean and variance as the empirical data, a brown normal density line is added. The rugs on the x-line provide further helpful information to observe the density in the tails.

```
> histPlot(SPI.RET)
```

How to display a density plot and kernel density estimates

The function `densityPlot()` computes a kernel density estimate by calling the `density()` function, and then displays it graphically. The algorithm used disperses the mass of the empirical distribution function over a regular grid of at least 512 points and then uses the fast Fourier transform to convolve this approximation with a discretized version of the kernel. It then uses linear approximation to evaluate the density at the specified points.

```
> args(densityPlot)
```

```
function (x, labels = TRUE, col = "steelblue", fit = TRUE, hist = TRUE,
    title = TRUE, grid = TRUE, rug = TRUE, skip = FALSE, ...)
NULL
```

The default plot adds a histogram, `hist=TRUE`, and overlays the density with a fitted normal distribution function, `fit=TRUE`.

```
> densityPlot(SPI.RET)
```

Optional dot arguments are passed to the `density()` function. This allows us to adapt the bandwidth, or to select an alternative smoothing kernel (the default kernel is Gaussian). For details we refer to the help page of the `density()` function.

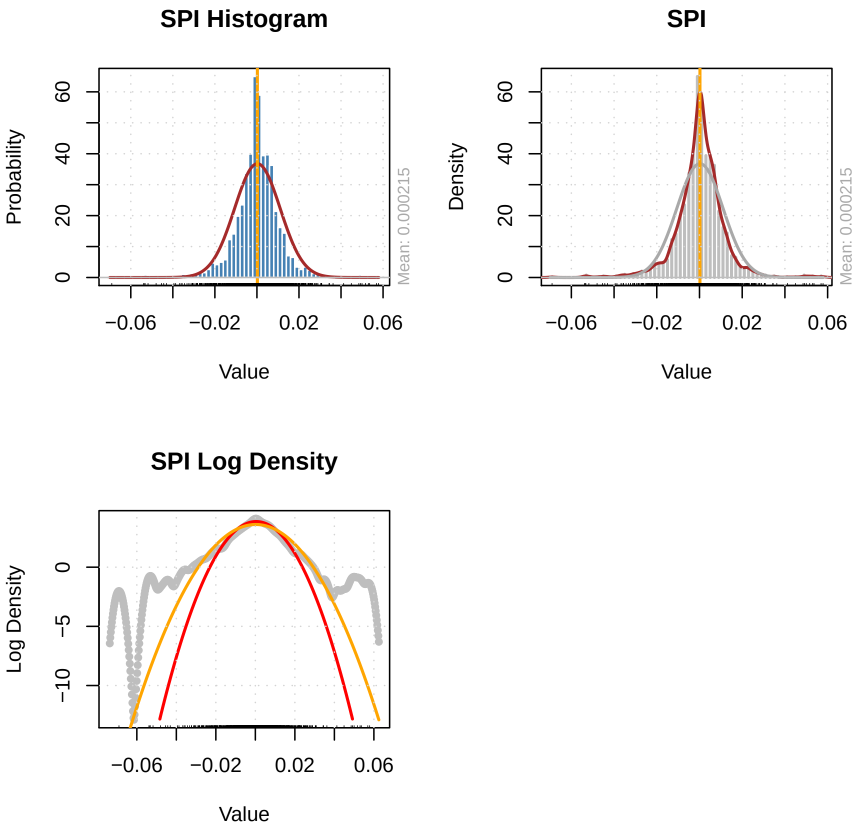


Figure 5.9 Histogram and density plots of Swiss pension fund assets: Upper Left: Histogram plot of the log returns of the Swiss Performance Index, SPI. The blue bins display the probability for the returns, the orange line the mean value, and the brown curve a normal density estimate with the same mean and variance as the empirical returns. Upper right: Kernel density estimate. Lower Left: Log Density Plot with a sample estimator and a robust estimate for the normal density fit.

How to display a log-density plot

The function `logDensityPlot()` creates a further view of the distributional properties of financial returns.

```
> args(logDensityPlot)

function (x, labels = TRUE, col = "steelblue", robust = TRUE,
    title = TRUE, grid = TRUE, rug = TRUE, skip = FALSE, ...)
NULL
```

The function displays the distribution on a logarithmic scale. Thus, in the case of normally distributed returns, we expect a parabolic shape, and heavy tails will be displayed as straight lines or are even bended upwards. The graph displays ...

```
> logDensityPlot(SPI.RET)
```

5.4 Quantile-Quantile Plots

The quantile-quantile plot, or qq-plot, is a graphical technique for determining if two data sets come from populations with a common distribution. A qq-plot is a plot of the quantiles of the first data set against the quantiles of the second data set. A 45-degree reference line is also plotted. If the two sets come from a population with the same distribution, the points should fall approximately along this reference line. The greater the departure from this reference line, the greater the evidence for the conclusion that the two data sets come from populations with different distributions.

To display a quantile-quantile plot for `timeSeries` objects we can use R's base functions `qqnorm()`, `qqline()`, and `qqplot()`. `qqnorm()` is a generic function, the default method of which produces a normal quantile-quantile plot. `qqline()` adds a line to a normal quantile-quantile plot which passes through the first and third quartiles. `qqplot()` produces a quantile-quantile plot of two data sets. In addition to these plots, Rmetrics offers three tailored plots to display distributional properties of financial returns fitted by a normal, a normal inverse Gaussian, and a generalized hyperbolic Student's t distribution. These distributions are heavily used in modelling financial returns[3].

```
Function:
qqnormPlot          returns a normal quantile-quantile plot
qqnigPlot           returns a NIG quantile-quantile plot
qqghtPlot           returns a GHT quantile-quantile plot

Arguments:
x                   a 'timeSeries' object
```

Listing 5.6 Quantile-quantile plot functions

A qq-plot helps to answer the following questions:

qqplotQ Do two data sets come from populations with a common distribution?

qqplotQ Do two data sets have common location and scale?

qqplotQ Do two data sets have similar distributional shapes?

qqplotQ Do two data sets have similar tail behaviour?

How to display a normal quantile-quantile plot

The normal quantile-quantile plot

```
> args(qqnormPlot)

function (x, labels = TRUE, col = "steelblue", pch = 19, title = TRUE,
    mtext = TRUE, grid = FALSE, rug = TRUE, scale = TRUE, ...)
NULL
```

displays the empirical data points versus the quantiles of a normal distribution function. By default, the empirical data are scaled by their mean and standard deviation. If a non-scaled view is desired we have to set the argument `scale=FALSE`. If the empirical data points are drawn from a normal distribution then we expect them to all lie on the diagonal line added to the plot. In addition, the plot shows the 95% confidence intervals.

[3] The first three functions are from R's base package, the `fooPlot()` functions are from Rmetrics.

```
> set.seed(1953)
> x <- rnorm(250)
> qqnormPlot(x)
```

How to display a NIG quantile-quantile plot

In the case of the normal inverse Gaussian distribution, NIG, the empirical data are fitted by a log-likelihood approach. The `qqnigPlot()` function returns an invisible list with the plot coordinates `$x` and `$y`.

```
> y <- rnig(250)
> qqnigPlot(y)
```

How to display a GHT quantile plot

In the case of the generalized hyperbolic Student's t distribution, GHT, the empirical data are fitted by a log-likelihood approach to the generalized hyperbolic Student's t distribution function. The function returns an invisible list with the plot coordinates `$x` and `$y`.

```
> z <- rght(250)
> qqghtPlot(z)
```

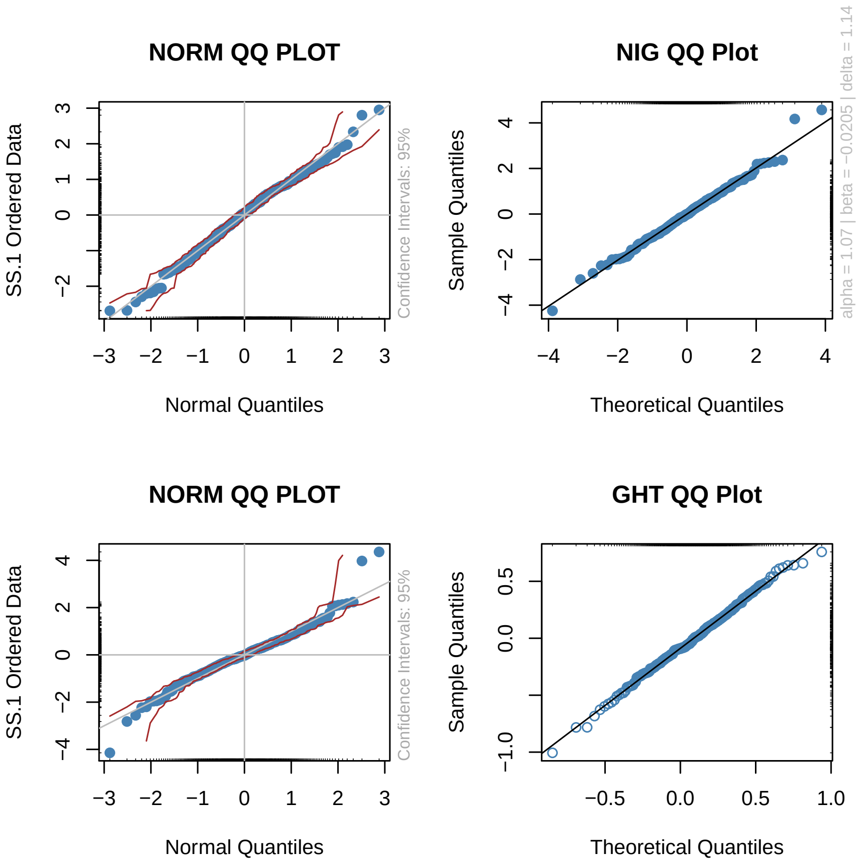


Figure 5.10 Quantile-Quantile Plots of simulated returns from a normal, a normal inverse Gaussian (NIG), and a generalized hyperbolic Student's t (GHT) distribution.

Chapter 6

Customization of Plots

Required R package(s):

```
> library(fPortfolio)
```

Rmetrics comes with several kinds of customized plots to display financial time series and their statistical properties. These plots can be adapted in many ways. The layout of the plot labels, including titles, labels and additional text information, can be modified by changing the content, the types of the fonts and the size of characters. Plot elements, such as lines and symbols, can be modified by changing their style, size and colours.

In the following we give a brief overview of how to customize plot labels, and how to select colours, fonts and plot symbols.

6.1 Plot Labels

Most of the Rmetrics tailored plots, such as `seriesPlot()`, have common arguments to customize their layout.

```
> args(seriesPlot)

function (x, labels = TRUE, type = "l", col = "steelblue", title = TRUE,
    grid = TRUE, box = TRUE, rug = TRUE, ...)
NULL
```

The main arguments for customization a plot are summarized in the following function listing.

```
Function:
plot                generic plot function
points              adds points to a plot
lines               adds connected line segments to a plot
abline              adds straight lines through a plot

Arguments:
type                determines the type of plot
col                 colour or colour palette for lines or symbols
title               should a default title be added?
grid                should a grid be added to the plot?
box                 should a box be added to the plot?
rug                 should rugs be added?
...                 optional arguments to be passed
```

Listing 6.1 Main arguments for plot, points and lines functions

For details we refer to the help functions for the `plot()` and `par()` functions. In the following we present some examples of how to customize a univariate time series plot.

How to create a series plot with default labels

The first graph shows a time series plot for the Swiss performance index created with default settings

```
> SPI <- SWX[, "SPI"]
> seriesPlot(SPI)
```

How to create a series plot with user-specified labels

The second graph shows the same plot but now with user-specified labels. Setting the argument `title=FALSE`

```
> seriesPlot(SPI, title = FALSE)
> title(main = "Swiss Performance Index", xlab = "", ylab = "SPI Index")
> text(as.POSIXct("2006-11-25"), rev(SPI)[1], as.character(rev(SPI)[1]),
     font = 2)
> mtext("Source: SWX", side = 4, col = "grey", adj = 0, cex = 0.7)
```

displays an untitled plot. Thus we can use the R base function `title()` to add a main title, subtitle, as well as x and y labels. Further text attributes can be added using R's base functions `text()` and `mtext()`.

For details please consult the help functions.

```
Function:
title                adds a title, a subtitle, and axis labels
text                 adds text string(s) to the plot
mtext                adds margin text string(s) to the plot
```

Listing 6.2 Title, text and margin text functions

How to create a series plot with decorations

The third graph shows how to decorate the plot. By setting `rug = FALSE`, we remove the rug. Next, we add a horizontal grid and replace the framed box with an L-shaped box. Finally, we add a copyright string as margin text, and add an orange horizontal line on index level 5000.

```
> seriesPlot(SPI, grid = FALSE, box = FALSE, rug = FALSE)
> hgrid()
> boxL()
> copyright()
> abline(h = 5000, col = "orange")
```

For details of decoration functions we refer to the help pages of the following functions:

```
Function:
decor                simple decoration function
hgrid                creates horizontal grid lines
vgrid                creates vertical grid lines
boxL                 creates a L-shaped box
box_                 creates a bottom line box
copyright            adds Rmetrics copyright to a plot
```

Listing 6.3 Plot decoration functions

How to create a series plot with optional dot arguments

The fourth graph demonstrates how to use optional plot parameters through the dot ... arguments. Here we have modified the plotting point symbol and changed the orientation of the axis label style. Further arguments are shown in Listing 6.4. For a complete list of all plot parameters we refer to `help(par)`.

```
> seriesPlot(SPI, grid = FALSE, rug = FALSE, type = "o", pch = 19,
     las = 1)
```

6.2 More About Plot Function Arguments

Here are some of the arguments you might want to specify for plots:

```
Function:
plot                generic plot function

Arguments:
type                what type of plot should be created?
axes                draw or suppress to plot the axes
ann                 draw or suppress to add title and axis labels
pch                 select the type of plotting symbol
cex                 select the size of plotting symbol and text
xlab, ylab          names of the labels for the x and y axes
main                the (main) title of the plot
xlim, ylim          the range of the x and y axes
log                 names of the axes which are to be logarithmic
col, bg             select colour of lines, symbols, background
lty, lwd            select line type, line width
las                 select orientation of the text of axis labels
```

Listing 6.4 Selected arguments for plot functions

Notice that some of the relevant parameters are documented in `help(plot)` or `plot.default()`, but many only in `help(par)`. The function `par()` is for setting or querying the values of graphical parameters in traditional R graphics.

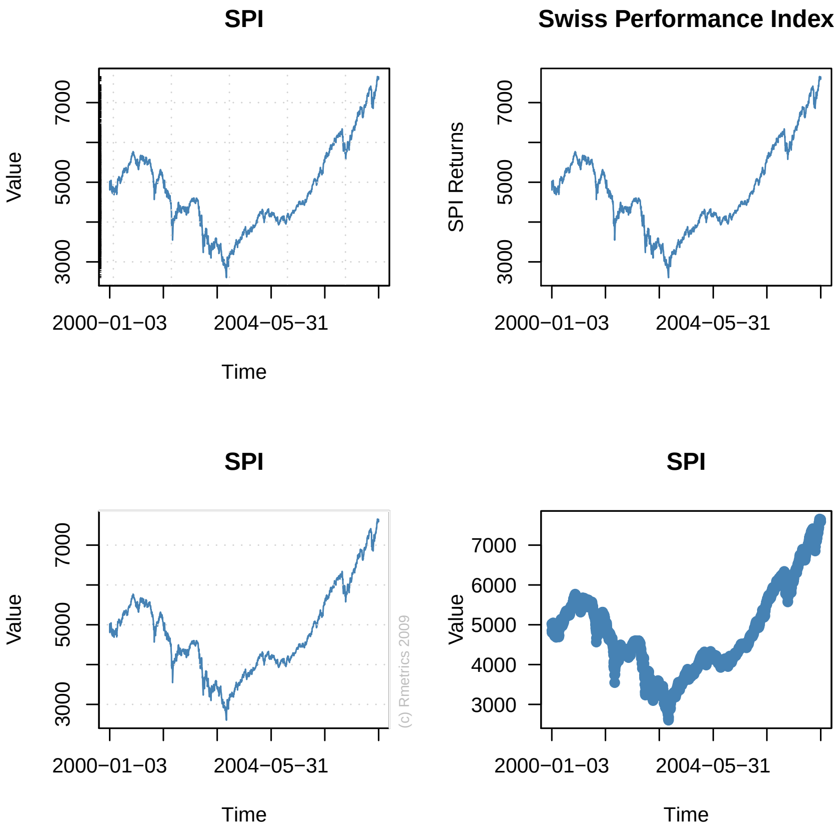


Figure 6.1 Customized time series plots: The first graph shows the default plot, the second a user entitled and labelled plot, the third a user decorated plot, and the fourth plot modifications by adding optional parameters through the dot argument.

How to modify the plot type

Settings for the plot `type` can be modified using the following identifiers:

```
Function:
plot                generic plot function

Argument:
type                specifies the type of plot
                    "p"        point plot (default)
                    "l"        line plot
                    "b"        both points and lines
                    "o"        overplotted points and lines
                    "h"        histogram like
                    "s"        steps
                    "n"        no plotting
```

Listing 6.5 Type argument specifications for plot functions

Note that by default, the `type` argument is set to `"p"`. If you want to draw the axes first and add points, lines and other graphical elements later, you should use `type="n"`.

How to select a font

With the `font` argument, an integer in the range from 1 to 5, we can select the type of fonts:

```
Function:
plot                generic plot function

Arguments:
font                integer specifying which font to use for text
font.axis           font number to be used for axis annotation
font.lab            font number to be used for x and y labels
font.main           font number to be used for plot main titles
font.sub            font number to be used for plot sub-titles
```

Listing 6.6 Font arguments for plot functions

If possible, device drivers arrange so that 1 corresponds to plain text (the default), 2 to bold face, 3 to italic and 4 to bold italic. Also, font 5 is expected to be the symbol font, in Adobe symbol encoding.

How to modify the size of fonts

With the argument `cex`, a numeric value which represents a multiplier, we can modify the size of fonts

```
Function:
plot                generic plot function

Arguments:
cex                 magnification of fonts/symbols relative to default
cex.axis            magnification for axis annotation relative to cex
cex.lab             magnification for x and y labels relative to cex
cex.main            magnification for main titles relative to cex
cex.sub             magnification for sub-titles relative to cex
```

Listing 6.7 cex arguments for plot functions

How to orient axis labels

The argument `las`, an integer value ranging from 0 to 3, allows us to determine the orientation of the axis labels

```
Function:
plot                generic plot function

Arguments:
las                 orientation
  0                   always parallel to the axis [default]
  1                   always horizontal
  2                   always perpendicular to the axis
  3                   always vertical
```

Listing 6.8 cex parameters for plot functions

Note that other string/character rotation (via argument `srt` to `par`) does not affect the axis labels.

How to select the line type

The argument `lty` sets the line type. Line types can either be specified as an integer, or as one of the character strings `"blank"`, `"solid"`, `"dashed"`, `"dotted"`, `"dotdash"`, `"longdash"`, or `"twodash"`, where `"blank"` uses invisible lines, i.e. does not draw them.

```
Function:
plot                generic plot function

Arguments:
lty                 sets line type to
  0                   blank
  1                   solid (default)
  2                   dashed
  3                   dotted
  4                   dotdash
  5                   longdash
  6                   twodash
```

Listing 6.9 lty argument for plot functions

6.3 Selecting Colours

Rmetrics provides tools and utilities to select individual colours by code numbers and sets of colours from colour palettes.

How to print the colour coding numbers

The function `colorTable()` displays a table of R's base colours together with their code numbers.

```
> colorTable()
```

Note that the colours are repeated cyclically.

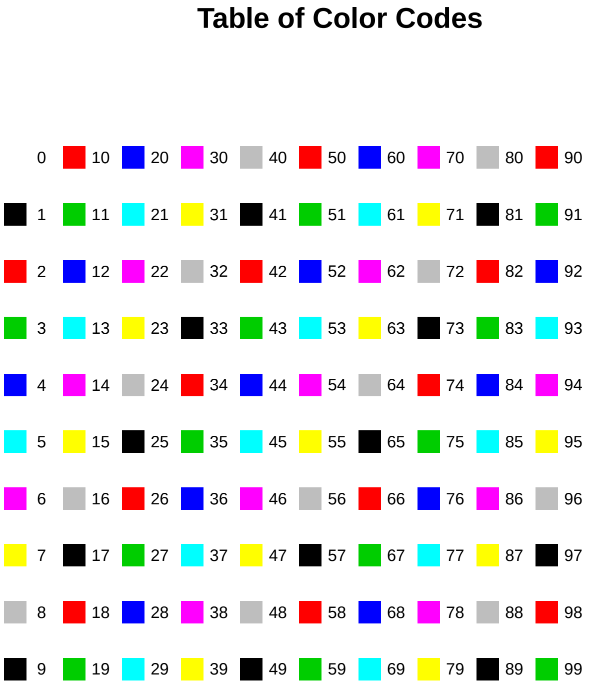


Figure 6.2 Colour table of R's base colours: The colours are shown together with their code numbers. Note that number 1 is white (invisible on white background), number 2 is black, and the next colours are red, green, blue, cyan, magenta, yellow, grey, then the cycle repeats with number 9 being black again.

How to use the colour name locator

The function `colorLocator()` displays R's 657 named colours for selection and optionally returns R's colour names. The idea and implementation of the colour locator originates in the contributed package `epitools` written and maintained by Aragon (2008).

Usually this function is used interactively, setting the argument to `TRUE`. Use the left mouse button to locate one or more colours and the stop the locator, using the method appropriate to the screen device you are using. Here is what you get on the console:

```
> colorLocator(TRUE)
```

```
   x  y     colour.names
1 15 15     lightslateblue
2 18 17     orange
3 27 22     yellowgreen
```

To return all colour names in alphabetical order you can call the function `colorMatrix()`. The first 20 are:

```
> head(sort(colorMatrix()), 20)
```

```
 [1] "aliceblue"     "antiquewhite"  "antiquewhite1" "antiquewhite2"
 [5] "antiquewhite3" "antiquewhite4" "aquamarine"    "aquamarine1"
 [9] "aquamarine2"   "aquamarine3"   "aquamarine4"   "azure"
[13] "azure1"        "azure2"        "azure3"        "azure4"
[17] "beige"         "bisque"        "bisque1"       "bisque2"
```

The associated colour locator display is shown in Figure 6.3.

Which colour palettes are available

To display a multivariate plot we would often like to use a personal set of colours. To this end, Rmetrics provides dozens of preimplemented colour palettes taken from several contributed R packages. The functions and their arguments have been modified, so that we have a common usage for all palettes. Here is a list of the palettes provided with Rmetrics:

```
Function:
rainbowPalette      Contiguous rainbow colour palette
```

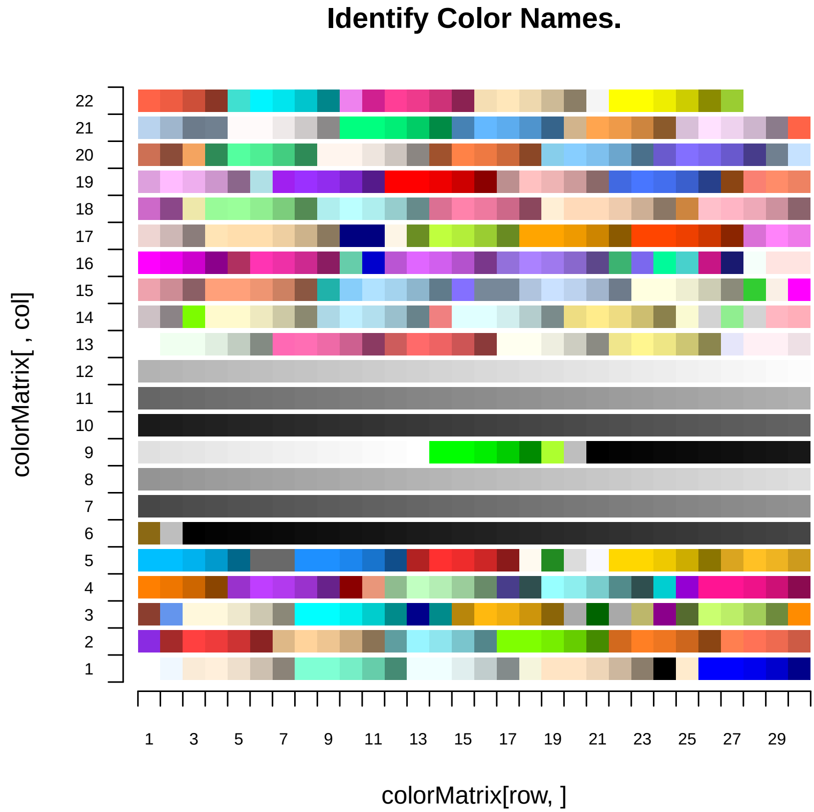


Figure 6.3 Colour locator with R's 657 named colours: The display shows the colour matrix. If you use the colour locator interactively, clicking on a colour square will return the name of the colour in the console window.

```
heatPalette        Contiguous heat colour palette
terrainPalette     Contiguous terrain colour palette
topoPalette        Contiguous topo colour palette
cmPalette          Contiguous cm colour palette
greyPalette        R's gamma-corrected gray palette
timPalette         Tim's MATLAB-like colour palette
rampPalette        Colour ramp palettes
seqPalette         Sequential colour brewer palettes
divPalette         Diverging colour brewer palettes
qualiPalette       Qualified colour brewer palettes
focusPalette       Red, green and blue focus palettes
monoPalette        Red, green and blue mono palettes
```

Listing 6.10 Colour palette functions

All Rmetrics' colour sets are named as `fooPalette`, where the prefix `foo` denotes the name of the underlying colour set.

R's Contiguous Colour Palettes

Palettes for n contiguous colours are implemented in the `grDevices` package. To conform with Rmetrics' naming convention for colour palettes, we have built a wrapper around the underlying functions. These are the `rainbowPalette()`, `heatPalette()`, `terrainPalette()`, `topoPalette()`, and the `cmPalette()`. Conceptually, all of these functions actually use (parts of) a line cut out of the 3-dimensional colour space, parametrized by the function `hsv(h,s,v,gamma)`, where `gamma=1` for the `fooPalette()` function, and hence, equispaced hues in RGB space tend to cluster at the red, green and blue primaries. Some applications, such as contouring, require a palette of colours which do not wrap around to give a final colour close to the starting one. If you want to pass additional arguments to the underlying functions, please consult `help(rainbow)`. With `rainbow`, the parameters `start` and `end` can be used to specify particular subranges of hues. Synonymous function calls are `rainbow()`, `heat.colors()`, `terrain.colors()`, `topo.colors()`, and `cm.colors()`.

```
> pie(rep(1, 12), col = rainbowPalette(12), xlab = "rainbowPalette")
> pie(rep(1, 12), col = heatPalette(12), xlab = "heatPalette")
> pie(rep(1, 12), col = terrainPalette(12), xlab = "terrainPalette")
```

```
> pie(rep(1, 12), col = topoPalette(12), xlab = "topoPalette")
> pie(rep(1, 12), col = cmPalette(12), xlab = "cmPalette")
```

R's Gamma-Corrected Gray Palette

The function `grayPalette()` chooses a series of n gamma-corrected grey levels. The range of the grey levels can be optionally monitored through the `...` arguments, for details we refer to `help(gray.colors)`, which is a synonymous function call used in the `grDevices` package.

```
> pie(rep(1, 12), col = greyPalette(12), xlab = "greyPalette")
```

Tim's MATLAB*-Like Colour Palette*

The function `timPalette()` creates a colour set ranging from blue to red, and passes through the colours cyan, yellow, and orange. It is an implementation of the MATLAB `jet` colour palette, originally used in fluid dynamics simulations. The function here is a copy from R's contributed package `fields`, doing a spline interpolation on n=64 colour points.

```
> pie(rep(1, 12), col = timPalette(12), xlab = "timPalette")
```

Colour Ramp Palettes

The function `rampPalette()` creates several colour ramps. The function is implemented in the contributed R package `colorRamps` (Keitt, 2007). The following colour ramp palettes are supported through the `name` argument: `"blue2red"`, `"green2red"`, `"blue2green"`, `"purple2green"`, `"blue2yellow"`, and `"cyan2magenta"`.

```
> palettes <- c("blue2red", "green2red", "blue2green", "purple2green",
      "blue2yellow", "cyan2magenta")
> for (palette in palettes) pie(rep(1, 12), col = rampPalette(12,
      palette), xlab = paste("rampPalette -", palette))
```

Colour Brewer Palettes

The functions `seqPalette()`, `divPalette()`, and `qualiPalette()` create colour sets according to R's contributed `RColorBrewer` package. The first letter in the function name denotes the type of the colour set: `"s"` for sequential palettes, `"d"` for diverging palettes, and `"q"` for qualitative palettes.

Sequential palettes are suited to ordered data that progress from low to high. Lightness steps dominate the look of these schemes, with light colours for low data values to dark colours for high data values. The sequential palettes names are: Blues, BuGn, BuPu, GnBu, Greens, Greys, Oranges, OrRd, PuBu, PuBuGn, PuRd, Purples, RdPu, Reds, YlGn, YlGnBu, YlOrBr, YlOrRd.

```
> palettes <- c("Blues", "BuGn", "BuPu", "GnBu", "Greens",
     "Greys", "Oranges", "OrRd", "PuBu", "PuBuGn", "PuRd",
     "Purples", "RdPu", "Reds", "YlGn", "YlGnBu", "YlOrBr",
     "YlOrRd")
> for (palette in palettes) pie(rep(1, 12), col = seqPalette(12,
     palette), xlab = paste("seq -", palette))
```

Diverging palettes put equal emphasis on mid-range critical values and extremes at both ends of the data range. The critical class or break in the middle of the legend is emphasized with light colours and low and high extremes are emphasized with dark colours that have contrasting hues. The diverging palettes names are: `"BrBG"`, `"PiYG"`, `"PRGn"`, `"PuOr"`, `"RdBu"`, `"RdGy"`, `"RdYlBu"`, `"RdYlGn"`, `"Spectral"`.

```
> palettes <- c("BrBG", "PiYG", "PRGn", "PuOr", "RdBu", "RdGy",
     "RdYlBu", "RdYlGn", "Spectral")
> for (palette in palettes) pie(rep(1, 12), col = divPalette(12,
     palette), names = NULL, xlab = paste("div -", palette))
```

Qualitative palettes do not imply magnitude differences between legend classes, and hues are used to create the primary visual differences between classes. Qualitative schemes are best suited to representing nominal or categorical data. The qualitative palettes names are: `"Accent"`, `"Dark2"`, `"Paired"`, `"Pastel1"`, `"Pastel2"`, `"Set1"`, `"Set2"`, `"Set3"`.

```
> palettes <- c("Accent", "Dark2", "Paired", "Pastel1", "Pastel2",
     "Set1", "Set2", "Set3")
> for (palette in palettes) pie(rep(1, 12), col = qualiPalette(12,
     palette), xlab = paste("quali -", palette))
```

In contrast to the original colour brewer palettes, the palettes here are created by spline interpolation from the colour variation with the most different values, i.e for the sequential palettes these are 9 values, for the diverging palettes these are 11 values, and for the qualitative palettes these are between 8 and 12 values, depending on the colour set.

The brewer colour palettes are originally from the contributed R package `RColorBrewer`, written by Neuwirth (2007).

Graph Colour Palettes

The functions `focusPalette()` and `monoPalette()` create colour sets inspired by R's contributed package `PerformanceAnalytics` (Carl & Peterson, 2008). These colour palettes have been designed to create readable, comparable line and bar graphs with specific objectives.

Focused Colour Palettes: Colour sets designed to provide focus on the data graphed as the first element. This palette is best used when there is clearly an important data set for the viewer to focus on, with the remaining data being secondary, tertiary, etc. Later elements graphed in diminishing values of grey. The focus palette names are: `"redfocus"`, `"greenfocus"`, `"bluefocus"`.

Monochrome Colour Palettes: These include colour sets for monochrome colour displays. The mono palette names are: `"redmono"`, `"greenmono"`, `"bluemono"`.

Inspect the functions for the colour palettes and feel free to add your own palettes. For the developer we would like to mention the following undocumented functions:

```
Function:
.asRGB              converts any R colour to RGB (red/green/blue)
.chcode             changes from one to another number system
.hex.to.dec         converts heximal numbers do decimal numbers
.dec.to.hex         converts decimal numbers do heximal numbers
```

Listing 6.11 Undocumented colour functions

6.4 Selecting Character Fonts

The function `characterTable()` displays the character for a given font. The font is specified by an integer number ranging from 1 to 5. This integer

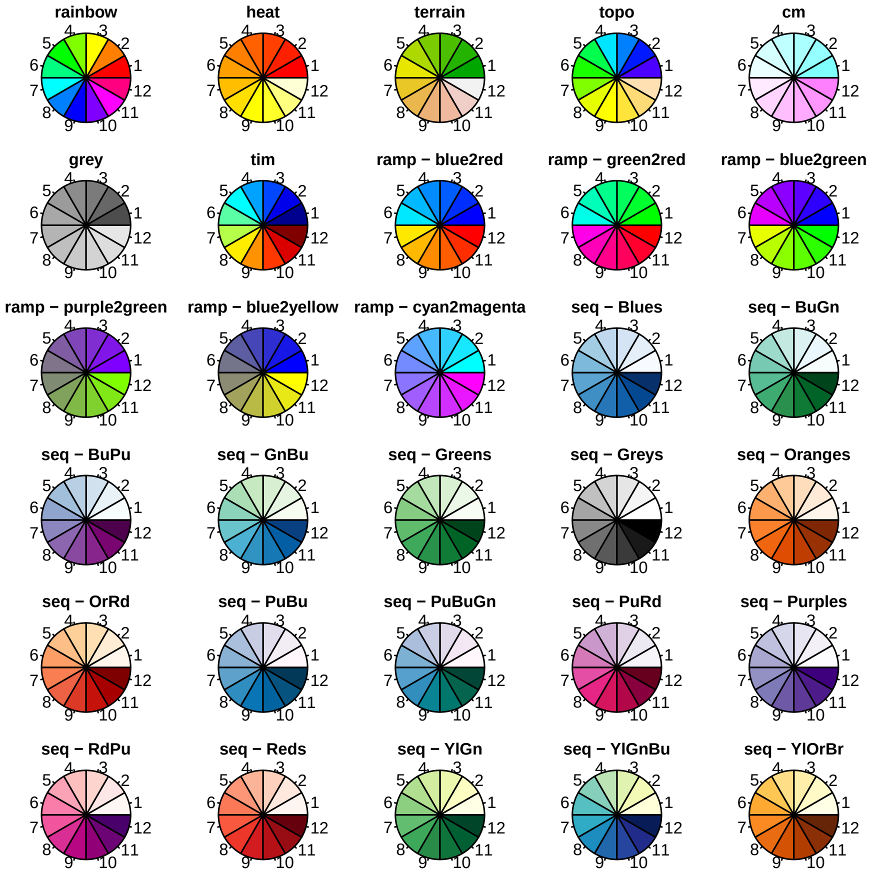


Figure 6.4 Selected colour palettes: The colour palettes provided by Rmetrics include R's base palettes, a grey palette, the MATLAB-like colour palette, colour ramp palettes, sequential colour brewer palettes, diverging colour brewer palettes, qualified colour brewer palettes, red/green/blue focus palettes, and red/green/blue mono palettes.

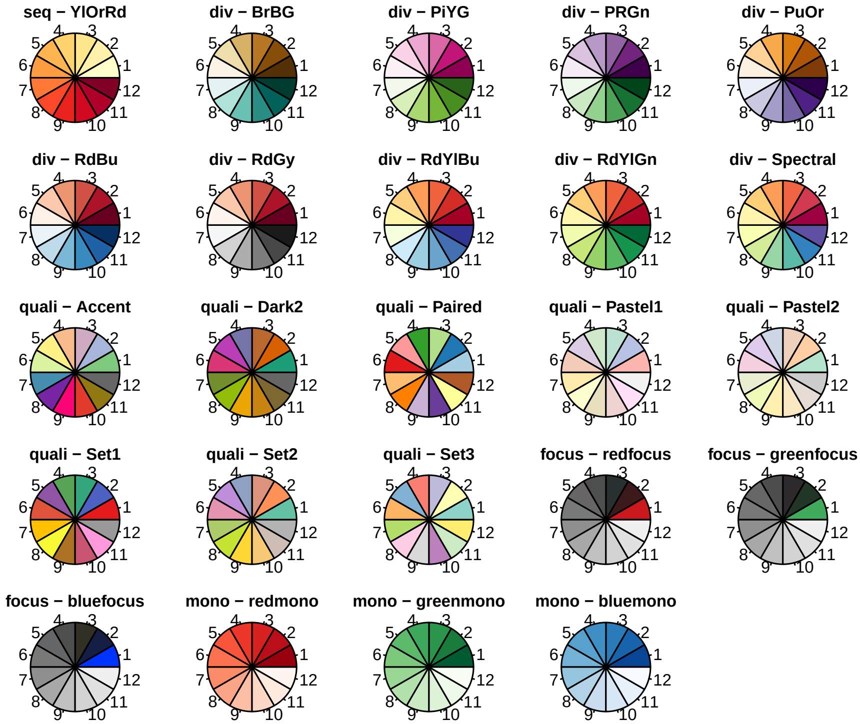


Figure 6.5 Selected colour palettes, continued: The remaining colour palettes from the previous figure.

specifies which font to use for text. If possible, device drivers arrange the fonts in the following sequence:

```
Function:
characterTable      displays a table of characters

Arguments:
font                specifies font number
  1                   plain text (the default)
  2                   bold face
  3                   italic
  4                   bold italic
  5                   symbol font in Adobe symbol encoding
```

Listing 6.12 Function to display characters for a given font

To display a specific font in a graphics display we can use the command

```
> characterTable(font = 5)
```

6.5 Selecting Plot Symbols

Plot symbols are set within the `plot()` function by setting the `pch` parameter, equal to an integer between 0 and usually 25. Since it is hard to remember what symbol each integer represents, Figure 6.7 may serve as a reminder. The function `symbolTable()` displays the plot symbol for a given code.

```
> # The following example use latin1 characters: these may not
> # appear correctly (or be omitted entirely).
> symbolTable()
```

Plot symbols can be referenced in the following way:

```
> print("info\100rmetrics.ch")

[1] "info@rmetrics.ch"
```

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	≅	Α	Β	Χ	Δ	Ε	Φ	Γ
11	Η	Ι	ϑ	Κ	Λ	Μ	Ν	Ο
12	Π	Θ	Ρ	Σ	Τ	Υ	ς	Ω
13	Ξ	Ψ	Ζ	[	∴	]	⊥	_
14	‾	α	β	χ	δ	ε	ϕ	γ
15	η	ι	φ	κ	λ	μ	ν	ο
16	π	θ	ρ	σ	τ	υ	ϖ	ω
17	ξ	ψ	ζ	{	|	}	~	
20								
21								
22								
23								
24	€	ϒ	′	≤	⁄	∞	ƒ	♣
25	♦	♥	♠	↔	←	↑	→	↓
26	°	±	″	≥	×	∝	∂	•
27	÷	≠	≡	≈	…	⏐	⎯	↵
30	ℵ	ℑ	ℜ	℘	⊗	⊕	∅	∩
31	∪	⊃	⊇	⊄	⊂	⊆	∈	∉
32	∠	∇	®	©	™	∏	√	⋅
33	¬	∧	∨	⇔	⇐	⇑	⇒	⇓
34	◊	〈	®	©	™	∑	⎛	⎜
35	⎝	⎡	⎢	⎣	⎧	⎨	⎩	⎪
36		〉	∫	⌠	⎮	⌡	⎞	⎟
37	⎠	⎤	⎥	⎦	⎫	⎬	⎭	

Figure 6.6 Character font tables: This table shows the characters for font number 5.

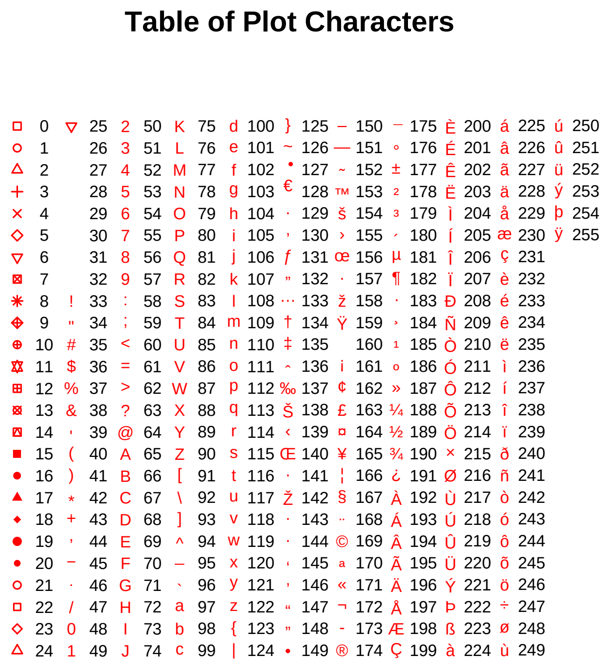


Figure 6.7 Table of plot symbols: Displayed are the plot symbols for the current font.

Here, the code symbol `100` prints the `@` sign, to print the © symbol we use code symbol `251`.

Chapter 7

Modelling Asset Returns

Required R package(s):

```
> library(fPortfolio)
```

In many cases we want to generate artificial data sets of assets which have the same statistical properties as a given set of empirical returns. In the simple case of the multivariate normal and Student's t as well as their skewed versions Rmetrics provides functions to fit the parameters for this family of elliptical distributions and to generate new random data sets from these parameters. To find out if a set of empirical financial asset returns is multivariate normally distributed we can perform an hypothesis test.[1]

7.1 Testing Asset returns for Normality

The function `assetsTest()` is a suite of (currently two) functions to test whether or not a set of asset returns is (multivariate) normally distributed.

The implemented tests are the *multivariate Shapiro test* (Royston, 1982) from the R package `mvnormtest` contributed by Jarek (2009), and the *nonparametric E-statistics test*, also called energy test (Szekely, 1989; Rizzo, 2002; Szekely &

[1] An alternative way to model multivariate assets sets and their dependency structure uses copulae. Rmetrics functions for testing, fitting, and simulating copulae are described in the ebook *Managing Risk with R/Rmetrics*.

Rizzo, 2005; Szekely, Rizzo & Bakirov, 2007) from the contributed R package energy (Rizzo & Szekely, 2008)[2].

```
Function:
assetsTest          tests for multivariate normal assets
assetsFit           estimates the parameters of a set of assets
assetsSim           simulates artificial data sets of assets
```

Listing 7.1 Functions to test a multivariate data set of returns for normality, to fit the model parameters, and to simulate artificial data sets with the same statistical properties as the empirical data set

```
> args(assetsTest)

function (x, method = c("shapiro", "energy"), Replicates = 100,
    title = NULL, description = NULL)
NULL
```

The function `assetsTest()` requires a multivariate `timeSeries` object `x` as input and performs the test specified by the `method` argument, by default the multivariate Shapiro test. An object of S4 class `fHTEST` is returned.

Let us now investigate whether the Swiss bond returns, Swiss equities, and Swiss Reits in the LPP2005 data are normally distributed or not, and let us compare the results of the two methods, `shapiro` and `energy`.

How to perform a multivariate Shapiro test

To perform a multivariate Shapiro test, we call the function `assetsTest()` with `method="shapiro"`. This is the default setting, the specification of the argument `method` is therefore optional.

```
> shapiroTest <- assetsTest(LPP2005.RET[, 1:3], method = "shapiro")
> print(shapiroTest)

Title:
 Multivariate Shapiro Test

Test Results:
```

[2] These are implemented as built-in functions in Rmetrics

```
  STATISTIC:
    W: 0.9521
  P VALUE:
    1.018e-09

Description:
 Tue May  5 11:52:50 2009 by user: Rmetrics
```

The function returns an object of class fHTEST, with the following slots

```
> slotNames(shapiroTest)

[1] "call"        "data"        "test"        "title"       "description"
```

The slot named `@test` of the result returned by the Shapiro test returns a list with all entries from the original test as implemented in in the R package `mvnormtest`. The printout from the Shapiro test tells us that the hypothesis of a multivariate normal return distribution is rejected.

How to perform a multivariate E-Statistics test

Alternatively, we can perform a multivariate E-Statistics Test. To do this, we have to set the argument `method="energy"` explicitly.

```
> assetsTest(LPP2005.RET[, 1:3], method = "energy")

Title:
 Energy Test

Test Results:
  STATISTIC:
    E-Statistic: 3.0382
  P VALUE:
    < 2.2e-16

Description:
 Tue May  5 11:52:50 2009 by user: Rmetrics
```

Again, the slot named `@test` of the result returned by the E-statistics test gives a list with all entries from the original test as implemented in the R package `energy` (Rizzo & Szekely, 2008). The energy test also rejects the hypothesis that the returns are multivariate normally distributed.

7.2 Fitting Asset returns

Rmetrics also provides functions to fit a data set of asset returns to the most common multivariate distribution functions for financial returns, the normal and the Student's t distributions.

```
> args(assetsFit)

function (x, method = c("st", "snorm", "norm"), title = NULL,
    description = NULL, fixed.df = NA, ...)
NULL
```

The function `assetsFit()` expects a multivariate `timeSeries` object of asset returns `x` as input and estimates the parameters of the specified distribution by the argument `method`. The choice can be a multivariate normal distribution, `method="norm"`, a multivariate skew-normal distribution, `"snorm"`, or a multivariate skew-Student's t distribution, `"st"`. For the latter, the number of degrees of freedom (df) can be held fixed during the parameter estimation.

How to fit a normal or Student's t distribution

The most common multivariate distribution functions for financial returns include the *normal distribution* and the *Student's t distribution* together with their skewed versions. The `method` argument determines which distribution should be fitted to the asset returns, by default the skewed Student's t distribution. The following example shows how to fit a skew Student's t distribution to the set of Swiss asset returns including the SPI, SBI, and SII.

```
> fit <- assetsFit(LPP2005.RET[, 1:3], method = "st")
> print(fit)

Title:
Fitted Asset Data Model:  st

Call:
assetsFit(x = LPP2005.RET[, 1:3], method = "st")

Model Parameters:
$mu
        SBI         SPI         SII
 9.1424e-05  2.4140e-03 -2.0953e-04
```

```
$Omega
            SBI         SPI        SII
SBI  1.2362e-06 -8.2959e-07 1.2687e-07
SPI -8.2959e-07  3.9629e-05 1.6154e-06
SII  1.2687e-07  1.6154e-06 6.1579e-06

$alpha
     SBI      SPI      SII
-0.14447 -0.32575  0.25231

$df
[1] 6.5043

Description:
[1] "Tue May  5 11:52:53 2009 by user: Rmetrics"
```

```
> plot(fit, 1)
```

The multivariate skew-normal distribution is implemented as discussed by Azzalini & Dalla Valle (1996); the (Omega, alpha) parametrization is the one used by Azzalini & Capitanio (1999).

The family of multivariate skew-t distributions is an extension of the multivariate Student's t family, via the introduction of a shape parameter which regulates skewness. In the symmetric case the skew-t distribution reduces to the regular symmetric t-distribution. When the number of degrees becomes infinity the distribution reduces to the multivariate skew-normal one (Azzalini & Capitanio, 2003).

The fits are done using maximum likelihood estimation (Azzalini & Capitanio, 1999, 2003). The function `assetsFit` returns an object of S4 class `fASSETS`. The `@model` slot provides the estimated parameters.

How to fit distributions using the copula approach

Rmetrics also offers functions to analyze, to model and to fit distribution functions using the copula approach. The relevant functions are available in the Rmetrics package `fCopulae` (Würtz, 2009b). For details we refer to the ebook *Managing Risk with R/Rmetrics*.

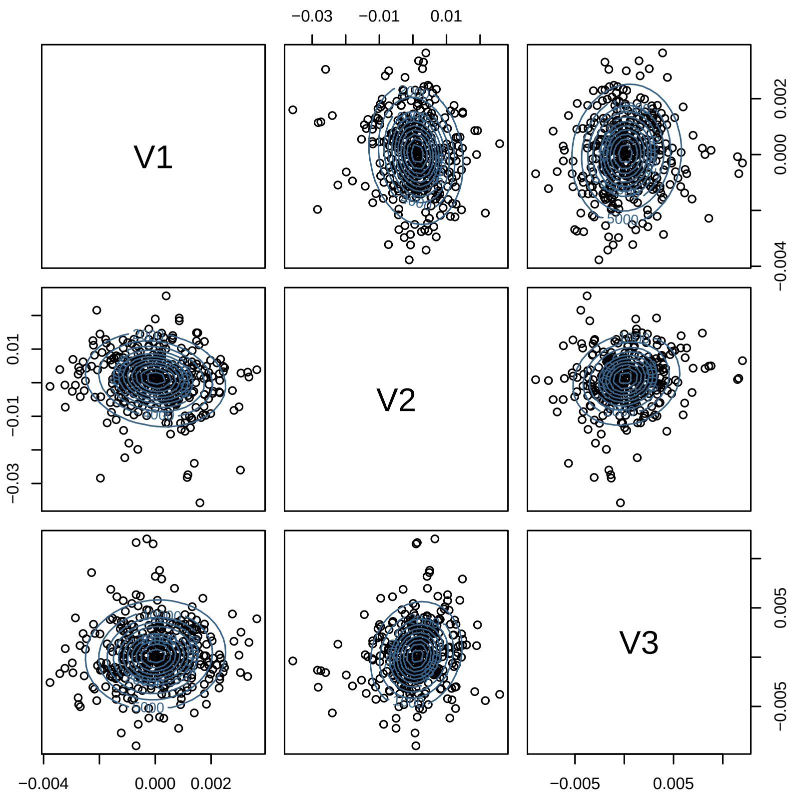


Figure 7.1 Fitted asset return distribution for major Swiss Indexes: Scatter plot of the Swiss Performance Index SPI, the Swiss Bond Index SBI, and the Swiss Immofunds Index SII returns overlayed by the contours for the fitted data.

7.3 Simulating Asset Returns from a given Distribution

From the fitted parameters we can generate assets sets with the same distributional properties as the empirical data. The simulation of artificial data sets is performed by the function `assetsSim()`.

```
> args(assetsSim)

function (n, dim = 2, model = list(mu = rep(0, dim), Omega = diag(dim),
    alpha = rep(0, dim), df = Inf), assetNames = NULL)
NULL
```

The first argument, `n`, sets the number of records to be simulated, the second, `dim`, sets the number of dimension (number of columns) of the data set and the third, `model`, sets the list of parameters. Alternatively, we can use as input the fitted model from the parameter estimation as returned by the function `assetsFit()`.

```
> slotNames(fit)

[1] "call"        "method"      "model"       "data"        "fit"
[6] "title"       "description"

> SIM <- EMP <- LPP2005.RET[, 1:3]
> X <- assetsSim(n = nrow(EMP), dim = ncol(EMP), model = fit@model)
```

This simulation has generated 10 random records of asset returns with the same distributional properties as the fitted empirical SPI equity, SBI bond, and SII real estate indices. The object returned `X` is a time series, which can be easily transformed into a `timeSeries` object with the same instrument names (column names) and date stamps (row names), using the function `series()`.

```
> series(SIM) <- as.matrix(X)
> head(SIM)

GMT
                   SBI        SPI         SII
2005-11-01  0.00031643 -0.0014141 -0.00379416
2005-11-02 -0.00014381  0.0212095  0.00102455
2005-11-03  0.00052415 -0.0079900  0.00031407
2005-11-04 -0.00020105 -0.0018232 -0.00275364
2005-11-07  0.00010807  0.0026618  0.00374866
2005-11-08 -0.00135207 -0.0035865  0.00449273
```

Chapter 8
Selecting Similar or Dissimilar Assets

Required R package(s):

```
> library(fPortfolio)
```

In many cases we want to select in a data pre-processing step the most dissimilar assets in a large data set of assets to reduce the number of assets in portfolio design. This can be done using statistical approaches which sort out the assets in groups with similar behaviour and similar properties. To those approaches belong cluster algorithms, like hierarchical clustering or k-means clustering, which group similar assets together and separate dissimilar ones (Kaufman & Rousseeuw, 1990). Another popular approach uses eigenvalue analysis. In addition we show how to add additional cluster methods, such as a robust k-means approach, to the cluster approaches already supported by Rmetrics.

8.1 Functions for Grouping Similar Assets

The Rmetrics function for selecting similar or dissimilar assets from a data set is the function `assetsSelect()`.

```
> args(assetsSelect)
```

```
function (x, method = c("hclust", "kmeans"), control = NULL,
    ...)
NULL
```

```
Function:
assetsSelect        selects assets from a cluster approach
assetsCorEigenPlot  performs eigenvalue analysis of assets
assetsArrange       rearranges columns in a data set of assets
```

Listing 8.1 Functions to select similar or dissimilar assets from a data set by clustering methods

The `method` argument of the `assetsSelect()` function allows you to select the type of grouping, either according to *hierarchical clustering* `method="hclust"` or according to *k-means clustering* `method="kmeans"`.

8.2 Grouping Asset Returns by Hierarchical Clustering

Setting `method="hclust"` performs a hierarchical cluster analysis on a set of dissimilarities and methods for analyzing it.

```
Function:
assetsSelect        for hierarchical clustering of dissimilarities

Arguments:
  x                 a 'timeSeries' object
  method            "hclust"
  control           list of optional cluster controls,
                    method - the name of the clustering method
                      "ward", "single", "complete", "average",
                      "mcquitty", "median", "centroid",
                    measure - the name of the distance measure
                      "euclidean", "maximum", "manhattan",
                      "canberra", "binary", "minkowski"
   ...              optional arguments
```

Listing 8.2 Functions to select similar or dissimilar assets from a data set by hierarchical clustering. The entry `method` in the `control` list determines the method used for the hierarchical clustering, and the entry `measure` determine the measure for the distance matrix

To perform a hierarchical clustering on asset returns, we call the function `assetsSelect()` with the argument `method="hclust"`. The underlying function is the R functions `hclust()` from the `stats` package. Arguments to this function can be passed to the `assetsSelect()` function through the `control` argument and the dots ... argument.

Internally, the function `hclust()` performs a clustering using a set of dissimilarities for the n objects being clustered. Initially, each object is assigned to its own cluster and then the algorithm proceeds iteratively, at each stage joining the two most similar clusters, continuing until there is just a single cluster. At each stage distances between clusters are recomputed by the Lance-Williams dissimilarity update formula according to the particular clustering method being used.

Let us start to group the assets and the benchmark series from the data set `LPP2005.RET`. We use the default settings, the *complete linkage method* with an *euclidean distance measure*.

```
> lppData <- LPP2005.RET
> hclustComplete <- assetsSelect(lppData, method = "hclust")
> hclustComplete

Call:
hclust(d = dist(t(x), method = measure), method = method)

Cluster method   : complete
Distance         : euclidean
Number of objects: 9

> plot(hclustComplete, xlab = "LPP2005 Assets")
> mtext("Distance Metric: Euclidean", side = 3)
```

A number of different clustering methods can be provided through the dots argument. Ward's minimum variance method aims at finding compact, spherical clusters. The complete linkage method finds similar clusters. The single linkage method (which is closely related to the minimal spanning tree) adopts a friends-of-friends clustering strategy. The other methods can be regarded as aiming for clusters with characteristics somewhere between the single and complete link methods. Note however, that methods `"median"` and `"centroid"` do not lead to a monotone distance measure, or equivalently the resulting dendrograms can have so called inversions (which are hard to interpret). For details we refer to the help page of the function `hclust()`.

The next example shows how to set up an alternative hierarchical clustering with with the *euclidean distance measure*, but now with Ward's method of *minimum variance agglomeration*

```
> hclustWard <- assetsSelect(lppData, method = "hclust", control = c(measure =
        "euclidean",
     method = "ward"))
> hclustWard

Call:
hclust(d = dist(t(x), method = measure), method = method)

Cluster method   : ward
Distance         : euclidean
Number of objects: 9

> plot(hclustWard)
> mtext("Distance Metric: Euclidean", side = 3)
```

The two plots have created dendograms which are shown in Figure 8.1 and Figure 8.2.

Further information can be extracted from the results returned by the `hclust` function. This function returns a list of S3 class `hclust` with the following values:

```
Function:
hclust              hierarchical clusters of dissimilarities

Values:
merge               merging process of clusters
height              the clustering height
order               permutation of the original observations
labels              labels for the objects being clustered
call                the call which produced the result
method              the cluster method that has been used
dist.method         the distance that has been used
```

Listing 8.3 Returned Values from hierarchical clustering

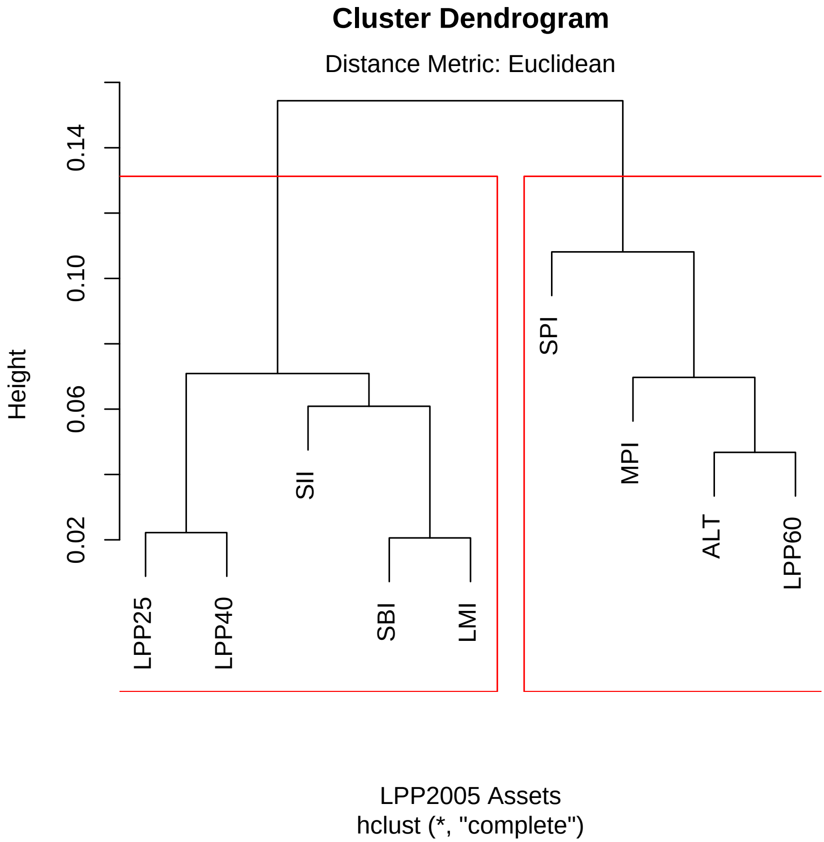


Figure 8.1 Hierarchical clustering of Swiss pension fund assets: Dendrogram plot (as obtained from default settings) for the Swiss pension fund assets set SBI, SBI, SII, LMI, MPI, and ALT, including the three benchmarks LPP25, LPP40, and LPP60.

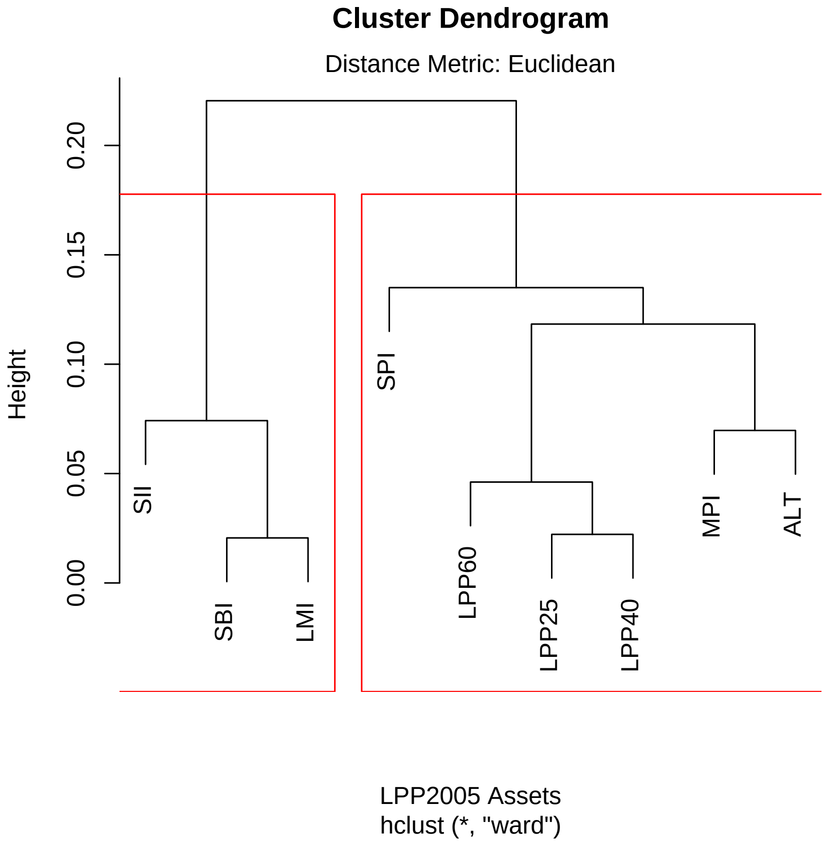


Figure 8.2 Hierarchical clustering of Swiss pension fund assets: Dendrogram plot as obtained using an euclidean distance measure and Ward's method for clustering for the Swiss pension fund assets set SBI, SBI, SII, LMI, MPI, and ALT, including the three benchmarks LPP25, LPP40, and LPP60.

8.3 Grouping Asset Returns by k-means Clustering

If we set `method="kmeans"` then the function `assetsSelect()` performs a *k-means clustering* on the financial time series, i.e. the time series of financial returns.

```
Function:
assetsSelect       for k-means cluster analysis

Arguments:
  x                a 'timeSeries' object
  method           "kmeans"
  control          list of cluster controls:
                   center - the number of clusters
                   algorithm - name of the algorithm
                     "Hartigan-Wong", "Lloyd",
                     "Forgy", "MacQueen"
```

Listing 8.4 Functions to select similar or dissimilar assets by k-means clustering. The entry `center` in the `control` list sets the number of clusters, and the entry `algorithm` selects the name of the clustering algorithm to be used

The transposed data `t(x)` given by the `@data` slot of the time series `x` is clustered by the k-means method, which aims to partition the points into `k` groups such that the sum of squares from points to the assigned cluster centres is minimized. At the minimum, all cluster centres are at the mean of their Voronoi sets, i.e. the set of data points which are nearest to the cluster centre.

The algorithm of Hartigan & Wong (1979) is used by default. Note that some authors use k-means to refer to a specific algorithm rather than the general method: most commonly the algorithm given by MacQueen (1967) but sometimes that given by Lloyd (1982) and Forgy (1965). The Hartigan-Wong (Hartigan & Wong, 1979) algorithm generally does a better job than either of those, but trying several random starts is often recommended.

Except for the Lloyd-Forgy method, `k` clusters will always be returned if a number is specified. If an initial matrix of centres is supplied, it is possible that no point will be closest to one or more centres, which is currently an error in the Hartigan-Wong method.

The number of centres and the name of the desired algorithm can be passed by the `control` argument,

```
> control <- c(centers = 2, algorithm = "H")
```

The list of algorithms includes `"Hartigan-Wong"`, `"Lloyd"`, `"Forgy"`, and `"MacQueen"`. Note that the names can be abbreviated in the `control` argument. The default settings for the number of centres is three, and the default algorithm is the algorithm of Hartigan and Wong. Let us consider the case with two groups

```
> kmeans <- assetsSelect(lppData, method = "kmeans", control <- c(centers = 2,
    algorithm = "Hartigan-Wong"))
> sort(kmeans$cluster)

  SBI   SII   LMI LPP25 LPP40   SPI   MPI   ALT LPP60
    1     1     1     1     1     2     2     2     2
```

The result shows us that the assets are clustered in two groups, one with lower risk and the other with higher risky assets. In group 1 (lower risk) we find the assets SBI, SII, LMI as well as the L25 and LPP40 benchmarks), in group 2 (higher risk) we find the assets SPI, MPI, ALT, and the benchmark LPP60, a quite natural grouping which we would have expected.

Further information can be extracted from the results returned from `kmeans` clustering. The function returns a list of S3 class `kmeans` with the following values:

```
Function:
kmeans              performs k-means cluster analysis

Values:
cluster             integer vector indicating the cluster to
                    which each point is allocated
centers             matrix of cluster centres
withinss            within-cluster sum of squares for each cluster
size                number of points in each cluster
```

Listing 8.5 Returned Values from k-means clustering

8.4 Grouping Asset Returns through Eigenvalue Analysis

A third approach groups the individual assets according to an eigenvalue analysis. This can be done calling the function `assetsCorEigenPlot()`.

```
> args(assetsCorEigenPlot)

function (x, labels = TRUE, title = TRUE, box = TRUE, method = c("pearson",
    "kendall", "spearman"), ...)
NULL
```

The function takes a multivariate `timeSeries` object `x` as input and performs according to the specified method for the computation of the correlation matrix an eigenvalue analysis.

The function calculates the first two eigenvectors of the correlation matrix and plot their components against the x and y directions. This results in a grouping of the assets. Three methods are available to compute the correlation matrix, `"person"`, `"kendall"`, and `"spearman"`. If `method` is `"kendall"` or `"spearman"`, Kendall's tau or Spearman's rho statistic is used to estimate a rank-based measure of association. These are more robust and have been recommended if the data do not necessarily come from a bivariate normal distribution. If `method` is `"pearson"`, the usual correlation coefficient will be returned.

```
> assetsCorEigenPlot(lppData, method = "kendall")
```

8.5 Grouping Asset Returns by Contributed Cluster Algorithms

There are several other contributed R packages an the CRAN server which provide alternative cluster algorithms or alternative implementations. One of the most prominent packages is the R package `cluster` contributed by Maechler, Rousseeuw, Struyf & Hubert (2009).

```
Function:
                    Hierarchical Clustering:
```

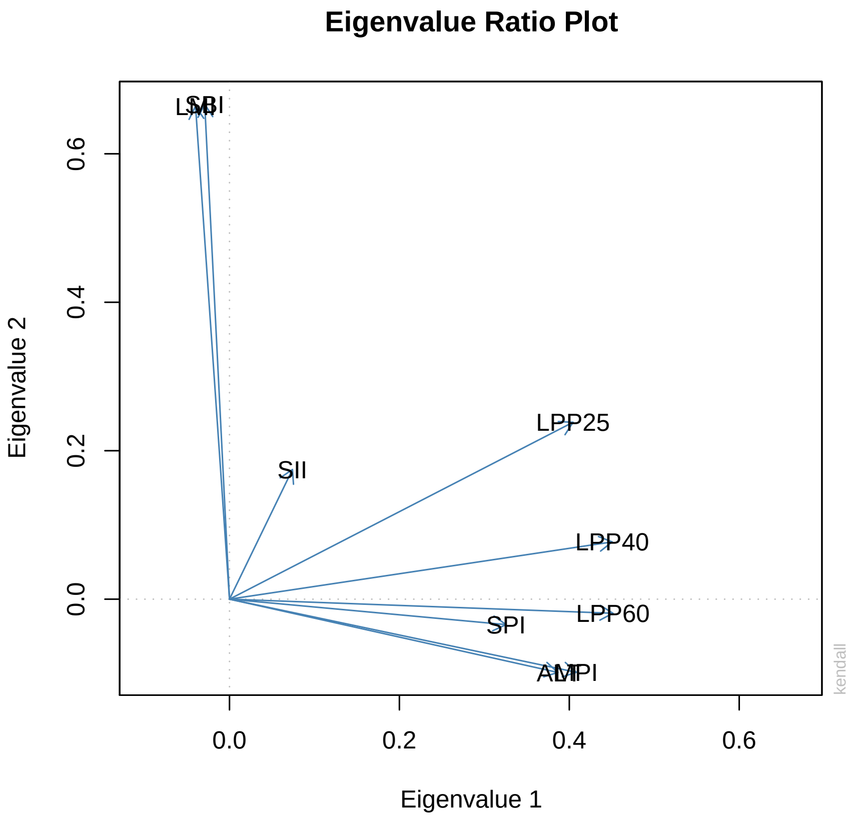


Figure 8.3 Grouping assets by eigenvalue analysis: The closeness of the arrows is a measure for similarities between individual assets.

```
diana                divisive hierarchical clustering
mona                 divisive hierarchical clustering with binary variables
agnes                agglomerative hierarchical clustering
                   Partitioning Methods:
pam                  partitioning into clusters around medoids,
                     a more robust version of k-means
clara                partitioning method for much larger data sets
fanny                fuzzy clustering of the data into k clusters
                   Pairwise Dissimilarities:
daisy                pairwise dissimilarities between observations
```

Listing 8.6 Further functions for clustering from R's contributed `cluster` package

For the grouping of financial returns, one can use these functions with the transposed data matrix directly, `t(series(x))`, or one can add additional methods to the `assetsSelect()` function. This can be done in the following way, for example using the more robust k-means algorithm implemented in `pam`:

```
> .pamSelect <- function(x, control = NULL, ...) {
     if (is.null(control))
         control = c(k = 2, metric = "euclidean")
     k <- as.integer(control[1])
     metric <- control[2]
     pam(x <- as.matrix(x), k = k, metric = metric, ...)
 }
```

Now apply Rmetrics' `assetSelect()` function:

```
> library(cluster)
> pam <- assetsSelect(LPP2005.RET, method = "pam", control <- c(k = 2,
     metric = "euclidean"))
> plot(pam, which.plots = 1)
```

Note that the plot of a cluster partition consists of a two-dimensional representation of the observations, in which the clusters are indicated by ellipses.

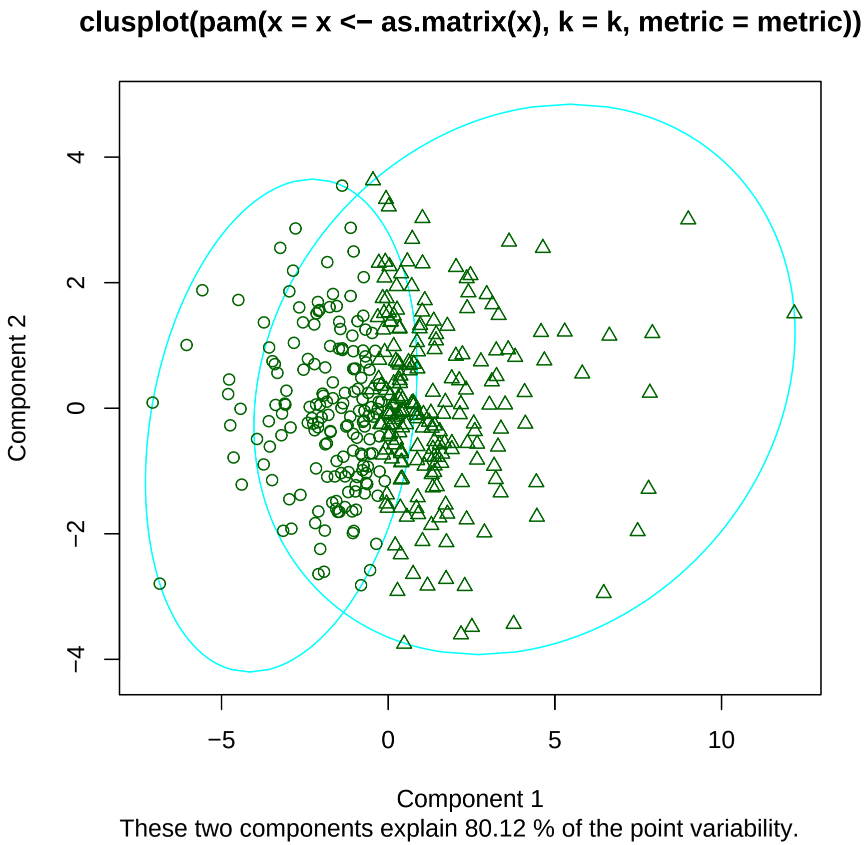


Figure 8.4 Grouping assets by partitioning around medoids: The graph shows a two-dimensional representation of the observations, in which the clusters are indicated by ellipses.

8.6 Ordering Data Sets of Assets

Pairwise correlations in financial data sets give important information on the pairwise dependencies of the individual asset returns. Graphical displays help us to visualize the correlations. This can be happen in many different ways depending on the ordering of the columns of the multivariate time series.

There are several ways to order a set of assets column by column. The most obvious ordering of assets may be the sorting in alphabetical order. From a statistical point of view we can use more sophisticated schemes, for example ordering by a PCA analysis (the default) or by hierarchical clustering. For this we can use the function `assetsArrange()`

```
> args(assetsArrange)

function (x, method = c("pca", "hclust", "abc"), ...)
NULL
```

with the following options for rearranging the data set of assets: `method="pca"` returns PCA correlation ordered column names, `"hclust"` returns hierarchical clustered column names, and `"abc"` returns alphabetically sorted column names.

In the following investigation of the pairwise correlations we sort the `LPP2005` data sets by hierarchical clustering:

```
> colnames(lppData[, 1:6])

[1] "SBI" "SPI" "SII" "LMI" "MPI" "ALT"

> Assets <- assetsArrange(lppData[, 1:6], method = "hclust")
> LPP2005HC <- 100 * lppData[, Assets]
> head(round(LPP2005HC, 5))

GMT
                SII      SBI      LMI      SPI     MPI      ALT
2005-11-01 -0.31909 -0.06127 -0.11089  0.84146 0.15481 -0.25730
2005-11-02 -0.41176 -0.27620 -0.11759  0.25193 0.03429 -0.11416
2005-11-03 -0.52094 -0.11531 -0.09925  1.27073 1.05030  0.50074
2005-11-04 -0.11272 -0.32358 -0.11985 -0.07028 1.16796  0.94827
2005-11-07 -0.17958  0.13110  0.03604  0.62052 0.27096  0.47240
2005-11-08  0.21034  0.05393  0.23270  0.03293 0.03468  0.08536
```

Chapter 9

Comparing Multivariate Return and Risk Statistics

Required R package(s):

```
> library(fPortfolio)
```

The star and segment plots introduced by Chambers et al. (1983) allows you to display multivariate data sets. Each star in a star plot represents a single observation. Typically, star and segment plots are generated in a multi-plot format with many stars or segments on each page and each star or segment representing one observation. Star plots are used to examine the relative values for a single data point and to locate similar or dissimilar points.

9.1 Star and Segment Plots

For the investigation of financial assets star plots can be used to answer the following questions:

- Which assets are dominant for a given observation?
- Which observations are most similar, i.e., are there clusters of observations?
- Are there outliers in the data set of assets?

Star plots are helpful for small-to-moderately-sized multivariate data sets. Their primary weakness is that their effectiveness is limited to data sets with less than a few hundred points. With data sets comprising more data points, they tend to be overwhelming. Rmetrics has implemented star plots to investigate several aspects of data sets of assets.

```
Function:
stars                   Star/Segment plots of a multivariate data
assetsStarsPlot         Segment/star diagrams of multivariate data sets
assetsBasicStatsPlot    Segment plot of basic return statistics
assetsMomentsPlot       Segment plot of distribution moments
assetsBoxStatsPlot      Segment plot of box plot statistics
```

Listing 9.1 Functions for star and segment plots

R's basic function to create star and segment plots is named `stars()`. It has the following arguments:

```
> args(stars)

function (x, full = TRUE, scale = TRUE, radius = TRUE, labels = dimnames(x)[[1
     L]],
    locations = NULL, nrow = NULL, ncol = NULL, len = 1, key.loc = NULL,
    key.labels = dimnames(x)[[2L]], key.xpd = TRUE, xlim = NULL,
    ylim = NULL, flip.labels = NULL, draw.segments = FALSE, col.segments = 1L:
     n.seg,
    col.stars = NA, axes = FALSE, frame.plot = axes, main = NULL,
    sub = NULL, xlab = "", ylab = "", cex = 0.8, lwd = 0.25,
    lty = par("lty"), xpd = FALSE, mar = pmin(par("mar"), 1.1 +
        c(2 * axes + (xlab != ""), 2 * axes + (ylab != ""), 1,
            0)), add = FALSE, plot = TRUE, ...)
NULL
```

The general Rmetrics star plot, `assetsStarsPlot()` is just a synonym for the basic function `stars()`

```
> args(assetsStarsPlot)

function (x, method = c("segments", "stars"), locOffset = c(0,
    0), keyOffset = c(0, 0), ...)
NULL
```

The specific star plots `assetsBasicStatsPlot()`, `assetsBoxStatsPlot()`, and `assetsMomentsPlot()` are built on top of the function `assetsStarsPlot()`. All three functions have the same list of arguments as input.

```
> args(assetsBasicStatsPlot)

function (x, par = TRUE, oma = c(0, 0, 0, 0), mar = c(4, 4, 4,
    4), keyOffset = c(-0.65, -0.5), main = "Assets Statistics",
    title = "Assets", titlePosition = c(3, 3.65), description = "Basic Returns
      Statistics",
    descriptionPosition = c(3, 3.5), ...)
NULL
```

9.2 Segment Plots of Basic Return Statistics

Let us consider the returns of the Swiss Pension Fund Index LPP2005.RET and let us compute the basic statistics as returned by the function `basicStats()`. The following statistics are considered: minimum and maximum values, first and third quartiles, mean and median, sum, SE mean, LCL mean and UCL mean, variance, standard deviation, skewness, and kurtosis. These observations are displayed as a segment plot. Which assets look similar and which look dissimilar?

```
> lppData <- LPP2005.RET
> assetsBasicStatsPlot(lppData[, -8], title = "", description = "")
```

This question is answered by Figure 9.1. Similar are SBI and LMI, and on the other hand SPI, MPI, and ALT. The SII seems less similar to the bonds as well as to the equities. LPP25 can be interpreted to represent more the bond assets, and the LPP60 more the equities and the alternative investment asset class.

9.3 Segment Plots of Distribution Moments

This segment plot displays four distributional sample estimates from the empirical asset returns including the mean, standard deviation, skewness, and kurtosis. When it is sufficient to characterize a distribution by its first four moments, then this plot allows for a simple comparison of the assets.

```
> assetsMomentsPlot(lppData[, -8], title = "", description = "")
```

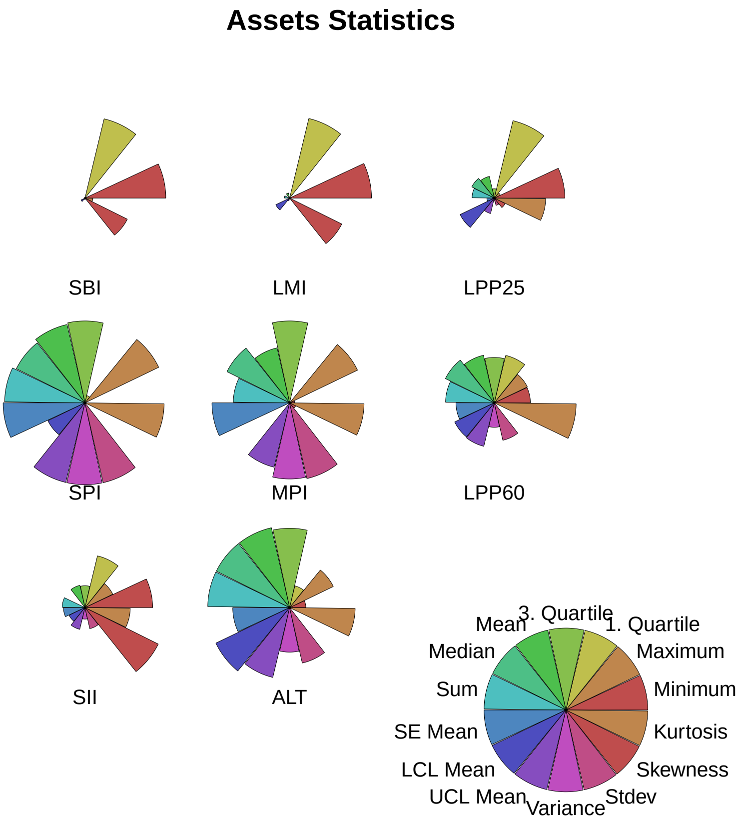


Figure 9.1 Segment plots based on the comparison of return statistics. 14 statistics are taken into account which represent the distributional properties of the six asset classes SBI, LMI, SII, SPI, MPI, and ALT as well as the two benchmarks LPP26, and LPP60.

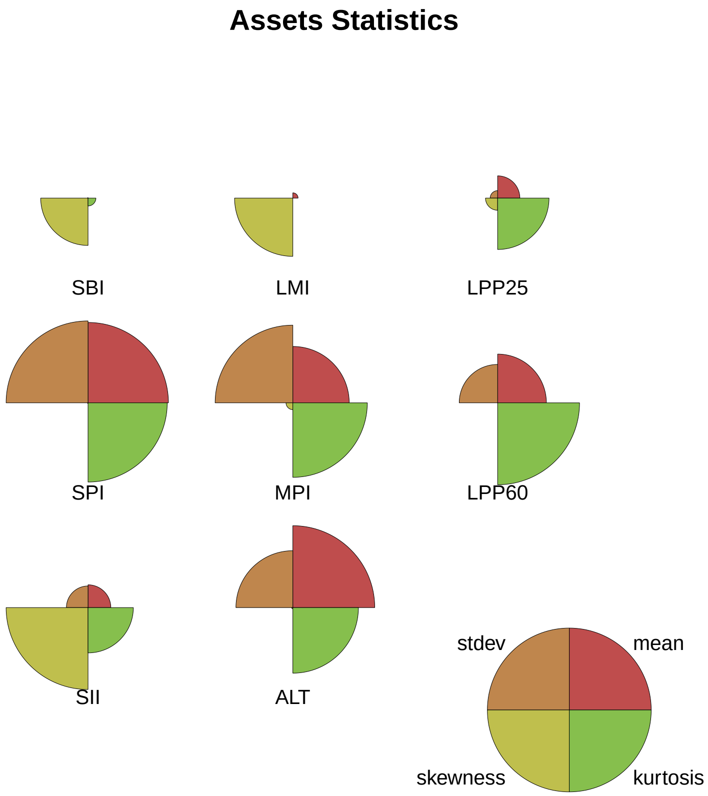


Figure 9.2 Star plots from distributional moments including the mean, standard deviation, skewness and kurtosis. The segments for the Swiss, SBI, and foreign bonds, LMI, look similar, and also the segments for the equity investments, the SPI, MPI, and ALT. The Swiss Immofunds Index, SII, differs from the remaining assets. We can also say, that the LPP60 benchmark is dominated by equities.

Figure 9.2 shows the same results as already observed in the segment plots for the basic return statistics and the box plot statistics.

9.4 Segment Plots of Box Plot Statistics

These segment plot uses as observations the values returned by the function `boxPlot()`, i.e. lower and upper hinge, lower and upper whisker, and the median.

```
> assetsBoxStatsPlot(lppData[, -8], title = "", description = "")
```

This segment graph in Figure 9.3 allows us to compare the assets from the view of the box plot statistics. Similarities are obvious between the Swiss and foreign bonds, and the Swiss and foreign equities together with the alternative investments. The Swiss Immo Funds Index is in between. The LPP25 benchmark is closer to the bonds, and the LPP60 benchmark is closer to the equities and alternative investments.

9.5 How to Position Stars and Segments in Star Plots

The default positions of stars or segments in a star plot are tailored for up to eight assets and a colour wheel legend. The positions for any other numbers of assets have to be adjusted individually by hand, and requires some process of trial and error. In most cases it is sufficient to modify the arguments `oma`, `mar`, `locOffset` and `keyOffset`. Another alternative is to use the low level function `stars()` directly.

It is also worth noting that star plots are most helpful for small to moderate sized multivariate data sets. Their primary weakness is that their effectiveness is limited to data sets with fewer than a few dozens points. For larger data sets, they tend to be overwhelming.

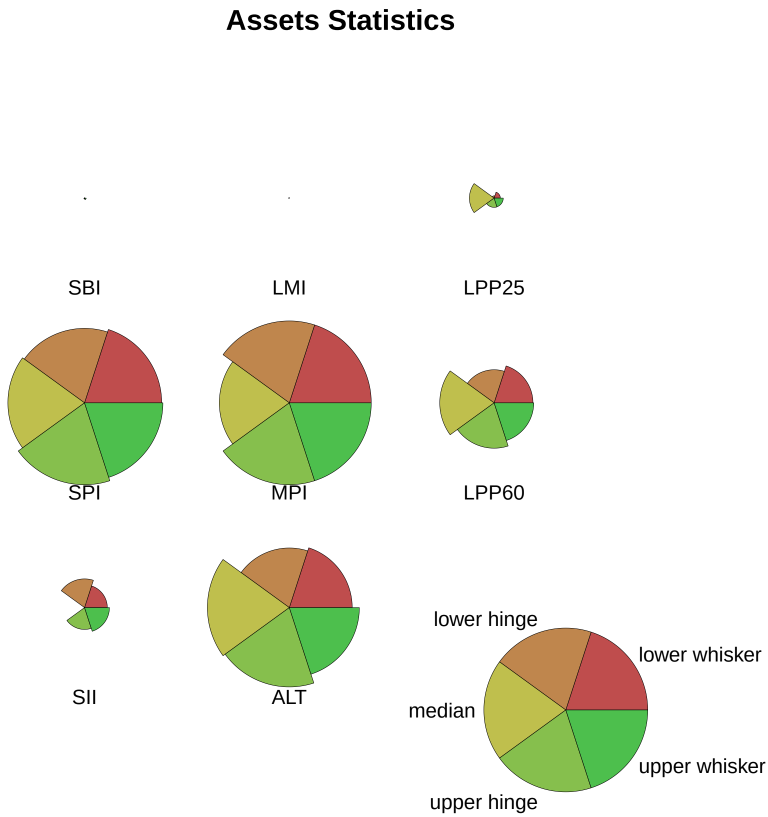


Figure 9.3 Star Plots from box plot statistics grouping the assets with respect to lower and upper hinge, lower and upper whisker, and the median of the assets and LPP benchmark series.

Chapter 10

Pairwise Dependencies of Assets

Required R package(s):

```
> library(fPortfolio)
```

To display dependencies, similarities or correlations between two individual assets R offers the function `pairs()`. If you use the default settings, this function produces a scatter plot. Rmetrics adds customized plots to provide different views on pairs of assets. This allows us to judge on different aspects of pairwise correlations and dependencies. The views include bivariate scatterplots, correlation tests, image plots, and histogram binning, among other exploratory data analysis techniques.

10.1 Simple Pairwise Scatter Plots of Assets

The function `assetsPairsPlot()` is a simple wrapper for R's base function `pairs()`. The function just transforms the data set of assets into a matrix object and plots `pairs(series(x), ...)`. Usually, the input `x` is given in form of a multivariate `timeSeries` object, then the `@data` slot is extracted by the function `series()`, and finally, a scatter plot of the data matrix is displayed.

```
> args(pairs.default)
```

```
function (x, labels, panel = points, ..., lower.panel = panel,
    upper.panel = panel, diag.panel = NULL, text.panel = textPanel,
    label.pos = 0.5 + has.diag/3, cex.labels = NULL, font.labels = 1,
    row1attop = TRUE, gap = 1)
NULL
```

```
Function:
pairs              displays pairs of scatterplots of assets
assetsPairsPlot    displays pairs of scatterplots of assets
assetsCorgramPlot  displays correlations between assets
assetsCorTestPlot  displays and tests pairwise correlations
assetsCorImagePlot displays an image plot of correlations
squareBinning      does a square binning of data points,
hexBinning         does a hexagonal binning of data points
```

Listing 10.1 Functions for pairwise assets plots

How to create a simple scatter plot

In the following example we create a simple scatter plot for all pairwise asset returns using the function `assetsPairsPlot()`.

```
> args(assetsPairsPlot)

function (x, labels = TRUE, ...)
NULL
```

We will rearrange the assets as suggested by hierarchical clustering. This yields a nicer arrangement and view of the off-diagonal scatterplot panels of the graph.

```
> Assets <- assetsArrange(LPP2005.RET[, 1:6], method = "hclust")
> LPP2005HC <- 100 * LPP2005.RET[, Assets]
> assetsPairsPlot(LPP2005HC, pch = 19, cex = 0.5, col = "royalblue4")
```

We have tailored the plot layout of the graph, using small (`cex=0.5`) full dots (`pch=19`) and the colour `royalblue4`.

The optional dot arguments which are allowed to be passed, are the same as those for R's `pairs` function, see `help(pairs.default)`.

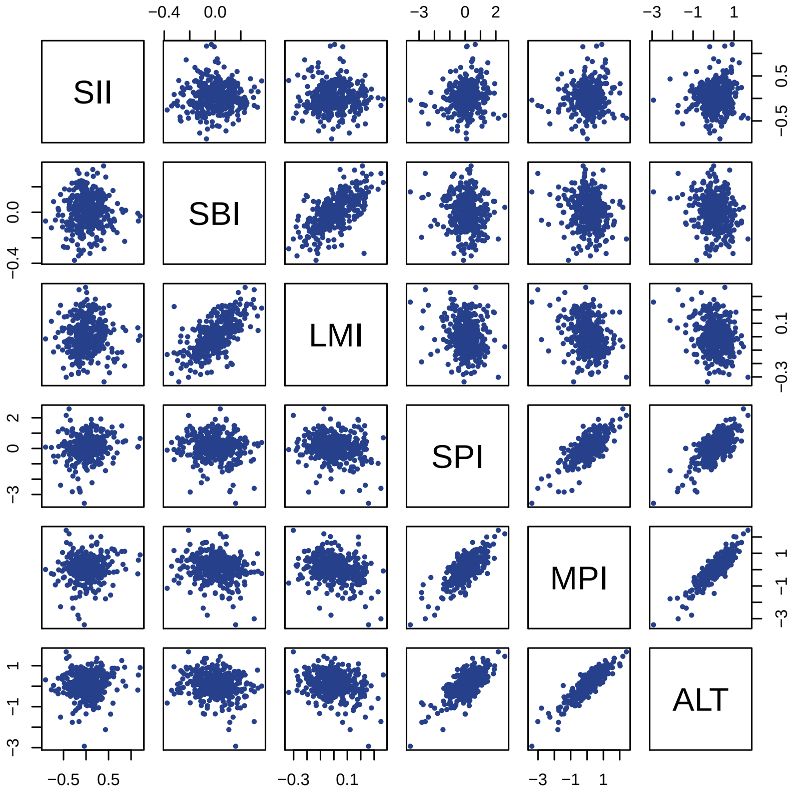


Figure 10.1 Pairwise scatter plots of assets from the Swiss pension fund index: The graph shows scatterplots for financial asset returns with default panels. In the default case both the lower and upper off-diagonal panels show a scatter plot, the diagonal panel is a text panel showing the names of the assets.

How to add a diagonal histogram panel

The possibility to modify and to define new panels makes these functions quite powerful. For example, to add histograms of the asset returns to the diagonal panels, we proceed as follows: First, we define the diagonal histogram panels,

```
> histPanel <- function(x, ...) {
      usr <- par("usr")
      on.exit(par(usr))
      par(usr = c(usr[1:2], 0, 1.5))
      h <- hist(x, plot = FALSE)
      breaks <- h$breaks
      nB <- length(breaks)
      y <- h$counts
      y <- y/max(y)
      rect(breaks[-nB], 0, breaks[-1], y, ...)
  }
```

and then we plot the correlations together with the histograms:

```
> assetsPairsPlot(LPP2005HC, diag.panel = histPanel, pch = 19,
      cex = 0.5, col = "red4", tick = 0, col.axis = "white")
```

The result is shown in Figure 10.2.

How to remove the axis labelling from a pairs plot

The two additional arguments `tick=0` and `col.axis="white"` just have to be added to suppress the ticks and the tick labels on the individual panel graphs.

Note that the scatter plots can be further customized by setting panel functions to appear as something completely different. The off-diagonal panel functions are passed the appropriate columns, the diagonal panel function (if any) is passed a single column, and the text panel function is passed a single location and the column name.

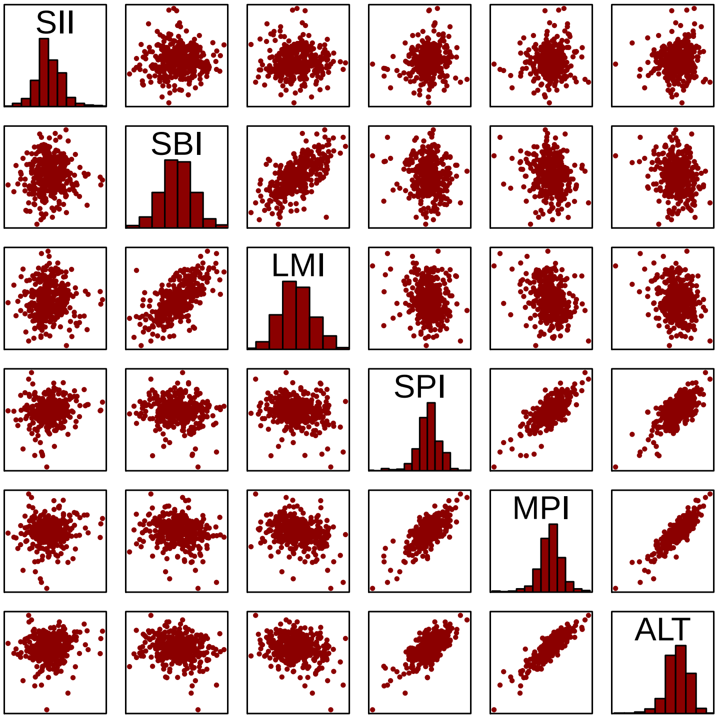


Figure 10.2 The plot shows customized pairwise scatter plots of assets with histograms in the diagonal panels. In addition, we have removed the axis labels on the panels.

10.2 Pairwise Correlations Between Assets

The function `assetsCorgramPlot()` displays a view of correlations as introduced by Friendly (2002), and is based on the implementations of the contributed R package `corrgram`[1] (Wright, 2009). This plot is also called a *correlogram plot*.

```
> args(assetsCorgramPlot)

function (x, labels = TRUE, method = c("pie", "shade"), ...)
NULL
```

The Rmetrics implementation works seamlessly with time series objects allows for two different correlogram views, a `method="pie"` and a `method="shade"` display of the off-diagonal panels. It is left to the user to extend the function to other off-diagonal displays as shown in Figure 10.3.

In the implementation of Friendly (2002), a matrix of correlations can be displayed schematically in a variety of forms: as numbers, shaded squares, bars, ellipses, or as circular pac-man symbols[2]. These schemes all attempt to show both the sign and magnitude of the correlation value, using a colour mapping of two hues in varying lightness, where the intensity of colour increases uniformly as the correlation value moves away from 0. Colour (blue for positive values, red for negative values) is used to encode the sign of the correlation, but the renderings are designed so that the sign may still be discerned when reproduced in black and white.

In the shaded row, each cell is shaded blue or red depending on the sign of the correlation, and with the intensity of colour scaled in proportion to the magnitude of the correlation. Such scaled colours are easily computed using RGB coding from red through white to blue. For simplicity, we ignore the non-linearities of colour reproduction and perception, but note that these are easily accommodated in the colour mapping function. White diagonal lines are added so that the direction of the correlation may still be discerned in black and white. This bipolar scale of colour was chosen to leave correlations near 0 empty (white), and to make positive and negative

[1] The `corrgram` functions are available as built-ins in Rmetrics.

[2] Rmetrics has implemented the pac-man scheme, `method="pie"`, which is the default, and the shaded squares scheme, `method="shade"`.

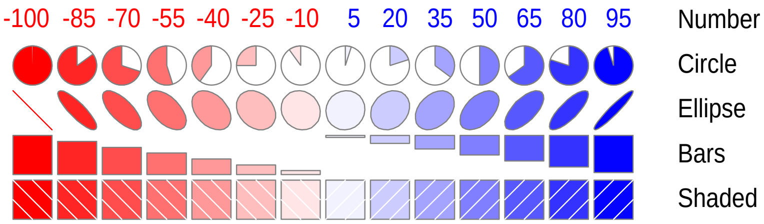


Figure 10.3 Correlation patterns for pairwise scatter plots: The patterns show how a matrix of correlations can be displayed schematically in the following forms: as numbers, as shaded squares, as bars, as ellipses, or as circular pac-man symbols. Source: Friendly (2002)

values of equal magnitude approximately equally intensely shaded. Gray scale and other colour schemes are implemented in the software, but not illustrated here.

The function `assetsCorgramPlot()` offers two options for the display of correlations. The lower panel is either a pie (pac-man) panel or a shaded panel overlayed by the scatter points, and the upper panel displays correlations as ellipses overlayed by a smooth fit of the data. Internally, smoothing is done by locally-weighted polynomial regression using the function `lowess()`. The first example displays a pac-man view

```
> assetsCorgramPlot(LPP2005HC, pch = 19, cex = 0.5)
```

and the graph uses shaded off-diagonal panels.

```
> assetsCorgramPlot(LPP2005HC, method = "shade", pch = 19,
    cex = 0.5)
```

If you want to develop your own correlogram panels, several panel functions are available as hidden functions. They can be used to generate alternative views on correlogram plots with panel displays customized by the developer. These include diagonal panel functions:

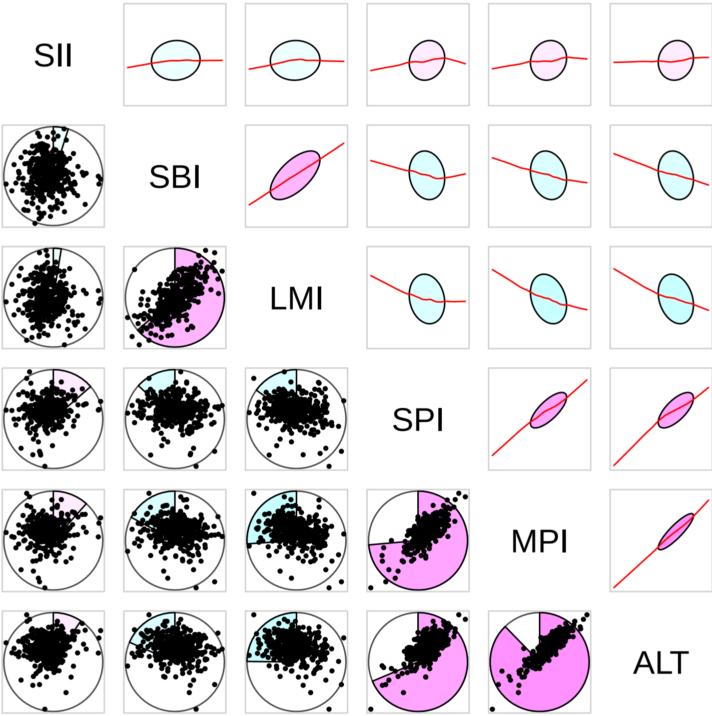


Figure 10.4 Display of sorted pairwise correlations between assets: The assets are sorted according the grouping as obtained from hierarchical clustering. The lower off-diagonal panel returns a combination of the `piePanel()` together with the a scatter plot as returned from the `pointsPanel()`. The upper off-diagonal panel returns a combination of the `ellipsePanel()` together with `lowess()` as returned from the `lowessPanel()` function. In the diagonal panel the names of the assets are shown.

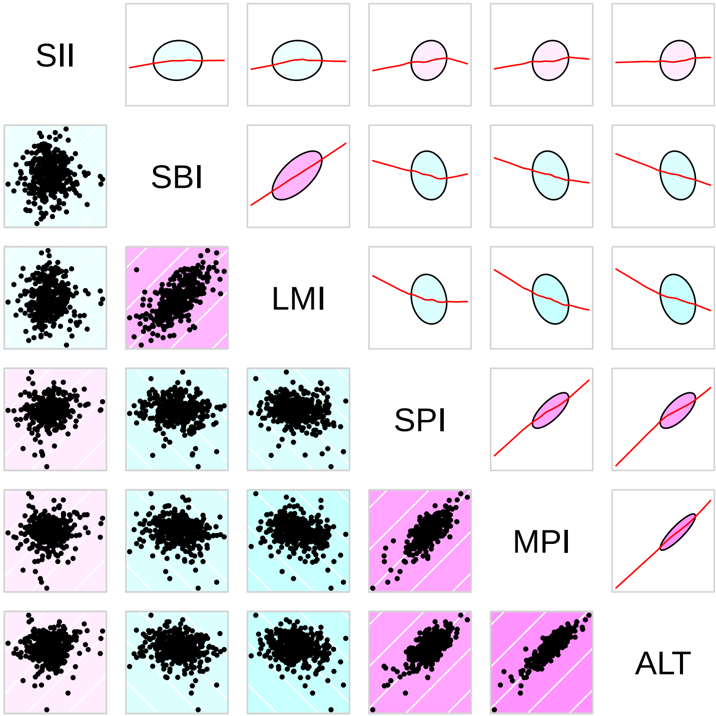


Figure 10.5 Display of sorted pairwise correlations between assets: The assets are sorted according the grouping as obtained from hierarchical clustering. The lower off-diagonal panel returns a combination of the `piePanel()` together with the a scatter plot as returned from the `pointsPanel()`. The upper off-diagonal panel returns a combination of the `ellipsePanel()` together with `lowess()` as returned from the `lowessPanel()` function. In the diagonal panel the names of the assets are shown.

```
Function:
.txtPanel          displays a text panel with asset names
.minmaxPanel       displays min and max values
.histPanel         displays a histogram plot
```

Listing 10.2 Diagonal panel functions

and off-diagonal panel functions:

```
Function:
.ptsPanel          displays a scatter plot panel
.piePanel          displays a pie (pac man) panel
.shadePanel        displays a shaded panel
.ellipsePanel      displays a coloured ellipse panel
.cortestPanel      displays a correlation test panel
.lowessPanel       displays a lowess fit panel
.numberPanel       displays correlations as numbers
.piePtsPanel       overlays a pie with a points panel
```

Listing 10.3 Off-diagonal panel functions

If you require further information, we recommend inspecting the source code of the provided hidden panel functions.

10.3 Tests of Pairwise Correlations

The function `assetsCorTestPlot()`

```
> args(assetsCorTestPlot)

function (x, labels = TRUE, ...)
NULL
```

combines a graphical view of the correlations together with the results from correlation tests.

```
> assetsCorTestPlot(LPP2005HC)
```

The lower off-diagonal panel displays the results from a scatter plot combined with a `lowess()` fit and the upper off diagonal panel shows the results returned from the function `cor.test()` which performs a test for association between paired samples, using one of Pearson's product moment correlation coefficient, Kendall's tau or Spearman's rho.

In the upper off-diagonal panels the numbers for the correlation coefficients are displayed as obtained from the function `cor()` together with the symbolic cutpoints `"***"` for <0.001, `"**"` for <0.01, `"*"` for <0.05, and `"."` for 0.1 for the p-values as obtained from the function `symnum()`. This function symbolically encodes a given numeric or logical vector or array and is thus particularly useful for the visualization of structured matrices, such as correlations.

The diagonal panel shows the names of the assets.

10.4 Image Plot of Correlations

The image plot of correlations `assetsCorImagePlot()` can be used for a larger number of assets in data sets, mainly of financial returns. It gives another alternative view. The function `assetsCorImagePlot()`

```
> args(assetsCorImagePlot)

function (x, labels = TRUE, show = c("cor", "test"), use = c("pearson",
    "kendall", "spearman"), abbreviate = 3, ...)
NULL
```

returns a quadratic plot of squared images coloured according to the computed values either of the correlation coefficient, `show="cor"`, or of the correlation tests, `show="test"`.

```
Function:
cor                 computes correlations
test                computes correlation test

Arguments:
show                specifies the image to be used, "cor" or "test"
use                 specifies the correlation coefficient to be used,
                    "pearson", "kendall" or "spearman"
```

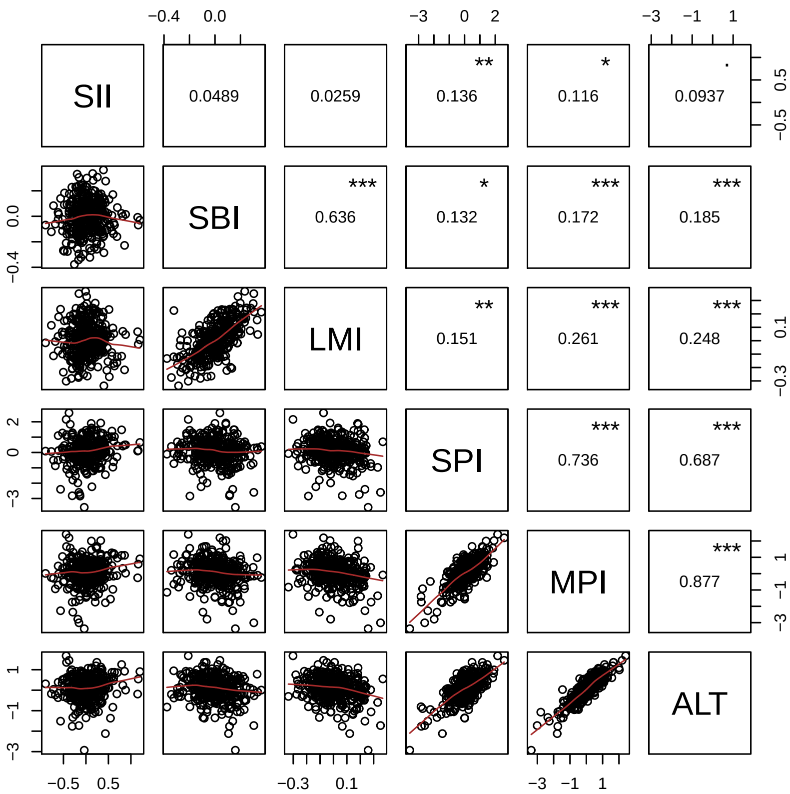


Figure 10.6 Scatter plots in combination with correlation tests: The plot represents a graphical view of correlations in combination with pairwise correlation tests. The lower off-diagonal panel displays a scatter plot combined with a `lowess()` fit. The upper off diagonal panel shows the results returned from the correlation test. The diagonal panel shows the names of the assets.

```
abbreviate          abbreviates labels to specified length
```

Listing 10.4 Correlation functions

If we have many assets, we can specify the argument `abbreviate` which allows to abbreviate assets name strings to the specified number of characters, such that they remain unique, if they were.

The following two graphs display different views on correlation images for the assets and benchmark series of the Swiss pension fund data set. In the first graph, Figure 10.7, the assets are ordered according to the degree of similarity as suggested by hierarchical clustering,

```
> assetsCorImagePlot(LPP2005HC)
```

and in the second graph, Figure 10.8, the columns of the data set are selected at random

```
> set.seed(1953)
> index <- sample(1:ncol(LPP2005HC))
> assetsCorImagePlot(LPP2005HC[, index])
```

Instead of using the the sample correlations, one can also think to use robust estimates for the correlation. The function `assetsCorImagePlot()` can easily be extended in this direction and is left as an example to the reader.

10.5 Bivariate Histogram Plots

Hexagon binning is useful for visualizing the structure in bivariate data sets of assets with a large number of records. The underlying concept is extremely simple, the plane of bivariate returns is tessellated by a regular grid of hexagons. Then the counts of points falling in each hexagon are counted, and finally the hexagons with at least one and more counts are plotted underlying a colour palette to the hexagons in proportion to the counts. If the size of the grid and the cuts in the colour palette are chosen in a proper fashion than the structure inherent in the data should emerge in the binned plots. Alternatively we can use squares instead of hexagons.

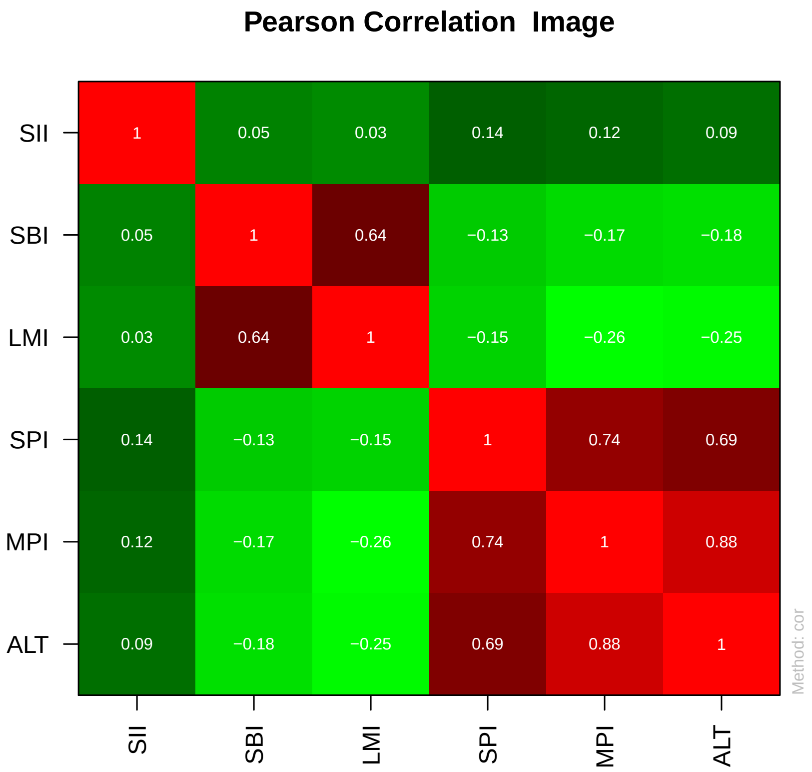


Figure 10.7 Image plots of pairwise correlations: The plot shows a symmetric coloured image with default settings: The numbers represent values for Pearson's correlation coefficient. Alternatively we can compute correlation tests. In both cases the underlying algorithms can use either Pearson's correlation coefficient, Kendall's rank correlation coefficient, or Spearman's rank correlation coefficient.

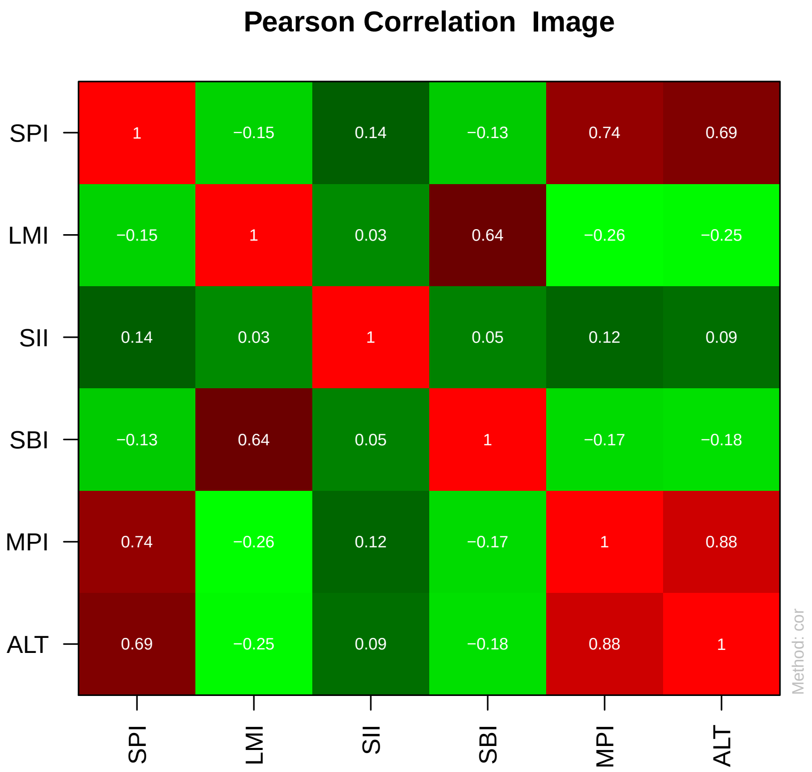


Figure 10.8 Image plots of pairwise correlations: The plot shows a symmetric coloured image with default settings: The numbers represent values for Pearson's correlation coefficient. Alternatively we can compute correlation tests. In both cases the underlying algorithms can use either Pearson's correlation coefficient, Kendall's rank correlation coefficient, or Spearman's rank correlation coefficient. In contrast to the previous graph, here the ordering of the assets is randomly chosen.

To plot histograms of pairs of assets we can use the functions `squareBinning()` and `hexBinning()`. The functions return an S3 object either of class `squareBinning` or `hexBinning`. For both objects generic plot functions exist.

```
Function:
squareBinning       does a square binning of data points
hexBinning          does a hexagonal binning of data points
plot                generic bivariate binning plot function
```

Listing 10.5 Bivariate histogram functions

The following example creates a bivariate histogram of two assets composed by hexagonal bins calling the function `hexBinning()`.

```
> args(hexBinning)

function (x, y = NULL, bins = 30)
NULL
```

The input can be either a bivariate `timeSeries` object `x`, or univariate `timeSeries` objects `x` and `y`. In the first case we set `y=NULL`, the default setting. In the next example we show the hexagonal binned histogram for Swiss bond, `SBI`, and Swiss performance index, `SPI`.

```
> hexHist <- hexBinning(SWX.RET[, c("SBI", "SPI")], bin = 20)
> plot(hexHist, xlab = "SBI", ylab = "SPI", col = rev(greyPalette(20)))
> title(main = "Bivariate Histogram Plot")
```

It is left to the reader to write his own panel functions with pairwise binned off-diagonal panels.

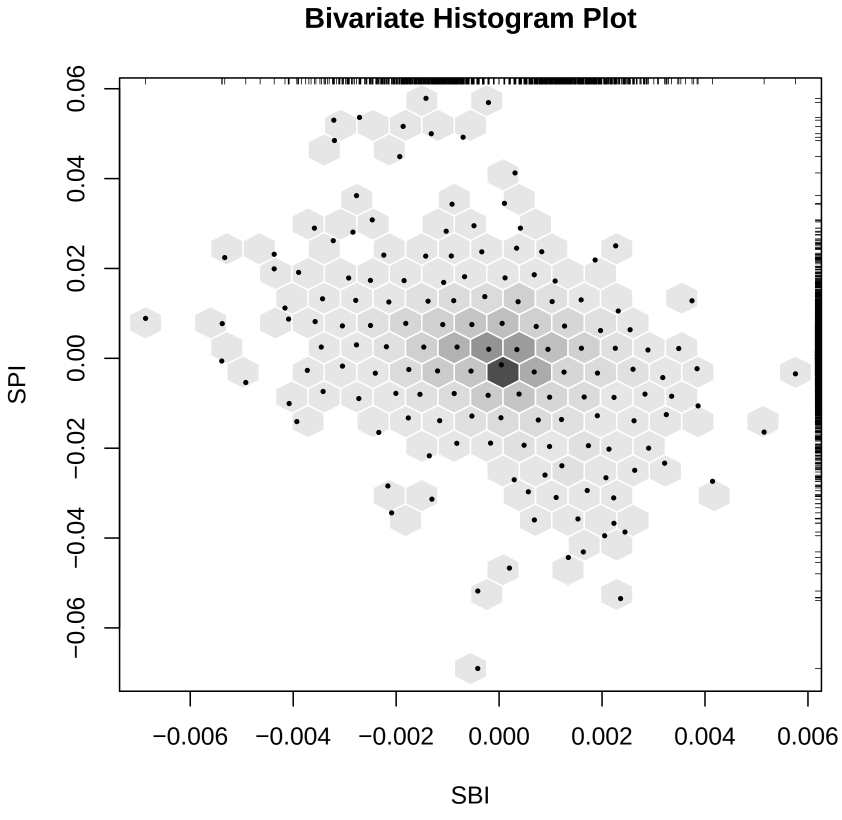


Figure 10.9 Bivariate histogram plots: The plots show bivariate histogram plots of the Swiss Bond, `SBI`, and Swiss Performance Index, `SPI`, expressed by hexagonal bins. The small dots in each bin express the centre of mass. The colours, here from a grey palette, indicate the frequency (counts) or probability of the counts.

Part III
Portfolio Framework

Introduction

Rmetrics provides functions to optimize a portfolio of assets. The goal of these functions is to determine the asset weights that minimize the risk for a desired return or, alternatively, that maximize the return for a given risk After determining the optimal weights it is advisable to conduct a performance analysis on the optimal portfolio.

However, before you can optimize a portfolio, you have to create an environment which specifies a portfolio from the beginning and defines all the parameters and values that are required to perform the optimization. In the following chapters we define three S4 classes that describe the portfolio environment including (i) the specification of all parameters describing the portfolio, (ii) the selection and description of the assets data set for which we want to optimize the portfolio, and (ii) to set the constraints under which the portfolio will be optimized.

In Chapter 11 we introduce the S4 portfolio specification class. This process consists of three parts: First, we have to decide what kind of portfolio model we want to apply, e.g. an MV portfolio or a mean-CVaR portfolio. Secondly, we have to set the required portfolio parameters; these include, for example, the weights, the target return and risk, the risk-free rate, the number of frontier points and the status of the solver. Thirdly, we have to deal with the optimization parameter options. This involves setting the name of the solver to be used in optimization, e.g. linear, quadratic or nonlinear. Further, we can set the logical flag that tells us if the optimization should be traced.

We explicitly show how to set, how to extract and how to modify these settings.

In Chapter 12 we discuss the S4 portfolio data class. In Rmetrics, data sets are represented by S4 `timeSeries` objects. These objects are used to represent financial asset returns for portfolio optimization. We describe the definition of the data, and the computation of the required statistical measures, including measures for the expected return and risk.

In Chapter 13 we describe the S4 portfolio constraints class. This is the most complex of the three classes. Constraints are defined by character strings or vectors of character strings. These strings can be used like a language to express lower and upper bounds on the ranges of weights that have to be satisfied for box, group, covariance risk budgets and general non-linear constraint settings. We give detailed examples of how to specify these constraints.

In Chapter 14 we give a brief overview of the functions that are available to optimize portfolios. This includes the case of single portfolios as well as the case of the whole portfolio frontier.

Chapter 11

S4 Portfolio Specification Class

Required R package(s):

```
> library(fPortfolio)
```

To compose and optimize a portfolio of assets we first have to specify it. This process includes choosing the kind of portfolio model we want to investigate, choosing the required portfolio parameter settings, and choosing which type of programming solver (linear, quadratic, nonlinear) should be applied.

In this chapter we introduce the S4 portfolio specification class `fPFOLIOSPEC` and describe each of its slots. These are the `@model`, the `@portfolio`, the `@optim`[1] and the `@messages` slot. All four slots are represented by lists. In addition, we show how to extract and modify individual entries from these lists. A short discussion concerning consistency of the input parameters closes this chapter.

11.1 Class Representation

All settings that specify a portfolio of assets are represented by an S4 class called `fPFOLIOSPEC`.

[1] optimization

```
> showClass("fPFOLIOSPEC")

Class "fPFOLIOSPEC" [package "fPortfolio"]

Slots:

Name:      model portfolio     optim  messages
Class:      list      list      list      list
```

An object of class `fPFOLIOSPEC` has four slots, named `@model`, `@portfolio`, `@optim`, and `@messages`. The first slot, `@model`, holds the model information, the second slot, `@portfolio`, the portfolio information and results, the `@optim` slot contains the information about the solver used for optimization, and the last slot, named `@messages`, holds a list of optional messages.

How to create a portfolio specification object

The function `portfolioSpec()` allows us to define specification settings from scratch. The default settings are for a mean-variance portfolio. To show the arguments of the function `portfolioSpec()`, you can use the function `formals()`, which prints an easy-to-read summary of the formal arguments.

```
> formals(portfolioSpec)

$model
list(type = "MV", optimize = "minRisk", estimator = "covEstimator",
    tailRisk = list(), params = list(alpha = 0.05, a = 1))

$portfolio
list(weights = NULL, targetReturn = NULL, targetRisk = NULL,
    riskFreeRate = 0, nFrontierPoints = 50, status = NA)

$optim
list(solver = "solveRquadprog", objective = NULL, options = list(meq = 2),
    control = list(), trace = FALSE)

$messages
list()
```

The settings are created specifying the values for the `model` list, for the `portfolio` list, for the optimization list `optim`, and for the `messages` list[2].

---

[2] A much more convenient way is to update an existing specification and to modify one or more of its parameters. In the next sections we present this in more detail.

A more comprehensive listing of the arguments for the default settings is shown below[3]:

```
Arguments:
model slot
    type = "MV"                    a string value
    optimize = "minRisk"           a string value
    estimator = "covEstimator"     a function name
    tailRisk = list()              a list
    params =
      list(alpha=0.05, a=1, ...) a list

portfolio slot                     a list
    weights = NULL                 a numeric vector
    targetReturn = NULL            a numeric value
    targetRisk = NULL              a numeric value
    riskFreeRate = 0               a numeric value
    nFrontierPoints = 50           an integer value
    status = NA)                   a integer value

optim slot                         a list
    solver = "solveRquadprog"      a function names
    objective = NULL               function names
    options = list()               a list with parameters
    control = list()               a list with controls
    trace = FALSE)                 a logical

messages slot:                     a list
    list = list()                  a list
```

Listing 11.1 Arguments of the function `portfolioSpec()`

We can create the default settings for a mean-variance portfolio by calling the function `portfolioSpec()` without arguments[4].

```
> defaultSpec <- portfolioSpec()
```

If we want to create a CVaR portfolio, we have to specify at least the model type, and the solver for the optimization.

[3] Note that when an argument is set to `NULL`, there is no default setting available and it is not required for the default portfolio.

[4] In this case, all arguments will just be set to their default value

```
> cvarSpec <- portfolioSpec(
    model = list(type = "CVaR", optimize = "minRisk",
                 estimator = "covEstimator", tailRisk = list(),
                 params = list(alpha = 0.05)),
    portfolio = list(weights = NULL,
                     targetReturn = NULL, targetRisk = NULL,
                     riskFreeRate = 0, nFrontierPoints = 50,
                     status = 0),
    optim = list(solver = "solveRglpk", objective = NULL,
                 params = list(), control = list(), trace = FALSE))
```

How to display the structure of a portfolio specification object

To look inside a portfolio's specification structure you can call the function `str()`. This function compactly displays the internal structure of the portfolio specification object[5]. It can be considered as a diagnostic function and as a simple way to summarize the internal structure of the object. Let us inspect the structure of the default settings:

```
> str(defaultSpec)

Formal class 'fPFOLIOSPEC' [package "fPortfolio"] with 4 slots
  ..@ model    :List of 5
  .. ..$ type     : chr "MV"
  .. ..$ optimize : chr "minRisk"
  .. ..$ estimator: chr "covEstimator"
  .. ..$ tailRisk : list()
  .. ..$ params   :List of 2
  .. .. ..$ alpha: num 0.05
  .. .. ..$ a    : num 1
  ..@ portfolio:List of 6
  .. ..$ weights        : NULL
  .. ..$ targetReturn   : NULL
  .. ..$ targetRisk     : NULL
  .. ..$ riskFreeRate   : num 0
  .. ..$ nFrontierPoints: num 50
  .. ..$ status         : logi NA
  ..@ optim    :List of 5
  .. ..$ solver   : chr "solveRquadprog"
```

[5] The entry `tailRisk` is only effective for portfolios constrained by tail risk budgets. How to use tail risk budgets in portfolio optimization is discussed in the ebook *Advanced Portfolio Optimization with R/Rmetrics*. The parameter `params$a=1` in the model slot is used as a risk aversion measure in mean-LPM portfolios. Lower partial moment, LPM, portfolios are also considered in the ebook *Advanced Portfolio Optimization with R/Rmetrics*.

```
  .. ..$ objective: NULL
  .. ..$ options  :List of 1
  .. .. ..$ meq: num 2
  .. ..$ control  : list()
  .. ..$ trace    : logi FALSE
  ..@ messages : list()
```

How to print a portfolio specification object

A nicely printed output of the same information can be obtained by using the generic `print()` function. Let us do this for the above-specified mean-CVaR portfolio.

```
> print(cvarSpec)

Model List:
 Type:                      CVaR
 Optimize:                  minRisk
 Estimator:                 covEstimator
 Tail Risk:                 list()
 Params:                    alpha = 0.05

Portfolio List:
 Target Weights:            NULL
 Target Return:             NULL
 Target Risk:               NULL
 Risk-Free Rate:            0
 Number of Frontier Points: 50
 Status:                    0

Optim List:
 Solver:                    solveRglpk
 Objective:                 list()
 Options:                   meq = 2
 Control:                   list()
 Trace:                     FALSE

Message List:
 List:                      NULL
```

11.2 The Model Slot

The `@model` slot covers all settings to specify a model portfolio. This includes the type of the portfolio, the objective function to be optimized, the estimators for mean and covariance, the tail risk[6], and optional model parameters. To extract the current model specification we can use the extractor functions

```
Model Slot - Extractor Functions:
getType             Extracts portfolio type from specification
getOptimize         Extracts what to optimize from specification
getEstimator        Extracts type of covariance estimator
getTailRisk         Extracts list of tail dependency risk matrices
getParams           Extracts parameters from specification
```

Listing 11.2 Extractor functions for the `@model` slot

and to modify the settings from a portfolio specification we can use the following assignment functions:

```
Model Slot - Constructor Functions:
setType             Sets type of portfolio optimization
setOptimize         Sets what to optimize, min risk or max return
setEstimator        Sets names of mean and covariance estimators
setParams           Sets optional model parameters
```

Listing 11.3 Constructor functions for the `@model` slot

How to modify the type of the portfolio model

The list entry `$type` from the `@model` slot describes the type of the desired portfolio. In the current implementation, `type` can take different values to represent the type of portfolios[7], such as

```
Model Slot - Argument: type
```

[6] see footnote 2

[7] The portfolio types "QLPM", "MAD", "SPS" are described in the ebook *Advanced Portfolio Optimization with R/Rmetrics*

```
Values:
"MV"                mean-variance (Markowitz) portfolio
"CVAR"              mean-conditional Value at Risk portfolio
"QLPM"              mean-quadratic-lower-partial-moment portfolio
"SPS"               Stone, Pedersen and Satchell type portfolios
"MAD"               mean-absolute-deviance Portfolio
```

Listing 11.4 The type argument for the `@model` slot

One can now use the function `getType()` to retrieve the current setting and the assignment function `setType()` to modify this selection, e.g.

```
> mySpec <- portfolioSpec()
> getType(mySpec)

[1] "MV"

> setType(mySpec) <- "CVAR"
> getType(mySpec)

[1] "CVAR"
```

In this example we changed the specification from a mean-variance portfolio to a mean-conditional value-at-risk portfolio[8].

Which objective to optimize

The list entry `$optimize` from the `@model` slot describes which objective function should be optimized. Possible choices are

```
Model Slot - Argument: optimize
Values:
"minRisk"           minimizes the risk for a given target return
"maxReturn"         maximizes the return for a given target risk
"objRisk"           gives the name of an alternative objective function
```

Listing 11.5 The optimize argument for the `@model` slot

[8] Note that we now also have to modify the solver, since for CVAR portfolios, a linear solver is required. It is currently up to the user to make sure that the specification contains no conflicts.

The first two options consider the most common choices; these are either minimizing the portfolio's risk for a given target return or maximizing the portfolio's return for a given target risk. In the default case of the mean-variance portfolio, the target risk is calculated from the sample covariance (or an alternative measure, e.g. a robust covariance estimate). The target return is computed by the sample mean of the assets if not otherwise specified. The third option leaves the user with the possibility to define any other portfolio objective function, such as maximizing the Sharpe ratio, for example[9]. You can use the function `getOptimize()` to retrieve and the assignment function `setOptimize()` to modify the current settings.

How to estimate mean and covariance of asset returns

The list entry `$estimator` from the `@model` slot requires a string denoting the function name of the covariance estimator that should be used for estimating risk. In Markowitz' mean-variance portfolio model, `type="MV"`, the default function `covEstimator()` is used, which computes the sample column means and the sample covariance matrix of the multivariate assets data series. Alternative estimators include Kendall's and Spearman's rank based covariance estimators, robust estimators, and furthermore, a shrinkage and a bagged estimator.

The minimum covariance determinant estimator `mcdEstimator()` and the minimum volume ellipsoid estimator `mveEstimator()` are based on the robust covariance estimators from R's recommended `MASS` package (Venables & Ripley, 2008), which is part of R's base environment.

The estimators `covMcdEstimator()` and `covOGKEstimator()` require functions to be loaded from the contributed R package `robustbase` (Rousseeuw et al., 2008). `covMcdEstimator()` is an alternative implementation of the MCD estimator, and is faster than the one implemented in the `MASS` package. The Orthogonalized Gnanadesikan-Kettenring estimator, OGK, can be used when the dimensionality of the covariance matrix becomes large.

The covariance estimators `shrinkEstimator()`, and `baggedEstimator()` use functions from the contributed R package `corpcor` (Schaefer et al., 2008). The shrinkage estimator computes the empirical variance of each considered random variable, and shrinks them towards their median (Schäfer

[9] For examples of user defined objective functions for specific risk measure we refer to the ebook *Advanced Portfolio Optimization with R/Rmetrics*.

& Strimmer, 2005; Opgen-Rhein & Strimmer, 2007). The bagged estimator uses bootstrap aggregating. This is a meta-algorithm to improve models in terms of stability and accuracy (Kotsiantis & Pintelas, 2004). Note that the R package `corpcor` does not have to be loaded explicitly, the required functions are available as built-ins[10].

The function `nnveEstimator()` performs robust covariance estimation by the nearest neighbour variance estimation, NNVE, method of Wang & Raftery (2002). The function is built-in from the contributed package `covRobust` (Wang et al., 2008).

```
Function:
portfolioSpec        specifies a portfolio

Model Slot:          specifies the type of estimator
  List Entry:
  estimator
    "covEstimator"       Covariance sample estimator
    "kendallEstimator"   Kendall's rank estimator
    "spearmanEstimator"  Spearman's rank estimator
    "mcdEstimator"       Minimum covariance determinant estimator
    "mveEstimator"       Minimum volume ellipsoid estimator
    "covMcdEstimator"    Minimum covariance determinant estimator
    "covOGKEstimator"    Orthogonalized Gnanadesikan-Kettenring
    "shrinkEstimator"    Shrinkage estimator
    "baggedEstimator"    Bagged Estimator
    "nnveEstimator"      Nearest neighbour variance estimator
```

Listing 11.6 Model slot of function `portfolioSpec()`

You can add you own functions to estimate the mean and covariance of the multivariate assets data series. If you want to do so, you have to write a function, e.g. named

```
myEstimator <- function(x, spec)
{
    <...>
```

[10] Built-in functions are often modified or customized functions copied from external sources, usually from contributed packages. Rmetrics uses built-ins when only a small part of the code is required, or if the functions require slight modifications to work seamlessly in the Rmetrics environment.

```
    list(mu = <...> , Sigma = <...>)
}
```

Listing 11.7 Template for a custom estimator function

where `x` is the multivariate time series object of assets, and `spec` is the portfolio specification. The argument `spec` allows additional parameters to be passed in. To be more specific, these arguments can usually be passed in through the list `@model$param`. Note that `myEstimator()` must return a named list, with at least the following two named entries: `$mu` and `$Sigma`. They represent the estimated values for the mean and covariance, respectively.

You can use the function `getEstimator()` to retrieve the current setting and the assignment function `setEstimator()` to modify the name of the estimator function to be used.

What is the tail risk list?

The list entry `tailRisk` from the `@model` slot is an empty list. It can be used to add tail risk budget constraints to the optimization. In this case a square matrix of pairwise tail dependence coefficients has to be specified as list entry. Usually, the matrix contains bivariate tail risk measures estimated via a copulae approach[11].

```
Model Slot - Argument: tailRisk
List Entries:
...                 a numeric matrix of tail dependence coefficients
```

Listing 11.8 The tailRisk argument

You can use the function `getTailRisk()` to inspect the current setting and `setTailRisk()` to assign a tail risk matrix.

How to set and modify model parameters

The list entry `$params` from the `@model` slot is a list with additional parameters used in different situations. It can be enhanced by the user if needed.

[11] Modelling tail dependence coefficients using a copula approach is presented in the ebook *Advanced Portfolio Optimization with R/Rmetrics*.

```
Model Slot - Argument: params
List Entries:
alpha               a numeric value, the VaR significance level alpha
a                   a numeric value, the LPM risk measure exponent
...                 optional parameters added by the user
```

Listing 11.9 The params argument

By default, it contains the the confidence level for "CVaR" portfolio optimization, `alpha=0.05`, and the exponent `a=1`, the parameter needed for portfolio optimization based on quadratic lower partial moments[12]. Note that you can add additional parameters. For example, you could write your own robust covariance estimator, and if this function requires some parameters, you can pass them in through the model parameter list.

Use the function `getModelParams()` and `setModelParams()` to inspect the current parameter settings, and to modify the values.

11.3 The Portfolio Slot

The `@portfolio` slot covers all settings to specify the parameters for a portfolio. This includes the weights, the target return and risk, the risk-free rate, the number of frontier points and the status of the solver.

Again, we can use the extractor functions to retrieve the current settings of the portfolio slot

```
Portfolio Slot - Extractor Functions:
getWeights          Extracts weights from a portfolio object
getTargetReturn     Extracts target return from specification
getTargetRisk       Extracts target risk from specification
getRiskFreeRate     Extracts risk-free rate from specification
getNFrontierPoints  Extracts number of frontier points
getStatus           Extracts the status of optimization
```

Listing 11.10 Extractor functions for the `@portfolio` slot

[12] Optimizing portfolios based on the quadratic lower partial moment approach is discussed in the ebook *Advance Portfolio Optimization with R/Rmetrics*.

The assignment functions can be used to modify these settings

```
Portfolio Slot - Assignment Functions:
setWeights          Sets weights vector
setTargetReturn     Sets target return value
setTargetRisk       Sets target risk value
setRiskFreeRate     Sets risk-free rate value
setNFrontierPoints  Sets number of frontier points
setStatus           Sets status value
```

Listing 11.11 Assignment functions for the `@portfolio` slot

How to set the values of weights, target return and risk

The list entries `$weights`, `$targetReturn` and `$targetRisk` from the `@portfolio` slot have to be considered collectively.

```
Portfolio slot - Arguments:
weights            a numeric vector of weights
targetReturn       a numeric value of the target return
targetRisk         a numeric value of the target risk
```

Listing 11.12 Arguments of the `@portfolio` slot

For example, if the weights for a portfolio are given, then the target return and target risk are determined, i.e. they are no longer free. As a consequence, if we set the weights to a new value, then the target return and risk also take new values, determined by the portfolio optimization. Since we do not know these values in advance, i.e. when we reset the weights, the values for the target return and risk are both set to `NA`. The same holds if we assign a new value to the target return or target risk; both of the other values are set to `NA`. By default, all three values are set to `NULL`. If this is the case, then it is assumed that an equal-weights portfolio should be calculated.

In summary, if only one of the three values is different from `NULL`, then the following procedure will be started:

1. If the weights are specified, it is assumed that a feasible portfolio should be considered.

2. If the target return is fixed, it is assumed that the efficient portfolio with the minimal risk should be considered.
3. And finally if the risk is fixed, the return should be maximized.

Use the functions `setWeights()`, `setTargetReturn()`, and `setTargetRisk()` to modify this selection. Note that a change in one of the three functions will influence the settings of the other two.

Let us look at an example of how to set the weights. First, let us display the default settings:

```
> mySpec <- portfolioSpec()
> getWeights(mySpec)

NULL

> getTargetReturn(mySpec)

NULL

> getTargetRisk(mySpec)

NULL
```

None of the three settings are available, therefore the extractor functions return `NULL`. Now, let us define a set of new weights, for example an equal-weights setting for four assets:

```
> setWeights(mySpec) <- c(1, 1, 1, 1)/4
> getWeights(mySpec)

[1] 0.25 0.25 0.25 0.25

> getTargetReturn(mySpec)

[1] NA

> getTargetRisk(mySpec)

[1] NA

> getOptimize(mySpec)

[1] "minRisk"
```

Now the target return and risk are set to NA, since we do not know the return and risk of the equal weights portfolio. On the other hand, if we want to fix the target return, for example to 2.5%, we can proceed as follows:

```
> setTargetReturn(mySpec) <- 0.025
> getWeights(mySpec)

[1] NA

> getTargetReturn(mySpec)

[1] 0.025

> getTargetRisk(mySpec)

[1] NA

> getOptimize(mySpec)

[1] "minRisk"
```

The weights and the target risk are now set to `NA`, since they are not known. In addition, the `getOptimize()` function returns `"minRisk"`, since we have specified the target return. If we set the target risk, for example to 30%, then we obtain the following settings:

```
> setTargetRisk(mySpec) <- 0.3
> getWeights(mySpec)

[1] NA

> getTargetReturn(mySpec)

[1] NA

> getTargetRisk(mySpec)

[1] 0.3

> getOptimize(mySpec)

[1] "maxReturn"
```

Note that the `getOptimize()` function now returns the value "maxReturn". The name of the optimizer also has to be changed, since we are now dealing with quadratic constraints.

How to set the risk-free rate

The risk-free rate is the theoretical rate of return of an asset with zero risk. Its value, `riskFreeRate=0`, is stored in the `@portfolio` slot and set to zero by default.

```
Portfolio Slot - Argument:
riskFreeRate        a numeric value of the risk-free rate
```

Listing 11.13 The riskFreeRate argument of the `@portfolio` slot

You can use the function `setRiskFreeRate()` to change the value of the risk-free rate, and the function `getRiskFreeRate()` to inspect its current value.

How to set the number of frontier points

The number of frontier points required by the calculation of the `portfolioFrontier` is obtained from the value of `nFrontierPoints` held in the `portfolio` slot. `nFrontierPoints` is set to 50 by default. You can change this with the function `setNFrontierPoints()`. The function `setNFrontierPoints()` returns the current setting for the number of frontier points.

```
Portfolio Slot - Argument:
nFrontierPoints     an integer value specifying the number of
                    frontier points
```

Listing 11.14 The nFrontierPoints argument of the `@portfolio` slot

Bear in mind that if when considering a single portfolio, e.g. the tangency portfolio, the minimum-variance portfolio or any other efficient portfolio, the setting for the number of frontier points will be ignored.

How to obtain the solver status information

The final `status` of portfolio optimization is returned and stored in the `@portfolio` slot. Before optimization, the value is unset to `NA`, after optimization a value of `status=0` indicates a successful termination. For other values, we recommend that you inspect the help page of the selected solver. The name of the solver can be returned by the function `getSolver()`.

```
Portfolio Slot - Argument:
status             an integer value of the status returned
                   by a portfolio optimization function
```

Listing 11.15 The status argument

Note that the function `setStatus()` should only be used internally in solver functions to save and report the exit status.

11.4 The Optim Slot

The `@optim` slot deals with the solver settings, the name of the solver to be used in optimization, the logical flag which tells us if the optimization should be traced, and the message list.

For the optimization slot we have the following extractor functions

```
Optim slot - Extractor functions:
getSolver          Extracts the name of the solver
getTrace           Extracts solver's trace flag
getObjective       Extracts the name of the objective function
getOptions         Extracts optional solver parameters
getControl         Extracts the control list of the solver
```

Listing 11.16 Extractor functions for the `@optim` slot

and assignment functions to modify these settings

```
Portfolio slot - Constructor functions:
setSolver          sets the name of the solver
setTrace           sets solver's trace flag
setObjective       sets the name of the objective function
setOptions         sets optional solver parameters
setControl         sets the control list of the solver
```

Listing 11.17 Constructor functions for the `@optim` slot

How to select an appropriate solver

The name of the default solver used for the optimization of the mean-variance Markowitz portfolio, which is the default portfolio, is a quadratic programming (QP) solver, named `solveRquadprog()` in Rmetrics. This solver implements the approach of Goldfarb & Idnani (1982).

For mean-CVaR portfolio optimization, we use a linear programming (LP) solver, named `solveRglpk()` in Rmetrics. This solver uses R's interface to the GNU linear programing kit (GLPK) (Makhorin, 2008). Rmetrics provides a wide range of additional solvers:

```
Optim Slot - Argument: solver
Values:
"solveRquadprog"    Rmetrics default QP solver
"solveRglpk"        Rmetrics default LP solver
"solveRshortExact"  analytical short selling QP solver
"solveRipop"        alternative QP solver
"solveRlpSolveAPI"  alternative LP solver
"solveRsymphony"    alternative LP solver
"solveRsocp"        QP solver for quadratic constraints
"solveRdonlp2"      NL solver for non-linear constraints
...                 for additional solvers
```

Listing 11.18 Solver arguments in the optim slot

If you change the type of portfolio, remember to check whether you have specified a solver that is compatible with that type of portfolio . You can also choose the solver by calling the function `setSolver()`.

How to trace the iteration path

The logical flag `trace` in the `@optim` slot allows (most) solvers to trace the portfolio optimization process. By default, this will not be the case, i.e. `trace=FALSE`.

```
Optim Slot - Argument:
trace               a logical flag to trace or not optimization
                    diagnostics from portfolio optimization
```

Listing 11.19 The trace argument for the `@optim` slot

Tracing the process of portfolio optimization may be especially useful if we run into problems with the solver. By setting `trace=TRUE`, we can usually find out where the problems arise. You can use the function `setTrace()` to set or reset the selection.

How to add a user-defined objective function

When we optimize a portfolio for which the objective function to be optimized is neither the mean return nor the covariance risk, or any other predefined return or risk measure, then we can use a user-defined objective function, which we can pass in through the `objective` list entry of the `@optim` slot.

```
Optim Slot - Argument:
objective          a character vector of three strings,
                   the objective function, the return,
                   and the risk function to be used.
```

Listing 11.20 The objective argument for the `@optim` slot

You can use the function `setObjective()` to set or reset the selection.

How to add optional parameters

If a user-defined objective function requires additional options, you can pass them in through the `options` list entry of the `@optim` slot.

```
Optim Slot - Argument:
options            a list of optional user supplied parameters
                   for the portfolio solvers
```

Listing 11.21 The options argument for the `@optim` slot

You can use the function `setOptions()` to set or reset the selection.

How to add control parameters for the solver

The argument `control` in the `@optim` slot allows you to control the parameters of the solvers. These are quantities such as the maximum number of iteration steps, or relative and absolute tolerances. Note that the entries in the list depend on which solver is used. An empty `list()` takes the default settings, which is what we recommend to control the parameters of the solvers.

```
Optim Slot - Argument:
control             a list of control parameters of the
                    portfolio solvers
```

Listing 11.22 The control argument for the `@optim` slot

Not all solvers allow you to modify their control settings, since these settings may be hard-coded. The QP `quadprog` solver, for instance, is one such solver. You can use the function `setControl()` to set or reset the selection, and the function `getControl()` to see the current control list.

11.5 The Message Slot

The message slots holds a list, into which you can save messages during the process of portfolio optimization. This option is especially helpful if you want to add your own portfolio models and solvers to the Rmetrics environment.

```
Messages Slot - Argument: list
list                an optional list of messages added during
                    the process of portfolio optimization
```

Listing 11.23 The argument list for the `@message` slot

11.6 Consistency Checks on Specifications

It is very important to be careful when modifying specification settings, because there are settings that are incompatible with certain other settings. For example, if you want to minimize the covariance risk for a mean-variance portfolio, you cannot assign a linear programming solver.

Currently we are working on implementing more consistency checks for the specification settings, so that you do not have to worry about creating conflicting settings. However, this has not been fully implemented in the current version of `fPortfolio`.

Chapter 12

S4 Portfolio Data Class

Required R package(s):

```
> library(fPortfolio)
```

In Rmetrics, data sets are represented by S4 `timeSeries` objects. These objects are used to represent financial returns series for portfolio optimization. Returns for a price or index series can be computed using the function `returns()`. The data summary information is stored in an object of class `fPFOLIODATA`. An object of this class holds all the information about the data set of assets that is required for portfolio optimization. In this chapter, we introduce the S4 portfolio data class and describe each of the slots. These are the `@data` slot, the `@statistics` slot and the `@tailRisk` slot. In addition, we show how to extract and modify individual slots.

12.1 Class Representation

An S4 `timeSeries` object only contains information on the series data themselves and the information on the date/time positions. Therefore, Rmetrics creates an S4 object of class `fPFOLIODATA` with the function `portfolioData()`.

```
> showClass("fPFOLIODATA")

Class "fPFOLIODATA" [package "fPortfolio"]
```

```
Slots:

Name:        data statistics   tailRisk
Class:       list       list       list
```

This S4 object holds additional information about the `timeSeries` data[1].

How to create a portfolio data object

The function `portfolioSpec()` allows you to define data settings for use in portfolio functions. The arguments of the function are

```
> args(portfolioData)

function (data, spec = portfolioSpec())
NULL
```

The settings are created by specifying the values for the time series data set and for the portfolio `spec`. First, we choose a subset of the `LPP2005.RET` returns data set, i.e. the `"SBI"`,`"SPI"`, `"LMI"` and `"MPI"`[2] columns. Then we create a portfolio object using the specified data and the default portfolio specification:

```
> lppAssets <- 100 * LPP2005.RET[, c("SBI", "SPI", "LMI", "MPI")]
> lppData <- portfolioData(data = lppAssets, spec = portfolioSpec())
```

How to display the structure of a portfolio data object

To look inside a portfolio's data structure you can call the function `str()`. This function compactly displays the internal structure of the portfolio data object. As in the case of the portfolio specification, the output can be considered as a diagnostic output and as a simple way to summarize the internal data structure.

```
> str(lppData, width = 65, strict.width = "cut")
```

---

[1] The `@tailRisk` slot is only effective for portfolios constrained by tail risk budgets. We discuss how to use tail risk budgets in portfolio optimization in the ebook *Advanced Portfolio Optimization with R/Rmetrics*.

[2] For more information on the LPP2005, see Section B.2

```
Formal class 'fPFOLIODATA' [package "fPortfolio"] with 3 slots
  ..@ data      :List of 3
  .. ..$ series :Time Series:
 Name:               object
Data Matrix:
 Dimension:          377 4
 Column Names:       SBI SPI LMI MPI
 Row Names:          2005-11-01  ...  2007-04-11
Positions:
 Start:              2005-11-01
 End:                2007-04-11
With:
 Format:             %Y-%m-%d
 FinCenter:          GMT
 Units:              SBI SPI LMI MPI
 Title:              Time Series Object
 Documentation:      Tue Jan 20 17:49:06 2009 by user:
  .. ..$ nAssets: int 4
  .. ..$ names  : chr [1:4] "SBI" "SPI" "LMI" "MPI"
  ..@ statistics:List of 5
  .. ..$ mean     : Named num [1:4] 4.07e-05 8.42e-02 5.53e-03 ..
  .. .. ..- attr(*, "names")= chr [1:4] "SBI" "SPI" "LMI" "MPI"
  .. ..$ Cov      : num [1:4, 1:4] 0.0159 -0.0127 0.0098 -0.015..
  .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. ..$ : chr [1:4] "SBI" "SPI" "LMI" "MPI"
  .. .. .. ..$ : chr [1:4] "SBI" "SPI" "LMI" "MPI"
  .. ..$ estimator: chr "covEstimator"
  .. ..$ mu       : Named num [1:4] 4.07e-05 8.42e-02 5.53e-03 ..
  .. .. ..- attr(*, "names")= chr [1:4] "SBI" "SPI" "LMI" "MPI"
  .. ..$ Sigma    : num [1:4, 1:4] 0.0159 -0.0127 0.0098 -0.015..
  .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. ..$ : chr [1:4] "SBI" "SPI" "LMI" "MPI"
  .. .. .. ..$ : chr [1:4] "SBI" "SPI" "LMI" "MPI"
  ..@ tailRisk  : list()
```

The internal structure shows us that we have three slots; the first is the `@data` slot, the second is the `@statistics` slot, and the third is the `@tailRisk` slot.

How to print a portfolio data object

A nicely printed output of the same information can be obtained by using the generic `print()` function. Let us do this for the LPP2005 portfolio data object specified above.

```
> print(lppData)

Head/Tail Series Data:

GMT
                 SBI     SPI       LMI      MPI
2005-11-01 -0.061275 0.84146 -0.110888 0.154806
2005-11-02 -0.276201 0.25193 -0.117594 0.034288
2005-11-03 -0.115309 1.27073 -0.099246 1.050296
GMT
                 SBI      SPI      LMI       MPI
2007-04-09  0.000000  0.00000 -0.10324  0.817915
2007-04-10 -0.068900  0.63294 -0.00315 -0.142829
2007-04-11  0.030628 -0.10442 -0.00909 -0.099106

Statistics:

$mean
       SBI        SPI        LMI        MPI
4.0663e-05 8.4175e-02 5.5315e-03 5.9052e-02

$Cov
           SBI       SPI        LMI       MPI
SBI  0.0158996 -0.012741  0.0098039 -0.015888
SPI -0.0127414  0.584612 -0.0140747  0.411598
LMI  0.0098039 -0.014075  0.0149511 -0.023322
MPI -0.0158884  0.411598 -0.0233222  0.535033

$estimator
[1] "covEstimator"

$mu
       SBI        SPI        LMI        MPI
4.0663e-05 8.4175e-02 5.5315e-03 5.9052e-02

$Sigma
           SBI       SPI        LMI       MPI
SBI  0.0158996 -0.012741  0.0098039 -0.015888
SPI -0.0127414  0.584612 -0.0140747  0.411598
LMI  0.0098039 -0.014075  0.0149511 -0.023322
MPI -0.0158884  0.411598 -0.0233222  0.535033
```

The output displays the first and last three lines of the data set from the `@data` slot, and the sample mean and covariance estimates from the `@statistics` slot. Alternative or robust mean and covariance estimates, which are computed by the specified `covEstimator` function, are also printed. The name of the alternative estimator function has to be defined in the portfolio specification using the function `setEstimator()`. Note that the tail risk is only shown if it is not an empty list.

12.2 The Data Slot

The `@data` slot keeps the S4 time series object, the number of assets, and their names in a list.

```
Data Slot - List Elements:
series              S4 timeSeries object
nAssets             number of assets
names               names of the assets
```

Listing 12.1 The `@data` slot

The contents of the `@data` slot can be extracted with the help of the function `getData()`.

```
> Data <- portfolioData(lppData)
> getData(Data)[-1]

$nAssets
[1] 4

$names
[1] "SBI" "SPI" "LMI" "MPI"
```

Since the time series in the first list element is quite long, we have excluded it from being printed.

12.3 The Statistics Slot

The `@statistics` slot holds information on the mean and covariance matrix of the `timeSeries` in a list.

```
Statistics Slot - List Elements:
mean                sample mean estimate
Cov                 sample covariance estimate
estimator           name of alternative estimator function
mu                  alternative mean estimate
Sigma               alternative covariance estimate
```

Listing 12.2 The `@statistics` slot

To be more precise, the `@statistics` slot holds the sample mean, `$mean`, and sample covariance matrix, `$Cov`, and additionally alternative measures for these two statistical measures, e.g. a robust estimate for the mean, `$mu`, and for the covariance matrix, `$Sigma`. The name of the estimator function used for the mean and covariance estimation can be retrieved from the character variable `$estimator`. A list of alternative mean and covariance estimators is given in chapter 4.

The contents of the `@statistics` slot can be extracted with the help of the function `getStatistics()`.

```
> getStatistics(Data)

$mean
       SBI        SPI        LMI        MPI
4.0663e-05 8.4175e-02 5.5315e-03 5.9052e-02

$Cov
           SBI       SPI        LMI       MPI
SBI  0.0158996 -0.012741  0.0098039 -0.015888
SPI -0.0127414  0.584612 -0.0140747  0.411598
LMI  0.0098039 -0.014075  0.0149511 -0.023322
MPI -0.0158884  0.411598 -0.0233222  0.535033

$estimator
[1] "covEstimator"

$mu
       SBI        SPI        LMI        MPI
4.0663e-05 8.4175e-02 5.5315e-03 5.9052e-02

$Sigma
           SBI       SPI        LMI       MPI
SBI  0.0158996 -0.012741  0.0098039 -0.015888
SPI -0.0127414  0.584612 -0.0140747  0.411598
```

```
LMI  0.0098039 -0.014075  0.0149511 -0.023322
MPI -0.0158884  0.411598 -0.0233222  0.535033
```

Chapter 13
S4 Portfolio Constraints Class

Required R package(s):

```
> library(fPortfolio)
```

Constraints define restrictions and boundary conditions on the weights and functional measures, depending on, or derived from, the portfolio weights. Constraints are defined by a character string or a vector of character strings. The formal style of these strings can be used like a language to express lower and upper bounds on the ranges of weights that have to be satisfied for box, group, covariance risk budgets and general non-linear constraint settings.

In this chapter we introduce the rules to express constraints as strings and introduce the functions used to create a summary for all the constraints. The function `portfolioConstraints()` creates such a summary, which is an S4 object of class `fPFOLIOCON`. We describe each of the slots that hold the constraint strings, the box and group constraints, the quadratic covariance risk budget constraints, and general non-linear constraints.

13.1 Class Representation

In Rmetrics, portfolio constraints are represented by S4 `fPFOLIOCON` objects. These objects are used to represent all constraints settings for portfolio optimization. The function `portfolioConstraints()` creates the default settings and reports on all constraints in a compact from.

An S4 object of class `fPFOLIOCON` has the following representation:

```
> showClass("fPFOLIOCON")

Class "fPFOLIOCON" [package "fPortfolio"]

Slots:

Name:   stringConstraints    minWConstraints    maxWConstraints
Class:          character            numeric            numeric

Name:   eqsumWConstraints minsumWConstraints maxsumWConstraints
Class:             matrix             matrix             matrix

Name:     minBConstraints    maxBConstraints   listFConstraints
Class:            numeric            numeric               list

Name:     minFConstraints    maxFConstraints
Class:            numeric            numeric
```

How to create a portfolio constraints object

The function `portfolioConstraints()` takes as arguments

```
> args(portfolioConstraints)

function (data, spec = portfolioSpec(), constraints = "LongOnly",
    ...)
NULL
```

the data set of assets as an object of `timeSeries`, the portfolio specification object `spec`, and the `constraints`, a vector of strings. The result is returned as an object of class `fPFOLIOCON`. The data set must be explicitly defined, whereas the last two arguments have default settings. The `spec=portfolioSpec()` argument takes as its default setting the mean-variance portfolio specification with long-only constraints, i.e. `constraints="LongOnly"`. In general, the argument `constraints` takes a character vector of constraints strings. The string vector can be composed from the following individual constraints:

```
Argument: constraints
Values:
```

```
"LongOnly"              long-only constraints [0,1]
"Short"                 unlimited short selling, [-Inf,Inf]

"minW[<...>]=<...>      lower box bounds
"maxw[<...>]=<...>      upper box bounds

"minsumW[<...>]=<...>   lower group bounds
"maxsumW[<...>]=<...>   upper group bounds

"minB[<...>]=<...>      lower covariance risk budget bounds
"maxB[<...>]=<...>      upper covariance risk budget bounds

"listF=list(<...>)"     list of non-linear functions
"minf[<...>]=<...>"     lower non-linear function bounds
"maxf[<...>]=<...>"     upper covariance risk budget
```

Listing 13.1 Arguments of the `portfolioConstraints()` function

The returned S4 object of class `fPFOLIOCON` has eleven slots, the `@stringConstraints` slot (a character value or vector of constraints), and ten further slots for the individual constraints components. The following example creates the `"LongOnly"` default settings for the returns from the three Swiss assets from the `LPP2005` Swiss Pension Fund Benchmark[1].

```
> Data <- 100 * LPP2005.RET[, 1:3]
> Spec <- portfolioSpec()
> setTargetReturn(Spec) <- mean(Data)
> Constraints <- "LongOnly"
> defaultConstraints <- portfolioConstraints(Data, Spec, Constraints)
```

How to display the structure of a portfolio constraints object

To explore the whole structure of an S4 `fPFOLIOCON`, type

```
> str(defaultConstraints, width = 65, strict.width = "cut")

Formal class 'fPFOLIOCON' [package "fPortfolio"] with 11 slots
  ..@ stringConstraints : chr "LongOnly"
  ..@ minWConstraints   : Named num [1:3] 0 0 0
  .. ..- attr(*, "names")= chr [1:3] "SBI" "SPI" "SII"
  ..@ maxWConstraints   : Named num [1:3] 1 1 1
```

[1] Because the target return is not defined in the default `spec` settings, we set it to the grand mean of the assets data set.

```
.. ..- attr(*, "names")= chr [1:3] "SBI" "SPI" "SII"
..@ eqsumWConstraints : num [1:2, 1:4] 3.60e-02 -1.00 4.07e-0..
.. ..- attr(*, "dimnames")=List of 2
.. .. ..$ : chr [1:2] "Return" "Budget"
.. .. ..$ : chr [1:4] "ceq" "SBI" "SPI" "SII"
..@ minsumWConstraints: logi [1, 1] NA
..@ maxsumWConstraints: logi [1, 1] NA
..@ minBConstraints   : Named num [1:3] -Inf -Inf -Inf
.. ..- attr(*, "names")= chr [1:3] "SBI" "SPI" "SII"
..@ maxBConstraints   : Named num [1:3] 1 1 1
.. ..- attr(*, "names")= chr [1:3] "SBI" "SPI" "SII"
..@ listFConstraints  : list()
..@ minFConstraints   : num(0)
..@ maxFConstraints   : num(0)
```

How to print a portfolio constraints object

A nicely printed output of the same information can be obtained using the generic `print()` function. Let us do this for the above default mean-CVaR portfolio[2] .

```
> print(defaultConstraints)

Title:
 Portfolio Constraints

Lower/Upper Bounds:
      SBI SPI SII
Lower   0   0   0
Upper   1   1   1

Equal Matrix Constraints:
             ceq         SBI       SPI       SII
Return  0.036037  4.0663e-05  0.084175  0.023894
Budget -1.000000 -1.0000e+00 -1.000000 -1.000000
```

13.2 Long-Only Constraint String

Let us consider the settings for long-only constraints.

[2] Non-defined constraints are not printed.

```
Constraints Settings:
"LongOnly"          long-only constraints, sets lower and upper bounds of
                    weights as box constraints
```

Listing 13.2 Long-only constraints string

The long-only constraints generate the following set of weights:

```
> longConstraints <- "LongOnly"
> portfolioConstraints(Data, Spec, longConstraints)

Title:
 Portfolio Constraints

Lower/Upper Bounds:
      SBI SPI SII
Lower   0   0   0
Upper   1   1   1

Equal Matrix Constraints:
             ceq         SBI       SPI       SII
Return  0.036037  4.0663e-05  0.084175  0.023894
Budget -1.000000 -1.0000e+00 -1.000000 -1.000000
```

This is also the default setting for the constraints if not otherwise defined. Alternatively, you can use `longConstraints=NULL`. Long-only positions reflect the fact that all weights are allowed to be between zero and one. Do not confuse this setting with box constraints, where the weights are restricted by arbitrary negative and/or positive lower and upper bounds. The output from the function `portfolioConstraints()` generates

1. the lower and upper bounds for each asset between 0 and 1 (100%),
2. two equal group constraints, where the first sums up to the target return, and the second sums up to 1 (i.e. we are fully invested), and
3. the lower and upper bounds for the covariance risk budgets between $-\infty$ and 1.

13.3 Unlimited Short Selling Constraint String

Let us consider the settings for unlimited short constraints.

```
Constraints Settings:
"Short"             short constraints, sets lower and upper bounds of
                    weights as box constraints ranging between minus
                    and plus infinity for all assets
```

Listing 13.3 Short constraint string

The unlimited short constraints generate the following set of weights:

```
> shortConstraints <- "Short"
> portfolioConstraints(Data, Spec, shortConstraints)

Title:
 Portfolio Constraints

Lower/Upper Bounds:
       SBI  SPI  SII
Lower -Inf -Inf -Inf
Upper  Inf  Inf  Inf

Equal Matrix Constraints:
             ceq         SBI       SPI       SII
Return  0.036037  4.0663e-05  0.084175  0.023894
Budget -1.000000 -1.0000e+00 -1.000000 -1.000000
```

The setting allows for unlimited negative and positive weights. In this case, the mean-variance portfolio optimization problem can be solved analytically[3].

13.4 Box Constraint Strings

For arbitrary short-selling, use box constraints and set the lower bounds to negative values for those assets that are allowed for short selling. Weight-

[3] In order to solve the unlimited short-selling portfolio analytically, you have to set `setSolver(Spec)<-solveRshortExact`.

constrained portfolios, where the weights are limited by lower and upper bounds, are specified by the two character strings `minW` and `maxW`.

```
Constraints Settings:
"minW"              lower bounds of weights for box constraints
"maxW"              upper bounds of weights for box constraints
```

Listing 13.4 Constraints setting for box constraints

These character strings have to be used with indices between one and the number of assets, and appropriate values. If these values are all positive and between zero and one then we have constrained long-only portfolios. If they are allowed to become negative, then we have constrained (or limited) short portfolios.

Constraints are given as a vector composed of individual strings. For example, the constraints settings with the following strings form a box-constrained portfolio for a set of three assets:

```
> box.1 <- "minW[1:3] = 0.1"
> box.2 <- "maxW[c(1, 3)] = c(0.5, 0.6)"
> box.3 <- "maxW[2] = 0.4"
> boxConstraints <- c(box.1, box.2, box.3)
> boxConstraints

[1] "minW[1:3] = 0.1"             "maxW[c(1, 3)] = c(0.5, 0.6)"
[3] "maxW[2] = 0.4"
```

and the portfolio constraints become

```
> portfolioConstraints(Data, Spec, boxConstraints)

Title:
 Portfolio Constraints

Lower/Upper Bounds:
      SBI SPI SII
Lower 0.1 0.1 0.1
Upper 0.5 0.4 0.6

Equal Matrix Constraints:
              ceq         SBI       SPI       SII
Return  0.036037  4.0663e-05  0.084175  0.023894
Budget -1.000000 -1.0000e+00 -1.000000 -1.000000
```

The constraints tell us that we want to invest at least 10% in each asset, and no more than 50% in the first asset, a maximum of 40% in the second asset, and a maximum of 60% in number 3. Notice that we can repeat constraints settings, as for `maxW` in the example above. In this case, the previous settings for the assets will be overwritten if they are multiply defined. The variable nAssets is automatically recognized and the value of the total number of assets will be assigned.

13.5 Group Constraint Strings

Group constraints define the value of the total weight of a group of assets or lower and upper bounds on such groups. For this we can make use of the following strings:

```
Constraints Settings:
"eqsumW"            equality group constraints
"minsumW"           lower bounds group constraints
"maxsumw"           upper bounds group constraints
```

Listing 13.5 Constraints setting for group constraints

Here, `"eqsumW"` sets the total amount of an investment in a group of assets to a fixed value, whereas `"minsumW"` and `"maxsumW"` set lower and upper bounds, e.g.

```
> group.1 <- "eqsumW[c(\"SPI\", \"SII\")]=0.6"
> group.2 <- "minsumW[c(2, 3)]=0.2"
> group.3 <- "maxsumW[1:nAssets]=0.7"
> groupConstraints <- c(group.1, group.2, group.3)
> groupConstraints

[1] "eqsumW[c(\"SPI\", \"SII\")]=0.6" "minsumW[c(2, 3)]=0.2"
[3] "maxsumW[1:nAssets]=0.7"
```

The first string means that we should invest exactly 60% of our money in the group consisting of assets one and three. The second string tells us that we should invest at least 20% in assets number two and three, and the third means that we want to invest no more than 70% of our money in assets one and two.

The portfolio constraints become

```
> portfolioConstraints(Data, Spec, groupConstraints)

Title:
 Portfolio Constraints

Lower/Upper Bounds:
      SBI SPI SII
Lower   0   0   0
Upper   1   1   1

Equal Matrix Constraints:
              ceq         SBI       SPI       SII
Return  0.036037  4.0663e-05  0.084175  0.023894
Budget -1.000000 -1.0000e+00 -1.000000 -1.000000
eqsumW  0.600000  0.0000e+00  1.000000  1.000000

Lower Matrix Constraints:
      avec SBI SPI SII
lower  0.2   0   1   1

Upper Matrix Constraints:
      avec SBI SPI SII
upper  0.7   1   1   1
```

Notice that the above conditions are reflected in the output of the function `portfolioConstraints()`. For example, the `"eqsumW"` constraint is shown in the `"eqsumW"` row of the `Equal Matrix Constraints` table. The value is give in the `ceq` column, and which assets this group constraint applies to is denoted by either 0 or 1. In this case, we can see that the constraint applies to the group comprising the SBI and the SII.

13.6 Covariance Risk Budget Constraint Strings

By default, risk budgets are not included in the portfolio optimization. Covariance risk budgets have to be added explicitly, and have the following form:

```
Constraints Settings:
"minB"              lower bounds of the covariance risk budgets
"maxB"              upper bounds of the covariance risk budgets
```

Listing 13.6 Covariance risk budget constraints

`"minB"` and `"maxB"` have to be assigned to the lower and upper bounds of the covariance risk budgets. The assignments must be given to all assets. The following shows an example of covariance risk budget constraints for a portfolio with 3 assets:

```
> budget.1 <- "minB[1:nAssets]=-Inf"
> budget.2 <- "maxB[c(1, 2:nAssets)]=c(0.5, rep(0.6, times=2))"
> budgetConstraints <- c(budget.1, budget.2)
> budgetConstraints

[1] "minB[1:nAssets]=-Inf"
[2] "maxB[c(1, 2:nAssets)]=c(0.5, rep(0.6, times=2))"
```

and the portfolio constraints become

```
> portfolioConstraints(Data, Spec, budgetConstraints)

Title:
 Portfolio Constraints

Lower/Upper Bounds:
      SBI SPI SII
Lower   0   0   0
Upper   1   1   1

Equal Matrix Constraints:
             ceq         SBI       SPI       SII
Return  0.036037  4.0663e-05  0.084175  0.023894
Budget -1.000000 -1.0000e+00 -1.000000 -1.000000

Lower/Upper Cov Risk Budget Bounds:
       SBI  SPI  SII
Lower -Inf -Inf -Inf
Upper  0.5  0.6  0.6
```

Again, the variable nAssets is automatically recognized and the value of the total number of assets, here 3, will be assigned.

Risk budget constraints will enforce diversification at the expense of return generation. The resulting portfolios will thus lie below the unconstrained efficient frontier.

Note that adding risk budget constraints will modify the optimization problem since these constraints are quadratic, unlike the other constraints considered so far. This requires optimizers which can handle non-linear constraints[4].

13.7 Non-Linear Weight Constraint Strings

We can also make use of non-linear functional constraints. This can be achieved by using the following strings:

```
Constraints Settings:
"listF"            list of non-linear functions
"minF"             lower bounds of non-linear functions
"maxF"             upper bounds of non-linear functions
```

Listing 13.7 Non-linear weight constraints

If, for example, we want to constrain our portfolio by a maximum drawdown, we first have to write a function to compute the maximum drawdown. We call the function `maxdd()`, and define it such that the data `x` will be passed to the function `min(drawdowns())`.

```
> maxdd <- function(x, ...) min(drawdowns(x, ...))
```

Next, we have to decide on the lower and upper bounds.

```
> nonlin.1 <- "listF=list(maxdd=maxdd)"
> nonlin.2 <- "minF=-0.04"
> nonlin.3 <- "maxF=0"
> nonlinConstraints <- c(nonlin.1, nonlin.2, nonlin.3)
> nonlinConstraints
```

[4] Portfolio Solvers for quadratic constraints will be presented in the ebook *Advanced Portfolio Optimization with R/Rmetrics*.

```
[1] "listF=list(maxdd=maxdd)" "minF=-0.04"
[3] "maxF=0"
```

Then portfolio constraints become

```
> portfolioConstraints(Data, Spec, nonlinConstraints)

Title:
 Portfolio Constraints

Lower/Upper Bounds:
      SBI SPI SII
Lower   0   0   0
Upper   1   1   1

Equal Matrix Constraints:
             ceq         SBI       SPI       SII
Return  0.036037  4.0663e-05  0.084175  0.023894
Budget -1.000000 -1.0000e+00 -1.000000 -1.000000

Non-Linear Function Constraints:
      maxdd
Lower -0.04
Upper  0.00
```

These settings can be interpreted in the following way: `"listF"` lists all the function names of the non-linear constraints function, and `"minF"` and `"maxF"` hold the values of their lower and upper bounds. Notice that if there is more than one non-linear constraints function, then `"listF"` holds the names of all functions, and `"minF"` and `"maxF"` are composed as vectors of the same length as the number of non-linear functions[5].

13.8 Case study: How To Construct Complex Portfolio Constraints

Box, group, risk budget and non-linear constraints can now be combined. First, let us create a larger data set than in the previous examples, using

[5] Portfolio Solvers for non-linear constraints will be presented in the ebook *Advance Portfolio Optimization with R/Rmetrics*.

the "SBI", "SPI", "SII", "LMI", "MPI" and "ALT" columns from the LPP2005 returns data set.

```
> # create data set
> Data <- 100 * LPP2005.RET[,1:6]
> # create default portfolio spec
> Spec <- portfolioSpec()
> # set target return to grand mean of the data
> setTargetReturn(Spec) <- mean(Data)
```

Now that we have created our data set and portfolio specification, we can turn our attention to the individual constraint strings. These can all be created in the same way as before, but bear in mind that our data set now consists of six assets instead of three.

```
> box.1 <- "minW[1:6] = 0.1"
> box.2 <- "maxW[c(1:3, 5)] = c(rep(0.5, 3), 0.6)"
> box.3 <- "maxW[4] = 0.4"
> # combine individual strings
> boxConstraints <- c(box.1, box.2, box.3)
> boxConstraints

[1] "minW[1:6] = 0.1"
[2] "maxW[c(1:3, 5)] = c(rep(0.5, 3), 0.6)"
[3] "maxW[4] = 0.4"
```

```
> group.1 <- "eqsumW[c(2, 3, 5)]=0.2"
> group.2 <- "minsumW[c(2, 4)]=0.2"
> group.3 <- "maxsumW[c(4:6, 2)]=0.6"
> # combine individual strings
> groupConstraints <- c(group.1, group.2, group.3)
> groupConstraints

[1] "eqsumW[c(2, 3, 5)]=0.2" "minsumW[c(2, 4)]=0.2"
[3] "maxsumW[c(4:6, 2)]=0.6"
```

```
> budget.1 <- "minB[1:nAssets]=-Inf"
> budget.2 <- "maxB[c(1:3, 4:nAssets)]=rep(c(0.5, 0.6), each=3)"
> # combine individual strings
> budgetConstraints <- c(budget.1, budget.2)
> budgetConstraints

[1] "minB[1:nAssets]=-Inf"
[2] "maxB[c(1:3, 4:nAssets)]=rep(c(0.5, 0.6), each=3)"
```

The portfolio should be constrained by a maximum drawdown; therefore, we redefine the same function as used above to compute this:

```
> # create function to compute maximum drawdown
> maxdd <- function(x,...) min(drawdown(x,...))

> nonlin.1 <- "listF=list(maxdd=maxdd)"
> nonlin.2 <- "minF=-0.04"
> nonlin.3 <- "maxF=0"
> # combine individual strings
> nonlinConstraints <- c(nonlin.1, nonlin.2, nonlin.3)
> nonlinConstraints

[1] "listF=list(maxdd=maxdd)" "minF=-0.04"
[3] "maxF=0"
```

The constraint string can now be constructed from all the individual constraints in the following manner:

```
> Constraints <- c(boxConstraints, groupConstraints,
     budgetConstraints, nonlinConstraints)
> Constraints

 [1] "minW[1:6] = 0.1"
 [2] "maxW[c(1:3, 5)] = c(rep(0.5, 3), 0.6)"
 [3] "maxW[4] = 0.4"
 [4] "eqsumW[c(2, 3, 5)]=0.2"
 [5] "minsumW[c(2, 4)]=0.2"
 [6] "maxsumW[c(4:6, 2)]=0.6"
 [7] "minB[1:nAssets]=-Inf"
 [8] "maxB[c(1:3, 4:nAssets)]=rep(c(0.5, 0.6), each=3)"
 [9] "listF=list(maxdd=maxdd)"
[10] "minF=-0.04"
[11] "maxF=0"
```

Finally, we can now create the portfolio constraints by passing the complex constraints as an argument of the `portfolioConstraints()` function:

```
> portfolioConstraints(Data, Spec, Constraints)

Title:
 Portfolio Constraints

Lower/Upper Bounds:
      SBI SPI SII LMI MPI ALT
```

```
Lower 0.1 0.1 0.1 0.1 0.1 0.1
Upper 0.5 0.5 0.5 0.4 0.6 1.0

Equal Matrix Constraints:
             ceq         SBI       SPI       SII        LMI       MPI
Return  0.043077  4.0663e-05  0.084175  0.023894  0.0055315  0.059052
Budget -1.000000 -1.0000e+00 -1.000000 -1.000000 -1.0000000 -1.000000
eqsumW  0.200000  0.0000e+00  1.000000  1.000000  0.0000000  1.000000
             ALT
Return  0.085768
Budget -1.000000
eqsumW  0.000000

Lower Matrix Constraints:
      avec SBI SPI SII LMI MPI ALT
lower  0.2   0   1   0   1   0   0

Upper Matrix Constraints:
      avec SBI SPI SII LMI MPI ALT
upper  0.6   0   1   0   1   1   1

Lower/Upper Cov Risk Budget Bounds:
       SBI  SPI  SII  LMI  MPI  ALT
Lower -Inf -Inf -Inf -Inf -Inf -Inf
Upper  0.5  0.5  0.5  0.6  0.6  0.6

Non-Linear Function Constraints:
      maxdd
Lower -0.04
Upper  0.00
```

Chapter 14
Portfolio Functions

Required R package(s):

```
> library(fPortfolio)
```

After we have loaded the data, specified the portfolio settings, and defined the constraints, we are ready to optimize the portfolio. The portfolio optimization functions take the data, the specifications and the constraints as inputs. The returned value is an S4 object of class `fPORTFOLIO` which can be used to print reports and/or to display graphs.

The usage of the functions is described in detail in Part IV for mean-variance portfolios and in Part V for mean-CVaR portfolios.

14.1 S4 Class Representation

In Rmetrics, portfolios are represented by S4 `fPORTFOLIO` objects. An S4 object of class `fPORTFOLIO` has the following representation

```
> showClass("fPORTFOLIO")

Class "fPORTFOLIO"

Slots:

Name:         call        data        spec constraints   portfolio
Class:        call fPFOLIODATA fPFOLIOSPEC  fPFOLIOCON  fPFOLIOVAL
```

```
Name:        title description
Class:   character   character
```

These objects are returned by the computation and optimization of any portfolio.

Functions to compute and optimize portfolios

The functions to compute or optimize a single portfolio are:

```
Portfolio Functions:
feasiblePortfolio       returns a feasible portfolio given the
                        vector of portfolio weights

efficientPortfolio      returns the portfolio with the lowest
                        risk for a given target return
maxratioPortfolio       returns the portfolio with the highest
                        return/risk ratio
tangencyPortfolio       synonym for maxratioPortfolio
minriskPortfolio        returns a portfolio with the lowest
                        risk at all
minvariancePortfolio    synonym for minriskPortfolio
maxreturnPortfolio      returns the portfolio with the highest
                        return for a given target risk

portfolioFrontier       computes portfolios on the efficient frontier
                        and/or on the minimum covariance locus.
```

Listing 14.1 Functions for computing and optimizing portfolios

If the weights for the function `feasiblePortfolio()` are not specified in the portfolio specification, then the function assumes that the portfolio should be an equal-weights portfolio.

The portfolio functions are called with the following arguments

```
> args(feasiblePortfolio)

function (data, spec = portfolioSpec(), constraints = "LongOnly")
NULL
```

The portfolio specification and the constraints specification have defaults, which are those for a mean-variance portfolio with long-only constraints.

The returned values are those from an S4 object of class `fPORTFOLIO` with the following slots:

```
Returned Portfolio Values:
call                function call
data                data, an object of class fPFOLIODATA
spec                specification, an object of class fPFOLIOSPEC
constraints         constraints, an object of class fPFOLIOCON
portfolio           the portfolio result as returned by the
                    implied solver
title               an optional title, by default the portfolio
                    function name
description         an optional description, by default time
                    and name of the user
```

Listing 14.2 fPORTFOLIO slots

How to display the structure of a portfolio object

The structure of an S4 object of class `fPORTFOLIO` is quite comprehensive, because it contains the previously presented data, spec, and constraints objects. To explore the whole structure of an S4 `fPFOLIOCON` object, type:

```
> tgPortfolio <- tangencyPortfolio(100 * LPP2005.RET[, 1:6])
> str(tgPortfolio, width = 65, strict.width = "cut")
```

```
Formal class 'fPORTFOLIO' [package "fPortfolio"] with 7 slots
  ..@ call       : language maxratioPortfolio(data = data, spec..
  ..@ data       :Formal class 'fPFOLIODATA' [package "fPortfol..
  .. .. ..@ data      :List of 3
  .. .. .. ..$ series :Time Series:
 Name:               object
Data Matrix:
 Dimension:          377 6
 Column Names:       SBI SPI SII LMI MPI ALT
 Row Names:          2005-11-01  ...  2007-04-11
Positions:
 Start:              2005-11-01
 End:                2007-04-11
With:
 Format:             %Y-%m-%d
 FinCenter:          GMT
 Units:              SBI SPI SII LMI MPI ALT
```

```
Title:              Time Series Object
Documentation:      Tue Jan 20 17:49:06 2009 by user:
 .. .. .. ..$ nAssets: int 6
 .. .. .. ..$ names  : chr [1:6] "SBI" "SPI" "SII" "LMI" ...
 .. .. ..@ statistics:List of 5
 .. .. .. ..$ mean     : Named num [1:6] 4.07e-05 8.42e-02 2.3..
 .. .. .. .. ..- attr(*, "names")= chr [1:6] "SBI" "SPI" "SII"..
 .. .. .. ..$ Cov      : num [1:6, 1:6] 0.0159 -0.0127 0.0018 ..
 .. .. .. .. ..- attr(*, "dimnames")=List of 2
 .. .. .. .. .. ..$ : chr [1:6] "SBI" "SPI" "SII" "LMI" ...
 .. .. .. .. .. ..$ : chr [1:6] "SBI" "SPI" "SII" "LMI" ...
 .. .. .. ..$ estimator: chr "covEstimator"
 .. .. .. ..$ mu       : Named num [1:6] 4.07e-05 8.42e-02 2.3..
 .. .. .. .. ..- attr(*, "names")= chr [1:6] "SBI" "SPI" "SII"..
 .. .. .. ..$ Sigma    : num [1:6, 1:6] 0.0159 -0.0127 0.0018 ..
 .. .. .. .. ..- attr(*, "dimnames")=List of 2
 .. .. .. .. .. ..$ : chr [1:6] "SBI" "SPI" "SII" "LMI" ...
 .. .. .. .. .. ..$ : chr [1:6] "SBI" "SPI" "SII" "LMI" ...
 .. .. ..@ tailRisk  : list()
 ..@ spec       :Formal class 'fPFOLIOSPEC' [package "fPortfol..
 .. .. ..@ model    :List of 5
 .. .. .. ..$ type     : chr "MV"
 .. .. .. ..$ optimize : chr "minRisk"
 .. .. .. ..$ estimator: chr "covEstimator"
 .. .. .. ..$ tailRisk : list()
 .. .. .. ..$ params   :List of 2
 .. .. .. .. ..$ alpha: num 0.05
 .. .. .. .. ..$ a    : num 1
 .. .. ..@ portfolio:List of 6
 .. .. .. ..$ weights        : atomic [1:6] 0 0.000482 0.18244..
 .. .. .. .. ..- attr(*, "invest")= num 1
 .. .. .. ..$ targetReturn   : logi NA
 .. .. .. ..$ targetRisk     : logi NA
 .. .. .. ..$ riskFreeRate   : num 0
 .. .. .. ..$ nFrontierPoints: num 50
 .. .. .. ..$ status         : int 0
 .. .. ..@ optim    :List of 5
 .. .. .. ..$ solver   : chr "solveRquadprog"
 .. .. .. ..$ objective: NULL
 .. .. .. ..$ options  :List of 1
 .. .. .. .. ..$ meq: num 2
 .. .. .. ..$ control  : list()
 .. .. .. ..$ trace    : logi FALSE
 .. .. ..@ messages : list()
 ..@ constraints:Formal class 'fPFOLIOCON' [package "fPortfoli..
 .. .. ..@ stringConstraints : chr "LongOnly"
```

```
.. .. ..@ minWConstraints   : Named num [1:6] 0 0 0 0 0 0
.. .. .. ..- attr(*, "names")= chr [1:6] "SBI" "SPI" "SII" "L..
.. .. ..@ maxWConstraints   : Named num [1:6] 1 1 1 1 1 1
.. .. .. ..- attr(*, "names")= chr [1:6] "SBI" "SPI" "SII" "L..
.. .. ..@ eqsumWConstraints : num [1:2, 1:7] NA -1.00 4.07e-0..
.. .. .. ..- attr(*, "dimnames")=List of 2
.. .. .. .. ..$ : chr [1:2] "Return" "Budget"
.. .. .. .. ..$ : chr [1:7] "ceq" "SBI" "SPI" "SII" ...
.. .. ..@ minsumWConstraints: logi [1, 1] NA
.. .. ..@ maxsumWConstraints: logi [1, 1] NA
.. .. ..@ minBConstraints   : Named num [1:6] -Inf -Inf -Inf ..
.. .. .. ..- attr(*, "names")= chr [1:6] "SBI" "SPI" "SII" "L..
.. .. ..@ maxBConstraints   : Named num [1:6] 1 1 1 1 1 1
.. .. .. ..- attr(*, "names")= chr [1:6] "SBI" "SPI" "SII" "L..
.. .. ..@ listFConstraints  : list()
.. .. ..@ minFConstraints   : num(0)
.. .. ..@ maxFConstraints   : num(0)
..@ portfolio  :Formal class 'fPFOLIOVAL' [package "fPortfoli..
.. .. ..@ portfolio:List of 6
.. .. .. ..$ weights       : Named num [1:6] 0 0.000482 0.182..
.. .. .. .. ..- attr(*, "names")= chr [1:6] "SBI" "SPI" "SII"..
.. .. .. ..$ covRiskBudgets: Named num [1:6] 0 0.00143 0.1538..
.. .. .. .. ..- attr(*, "names")= chr [1:6] "SBI" "SPI" "SII"..
.. .. .. ..$ targetReturn  : Named num [1:2] 0.0283 0.0283
.. .. .. .. ..- attr(*, "names")= chr [1:2] "mean" "mu"
.. .. .. ..$ targetRisk    : Named num [1:4] 0.153 0.153 0.31..
.. .. .. .. ..- attr(*, "names")= chr [1:4] "Cov" "Sigma" "CV..
.. .. .. ..$ targetAlpha   : num 0.05
.. .. .. ..$ status        : int 0
.. .. ..@ messages : list()
..@ title      : chr "Tangency Portfolio"
..@ description: chr "Tue Apr  7 17:32:03 2009 by user: Rmetr..
```

The above shows the structure of a mean-variance tangency portfolio with long-only constraints.

How to print a portfolio object

A compact and nicely formatted printout of the most important information can be obtained by using the generic `print()` function. Let us do this for the mean-variance tangency portfolio that we optimized above.

```
> print(tgPortfolio)
```

```
Title:
 MV Tangency Portfolio
 Estimator:         covEstimator
 Solver:            solveRquadprog
 Optimize:          minRisk
 Constraints:       LongOnly

Portfolio Weights:
   SBI    SPI    SII    LMI    MPI    ALT
0.0000 0.0005 0.1824 0.5751 0.0000 0.2420

Covariance Risk Budgets:
   SBI    SPI    SII    LMI    MPI    ALT
0.0000 0.0014 0.1538 0.1121 0.0000 0.7326

Target Return and Risks:
   SBI    SPI    SII    LMI    MPI    ALT
0.0000 0.0014 0.1538 0.1121 0.0000 0.7326

Description:
 Tue Apr  7 17:32:03 2009 by user: Rmetrics
```

Part IV
Mean-Variance Portfolios

Introduction

Modern portfolio theory proposes how rational investors use diversification to optimize their portfolio(s) of risky assets. The basic concepts of the theory go back to Markowitz (1952)'s idea of diversification and the efficient portfolio frontier. His model considers asset returns as a random variable, and models a portfolio as a weighted combination of assets. Being a random variable, a portfolio's returns have an expected mean and variance. In this model, return and risk are estimated by the sample mean and the sample standard deviation of the asset returns.

In Chapter 15 we briefly describe the mean-variance portfolio theory and present its solution when no restrictions are set on the weights. This is the unlimited short selling case where an analytically closed form solution is possible. We derive the feasible set and the efficient frontier. Two special points on the frontier are discussed in detail: the minimum variance portfolio and the tangency portfolio. In the case of constraints we discuss the solutions for box and group constraints defining linear constraints. The case of the maximum return mean-variance portfolio and in the case of additional covariance risk budget constaints we have a new situation where quadratic forms of the constraints are becoming active.

In Chapter 16 we show how to specify a mean-variance portfolio which describes an S4 class in Rmetrics. We discuss the slots representing the portfolio data, the portfolio specification, and the portfolio constraints.

In Chapter 17 we show in several examples how to compute an optimize several type of mean-variance portfolios. These include feasible portfolios, portfolios with the lowest risk for a given return, the global minimum variance portfolio, the tangency portfolio, and portfolios with the highest return for a given risk. We also show how to handle non-linear constraints, like the maximum drawdown constrained portfolio, or the 130/30 extension strategy constrained portfolio.

In Chapter 18 we present how to compute and graphically display the whole efficient frontier of a portfolio. As special examples we consider the portfolio with long-only constraints, the unlimited short selling portfolio, box and/or group constrained portfolios, and covariance risk budget constrained portfolios. In addition we show how to create different reward/risk views on the efficient frontier.

In Chapter 19 we present a case study.

In Chapter 20 we discuss how to robustify portfolios using alternative covariance estimators. The standard estimator used in the mean-variance portfolio optimization is the sample covariance estimator. Here we show the influence on the portfolio if we use robust and related statistical estimators. These include the minimum covariance determinant estimator, the minimum volume ellipsoid estimator, the orthogonalized Gnanadesikan-Kettenring estimator for large covariance matrixes, and the shrinkage covariance estimator.

Chapter 15
Markowitz Portfolio Theory

Required R package(s):

```
> library(fPortfolio)
```

In this chapter we define the original mean-variance portfolio optimization problem and several related problems. The problem of minimizing the covariance risk for a given target return with optional box and group constraints is a quadratic programming problem with linear constraints. We call it QP1 or *minimum risk mean-variance portfolio*. The opposite case, fixing the risk and maximizing the return, has a linear opbjective function with quadratic constraints. We call this programming problem QP2 or *maximum return mean-variance portfolio* problem. The QP2 problem is much more complex than QP1. If we have even more complex constraints, i.e. nonlinear constraints, we need a new class of solvers, which we call NL1, or *non-linear constrained portfolio* problem. This allows us to handle the case of linear and quadratic objective functions with non-linear constraints. In all three cases we speak of Markowitz' portfolio optimization problem, although they require different classes of solvers with increasing complexity.

15.1 The Minimum Risk Mean-Variance Portfolio

Following Markowitz (1952) we define the problem of portfolio selection as follows:

$$
\begin{aligned}
&\min_{w} \quad w^T \ \hat{\Sigma} \ w \\
s.t. & \\
&w^T \hat{\mu} = \bar{r} \\
&w^T 1 = 1
\end{aligned}
$$

The formula expresses that we minimize the variance-covariance risk $\overline{\sigma}^2 = w^T \ \hat{\Sigma} \ w$, where the matrix $\hat{\Sigma}$ is an estimate of the covariance of the assets. The vector w denotes the individual investments subject to the condition $w^T 1 = 1$ that the available capital is fully invested. The expected or target return $\bar{r}$ is expressed by the condition $w^T \hat{\mu} = \bar{r}$, where the p-dimensional vector $\hat{\mu}$ estimates the expected mean of the assets.

Markowitz' portfolio model[1] has a unique solution:

$$
w^\star = \hat{\mu} w_0^\star + w_1^\star
$$

where

$$
\begin{aligned}
w_0^\star &= \frac{1}{\Delta}(B\hat{\Sigma}^{-1}\hat{\mu} - C\hat{\Sigma}^{-1}1) \\
w_1^\star &= \frac{1}{\Delta}(C\hat{\Sigma}^{-1}\hat{\mu} - A\hat{\Sigma}^{-1}1) \\
\Delta &= AB - C^2
\end{aligned}
$$

with

$$
\begin{aligned}
A &= \hat{\mu}^T\hat{\Sigma}^{-1}\hat{\mu} \\
B &= 1^T\hat{\Sigma}^{-1}1 \\
C &= 1^T\hat{\Sigma}^{-1}\hat{\mu} \ .
\end{aligned}
$$

[1] For a detailed listing of Markowitz' assumptions and technical conditions underlying his approach we refer to Vanini & Vignola (2001).

15.2 The Feasible Set and the Efficient Frontier

The corresponding standard deviation $\overline{\sigma}$ for the optimal portfolio with weights $w^\star$ is

$$\overline{\sigma} = \sqrt{\frac{1}{\Delta}(\hat{\mu}B - 2\hat{\mu}C + A)}$$
$$\overline{r} = w^T \ \hat{\mu} \ .$$

The locus of this set in the $\{\overline{\sigma}, \overline{r}\}$-space are hyperbolas. The set inside the hyperbola is the *feasible set* of mean/standard deviation portfolios, and the borders are the *efficient frontier* (upper border), and the minimum variance locus (lower border). Here, $r_\star$ is the return of the minimum variance portfolio.

15.3 The Minimum Variance Portfolio

The point with the smallest risk on the efficient frontier is called the global *minimum variance portfolio*, MVP. The MVP represents just the minimum risk point on the efficient frontier.The set of weight is:

$$w_\star = \frac{\Sigma^{-1}1}{1^T\Sigma^{-1}1}$$

15.4 The Capital Market Line and Tangency Portfolio

Reward/risk profiles of different combinations of a risky portfolio with a riskless asset, with expected return r_f, can be represented as a straight line, the so called capital market line, CML. The point where the CML touches the efficient frontier corresponds to the optimal risky portfolio. Mathematically, this can be expressed as the portfolio that maximizes the quantity

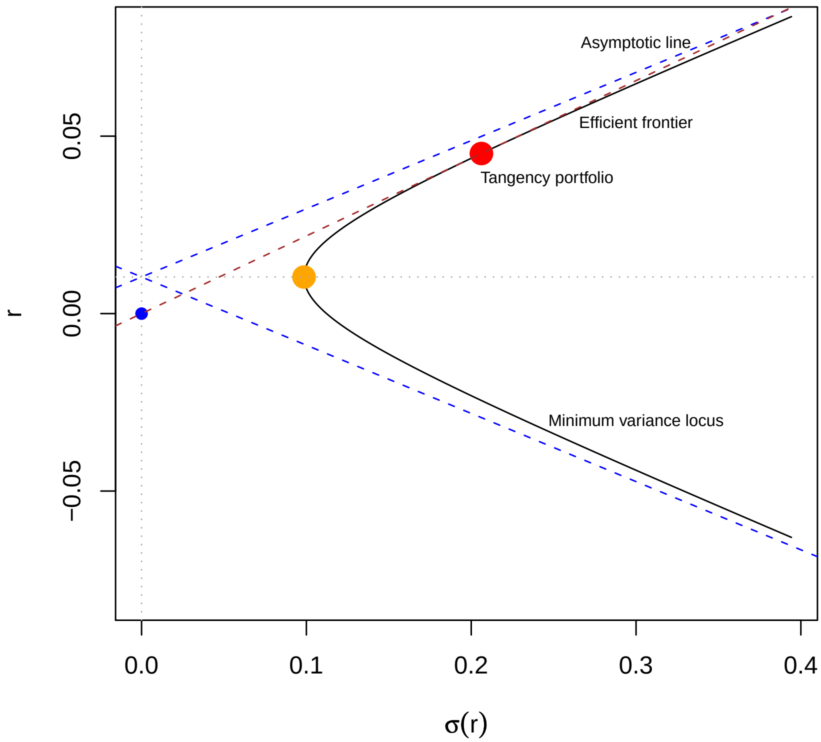


Figure 15.1 Risk versus return view of the mean-variance portfolio: Mean-variance portfolio illustrations in the $(\sigma(r), r)$ space. The minimum variance point $r_\star$ separates the efficient frontier $\partial_+\mathcal{A}$ from the minimum variance locus $\partial_-\mathcal{A}$.

$$\max_w \quad h(w) = \frac{\hat{\mu}^T w - r_f}{w^T \hat{\Sigma} w}$$
$$s.t.$$
$$w^T \hat{\mu} = \bar{r}$$
$$w^T 1 = 1$$

among all w. This quantity is precisely the *Sharpe ratio* introduced by Sharpe (1994).

15.5 Box and Group Constrained Mean-Variance Portfolios

As shown above, the unlimited short selling portfolio can be solved analytically. However, if the weights are bounded by zero, which forbids short selling, then the optimization has to be done numerically. The structure of the portfolio problems is quadratic and thus we can use a quadratic solver to compute the weights of the portfolio. In the following we consider as the standard Markowitz portfolio problem a portfolio which sets box and group constraints on the weights:

$$\min_w \quad w^T \; \Sigma \; w$$
$$s.t.$$
$$Aw \leq b$$

It can be shown that, if Σ is a positive definite matrix, the Markowitz portfolio problem is a convex optimization problem. As such, its local optimal solutions are also global optimal solutions.

The contributed R package `quadprog`(Weingessel, 2004) makes the function `solve.QP()` available, which interfaces a FORTRAN subroutine. This subroutine implements the dual method of Goldfarb & Idnani (1982, 1983) for solving quadratic programming problems of the form $min(-c^T x + 1/2x^T Cx)$ with the constraints $A^T x \geq b$. The Rmetrics solver function `solveRQuadprog(data, spec, constraints)` provides a direct interface to the FORTRAN subroutine provided in the `quadprog` package. The desired solver is selected through the specification structure, which means that the user need not interact with the underlying FORTRAN subroutine. If necessary, the setting can be modified by calling the function `setSolver()`. This allows also allows you to supply an alternative solver providing access to another algorithm or to write your own code.

15.6 Maximum Return Mean-Variance Portfolios

In contrast to minimum risk portfolios, where we minimize the risk for a given target return, maximum return portfolios work the opposite way: Maximize the return for a given target risk.

$$\begin{aligned} & \max_{w} \quad w^T \hat{\mu} \\ s.t. & \\ & Aw \leq b \\ & w^T \hat{\Sigma} w \leq \sigma \end{aligned}$$

Note that now we are concerned with a linear programming problem and quadratic constraints. This can be solved in Rmetrics using either the second order cone programming solver from the R package `Rsocp` or the less efficient non-linear programming solver from the R package `Rdonlp2`.

15.7 Covariance Risk Budgets Constraints

Risk budgeting is a way of taking a finite risk resource, and deciding how best to allocate it. In a mean-variance world, this defaults to Markowitz' portfolio optimization, where results are not only in terms of weights and monetary allocations but also in terms of risk contributions (Scherer & Martin, 2005). In order to quantify risk contributions, we address the questions of how the portfolio risk changes if we increase or decrease holdings in a set of assets. This change for a given asset i can be computed from the derivative

$$\sigma = \sqrt{w^T \: \hat{\Sigma} \: w} = \sum_i w_i \frac{d\sigma}{dw_i} \quad .$$

To make the interpretation easier, we divide through σ and arrive at normalized risk budgets that sum up to 100%, i.e. to 1.

$$1 = \sum_i \mathcal{B}_i = \sum_i w_i \frac{w_i}{\sigma} \frac{d\sigma}{dw_i} \ .$$

Now, adding risk budgeting constraints in portfolio optimization

$$\begin{aligned} & \min \quad w^T \: \hat{\Sigma} \: w \\ s.t. & \\ & w^T \hat{\mu} = \bar{r} \\ & w^T 1 = 1 \\ \mathcal{B}_i^{lower} \leq \frac{w_i}{\sigma} \frac{d\sigma}{dw_i} \leq \mathcal{B}_i^{upper} & \\ & \dots \end{aligned}$$

allows us to limit the maximum and minimum risk contributions arising from individual positions.

Covariance risk budgeting can be formulated for a risk minimizing portfolio as a portfolio with a quadratic objective function and quadratic constraints, in addition to the common linear constraints. The problem is discussed in the ebook *Advanced Portfolio Optimization with R/Rmetrics*.

Chapter 16
Mean-Variance Portfolio Settings

Required R package(s):

```
> library(fPortfolio)
```

Like all portfolios in Rmetrics, mean-variance portfolios are defined by the time series data set, the portfolio specification object, and the constraint strings. Specifying a portfolio thus requires three steps.

16.1 Step 1: Portfolio Data

The portfolio functions expect S4 `timeSeries` objects. You can create them from scratch using one of the functions from the Rmetrics `timeSeries` package for time series generation. Alternatively, you can load a data set from the demo examples provided in the `fPortfolio` package. Note that that the portfolio functions expect time-ordered data records. To sort S4 `timeSeries` objects, use the generic function `sort()`. To align time series objects and to manage missing values use the function `align()`. If you want to bind and merge several `timeSeries` to a data set of assets, you can use the functions `cbind()`, `rbind()` and `merge()`. These functions are explained in detail in Chapter 1[1].

[1] For further details please see the ebook *Chronological Objects with R/Rmetrics*

16.2 Step 2: Portfolio Specification

The representation of the portfolio specification and how to manage the slots is discussed in detail in Chapter 11. For Markowitz' mean-variance portfolio we can just use the default settings

```
> mvSpec <- portfolioSpec()
> print(mvSpec)

Model List:
 Type:                       MV
 Optimize:                   minRisk
 Estimator:                  covEstimator
 Tail Risk:                  list()
 Params:                     alpha = 0.05 a = 1

Portfolio List:
 Target Weights:             NULL
 Target Return:              NULL
 Target Risk:                NULL
 Risk-Free Rate:             0
 Number of Frontier Points: 50
 Status:                     NA

Optim List:
 Solver:                     solveRquadprog
 Objective:                  list()
 Options:                    meq = 2
 Control:                    list()
 Trace:                      FALSE

Message List:
 List:                       NULL
```

The printout tells us that the portfolio type is concerned with the mean-variance portfolio `"MV"`, that we want to optimize (minimize) the risk `"minrisk"` using the quadprog solver `"solveRquadprog"`, and that the sample covariance estimator `"covEstimator"` will be applied. The other two parameters shown are the risk-free rate and the number of frontier points. The first will only be used when we calculate the tangency portfolio and the Sharpe ratio, and the second when we calculate the whole efficient frontier.

16.3 Step 3: Portfolio Constraints

In many cases we will work with long-only portfolios. Specifying

```
> constraints <- "LongOnly"
```

will force the lower and upper bounds for the weights to zero and one, respectively.

Many alternative constraints have already been implemented in `fPortfolio`. These include unlimited short selling, lower and upper bounds, linear equality and inequality constraints, covariance risk budget constraints, and non-linear function constraints. For a full list, see Chapter 11. The solver for dealing with these constraints has to be selected by the user and assigned by the function `setSolver()`.

Chapter 17
Minimum Risk Mean-Variance Portfolios

Required R package(s):

```
> library(fPortfolio)
```

The following examples show how to compute the properties of a minimum risk mean-variance portfolio. These portfolios have a quadratic objective function defined by the covariance matrix of the financial assets and a fixed target return. Included are feasible, efficient, tangency and global minimum risk portfolios. We consider the case of linear constraints, as well as long-only, short selling, box and group constraints[1].

17.1 How to Compute a Feasible Portfolio

A *feasible portfolio* is an 'existing' portfolio described by the settings of the portfolio specification. 'Existing' means that the portfolio was specified by its parameters in such a way that in a risk versus return plot the portfolio has a solution and is part of the feasible set (including the efficient frontier and the minimum variance locus).

The generic way to define a feasible portfolio is to define the portfolio weights. For example, the *equal weights portfolio* is such a portfolio. To

[1] The case of non-linear constraints and the use of alternative solvers is described in the ebook *Advanced Portfolio Optimization with R/Rmetrics*.

specify the equal weights portfolio for the LPP2005[2] data set, we first list the names of the instruments which are part of this data set, and then we subset those which we want to include in the portfolio.

```
Functions:
feasiblePortfolio        feasible portfolio given the weights
efficientPortfolio       minimum risk portfolio for given return
tangencyPortfolio        portfolio with highest Sharpe ratio
minvariancePortfolio     global minimum risk portfolio
weightsPie               weights pie plot
weightedReturnsPie       weighted returns pie plot
covRiskBudgetsPie        covariance risk budget pie plot
```

Listing 17.1 The table lists functions to optimize linearly constrained mean-variance portfolios and to plot the results

```
> colnames(LPP2005.RET)

[1] "SBI"   "SPI"   "SII"   "LMI"   "MPI"   "ALT"   "LPP25" "LPP40" "LPP60"

> lppData <- 100 * LPP2005.RET[, 1:6]
```

Next we add the vector of weights[3] to the specification `spec`, using the function `setWeights()`. In this case, are using equal weights.

```
> ewSpec <- portfolioSpec()
> nAssets <- ncol(lppData)
> setWeights(ewSpec) <- rep(1/nAssets, times = nAssets)
```

Now we are ready to calculate the properties of this portfolio. To do so, we call the function `feasiblePortfolio()`

```
> ewPortfolio <- feasiblePortfolio(
      data = lppData,
      spec = ewSpec,
      constraints = "LongOnly")
> print(ewPortfolio)
```

[2] The LPP2005 data set has nine columns, the first six columns are domestic and foreign assets, the last three columns are the benchmark indices with increasing risk profiles.

[3] The default settings do not specify a weights vector; by default it is set to NULL. We therefore have to supply the portfolio weights explicitly

```
Title:
 MV Feasible Portfolio
 Estimator:         covEstimator
 Solver:            solveRquadprog
 Optimize:          minRisk
 Constraints:       LongOnly

Portfolio Weights:
   SBI    SPI    SII    LMI    MPI    ALT
0.1667 0.1667 0.1667 0.1667 0.1667 0.1667

Covariance Risk Budgets:
    SBI     SPI     SII     LMI     MPI     ALT
-0.0039  0.3526  0.0431 -0.0079  0.3523  0.2638

Target Return and Risks:
  mean     mu    Cov  Sigma   CVaR    VaR
0.0431 0.0431 0.3198 0.3198 0.7771 0.4472

Description:
 Mon May  4 13:44:57 2009 by user: Rmetrics
```

The output first reports the settings, then the portfolio weights, then the covariance risk budgets, and finally the target returns and risks. This includes the portfolio `mean`, and several portfolio risk measures, including the variance computed from the `covariance` matrix, the conditional value-at-risk, and the value-at-risk. Note that since we have specified no alternative covariance estimator, `mu` and `Sigma` are the same as `mean` and `Cov`.

Now let us display the results from the equal weights portfolio, the assignment of weights, and the attribution of returns and risk.

```
> col = divPalette(ncol(lppData), "RdBu")
> weightsPie(ewPortfolio, radius = 0.7, col = col)
> mtext(text = "Equally Weighted MV Portfolio", side = 3, line = 1.5,
     font = 2, cex = 0.7, adj = 0)
> weightedReturnsPie(ewPortfolio, radius = 0.7, col = col)
> mtext(text = "Equally Weighted MV Portfolio", side = 3, line = 1.5,
     font = 2, cex = 0.7, adj = 0)
> covRiskBudgetsPie(ewPortfolio, radius = 0.7, col = col)
> mtext(text = "Equally Weighted MV Portfolio", side = 3, line = 1.5,
     font = 2, cex = 0.7, adj = 0)
```

The pie plots are shown in the left-hand column of Figure 17.1. We have created a view of the pies with a legend listing the asset names and the percentual part of the pies. All pie plots are surrounded by a box, which is the default. The colours are taken from a red to blue diverging palette.

17.2 How to Compute a Minimum Risk Efficient Portfolio

A *minimum risk efficient portfolio* is a portfolio with the lowest risk for a given target return. As a first example for an efficient portfolio, we calculate the efficient mean-variance portfolio with the same target return as the equal weights portfolio, but with the lowest possible risk. Since the default settings of the `portfolioSpec()` function does not define a target return we should not forget to explicitly add the target return to the portfolio specification.

```
> minriskSpec <- portfolioSpec()
> targetReturn <- getTargetReturn(ewPortfolio@portfolio)["mean"]
> setTargetReturn(minriskSpec) <- targetReturn
```

The next step is then to optimize the portfolio for the specified target return.

```
> minriskPortfolio <- efficientPortfolio(
     data = lppData,
     spec = minriskSpec,
     constraints = "LongOnly")
> print(minriskPortfolio)

Title:
 MV Efficient Portfolio
 Estimator:         covEstimator
 Solver:            solveRquadprog
 Optimize:          minRisk
 Constraints:       LongOnly

Portfolio Weights:
   SBI    SPI    SII    LMI    MPI    ALT
0.0000 0.0086 0.2543 0.3358 0.0000 0.4013

Covariance Risk Budgets:
    SBI     SPI     SII     LMI     MPI     ALT
 0.0000  0.0184  0.1205 -0.0100  0.0000  0.8711

Target Return and Risks:
```

```
  mean     mu    Cov  Sigma   CVaR    VaR
0.0431 0.0431 0.2451 0.2451 0.5303 0.3412

Description:
 Mon May  4 13:44:58 2009 by user: Rmetrics
```

The weights and related pie plots are generated in the same way as for the equal weights portfolio shown in the previous section.

```
> col = qualiPalette(ncol(lppData), "Dark2")
> weightsPie(minriskPortfolio, radius = 0.7, col = col)
> mtext(text = "Minimal Risk MV Portfolio", side = 3, line = 1.5,
     font = 2, cex = 0.7, adj = 0)
> weightedReturnsPie(minriskPortfolio, radius = 0.7, col = col)
> mtext(text = "Minimal Risk MV Portfolio", side = 3, line = 1.5,
     font = 2, cex = 0.7, adj = 0)
> covRiskBudgetsPie(minriskPortfolio, radius = 0.7, col = col)
> mtext(text = "Minimal Risk MV Portfolio", side = 3, line = 1.5,
     font = 2, cex = 0.7, adj = 0)
```

The pie plots are shown in the right-hand column of Figure 17.1. We have created a view of the pies with the same settings as in the previous example, except for the colours; these are taken from a dark qualitative palette.

17.3 How to Compute the Global Minimum Variance Portfolio

The *global minimum variance portfolio* is the efficient portfolio with the lowest possible risk. The global minimum variance point is thus the point which separates the efficient frontier from the minimum variance locus.

```
> globminSpec <- portfolioSpec()
> globminPortfolio <- minvariancePortfolio(
     data = lppData,
     spec = globminSpec,
     constraints = "LongOnly")
> print(globminPortfolio)

Title:
 MV Minimum Variance Portfolio
 Estimator:         covEstimator
```

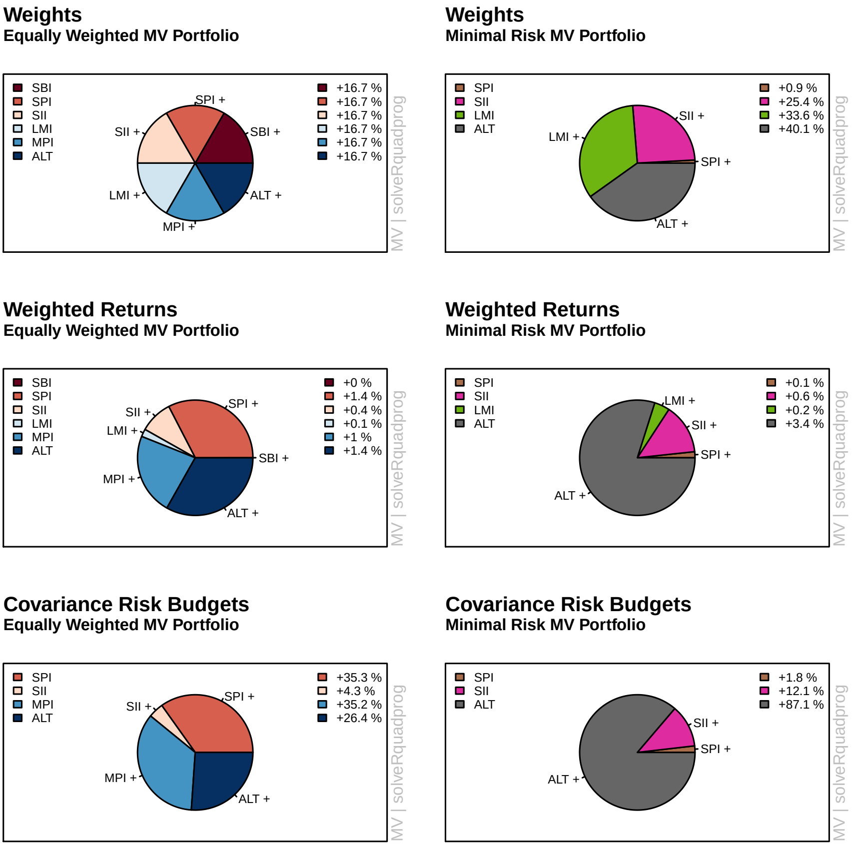


Figure 17.1 Weights, weighted returns, and covariance risk budgets plots for an equal weighted and a minimum variance portfolio: The equally weighted portfolio is shown to the left and the efficient portfolio with the same target return to the right. We have reduced the radius of the pies to 70% since we have a legend to the right and left. The legend to the left lists the assets and the legend to the right the percentual parts of the pie. The text to the right margin denotes the portfolio type, `MV`, and the solver, `solveRquadprog`, used for optimizing the portfolio.

```
 Solver:            solveRquadprog
 Optimize:          minRisk
 Constraints:       LongOnly

Portfolio Weights:
   SBI    SPI    SII    LMI    MPI    ALT
0.3554 0.0000 0.0891 0.4894 0.0025 0.0636

Covariance Risk Budgets:
   SBI    SPI    SII    LMI    MPI    ALT
0.3553 0.0000 0.0891 0.4893 0.0025 0.0637

Target Return and Risks:
  mean     mu    Cov  Sigma   CVaR    VaR
0.0105 0.0105 0.0986 0.0986 0.2020 0.1558

Description:
 Mon May  4 13:44:58 2009 by user: Rmetrics
```

Internally, the global minimum mean-variance portfolio is calculated by minimizing the efficient portfolio with respect to the target risk. This is a quadratic optimization problem with linear constraints.

```
> col <- seqPalette(ncol(lppData), "YlGn")
> weightsPie(globminPortfolio, box = FALSE, col = col)
> mtext(text = "Global Minimum Variance MV Portfolio", side = 3,
     line = 1.5, font = 2, cex = 0.7, adj = 0)
> weightedReturnsPie(globminPortfolio, box = FALSE, col = col)
> mtext(text = "Global Minimum Variance MV Portfolio", side = 3,
     line = 1.5, font = 2, cex = 0.7, adj = 0)
> covRiskBudgetsPie(globminPortfolio, box = FALSE, col = col)
> mtext(text = "Global Minimum Variance MV Portfolio", side = 3,
     line = 1.5, font = 2, cex = 0.7, adj = 0)
```

The pie plots for the global minimum mean-variance portfolio are shown in the left-hand column of Figure 17.2. Compared to the previous pie plots in Figure 17.1, we have chosen a different layout. The colours are taken from yellow to green sequential palettes, and the boxes around the pies have been suppressed.

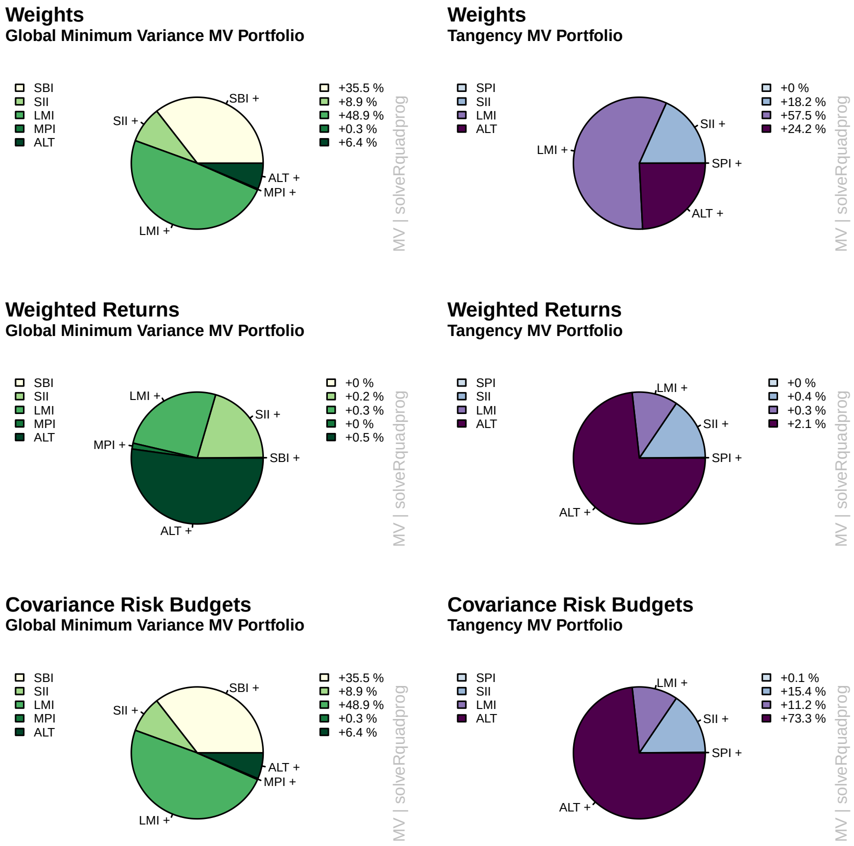


Figure 17.2 Weights, weighted returns, and covariance risk budget plots for the global minimum risk and the tangency portfolios: The global minimum risk portfolio is shown to the left and the tangency portfolio to the right. In this graph we have chosen a orange-to-red colour palette and removed the boxes from the pie charts.

17.4 How to Compute the Tangency Portfolio

The *tangency portfolio* is calculated by minimizing the Sharpe Ratio for a given risk-free rate. The Sharpe ratio is the ratio of the target return lowered by the risk-free rate and the covariance risk. The default risk-free rate is zero and can be reset to another value by modifying the portfolio's specification.

```
> tgSpec <- portfolioSpec()
> setRiskFreeRate(tgSpec) <- 0
```

The tangency portfolio is then obtained by calling the function `tangencyPortfolio()`

```
> tgPortfolio <- tangencyPortfolio(
     data = lppData,
     spec = tgSpec,
     constraints = "LongOnly")
> print(tgPortfolio)

Title:
 MV Tangency Portfolio
 Estimator:         covEstimator
 Solver:            solveRquadprog
 Optimize:          minRisk
 Constraints:       LongOnly

Portfolio Weights:
   SBI    SPI    SII    LMI    MPI    ALT
0.0000 0.0005 0.1824 0.5751 0.0000 0.2420

Covariance Risk Budgets:
   SBI    SPI    SII    LMI    MPI    ALT
0.0000 0.0014 0.1538 0.1121 0.0000 0.7326

Target Return and Risks:
  mean     mu    Cov  Sigma   CVaR    VaR
0.0283 0.0283 0.1534 0.1534 0.3097 0.2143

Description:
 Mon May  4 13:44:59 2009 by user: Rmetrics
```

and the pie plots are generated as in the previous examples, but this time using a blue to purple sequential palette.

```
> col <- seqPalette(ncol(lppData), "BuPu")
> weightsPie(tgPortfolio, box = FALSE, col = col)
> mtext(text = "Tangency MV Portfolio", side = 3, line = 1.5,
     font = 2, cex = 0.7, adj = 0)
> weightedReturnsPie(tgPortfolio, box = FALSE, col = col)
> mtext(text = "Tangency MV Portfolio", side = 3, line = 1.5,
     font = 2, cex = 0.7, adj = 0)
> covRiskBudgetsPie(tgPortfolio, box = FALSE, col = col)
> mtext(text = "Tangency MV Portfolio", side = 3, line = 1.5,
     font = 2, cex = 0.7, adj = 0)
```

The pie plots are shown in the right-hand column of Figure 17.2.

17.5 How to Customize a Pie Plot

The functions `weightsPie()`, `weightedReturnsPie()`, and `covRiskBudgetsPie()` have with several arguments that allow you to customize the plots.

```
Functions:
weightsPie          displays the weights composition
weightedReturnsPie  displays weighted returns, the investment
covRiskBudgetsPie   displays the covariance risk budgets

Arguments:
object              an S4 object of class fPORTFOLIO
pos                 the point position on a whole frontier
labels              should the graph be labelled?
col                 selects colour from a colour palette
box                 should a box drawn around the pies?
legend              should a legend added to the pies?
radius              the radius of the pie
...                 arguments to be passed
```

Listing 17.2 Functions to plot pie charts of portfolio weights

If you prefer a bar plot instead of a pie chart you can easily create it with R's base function `barplot()`. Here is an example of how to display the weights of the tangency portfolio optimized above as a horizontal bar chart.

```
> col = rampPalette(ncol(lppData), "purple2green")
> weights <- 100 * as.vector(getWeights(tgPortfolio))
> names <- as.vector(getNames(tgPortfolio))
> barplot(height = weights, names.arg = names,
         horiz = TRUE, las = 1, col = col)
> title(main = "Weights of Long-Only Tangency Portfolio",
       xlab = "Weights %")
```

The bar plots of weights, weighted returns, and covariance risk budgets are shown in Figure 17.3. It is left to the reader to write his or her own functions for customized bar plots with the same argument list as for the pie plots.

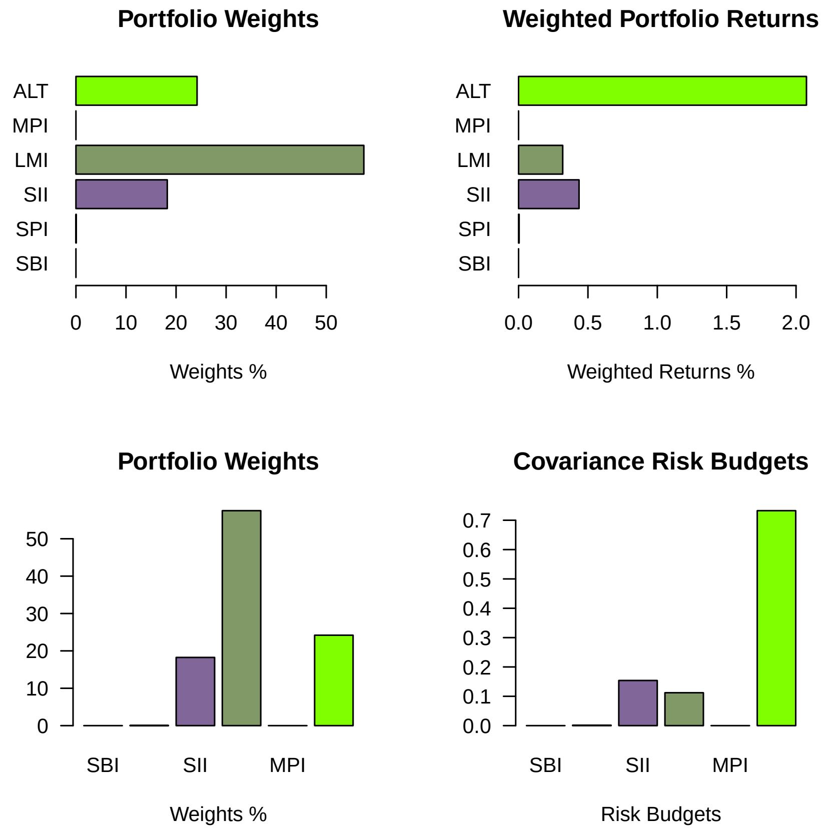


Figure 17.3 Weights, weighted returns, and covariance risk budget bar plots for the long-only tangency portfolio with zero risk-free rate. In this graph we have chosen a purple to green colour ramp palette and and a horizontal display for the first two and a vertical display (the default) for the remaining two graphs.

Chapter 18
Mean-Variance Portfolio Frontiers

Required R package(s):

```
> library(fPortfolio)
```

The efficient frontier together with the minimum variance locus form the 'upper border' and 'lower border' lines of the feasible set. To the right the feasible set is determined by the envelope of all pairwise asset frontiers. The region outside of the feasible set is unachievable by holding risky assets alone. No portfolios can be constructed corresponding to the points in this region. Points below the frontier are suboptimal. Thus, a rational investor will hold a portfolio only on the frontier. In this chapter we show how to compute the whole efficient frontier and minimum variance locus of a mean-variance portfolio with linear constraints[1] and show which functions can be used to display the results.

18.1 Frontier Computation and Graphical Displays

The Rmetrics function `portfolioFrontier()` allows you to calculate optimized portfolios along the efficient frontier and the minimum variance locus. For the default settings with long-only constraints, the range spans all values equidistantly ranging from the asset with the lowest return up to

[1] The case of non-linear constraints and the use of alternative solvers is described in the ebook *Advanced Portfolio Optimization with R/Rmetrics*.

the asset with the highest return. Allowing for box, group and other more complex constraints the range of the frontier will be downsized, i.e. the length of the frontier becomes shorter and shorter. Bear in mind that it is possible for the constraints to be too strong, and that a frontier might not even exist at all.

Many additional parameters can be set by the portfolio specification function, such as the number of frontier points.

```
Functions:
portfolioFrontier       efficient portfolios on the frontier
frontierPoints          extracts risk/return frontier points
frontierPlot            creates an efficient frontier plot
  cmlPoints               adds market portfolio
  cmlLines                adds capital market line
  tangencyPoints          adds tangency portfolio point
  tangencyLines           adds tangency line
  equalWeightsPoints      adds point of equal weights portfolio
  singleAssetPoints       adds points of single asset portfolios
  twoAssetsLines          adds frontiers of two assets portfolios
  sharpeRatioLines        adds Sharpe ratio line
  monteCarloPoints        adds randomly feasible portfolios
weightsPlot             weights bar plot along the frontier
weightedReturnsPlot     weighted returns bar plot
covRiskBudgetsPlot      covariance risk budget bar plot
```

Listing 18.1 This table lists functions to compute the efficient frontier of linearly constrained mean-variance portfolios and to plot the results.

How to compute the efficient frontier

The computation of the efficient frontier for the default MV portfolio just requires a few function calls. As a first example, we compute the efficient frontier for the six assets included in Pictet's pension fund benchmark portfolio `LPP2005.RET`. We multiply the series by 100 to convert them to returns in percentages.

```
> lppData <- 100 * LPP2005.RET[, 1:6]
> colnames(lppData)

[1] "SBI" "SPI" "SII" "LMI" "MPI" "ALT"
```

Let us compute the efficient frontier for example on 5 points.

```
> lppSpec <- portfolioSpec()
> setNFrontierPoints(lppSpec) <- 5
> longFrontier <- portfolioFrontier(lppData, lppSpec)
```

How to print an efficient frontier report

The printout of the S4 `frontier` object returned by the `portfolioFrontier()` function lists the major parameter settings, the portfolio weights, the covariance risk budgets, and the target return and risk values for each point along the frontier.

```
> print(longFrontier)

Title:
 MV Portfolio Frontier
 Estimator:         covEstimator
 Solver:            solveRquadprog
 Optimize:          minRisk
 Constraints:       LongOnly
 Portfolio Points:  5 of 5

Portfolio Weights:
     SBI    SPI    SII    LMI    MPI    ALT
1 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000
2 0.0193 0.0000 0.1481 0.6665 0.0000 0.1661
3 0.0000 0.0085 0.2535 0.3386 0.0000 0.3994
4 0.0000 0.0210 0.3458 0.0000 0.0000 0.6332
5 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000

Covariance Risk Budgets:
      SBI     SPI     SII     LMI     MPI     ALT
1  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000
2  0.0064  0.0000  0.1593  0.3359  0.0000  0.4984
3  0.0000  0.0183  0.1208 -0.0097  0.0000  0.8707
4  0.0000  0.0286  0.0890  0.0000  0.0000  0.8824
5  0.0000  0.0000  0.0000  0.0000  0.0000  1.0000

Target Return and Risks:
    mean     mu    Cov  Sigma   CVaR    VaR
1 0.0000 0.0000 0.1261 0.1261 0.2758 0.2177
2 0.0215 0.0215 0.1214 0.1214 0.2362 0.1760
3 0.0429 0.0429 0.2439 0.2439 0.5275 0.3392
4 0.0643 0.0643 0.3939 0.3939 0.8822 0.5886
```

```
5 0.0858 0.0858 0.5684 0.5684 1.3343 0.8978

Description:
 Mon May  4 13:44:26 2009 by user: Rmetrics
```

How to interactively plot the efficient frontier

Several plotting facilities are available to display the efficient frontier. For a quick interactive overview you can use the generic `plot()` function offering the following selections:

```
> longFrontier <- portfolioFrontier(lppData)
> plot(longFrontier)

Make a plot selection (or 0 to exit):

1:   Plot Efficient Frontier
2:   Add Minimum Risk Portfolio
3:   Add Tangency Portfolio
4:   Add Risk/Return of Single Assets
5:   Add Equal Weights Portfolio
6:   Add Two Asset Frontiers [0-1 PF Only]
7:   Add Wheel Pie of Weights
8:   Add Monte Carlo Portfolios
9:   Add Sharpe Ratio [MV PF Only]

Selection:
```

As an example, recalculate the `frontier` with the default number of frontier points (50), then call the interactive `plot()` function and add some optional graphs.

How to create a customized efficient frontier plot

For customized plots the function `frontierPlot()` with several add-on plot functions can be used.

```
Functions:
frontierPlot          efficient frontier plot
  cmlPoints             adds market portfolio
  cmlLines              adds capital market line
  tangencyPoints        adds tangency portfolio point
```

```
  tangencyLines       adds tangency line
  equalWeightsPoints  adds point of equal weights portfolio
  singleAssetPoints   adds points of single asset portfolios
  twoAssetsLines      adds frontiers of two assets portfolios
  sharpeRatioLines    adds Sharpe ratio line
  monteCarloPoints    adds randomly feasible portfolios

Arguments:
object                an S4 object of class fPORTFOLIO
frontier              which frontier part should be plotted?
col                   colours for the EF and the MV locus
add                   should another frontier added?
return                select from 'mean' or 'mu'
risk                  select from 'cov',' sigma', 'VaR', 'CVaR'
auto                  automatic risk/return selection
labels                should the plot be labelled?
title                 should a default title be added?
mcSteps               number of Monte Carlo steps
xlim, ylim            set the plot range
mText                 marginal text to be added
...                   optional arguments to be passed
```

Listing 18.2 Functions to display the efficient frontier.

These functions allow you to create your own customized plotting function, allowing you to create your own presentation of the frontier. The Rmetrics `fPortfolio` package already has such a function, named `tailoredFrontierPlot()`, which can be used as a starting point for your customized frontier plot function.

```
> args(tailoredFrontierPlot)

function (object, risk = c("Cov", "Sigma", "CVaR", "VaR"), mText = NULL,
    col = NULL, xlim = NULL, ylim = NULL, twoAssets = FALSE)
NULL
```

If you want to write your own display function, it is a good idea to also inspect the code of the function.

```
> tailoredFrontierPlot
```

The result of calling the `tailoredFrontierPlot()` is shown in Figure 18.1, now re-calculated on 25 frontier points.

```
> setNFrontierPoints(lppSpec) <- 25
> longFrontier <- portfolioFrontier(lppData, lppSpec)
> tailoredFrontierPlot(object = longFrontier, mText = "MV Portfolio - LongOnly
     Constraints",
    risk = "Cov")
```

How to create weights and related plots

Furthermore, bar and line plots for the weights, `weightsPlot()`, the performance attribution, `weightedReturnsPlot()`, and the covariance risk budgets, `covRiskBudgetsPlot()`, are available. Note that these plots can also be displayed as line plots.

```
> weightsPlot(longFrontier, mtext = FALSE)
> text <- "Mean-Variance Portfolio - Long Only Constraints"
> mtext(text, side = 3, line = 3, font = 2, cex = 0.9)
> weightedReturnsPlot(longFrontier, mtext = FALSE)
> covRiskBudgetsPlot(longFrontier, mtext = FALSE)
```

18.2 The 'long-only' Portfolio Frontier

The long-only constraints are the default constraints for the mean-variance portfolios. Remember that in this case all the weights are bounded between zero and one. In the previous example we optimized the default portfolio, and the results are shown in Figure 18.1 for the frontier and in Figure 18.2 for the weights.

Now we want to explore in more detail the feasible set of the long-only constrained mean-variance portfolio. For this, we plot the frontier, add randomly generated portfolios from a Monte Carlo simulation, and add the frontier lines of all two-asset portfolios.

```
> set.seed(1953)
> frontierPlot(object = longFrontier, pch = 19, xlim = c(0.05,
    0.85), cex = 0.5)
> monteCarloPoints(object = longFrontier, mcSteps = 1000, pch = 19,
    cex = 0.5)
```

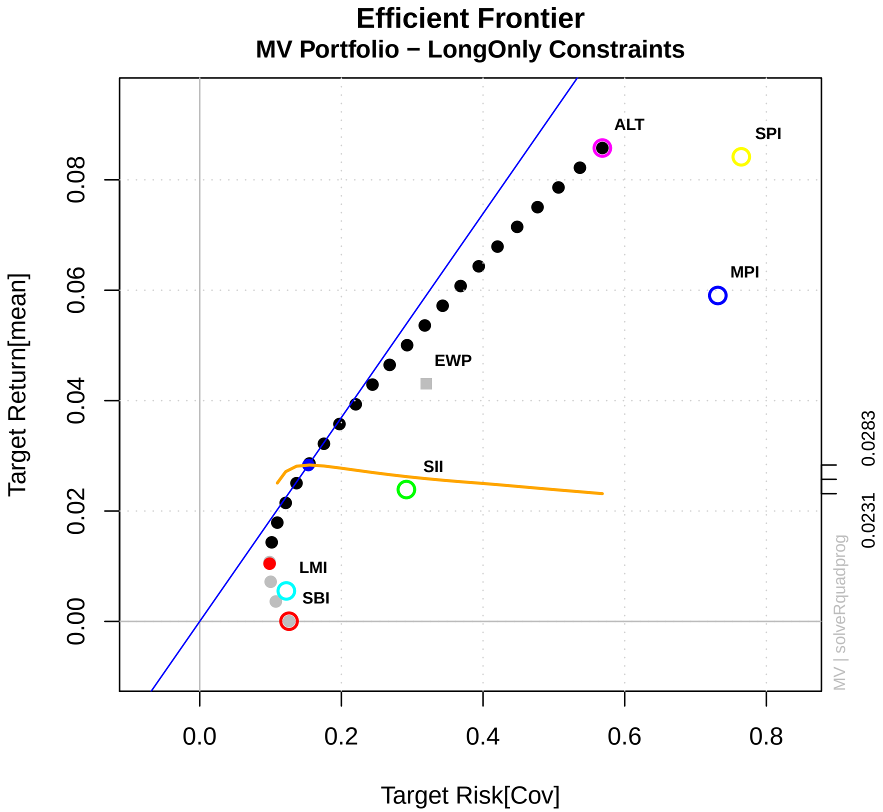


Figure 18.1 Efficient frontier of a long-only constrained mean-variance portfolio: The plot includes the efficient frontier, the tangency line and tangency point for a zero risk-free rate, the equal weights portfolio, EWP, all single assets risk vs. return points. The line of Sharpe ratios is also shown, with its maximum coinciding with the tangency portfolio point. The range of the Sharpe ratio is printed on the right hand side axis of the plot.

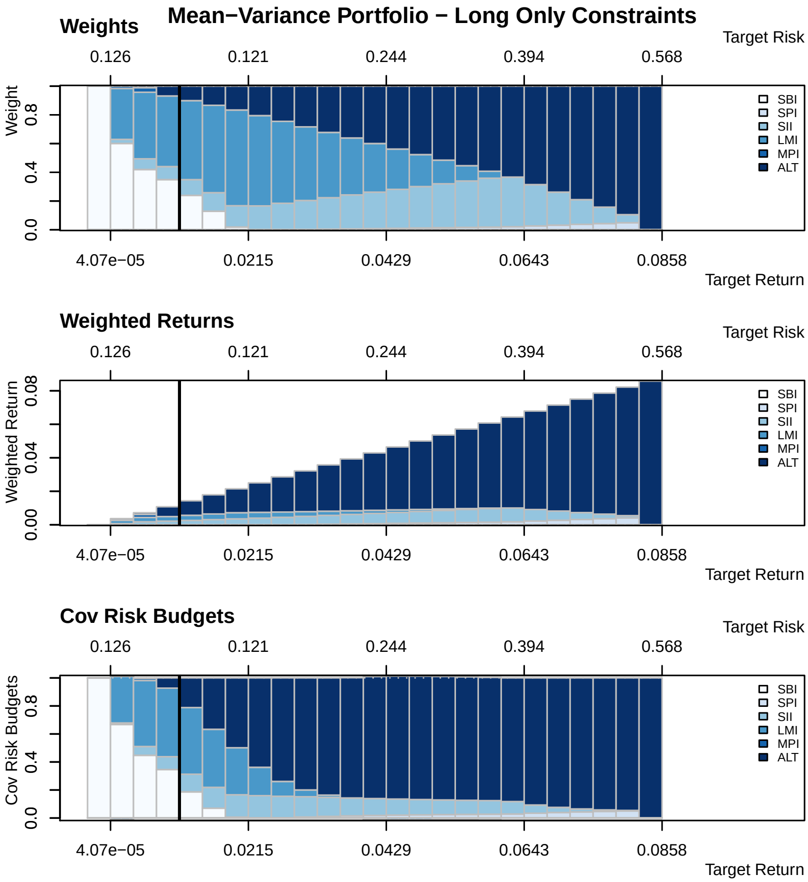


Figure 18.2 Weights along the efficient frontier of a long-only constrained mean-variance portfolio: The graphs from top to bottom show the weights, the weighted returns or in other words the performance attribution, and the covariance risk budgets which are a measure for the risk attribution. The upper axis labels the target risk, and the lower labels the target return. The thick vertical line separates the efficient frontier from the minimum variance locus. The risk axis thus increases in value to both sides of the separator line. The legend to the right links the assets names to colour of the bars.

```
> par(mfrow = c(1, 1))
> set.seed(1953)
> frontierPlot(object = longFrontier, pch = 19, xlim = c(0.05,
     0.85), cex = 0.5)
> monteCarloPoints(object = longFrontier, mcSteps = 1000, pch = 19,
     cex = 0.5)
> twoAssetsLines(object = longFrontier, col = "orange", lwd = 2)
> frontier <- frontierPoints(object = longFrontier)
> lines(frontier, col = "red", lwd = 2)
```

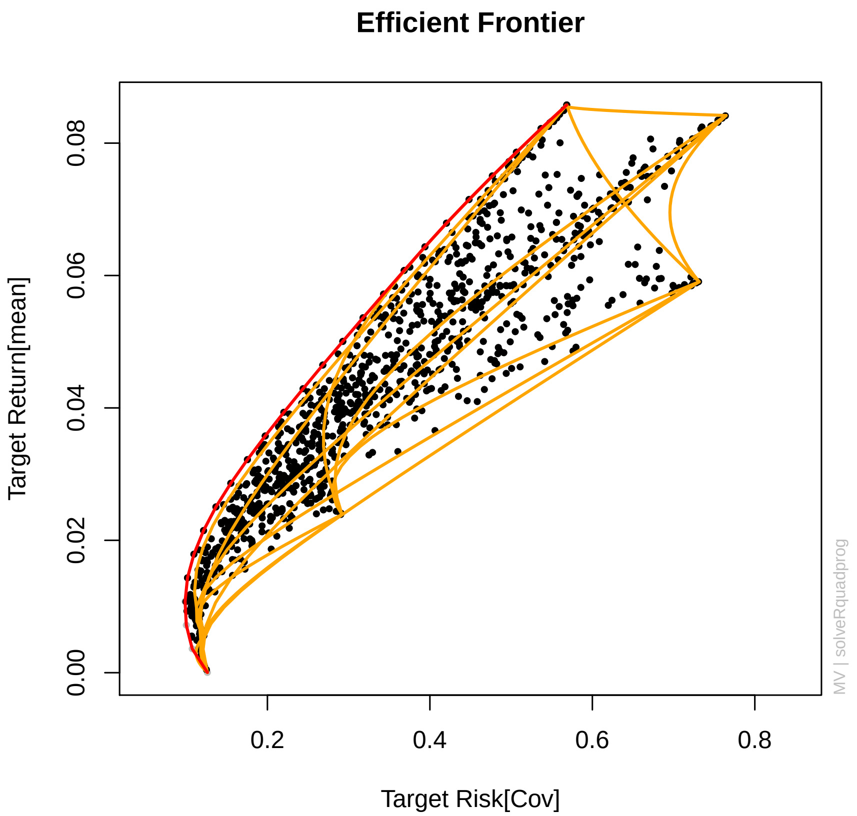


Figure 18.3 The feasible set for a long-only constrained mean-variance portfolio: The graph shows the risk/return plot for 1000 randomly generated mean-variance portfolios with long-only constraints. The plot is overlayed by the efficient frontier, the minimum variance locus, and the pairwise frontier lines of all combinations of two-asset portfolios. The corners of the lines coincide with the risk/return values for the six assets.

```
> twoAssetsLines(object = longFrontier, col = "orange", lwd = 2)
> frontier <- frontierPoints(object = longFrontier)
> lines(frontier, col = "red", lwd = 2)
```

Note that the Monte Carlo simulation provided by the function `monteCarloPoints()` is only meaningful for long-only constraints.

18.3 The Unlimited 'short' Portfolio Frontier

If all the weights are not restricted, we have the case of unlimited short selling. Since unlimited short selling portfolios can be solved analytically, we can replace the solver `"solveRquadprog"` with the solver `"solveRshortExact"`

```
> shortSpec <- portfolioSpec()
> setNFrontierPoints(shortSpec) <- 5
> setSolver(shortSpec) <- "solveRshortExact"
> shortFrontier <- portfolioFrontier(
     data = lppData,
     spec = shortSpec,
     constraints = "Short")
> print(shortFrontier)

Title:
 MV Portfolio Frontier
 Estimator:         covEstimator
 Solver:            solveRshortExact
 Optimize:          minRisk
 Constraints:       Short
 Portfolio Points:  5 of 5

Portfolio Weights:
      SBI     SPI     SII     LMI     MPI     ALT
1  0.5348 -0.0310  0.0493  0.4245  0.1054 -0.0830
2  0.1560  0.0210  0.1337  0.5648 -0.1013  0.2258
3 -0.2227  0.0730  0.2181  0.7051 -0.3081  0.5346
4 -0.6014  0.1250  0.3025  0.8453 -0.5149  0.8435
5 -0.9801  0.1769  0.3869  0.9856 -0.7216  1.1523

Covariance Risk Budgets:
      SBI     SPI     SII     LMI     MPI     ALT
```

```
1  0.5348  0.0267  0.0233  0.3730 -0.0322  0.0744
2  0.0788  0.0513  0.1412  0.3569 -0.1893  0.5610
3 -0.0038  0.1420  0.1230  0.1007 -0.4222  1.0602
4  0.0360  0.1658  0.1008  0.0260 -0.4700  1.1414
5  0.0674  0.1734  0.0885 -0.0003 -0.4813  1.1523

Target Return and Risks:
    mean     mu    Cov  Sigma   CVaR    VaR
1 0.0000 0.0000 0.1121 0.1121 0.2374 0.1921
2 0.0215 0.0215 0.1144 0.1144 0.2187 0.1710
3 0.0429 0.0429 0.1962 0.1962 0.3790 0.2712
4 0.0643 0.0643 0.2979 0.2979 0.5912 0.4078
5 0.0858 0.0858 0.4048 0.4048 0.8123 0.5374

Description:
 Mon May  4 13:44:46 2009 by user: Rmetrics
```

For the plot of the frontier we reset the number of frontier points to 20 and recalculate the frontier.

```
> setNFrontierPoints(shortSpec) <- 20
> shortFrontier <- portfolioFrontier(data = lppData, spec = shortSpec,
     constraints = "Short")
> tailoredFrontierPlot(object = shortFrontier, mText = "MV Portfolio - Short
     Constraints",
     risk = "Cov")
```

```
> weightsPlot(shortFrontier, mtext = FALSE)
> text <- "MV Portfolio - Short Constrained Portfolio"
> mtext(text, side = 3, line = 3, font = 2, cex = 0.9)
> weightedReturnsPlot(shortFrontier, mtext = FALSE)
> covRiskBudgetsPlot(shortFrontier, mtext = FALSE)
```

The results are shown in Figure 18.4 and Figure 18.5.

18.4 The Box-Constrained Portfolio Frontier

A box-constrained portfolio is a portfolio where the weights are constrained by lower and upper bounds, e.g. we want to invest at least in each asset 10% and no more than 50%.

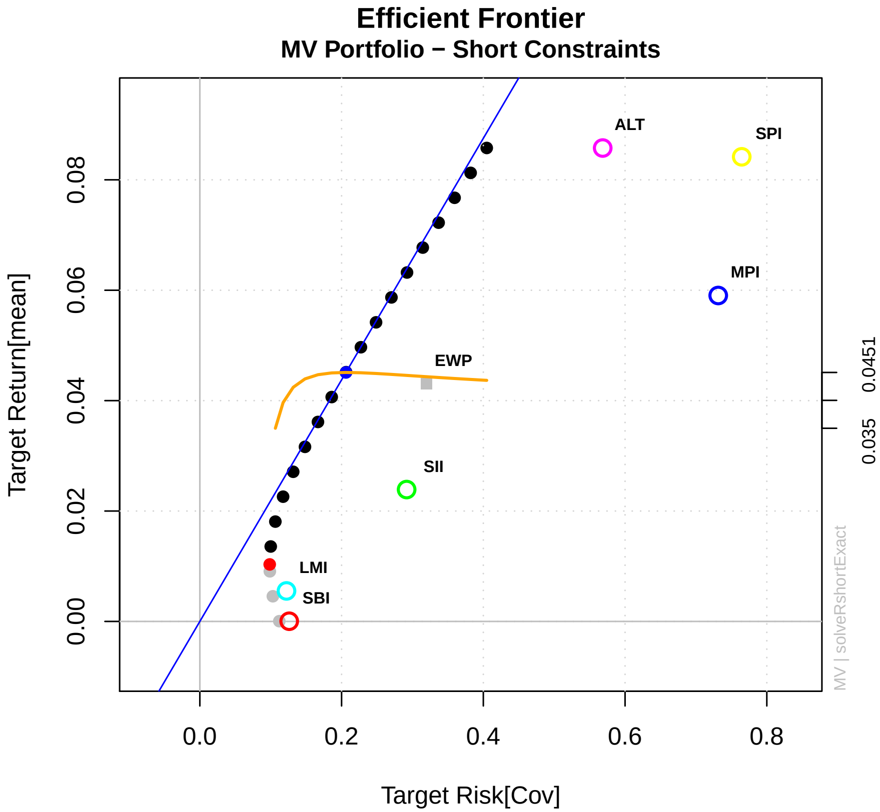


Figure 18.4 Efficient frontier of an unlimited short selling constrained mean-variance portfolio: The plot includes the efficient frontier, the tangency line and tangency point for a zero risk-free rate, the equal weights portfolio, EWP, all single assets risk vs. return points. The line of Sharpe ratios is also shown, with its maximum coinciding with the tangency portfolio point. The range of the Sharpe ratio is printed on the right hand side axis of the plot.

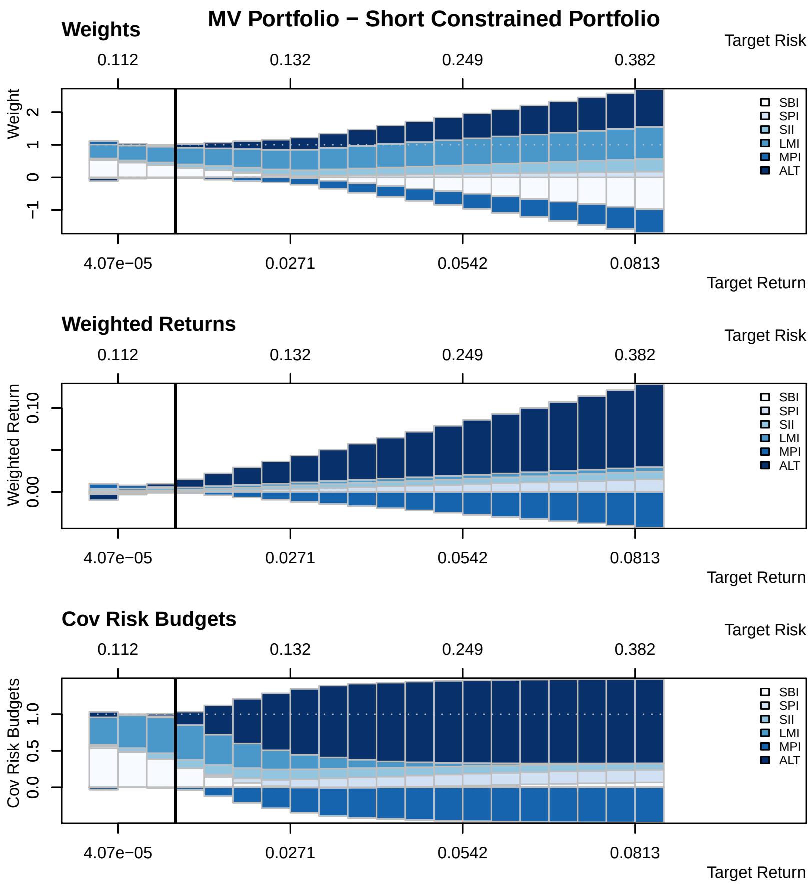


Figure 18.5 Weights along the efficient frontier of an unlimited short selling constrained mean-variance portfolio: The graphs from top to bottom show the weights, the weighted returns or in other words the performance attribution, and the covariance risk budgets which are a measure for the risk attribution. The upper axis labels the target risk, and the lower labels the target return. The thick vertical line separates the efficient frontier from the minimum variance locus. The risk axis thus increases in value to both sides of the separator line. The legend to the right links the assets names to colour of the bars.

```
> boxSpec <- portfolioSpec()
> setNFrontierPoints(boxSpec) <- 15
> boxConstraints <- c(
     "minW[1:6]=0.1",
     "maxW[1:6]=0.5")
> boxFrontier <- portfolioFrontier(
     data = lppData,
     spec = boxSpec,
     constraints = boxConstraints)
> print(boxFrontier)
```

```
Title:
 MV Portfolio Frontier
 Estimator:         covEstimator
 Solver:            solveRquadprog
 Optimize:          minRisk
 Constraints:       minW maxW
 Portfolio Points:  5 of 5

Portfolio Weights:
     SBI    SPI    SII    LMI    MPI    ALT
1 0.1000 0.1000 0.1836 0.4032 0.1000 0.1133
2 0.1000 0.1000 0.2140 0.3034 0.1000 0.1826
3 0.1000 0.1000 0.2445 0.2036 0.1000 0.2520
4 0.1000 0.1000 0.2749 0.1038 0.1000 0.3213
5 0.1000 0.1000 0.1808 0.1000 0.1000 0.4192

Covariance Risk Budgets:
      SBI     SPI     SII     LMI     MPI     ALT
1  0.0036  0.3125  0.1019  0.0142  0.3066  0.2613
2 -0.0006  0.2628  0.0981 -0.0060  0.2677  0.3780
3 -0.0027  0.2229  0.0937 -0.0098  0.2332  0.4627
4 -0.0038  0.1916  0.0896 -0.0064  0.2046  0.5245
5 -0.0041  0.1677  0.0369 -0.0061  0.1829  0.6227

Target Return and Risks:
    mean     mu    Cov  Sigma   CVaR    VaR
1 0.0307 0.0307 0.2056 0.2056 0.4804 0.2935
2 0.0368 0.0368 0.2429 0.2429 0.5695 0.3553
3 0.0429 0.0429 0.2826 0.2826 0.6631 0.3983
4 0.0490 0.0490 0.3240 0.3240 0.7597 0.4481
5 0.0552 0.0552 0.3683 0.3683 0.8844 0.5678

Description:
 Mon May  4 13:44:49 2009 by user: Rmetrics
```

For the plot of the frontier we reset the number of frontier points to 25 and recalculate the frontier.

```
> setNFrontierPoints(boxSpec) <- 25
> boxFrontier <- portfolioFrontier(data = lppData, spec = boxSpec,
     constraints = boxConstraints)
> tailoredFrontierPlot(object = boxFrontier, mText = "MV Portfolio - Box
     Constraints",
     risk = "Cov")
```

```
> weightsPlot(boxFrontier)
> text <- "MV Portfolio - Box Constrained Portfolio"
> mtext(text, side = 3, line = 3, font = 2, cex = 0.9)
> weightedReturnsPlot(boxFrontier)
> covRiskBudgetsPlot(boxFrontier)
```

The efficient frontier of the box-constrained MV portfolio is shown in Figure 18.6. The weights, weighted returns and covariance risk budgets are shown in the left-hand column of Figure 18.7.

18.5 The Group-Constrained Portfolio Frontier

A group-constrained portfolio is a portfolio where the weights of groups of selected assets are constrained by lower and upper bounds for the total weights of the groups, e.g. we want to invest at least in the group of bonds 30% and no more than 50% in the groups of assets.

```
> groupSpec <- portfolioSpec()
> setNFrontierPoints(groupSpec) <- 7
> groupConstraints <- c("minsumW[c(1,4)]=0.3",
                        "maxsumW[c(2,5)]=0.5")
> groupFrontier <- portfolioFrontier(
     data = lppData,
     spec = groupSpec,
     constraints = groupConstraints)
> print(groupFrontier)
```

```
Title:
 MV Portfolio Frontier
 Estimator:         covEstimator
```

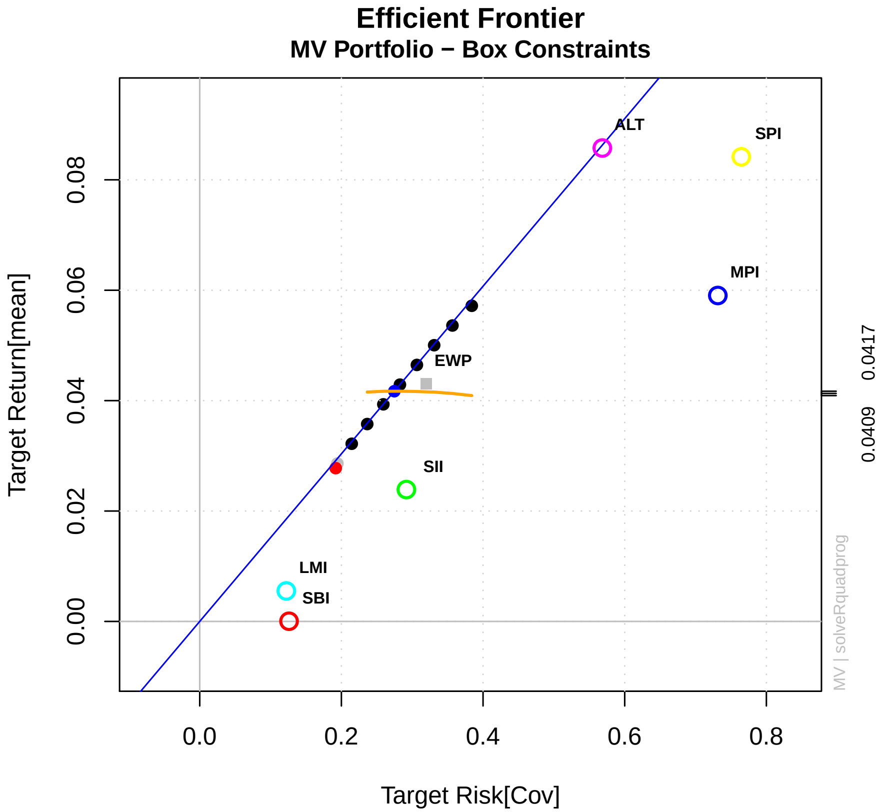


Figure 18.6 Efficient frontier of a box-constrained mean-variance portfolio: The plot includes the efficient frontier, the tangency line and tangency point for a zero risk-free rate, the equal weights portfolio, EWP, all single assets risk vs. return points. The line of Sharpe ratios is also shown, with its maximum coinciding with the tangency portfolio point. The range of the Sharpe ratio is printed on the right hand side axis of the plot.

```
 Solver:            solveRquadprog
 Optimize:          minRisk
 Constraints:       minsumW maxsumW
 Portfolio Points:  5 of 5

Portfolio Weights:
     SBI    SPI    SII    LMI    MPI    ALT
1 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000
2 0.2394 0.0000 0.1097 0.5500 0.0000 0.1009
3 0.0000 0.0006 0.1838 0.5705 0.0000 0.2450
4 0.0000 0.0085 0.2535 0.3386 0.0000 0.3994
5 0.0000 0.0284 0.0721 0.3000 0.0000 0.5995

Covariance Risk Budgets:
      SBI     SPI     SII     LMI     MPI     ALT
1  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000
2  0.1869  0.0000  0.1258  0.4757  0.0000  0.2117
3  0.0000  0.0019  0.1532  0.1061  0.0000  0.7388
4  0.0000  0.0183  0.1208 -0.0097  0.0000  0.8707
5  0.0000  0.0445  0.0097 -0.0150  0.0000  0.9609

Target Return and Risks:
    mean     mu    Cov  Sigma   CVaR    VaR
1 0.0000 0.0000 0.1261 0.1261 0.2758 0.2177
2 0.0143 0.0143 0.1016 0.1016 0.2002 0.1534
3 0.0286 0.0286 0.1549 0.1549 0.3136 0.2170
4 0.0429 0.0429 0.2439 0.2439 0.5275 0.3392
5 0.0572 0.0572 0.3516 0.3516 0.8182 0.5552

Description:
 Mon May  4 13:44:50 2009 by user: Rmetrics
```

For the plot of the frontier we reset the number of frontier points to 25 and recalculate the frontier.

```
> groupSpec <- portfolioSpec()
> setNFrontierPoints(groupSpec) <- 25
> groupFrontier <- portfolioFrontier(data = lppData, spec = groupSpec,
     constraints = groupConstraints)
> tailoredFrontierPlot(object = groupFrontier, mText = "MV Portfolio - Group
     Constraints",
     risk = "Cov")
```

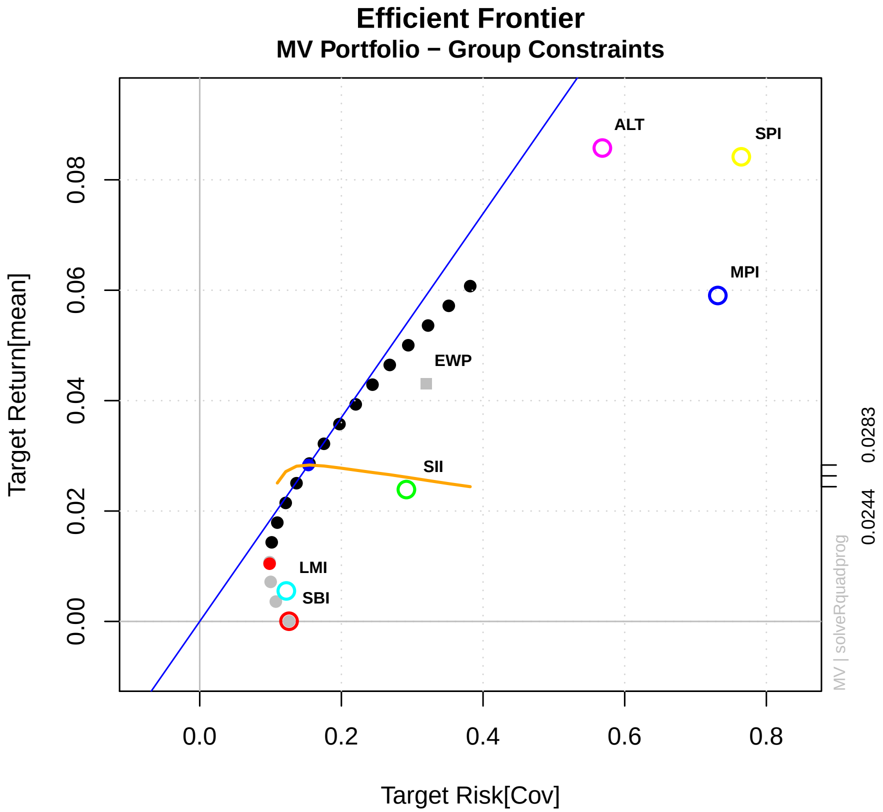


Figure 18.7 Efficient frontier of a group-constrained mean-variance portfolio: The plot includes the efficient frontier, the tangency line and tangency point for a zero risk-free rate, the equal weights portfolio, EWP, all single assets risk vs. return points. The line of Sharpe ratios is also shown, with its maximum coinciding with the tangency portfolio point. The range of the Sharpe ratio is printed on the right hand side axis of the plot.

The efficient frontier of the group-constrained MV portfolio is shown in Figure 18.7. The corresponding weights, weighted returns and covariance risk budgets are shown in the right-hand column of Figure 18.7.

```
> weightsPlot(groupFrontier, mtext = FALSE)
> text <- "MV Portfolio - Group Constrained Portfolio"
> mtext(text, side = 3, line = 3, font = 2, cex = 0.9)
> weightedReturnsPlot(groupFrontier, mtext = FALSE)
> covRiskBudgetsPlot(groupFrontier, mtext = FALSE)
```

18.6 The Box/Group-Constrained Portfolio Frontier

Box and group constraints can be combined

```
> boxgroupSpec <- portfolioSpec()
> setNFrontierPoints(boxgroupSpec) <- 15
> boxgroupConstraints <- c(boxConstraints,
                           groupConstraints)
> boxgroupFrontier <- portfolioFrontier(
     data = lppData,
     spec = boxgroupSpec,
     constraints = boxgroupConstraints)
> print(boxgroupFrontier)
```

```
Title:
 MV Portfolio Frontier
 Estimator:         covEstimator
 Solver:            solveRquadprog
 Optimize:          minRisk
 Constraints:       minW maxW minsumW maxsumW
 Portfolio Points:  4 of 4

Portfolio Weights:
     SBI    SPI    SII    LMI    MPI    ALT
1 0.1000 0.1000 0.1836 0.4032 0.1000 0.1133
2 0.1000 0.1000 0.2140 0.3034 0.1000 0.1826
3 0.1000 0.1000 0.2445 0.2036 0.1000 0.2520
4 0.1000 0.1000 0.1501 0.2000 0.1000 0.3499

Covariance Risk Budgets:
      SBI     SPI     SII     LMI     MPI     ALT
```

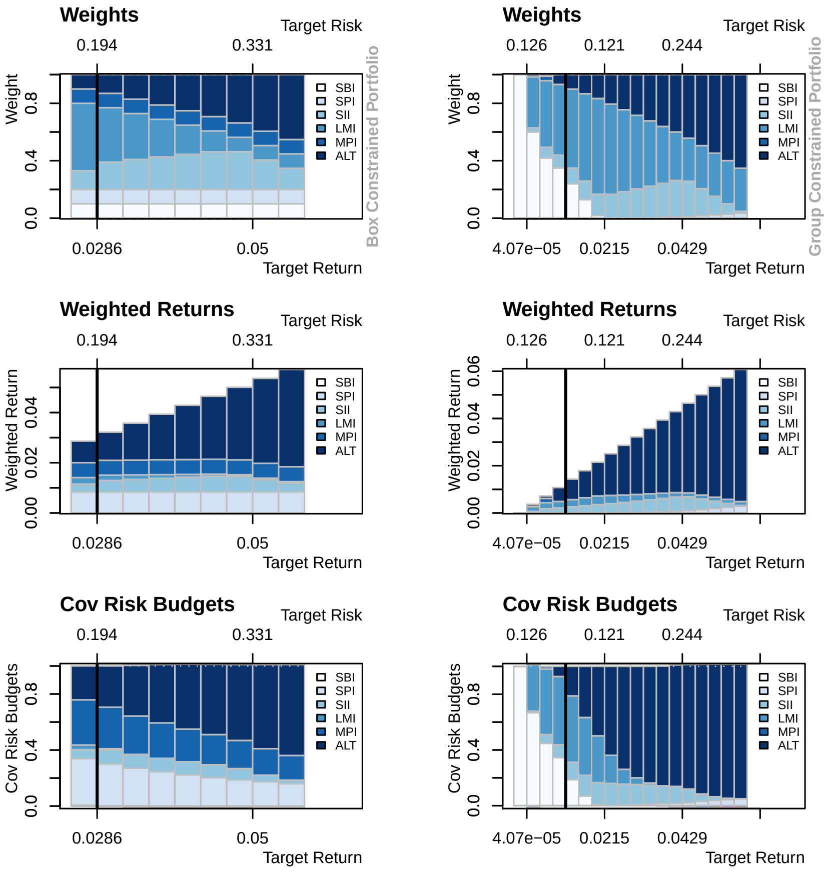


Figure 18.8 MV box (left) and group (right) constrained weights, weighted returns, and covariance risk budgets plots along the frontier.

```
1  0.0036  0.3125  0.1019  0.0142  0.3066  0.2613
2 -0.0006  0.2628  0.0981 -0.0060  0.2677  0.3780
3 -0.0027  0.2229  0.0937 -0.0098  0.2332  0.4627
4 -0.0035  0.1921  0.0340 -0.0106  0.2065  0.5815

Target Return and Risks:
    mean     mu    Cov  Sigma   CVaR    VaR
1 0.0307 0.0307 0.2056 0.2056 0.4804 0.2935
2 0.0368 0.0368 0.2429 0.2429 0.5695 0.3553
3 0.0429 0.0429 0.2826 0.2826 0.6631 0.3983
4 0.0490 0.0490 0.3263 0.3263 0.7848 0.5081

Description:
 Mon May  4 13:44:52 2009 by user: Rmetrics
```

For the plot of the frontier we reset the number of frontier points to 25 and recalculate the frontier.

```
> boxgroupSpec <- portfolioSpec()
> setNFrontierPoints(boxgroupSpec) <- 25
> boxgroupFrontier <- portfolioFrontier(
     data = lppData,
     spec = boxgroupSpec,
     constraints = boxgroupConstraints)
> tailoredFrontierPlot(
     object = boxgroupFrontier,
     mText = "MV Portfolio - Box/Group Constraints",
     risk = "Cov")
```

```
> weightsPlot(boxgroupFrontier, mtext = FALSE)
> text <- "MV Portfolio - Box/Group Constrained Portfolio"
> mtext(text, side = 3, line = 3, font = 2, cex = 0.9)
> weightedReturnsPlot(boxgroupFrontier, mtext = FALSE)
> covRiskBudgetsPlot(boxgroupFrontier, mtext = FALSE)
```

The results of combining box and group constraints are shown in Figure 18.9 and Figure 18.10.

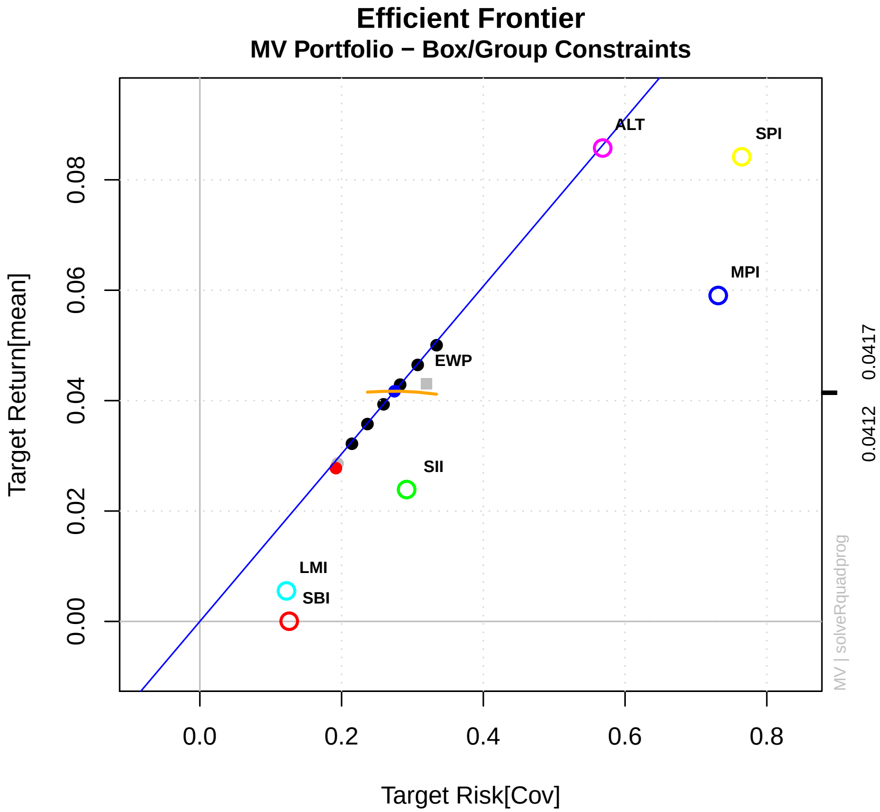


Figure 18.9 Efficient frontier of a box/group constrained mean-variance portfolio: The plot includes the efficient frontier, the tangency line and tangency point for a zero risk-free rate, the equal weights portfolio, EWP, all single assets risk vs. return points. The line of Sharpe ratios is also shown, with its maximum coinciding with the tangency portfolio point. The range of the Sharpe ratio is printed on the right hand side axis of the plot.

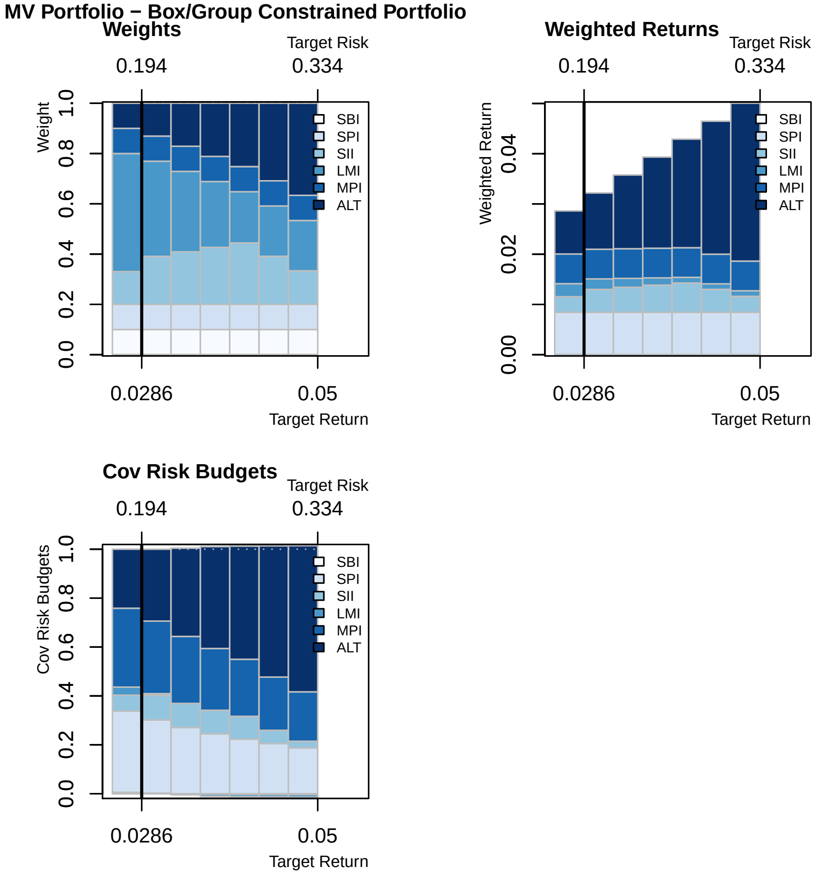


Figure 18.10 Weights along the efficient frontier of a mixed box/group constrained mean-variance portfolio: The graphs from top to bottom show the weights, the weighted returns or in other words the performance attribution, and the covariance risk budgets which are a measure for the risk attribution. The upper axis labels the target risk, and the lower labels the target return. The thick vertical line separates the efficient frontier from the minimum variance locus. The risk axis thus increases in value to both sides of the separator line. The legend to the right links the assets names to colour of the bars.

18.7 Creating Different 'Reward/Risk Views' on the Efficient Frontier

In the efficient frontier plots we have plotted the target return as a function of the covariance risk, expressed as the standard deviation. We can now ask what the efficient frontier looks like when we plot the sample mean versus the conditional Value-at-Risk. The `frontierPlot()` and add-on functions allow you to change the view specifying the arguments for the `return` and the `risk` in the `frontierPlot()` function.

As an example let us plot the efficient frontier for the sample mean return versus the covariance, the CVaR and VaR risk measures. Note that if we specify a risk/reward measure, we have to set the argument `auto=FALSE`, explicitly.

```
> frontierPlot(longFrontier, auto = TRUE)
> frontierPlot(longFrontier, return = "mean", risk = "Cov",
    auto = FALSE)
> frontierPlot(longFrontier, return = "mean", risk = "CVaR",
    auto = FALSE)
> frontierPlot(longFrontier, return = "mean", risk = "VaR",
    auto = FALSE)
```

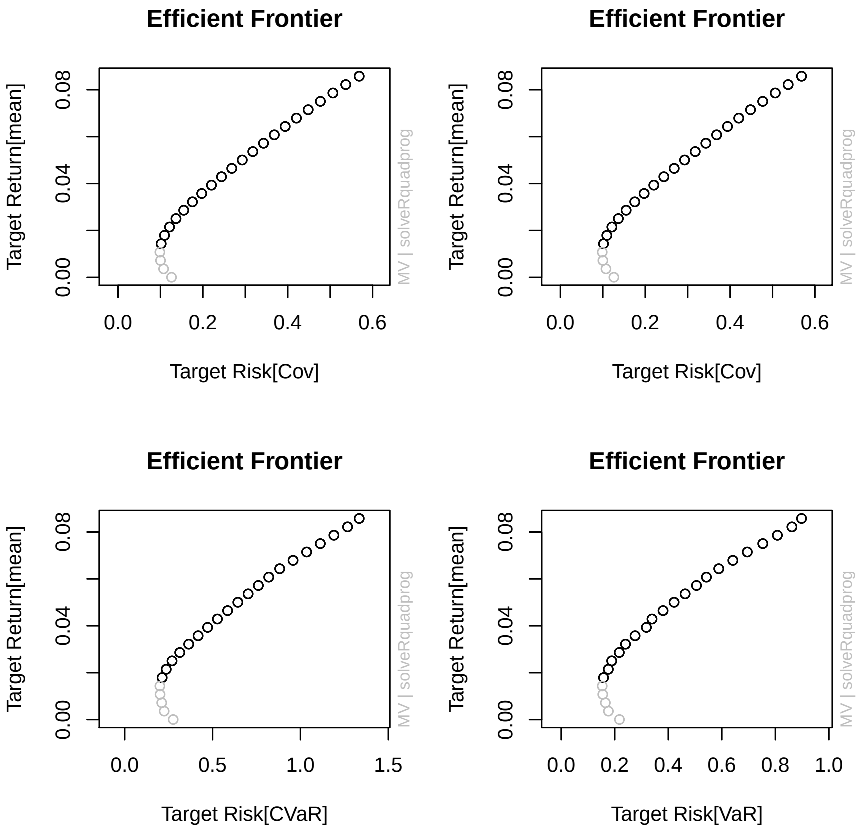


Figure 18.11 MV long-only constrained frontier plots for different risk measures: *Upper left*, the default, risk and reward type are selected automatically from the portfolio specifications, *upper right*, the same graph, the mean plotted versus the covariance risk, *lower left*, now the mean return plotted versus the conditional value at risk, and *lower right*, now the mean return plotted versus the value-at-risk.

Chapter 19

Case Study: Dow Jones Index

Required R package(s):

```
> library(fPortfolio)
> library(fEcofin)
> library(corpcor)
```

In this chapter we have prepared a real-world case study for optimizing a portfolio with the 30 shares given in the Dow Jones Index. We use the `DowJones30` data set provided in the `fEcofin` Package.

1. Load the data set

```
> djiData = as.timeSeries(DowJones30)
> djiData.ret <- 100 * returns(djiData)
> colnames(djiData)

 [1] "AA"   "AXP"  "T"    "BA"   "CAT"  "C"    "KO"   "DD"   "EK"   "XOM"
[11] "GE"   "GM"   "HWP"  "HD"   "HON"  "INTC" "IBM"  "IP"   "JPM"  "JNJ"
[21] "MCD"  "MRK"  "MSFT" "MMM"  "MO"   "PG"   "SBC"  "UTX"  "WMT"  "DIS"

> c(start(djiData), end(djiData))

GMT
[1] [1990-12-31] [2001-01-02]
```

The data cover 10 years of daily data. If you would like to use more recent data, please feel free to update the data from Yahoo Finance[1].

2. Perform an exploratory data analysis

Explore the returns series, and the series of share prices. Then investigate pairwise dependencies between the asset returns, including correlations and distributional properties from star plots. Which of the shares are similar or dissimilar? Use hierarchical clustering and a PCA analysis of the equities.

```
> par(mfrow = c(1, 1), ask = TRUE)
> for (i in 1:3) plot(djiData.ret[, (10 * i - 9):(10 * i)])
> for (i in 1:3) plot(djiData[, (10 * i - 9):(10 * i)])
> assetsCorImagePlot(djiData.ret)
> plot(assetsSelect(djiData.ret))
> assetsCorEigenPlot(djiData.ret)
```

3. Find the optimal weights for a long-only MV portfolio

Apply the mean-variance portfolio approach to explore the efficient frontier and to display the weights along the frontier.

```
> frontier <- portfolioFrontier(djiData.ret)
> tailoredFrontierPlot(frontier)
> weightsPlot(frontier)
```

4. Find the optimal weights for a group-constrained MV portfolio

Perform a clustering of the equities, grouping the data into 5 clusters. Limit the investment for each cluster to a maximum of 50%.

```
> selection <- assetsSelect(djiData.ret, method = "kmeans")
> cluster <- selection$cluster
> cluster[cluster == 1]
```

[1] http://finance.yahoo.com/

```
AXP   C JPM
  1   1   1

> cluster[cluster == 2]

 HD WMT
  2   2

> cluster[cluster == 3]

 AA  BA CAT  DD  GM HON  IP MMM UTX
  3   3   3   3   3   3   3   3   3

> cluster[cluster == 4]

  T  KO  EK XOM  GE JNJ MCD MRK  MO  PG SBC DIS
  4   4   4   4   4   4   4   4   4   4   4   4

> cluster[cluster == 5]

 HWP INTC  IBM MSFT
   5    5    5    5

> constraints <- c(
     'maxsumW[c("BA","DD","EK","XOM","GM","HON","MMM","UTX")] = 0.30',
     'maxsumW[c(T","KO","GE","HD","JNJ","MCD","MRK","MO","PG","SBC","WMT","DIS
     ")] = 0.30',
     'maxsumW[c(AXP","C","JPM")] = 0.30',
     'maxsumW[c(AA","CAT","IP")] = 0.30',
     'maxsumW[c(HWP","INTC","IBM","MSFT")] = 0.30')
```

Estimate the covariance matrix using the shrinkage estimator and compute the weights along the frontier. The weights are shown in Figure 19.1.

```
> djiSpec <- portfolioSpec()
> setNFrontierPoints(djiSpec) <- 25
> setEstimator(djiSpec) <- "shrinkEstimator"
> djiFrontier <- portfolioFrontier(djiData.ret, djiSpec)
> col = seqPalette(30, "YlOrRd")
> weightsPlot(djiFrontier, col = col)
```

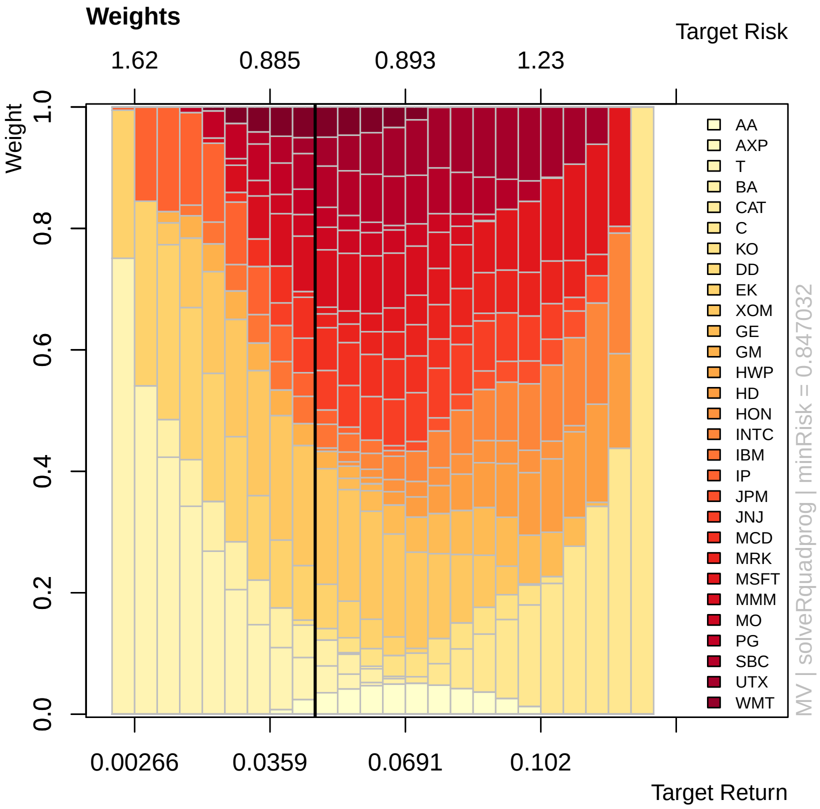


Figure 19.1 The graph shows the weights along the DJI mean-variance frontier. The legend to the right links the equity names to colour of the bars.

Chapter 20

Robust Portfolios and Covariance Estimation

Required R package(s):

```
> library(fPortfolio)
> library(robustbase)
> library(corpcor)
```

Mean-variance portfolios constructed using the sample mean and covariance matrix of asset returns often perform poorly out-of-sample due to estimation errors in the mean vector and covariance matrix. As a consequence, minimum-variance portfolios may yield unstable weights that fluctuate substantially over time. This loss of stability may also lead to extreme portfolio weights and dramatic swings in weights with only minor changes in expected returns or the covariance matrix. Consequentially, we observe frequent re-balancing and excessive transaction costs.

To achieve better stability properties compared to traditional minimum-variance portfolios, we try to reduce the estimation error using robust methods to compute the mean and/or covariance matrix of the set of financial assets. Two different approaches are implemented: robust mean and covariance estimators, and the shrinkage estimator[1].

If the number of time series records is small and the number of considered assets increases, then the sample estimator of covariance becomes more and more unstable. Specifically, it is possible to provide estimators that improve considerably upon the maximum likelihood estimate in terms of mean-squared error. Moreover, when the number of records is smaller

[1] For further information, we recommend the text book by Marazzi (1993)

than the number of assets, the empirical estimate of the covariance matrix becomes singular.

20.1 Robust Mean and Covariance Estimators

In the mean-variance portfolio approach, the sample mean and sample covariance estimators are used by default to estimate the mean vector and covariance matrix.

This information, i.e. the name of the covariance estimator function, is kept in the specification structure and can be shown by calling the function `getEstimator()`. The default setting is

```
> getEstimator(portfolioSpec())
```

```
[1] "covEstimator"
```

There are many different implementations of robust and related estimators for the mean and covariance in R's base packages and in contributed packages. The estimators listed below can be accessed by the portfolio optimization program.

```
Functions:
covEstimator        Covariance sample estimator
kendallEstimator    Kendall's rank estimator
spearmanEstimator   Spearman's rank estimator
mcdEstimator        MCD, minimum covariance determinant estimator
mveEstimator        MVE, minimum volume ellipsoid estimator
covMcdEstimator     Minimum covariance determinant estimator
covOGKEstimator     Orthogonalized Gnanadesikan-Kettenring estimator
shrinkEstimator     Shrinkage covariance estimator
baggedEstimator     Bagged covariance estimator
```

Listing 20.1 Rmetrics functions to estimate robust covariances for portfolio optimization

20.2 The MCD Robustified Mean-Variance Portfolio

The *minimum covariance determinant*, MCD, estimator of location and scatter looks for the $h > n/2$ observations out of n data records whose classical covariance matrix has the lowest possible determinant. The raw MCD estimate of location is then the average of these h points, whereas the raw MCD estimate of scatter is their covariance matrix, multiplied by a consistency factor and a finite sample correction factor (to make it consistent with the normal model and unbiased for small sample sizes).

The algorithm from the `MASS` library is quite slow, whereas the one from contributed package `robustbase` (Rousseeuw et al., 2008) is much more time-efficient. The implementation in `robustbase` uses the fast MCD algorithm of Rousseeuw & Van Driessen (1999). To optimize a Markowitz mean-variance portfolio, we just have to specify the name of the mean/covariance estimator function. Unfortunately, this can take some time since we have to apply the MCD estimator in every instance when we call the function `covMcdEstimator()`. To circumvent this, we perform the covariance estimation only once at the very beginning, store the value globally, and use its estimate in the new function `fastCovMcdEstimator()`.

```
> lppData <- 100 * LPP2005.RET[, 1:6]
> covMcdEstimate <- covMcdEstimator(lppData)
> fastCovMcdEstimator <-
     function(x, spec = NULL, ...)
     covMcdEstimate
```

Next we define the portfolio specification

```
> covMcdSpec <- portfolioSpec()
> setEstimator(covMcdSpec) <- "fastCovMcdEstimator"
> setNFrontierPoints(covMcdSpec) <- 5
```

and optimize the MCD robustified portfolio (with long-only default constraints).

```
> covMcdFrontier <- portfolioFrontier(
     data = lppData, spec = covMcdSpec)
> print(covMcdFrontier)

Title:
 MV Portfolio Frontier
```

```
 Estimator:         fastCovMcdEstimator
 Solver:            solveRquadprog
 Optimize:          minRisk
 Constraints:       LongOnly
 Portfolio Points:  5 of 5

Portfolio Weights:
     SBI    SPI    SII    LMI    MPI    ALT
1 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000
2 0.1379 0.0377 0.1258 0.5562 0.0000 0.1424
3 0.0000 0.0998 0.2088 0.3712 0.0000 0.3202
4 0.0000 0.1661 0.2864 0.0430 0.0000 0.5046
5 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000

Covariance Risk Budgets:
      SBI     SPI     SII     LMI     MPI     ALT
1  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000
2  0.0492  0.1434  0.1209  0.2452  0.0000  0.4413
3  0.0000  0.2489  0.0878 -0.0071  0.0000  0.6704
4  0.0000  0.2624  0.0660 -0.0027  0.0000  0.6743
5  0.0000  0.0000  0.0000  0.0000  0.0000  1.0000

Target Return and Risks:
    mean     mu    Cov  Sigma   CVaR    VaR
1 0.0000 0.0000 0.1261 0.1304 0.2758 0.2177
2 0.0215 0.0215 0.1242 0.1153 0.2552 0.1733
3 0.0429 0.0429 0.2493 0.2117 0.5698 0.3561
4 0.0643 0.0643 0.4023 0.3363 0.9504 0.5574
5 0.0858 0.0858 0.5684 0.5016 1.3343 0.8978

Description:
 Mon May  4 12:04:50 2009 by user: Rmetrics
```

Note that for the Swiss Pension Fund benchmark data set the `"covMcdEstimator"` is about 20 time slower than the sample covariance estimator, and the `"mcdEstimator"` is even slower by a factor of about 300.

For the plot we recalculate the frontier on 20 frontier points.

```
> setNFrontierPoints(covMcdSpec) <- 20
> covMcdFrontier <- portfolioFrontier(
     data = lppData, spec = covMcdSpec)
> tailoredFrontierPlot(
     covMcdFrontier,
     mText = "MCD Robustified MV Portfolio",
     risk = "Sigma")
```

The frontier plot is shown in Figure 20.1.

To display the weights, risk attributions and covariance risk budgets for the MCD robustified portfolio in the left-hand column and the same plots for the sample covariance MV portfolio in the right-hand column of a figure:

```
> ## MCD robustified portfolio
> par(mfcol = c(3, 2), mar = c(3.5, 4, 4, 3) + 0.1)
> col = qualiPalette(30, "Dark2")
> weightsPlot(covMcdFrontier, mtext = FALSE, col = col)
> text <- "MCD"
> mtext(text, side = 3, line = 3, font = 2, cex = 0.9)
> weightedReturnsPlot(covMcdFrontier, mtext = FALSE, col = col)
> covRiskBudgetsPlot(covMcdFrontier, mtext = FALSE, col = col)
> ## Sample covariance MV portfolio
> longSpec <- portfolioSpec()
> setNFrontierPoints(longSpec) <- 20
> longFrontier <- portfolioFrontier(data = lppData, spec = longSpec)
> col = qualiPalette(30, "Set1")
> weightsPlot(longFrontier, mtext = FALSE, col = col)
> text <- "COV"
> mtext(text, side = 3, line = 3, font = 2, cex = 0.9)
> weightedReturnsPlot(longFrontier, mtext = FALSE, col = col)
> covRiskBudgetsPlot(longFrontier, mtext = FALSE, col = col)
```

The weights, risk attributions and covariance risk budgets are shown in Figure 20.2.

20.3 The MVE Robustified Mean-Variance Portfolio

Rousseeuw & Leroy (1987) proposed a very robust alternative to classical estimates of mean vectors and covariance matrices, the Minimum Volume Ellipsoid, MVE. Samples from a multivariate normal distribution form ellipsoid-shaped ‘clouds’ of data points. The MVE corresponds to the smallest point cloud containing at least half of the observations, the uncontaminated portion of the data. These ‘clean’ observations are used for preliminary estimates of the mean vector and the covariance matrix. Using these estimates, the program computes a robust Mahalanobis distance for every observation vector in the sample. Observations for which the robust

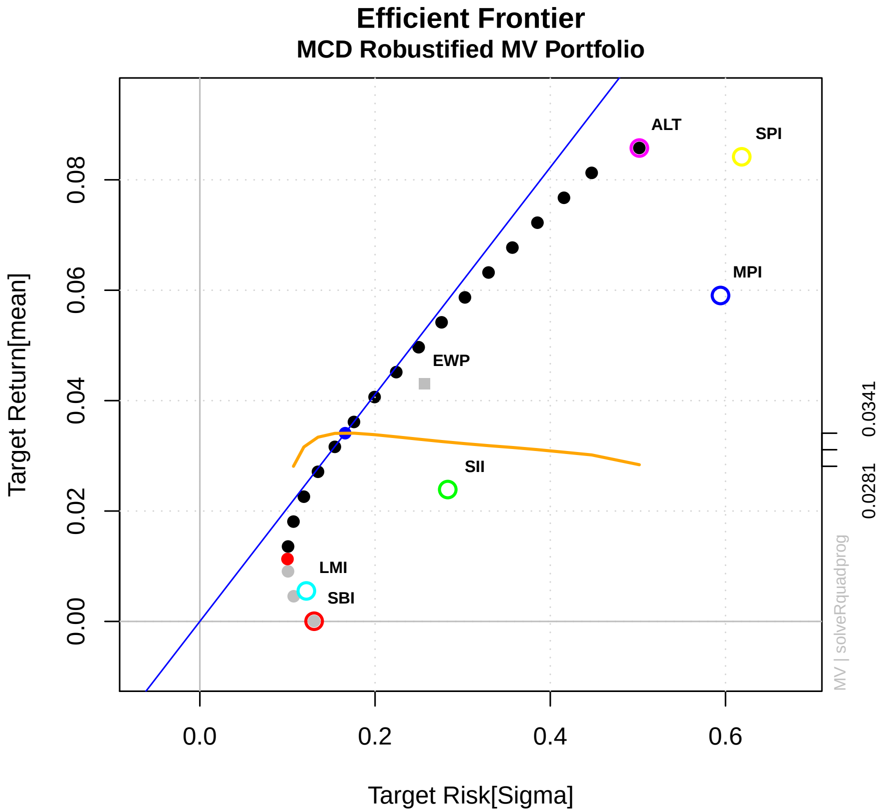


Figure 20.1 Efficient frontier of a long-only constrained mean-variance portfolio with robust MCD covariance estimates: The plot includes the efficient frontier, the tangency line and tangency point for a zero risk-free rate, the equal weights portfolio, EWP, all single assets risk vs. return points. The line of Sharpe ratios is also shown, with its maximum coinciding with the tangency portfolio point. The range of the Sharpe ratio is printed on the right hand side axis of the plot.

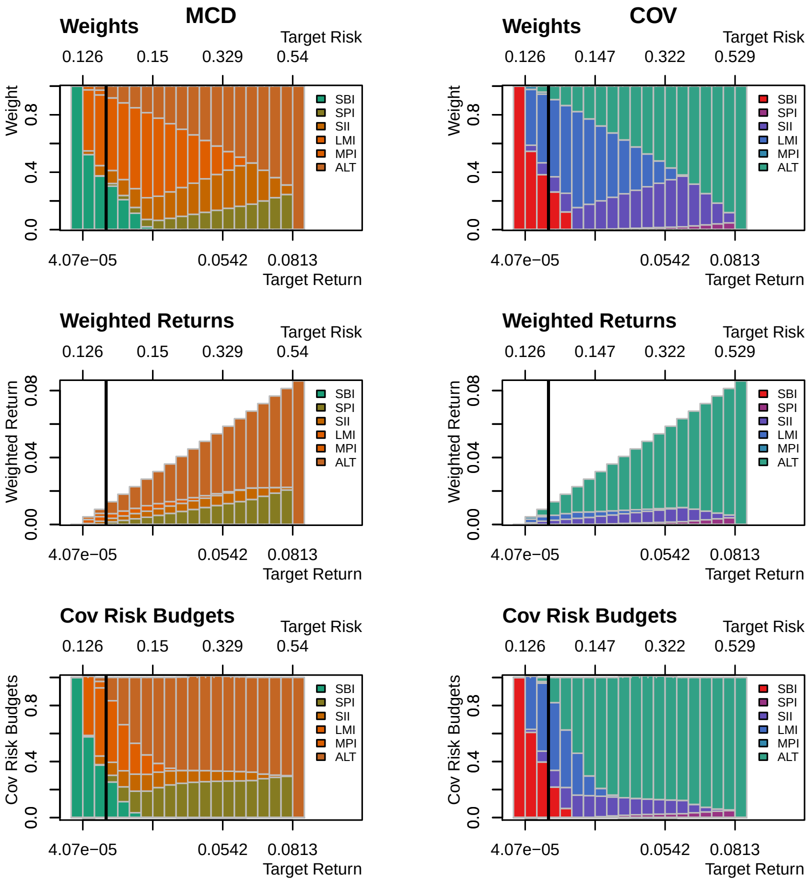


Figure 20.2 Weights plot for MCD robustified and COV MV portfolios. Weights along the efficient frontier of a long-only constrained mean-variance portfolio with robust MCD (left) and sample (right) covariance estimates: The graphs from top to bottom show the weights, the weighted returns or in other words the performance attribution, and the covariance risk budgets, which are a measure for the risk attribution. The upper axis labels the target risk, and the lower labels the target return. The thick vertical line separates the efficient frontier from the minimum variance locus. The risk axis thus increases in value to both sides of the separator line. The legend to the right links the assets names to colour of the bars.

Mahalanobis distances exceed the 97.5% significance level for the chi-square distribution are flagged as probable outliers.

Rmetrics provides a function, `mveEstimator()`, to compute the MVE estimator; it is based on the `cov.rob()` estimator from the `MASS` package. We define a function called `fastMveEstimator()`

```
> mveEstimate <- mveEstimator(lppData)
> fastMveEstimator <- function(x, spec = NULL, ...) mveEstimate
```

and set the portfolio specifications

```
> mveSpec <- portfolioSpec()
> setEstimator(mveSpec) <- "fastMveEstimator"
> setNFrontierPoints(mveSpec) <- 5
```

Then we compute the MVE robustified efficient frontier

```
> mveFrontier <- portfolioFrontier(
     data = lppData,
     spec = mveSpec,
     constraints = "LongOnly")
> print(mveFrontier)

Title:
 MV Portfolio Frontier
 Estimator:         fastMveEstimator
 Solver:            solveRquadprog
 Optimize:          minRisk
 Constraints:       LongOnly
 Portfolio Points:  5 of 5

Portfolio Weights:
     SBI    SPI    SII    LMI    MPI    ALT
1 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000
2 0.1188 0.0271 0.1520 0.5566 0.0000 0.1455
3 0.0000 0.0709 0.2643 0.3290 0.0000 0.3358
4 0.0000 0.1196 0.3433 0.0000 0.0000 0.5371
5 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000

Covariance Risk Budgets:
      SBI     SPI     SII     LMI     MPI     ALT
1  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000
2  0.0420  0.0974  0.1682  0.2477  0.0000  0.4447
3  0.0000  0.1693  0.1313 -0.0088  0.0000  0.7082
```

```
4  0.0000  0.1819  0.0894  0.0000  0.0000  0.7287
5  0.0000  0.0000  0.0000  0.0000  0.0000  1.0000

Target Return and Risks:
    mean     mu    Cov  Sigma   CVaR    VaR
1 0.0000 0.0000 0.1261 0.1229 0.2758 0.2177
2 0.0215 0.0215 0.1230 0.1094 0.2468 0.1728
3 0.0429 0.0429 0.2465 0.2024 0.5479 0.3459
4 0.0643 0.0643 0.3977 0.3221 0.9183 0.5535
5 0.0858 0.0858 0.5684 0.4781 1.3343 0.8978

Description:
 Mon May  4 12:04:54 2009 by user: Rmetrics
```

For the frontier plot, we recompute the robustified frontier on 20 points.

```
> setNFrontierPoints(mveSpec) <- 20
> mveFrontier <- portfolioFrontier(
     data = lppData, spec = mveSpec)
> tailoredFrontierPlot(
     mveFrontier,
     mText = "MVE Robustified MV Portfolio",
     risk = "Sigma")
```

The frontier plot is shown in Figure 20.3.

To complete this section, we will show the weights and the performance and risk attribution plots (left-hand column of Figure 20.4).

```
> col = divPalette(6, "RdBu")
> weightsPlot(mveFrontier, col = col,
     mtext = FALSE)
> boxL()
> text <- "MVE Robustified MV Portfolio"
> mtext(text, side = 3, line = 3, font = 2, cex = 0.9)
> weightedReturnsPlot(mveFrontier, col = col,
     mtext = FALSE)
> boxL()
> covRiskBudgetsPlot(mveFrontier, col = col,
     mtext = FALSE)
> boxL()
```

For the colours we have chosen a diverging red to blue palette. The `boxL()` function draws an alternative frame around the graph with axes to the left and bottom.

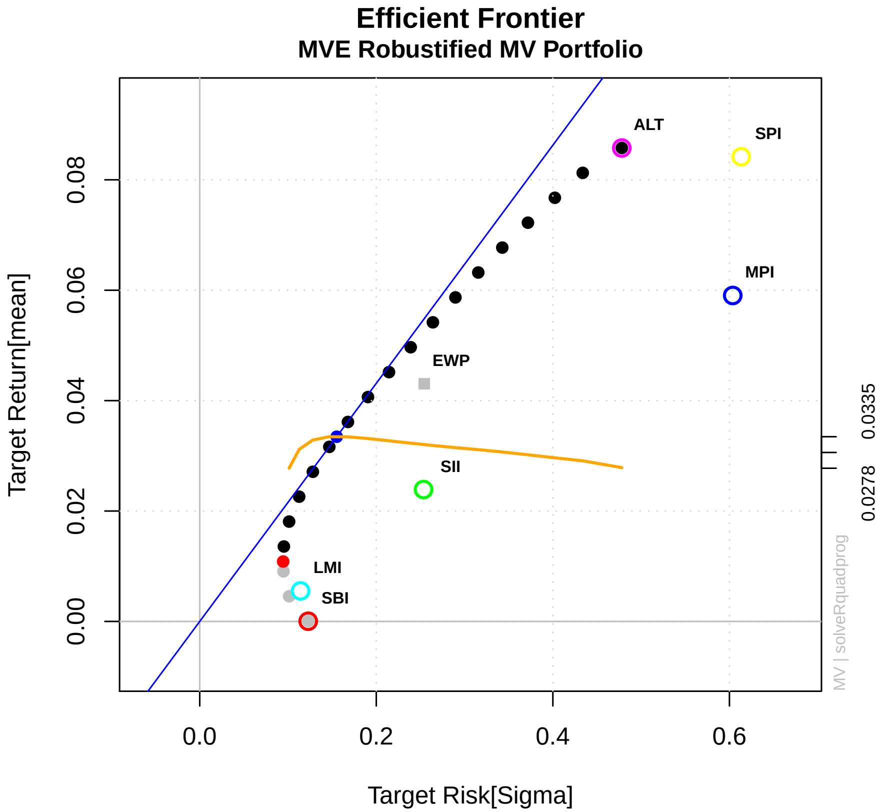


Figure 20.3 Efficient frontier of a long-only constrained mean-variance portfolio with robust MVE covariance estimates: The plot includes the efficient frontier, the tangency line and tangency point for a zero risk-free rate, the equal weights portfolio, EWP, all single assets risk vs. return points. The line of Sharpe ratios is also shown, with its maximum coinciding with the tangency portfolio point. The range of the Sharpe ratio is printed on the right hand side axis of the plot.

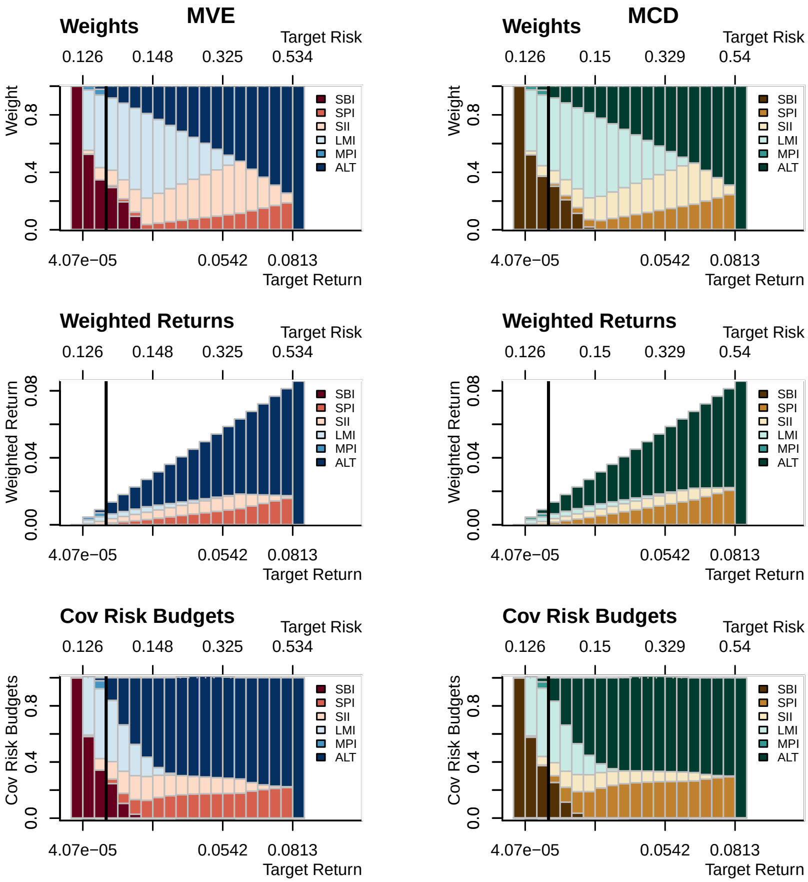


Figure 20.4 Weights along the efficient frontier of a long-only constrained mean-variance portfolio with robust MVE (left) and MCD (right) covariance estimates: The graphs from top to bottom show the weights, the weighted returns or in other words the performance attribution, and the covariance risk budgets which are a measure for the risk attribution. The upper axis labels the target risk, and the lower labels the target return. The thick vertical line separates the efficient frontier from the minimum variance locus. The risk axis thus increases in value to both sides of the separator line. The legend to the right links the assets names to colour of the bars. Note that the comparison of weights between the MVE and MCD with sample covariance estimates shows a much better diversification of the portfolio weights and also leads to a better diversification of the covariance risk budgets.

20.4 The OGK Robustified Mean-Variance Portfolio

The Orthogonalized Gnanadesikan-Kettenring (OGK) estimator computes the orthogonalized pairwise covariance matrix estimate described in Maronna & Zamar (2002). The pairwise proposal goes back to Gnanadesikan & Kettenring (1972).

We first write a fast estimator function, `fastCovOGKEstimator()`

```
> covOGKEstimate <- covOGKEstimator(lppData)
> fastCovOGKEstimator <- function(x, spec = NULL, ...) covOGKEstimate
```

then we set the portfolio specification

```
> covOGKSpec <- portfolioSpec()
> setEstimator(covOGKSpec) <- "fastCovOGKEstimator"
> setNFrontierPoints(covOGKSpec) <- 5
```

and finally we compute the OGK robustified frontier

```
> covOGKFrontier <- portfolioFrontier(
     data = lppData, spec = covOGKSpec)
> print(covOGKFrontier)

Title:
 MV Portfolio Frontier
 Estimator:         fastCovOGKEstimator
 Solver:            solveRquadprog
 Optimize:          minRisk
 Constraints:       LongOnly
 Portfolio Points:  5 of 5

Portfolio Weights:
     SBI    SPI    SII    LMI    MPI    ALT
1 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000
2 0.0990 0.0171 0.1593 0.5723 0.0000 0.1522
3 0.0000 0.0650 0.2661 0.3277 0.0000 0.3411
4 0.0000 0.1179 0.3433 0.0000 0.0000 0.5388
5 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000

Covariance Risk Budgets:
      SBI     SPI     SII     LMI     MPI     ALT
1  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000
2  0.0347  0.0583  0.1827  0.2605  0.0000  0.4639
```

```
3  0.0000  0.1540  0.1329 -0.0089  0.0000  0.7221
4  0.0000  0.1790  0.0895  0.0000  0.0000  0.7315
5  0.0000  0.0000  0.0000  0.0000  0.0000  1.0000

Target Return and Risks:
    mean     mu    Cov  Sigma   CVaR    VaR
1 0.0000 0.0000 0.1261 0.1270 0.2758 0.2177
2 0.0215 0.0215 0.1223 0.1197 0.2419 0.1741
3 0.0429 0.0429 0.2460 0.2222 0.5450 0.3418
4 0.0643 0.0643 0.3976 0.3532 0.9175 0.5523
5 0.0858 0.0858 0.5684 0.5236 1.3343 0.8978

Description:
 Mon May  4 12:04:56 2009 by user: Rmetrics

> setNFrontierPoints(covOGKSpec) <- 20
> covOGKFrontier <- portfolioFrontier(
     data = lppData, spec = covOGKSpec)
> tailoredFrontierPlot(
     covOGKFrontier,
     mText = "OGK Robustified MV Portfolio",
     risk = "Sigma")
```

The frontier plot is shown in Figure 20.5.

The weights, and the performance and risk attributions are shown in the left-hand column of Figure 20.6.

```
> col = divPalette(6, "RdYlGn")
> weightsPlot(covOGKFrontier, col = col, mtext = FALSE)
> text <- "OGK Robustified MV Portfolio"
> mtext(text, side = 3, line = 3, font = 2, cex = 0.9)
> weightedReturnsPlot(covOGKFrontier, col = col, mtext = FALSE)
> covRiskBudgetsPlot(covOGKFrontier, col = col, mtext = FALSE)
```

20.5 The Shrinked Mean-Variance Portfolio

A simple version of a shrinkage estimator of the covariance matrix is constructed as follows. We consider a convex combination of the empirical estimator with some suitable chosen target, e.g., the diagonal matrix. Subsequently, the mixing parameter is selected to maximize the expected accuracy

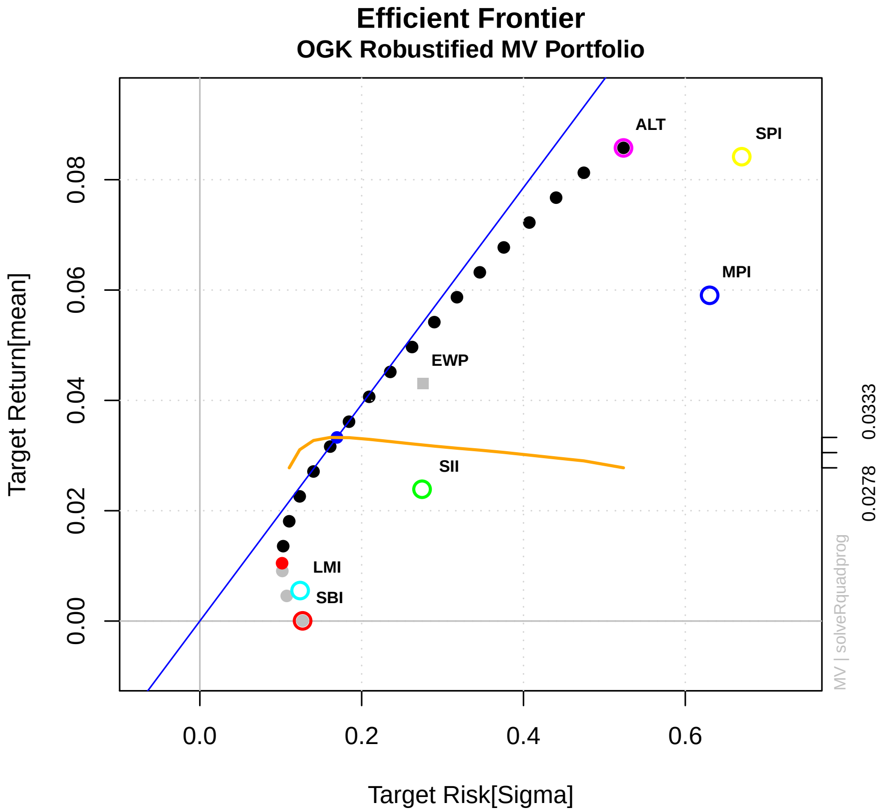


Figure 20.5 Efficient frontier of a long-only constrained mean-variance portfolio with robust OGK covariance estimates: The plot includes the efficient frontier, the tangency line and tangency point for a zero risk-free rate, the equal weights portfolio, EWP, all single assets risk vs. return points. The line of Sharpe ratios is also shown, with its maximum coinciding with the tangency portfolio point. The range of the Sharpe ratio is printed on the right hand side axis of the plot.

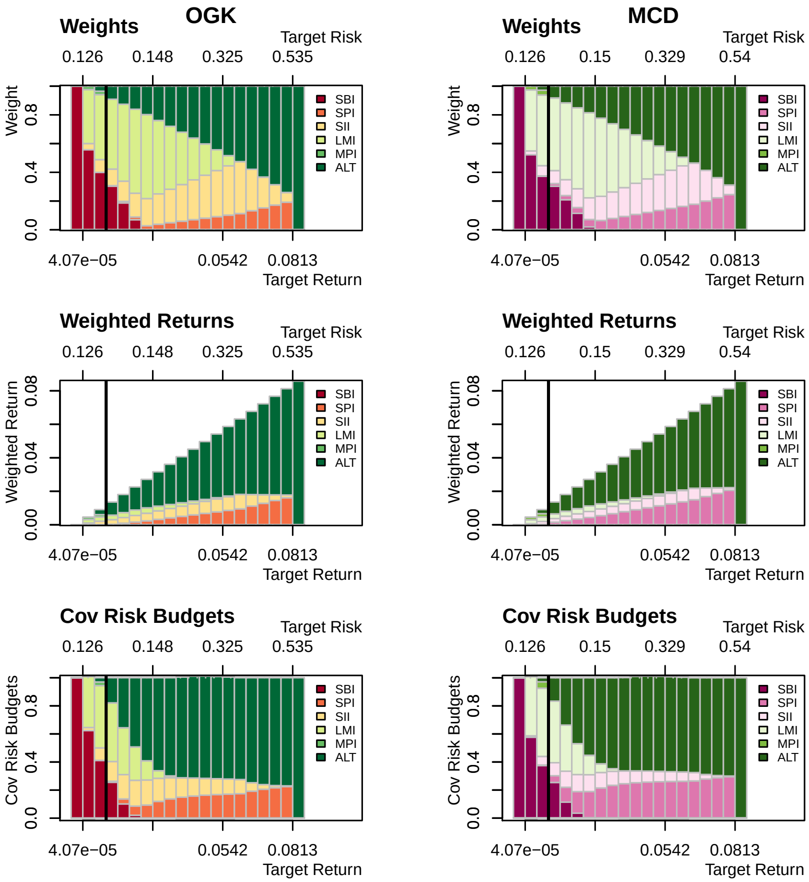


Figure 20.6 Weights along the efficient frontier of a long-only constrained mean-variance portfolio with robust OGK (left) and MCD (right) covariance estimates: The graphs from top to bottom show the weights, the weighted returns or in other words the performance attribution, and the covariance risk budgets which are a measure for the risk attribution. The upper axis labels the target risk, and the lower labels the target return. The thick vertical line separates the efficient frontier from the minimum variance locus. The risk axis thus increases in value to both sides of the separator line. The legend to the right links the assets names to colour of the bars. Note that both estimators result in a similar behaviour concerning the diversification of the weights. A remark, for larger data sets of assets the OGK estimator becomes favourable since it is more computation efficient.

of the shrinked estimator. This can be done by cross-validation, or by using an analytic estimate of the shrinkage intensity. The resulting regularized estimator can be shown to outperform the maximum likelihood estimator for small samples. For large samples, the shrinkage intensity will reduce to zero, therefore in this case the shrinkage estimator will be identical to the empirical estimator. Apart from increased efficiency, the shrinkage estimate has the additional advantage that it is always positive definite and well conditioned, (Schäfer & Strimmer, 2005)[2].

```
> shrinkSpec <- portfolioSpec()
> setEstimator(shrinkSpec) <- "shrinkEstimator"
> setNFrontierPoints(shrinkSpec) <- 5
> shrinkFrontier <- portfolioFrontier(
     data = lppData, spec = shrinkSpec)
> print(shrinkFrontier)

Title:
 MV Portfolio Frontier
 Estimator:         shrinkEstimator
 Solver:            solveRquadprog
 Optimize:          minRisk
 Constraints:       LongOnly
 Portfolio Points:  5 of 5

Portfolio Weights:
     SBI    SPI    SII    LMI    MPI    ALT
1 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000
2 0.1064 0.0022 0.1591 0.5649 0.0000 0.1674
3 0.0000 0.0207 0.2460 0.3441 0.0000 0.3892
4 0.0000 0.0410 0.3290 0.0126 0.0000 0.6174
5 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000

Covariance Risk Budgets:
      SBI     SPI     SII     LMI     MPI     ALT
1  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000
2  0.0378  0.0070  0.1812  0.2553  0.0000  0.5188
3  0.0000  0.0455  0.1154 -0.0094  0.0000  0.8485
4  0.0000  0.0576  0.0823 -0.0009  0.0000  0.8610
5  0.0000  0.0000  0.0000  0.0000  0.0000  1.0000

Target Return and Risks:
    mean     mu    Cov  Sigma   CVaR    VaR
```

[2] The covariance shrinkage estimator we use here is implemented in the Rpackage `corpcor` (Schaefer et al., 2008).

```
1 0.0000 0.0000 0.1261 0.1449 0.2758 0.2177
2 0.0215 0.0215 0.1219 0.1283 0.2386 0.1772
3 0.0429 0.0429 0.2440 0.2430 0.5328 0.3386
4 0.0643 0.0643 0.3941 0.3920 0.8923 0.5868
5 0.0858 0.0858 0.5684 0.5656 1.3343 0.8978

Description:
 Mon May  4 12:04:59 2009 by user: Rmetrics
```

The results are shown in Figure 20.7 and Figure 20.8.

20.6 How to Write Your Own Covariance Estimator

Since we have just to set the name of the mean/covariance estimator function calling the function `setEstimator()` it becomes straightforward to add user-defined covariance estimators.

Let us show an example. In R's recommended package `MASS` there is a function (`cov.trob()`) which estimates a covariance matrix assuming the data come from a multivariate Student's t distribution. This approach provides some degree of robustness to outliers without giving a high breakdown point[3].

```
> covtEstimator <- function (x, spec = NULL, ...) {
     x.mat = as.matrix(x)
     list(mu = colMeans(x.mat), Sigma = MASS::cov.trob(x.mat)$cov) }
> covtSpec <- portfolioSpec()
> setEstimator(covtSpec) <- "covtEstimator"
> setNFrontierPoints(covtSpec) <- 5
> covtFrontier <- portfolioFrontier(
     data = lppData, spec = covtSpec)
> print(covtFrontier)

Title:
 MV Portfolio Frontier
 Estimator:         covtEstimator
 Solver:            solveRquadprog
 Optimize:          minRisk
 Constraints:       LongOnly
```

[3] Intuitively, the breakdown point of an estimator is the proportion of incorrect observations an estimator can handle before giving an arbitrarily unreasonable result

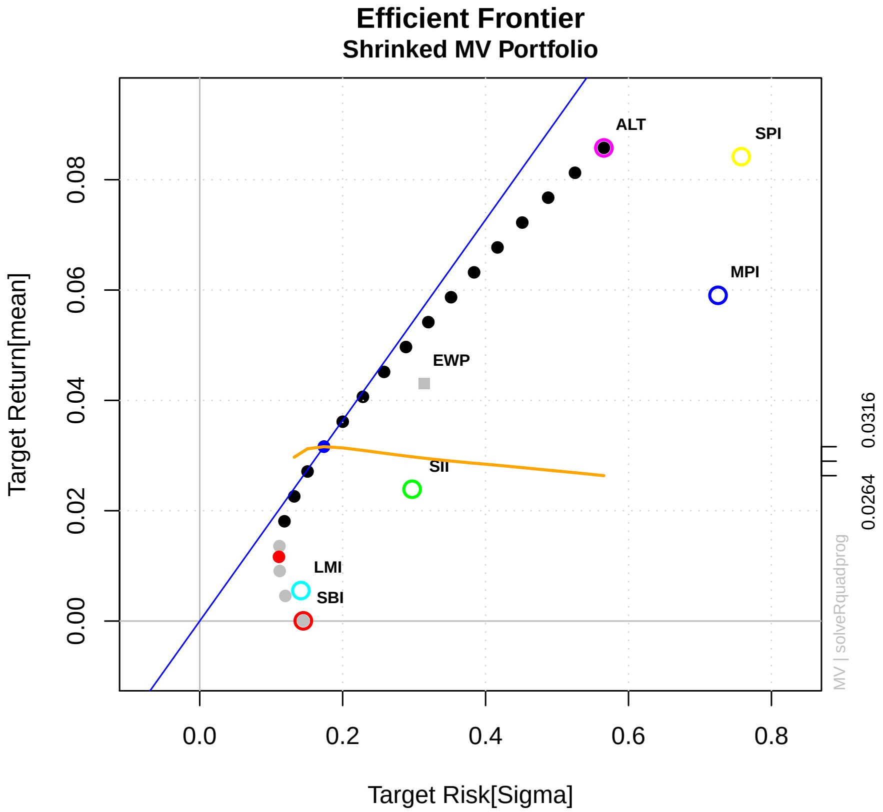


Figure 20.7 Efficient frontier of a long-only constrained mean-variance portfolio with shrinked covariance estimates: The plot includes the efficient frontier, the tangency line and tangency point for a zero risk-free rate, the equal weights portfolio, EWP, all single assets risk vs. return points. The line of Sharpe ratios is also shown, with its maximum coinciding with the tangency portfolio point. The range of the Sharpe ratio is printed on the right hand side axis of the plot.

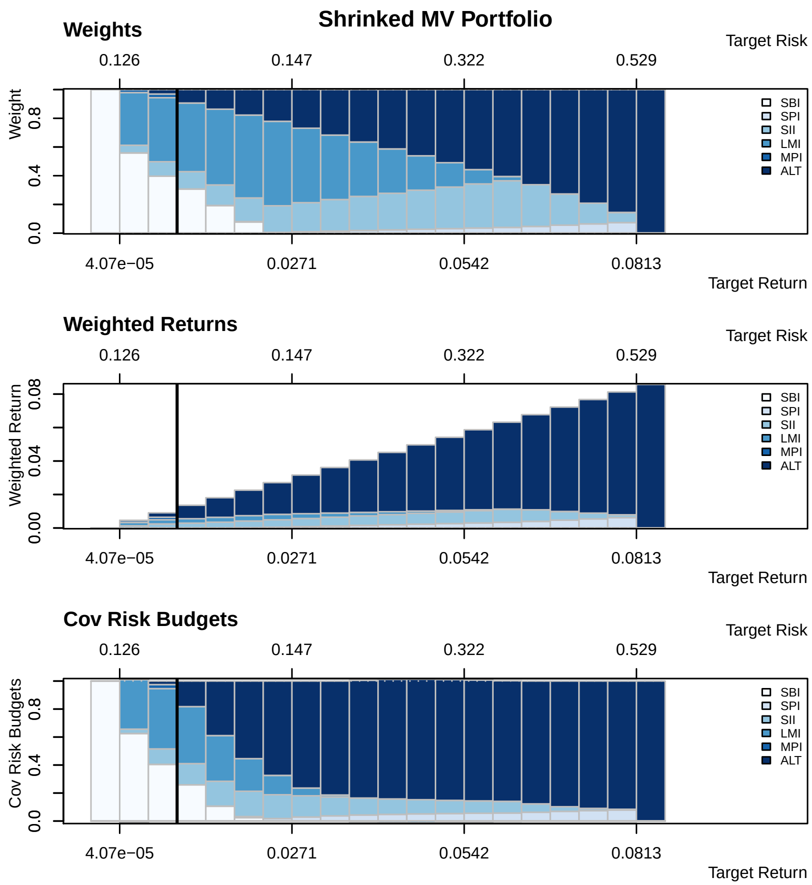


Figure 20.8 Weights along the efficient frontier of a long-only constrained mean-variance portfolio with shrinked covariance estimates: The graphs from top to bottom show the weights, the weighted returns or in other words the performance attribution, and the covariance risk budgets which are a measure for the risk attribution. The upper axis labels the target risk, and the lower labels the target return. The thick vertical line separates the efficient frontier from the minimum variance locus. The risk axis thus increases in value to both sides of the separator line. The legend to the right links the assets names to colour of the bars.

```
 Portfolio Points:  5 of 5

Portfolio Weights:
     SBI    SPI    SII    LMI    MPI    ALT
1 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000
2 0.0749 0.0156 0.1490 0.6061 0.0000 0.1544
3 0.0000 0.0517 0.2479 0.3420 0.0000 0.3583
4 0.0000 0.0896 0.3441 0.0000 0.0000 0.5663
5 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000

Covariance Risk Budgets:
      SBI     SPI     SII     LMI     MPI     ALT
1  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000
2  0.0260  0.0527  0.1627  0.2873  0.0000  0.4714
3  0.0000  0.1205  0.1179 -0.0089  0.0000  0.7706
4  0.0000  0.1326  0.0897  0.0000  0.0000  0.7777
5  0.0000  0.0000  0.0000  0.0000  0.0000  1.0000

Target Return and Risks:
    mean     mu    Cov  Sigma   CVaR    VaR
1 0.0000 0.0000 0.1261 0.1109 0.2758 0.2177
2 0.0215 0.0215 0.1220 0.1043 0.2420 0.1741
3 0.0429 0.0429 0.2451 0.2006 0.5424 0.3432
4 0.0643 0.0643 0.3958 0.3217 0.9066 0.5645
5 0.0858 0.0858 0.5684 0.4697 1.3343 0.8978

Description:
 Mon May  4 12:05:00 2009 by user: Rmetrics

> setNFrontierPoints(covtSpec) <- 20
> covtFrontier <- portfolioFrontier(
     data = lppData, spec = covtSpec)
> tailoredFrontierPlot(
     shrinkFrontier,
     mText = "Student's t MV Portfolio",
     risk = "Sigma")
```

The frontier plot is shown in Figure 20.9. The weights and related plots are computed in the usual way, and presented in Figure 20.10.

```
> weightsPlot(covtFrontier, mtext = FALSE)
> text <- "Student's t"
> mtext(text, side = 3, line = 3, font = 2, cex = 0.9)
> weightedReturnsPlot(covtFrontier, mtext = FALSE)
> covRiskBudgetsPlot(covtFrontier, mtext = FALSE)
```

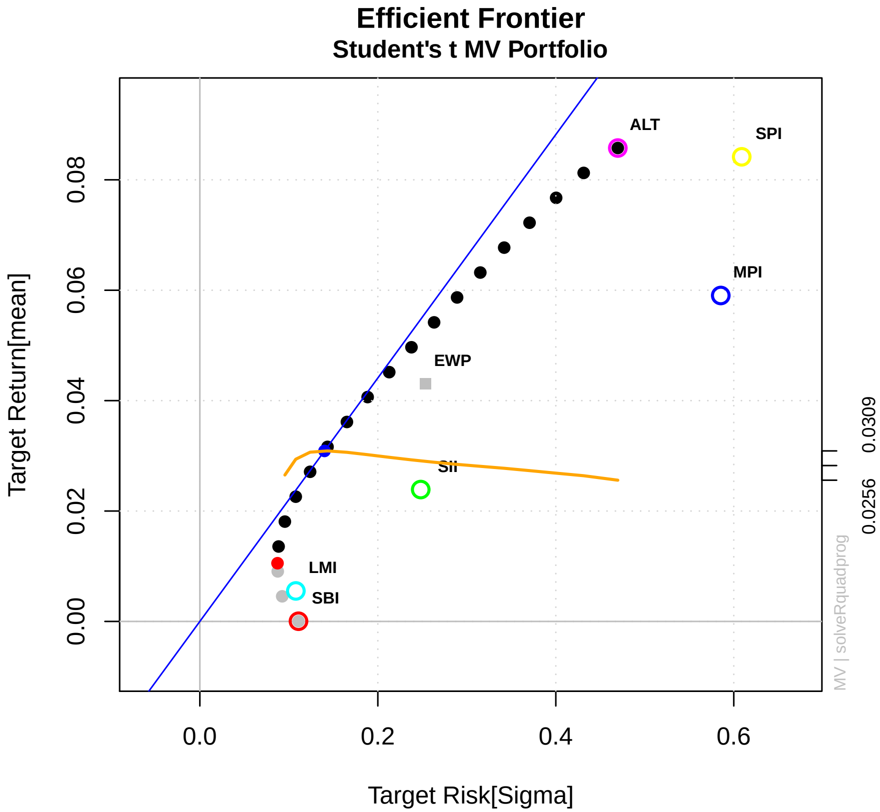


Figure 20.9 Efficient frontier of a long-only constrained mean-variance portfolio with Student's t estimated covariance estimates: The plot includes the efficient frontier, the tangency line and tangency point for a zero risk-free rate, the equal weights portfolio, EWP, all single assets risk vs. return points. The line of Sharpe ratios is also shown, with its maximum coinciding with the tangency portfolio point. The range of the Sharpe ratio is printed on the right hand side axis of the plot.

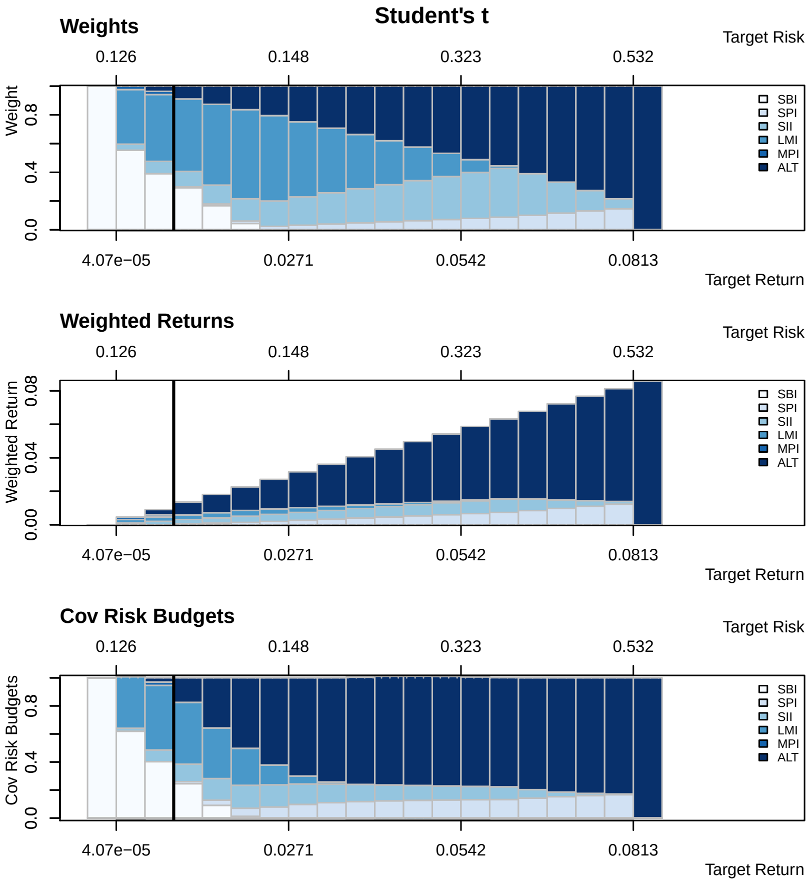


Figure 20.10 Weights along the efficient frontier of a long-only constrained mean-variance portfolio with robust Student's t covariance estimates: The graphs from top to bottom show the weights, the weighted returns or in other words the performance attribution, and the covariance risk budgets which are a measure for the risk attribution. The upper axis labels the target risk, and the lower labels the target return. The thick vertical line separates the efficient frontier from the minimum variance locus. The risk axis thus increases in value to both sides of the separator line. The legend to the right links the assets names to colour of the bars.

Part V
Mean-CVaR Portfolios

Introduction

An alternative risk measure to the covariance is the Conditional Value at Risk, CVaR, which is also known as mean excess loss, mean shortfall or tail Value at Risk, VaR. For a given time horizon and confidence level, CVaR is the conditional expectation of the loss above VaR for the time horizon and the confidence level under consideration.

Pflug (2000) was the first to show that CVaR is a coherent risk measure and Rockafellar & Uryasev (2000) has shown that CVaR has other attractive properties including convexity.

We briefly describe the mathematical formulation of mean-CVaR optimization problems, which can be formulated as an equivalent linear programming problem and can be solved using standard linear programming solvers.

In Chapter 21 we briefly describe mean-CVaR portfolio theory and present its solution. We derive the feasible set and the efficient frontier. Two special points on the frontier are discussed in detail.

In Chapter 23 we present examples of how to compute feasible mean-CVaR portfolios and efficient mean-CVaR portfolios. These include not only the general cases, i.e. computing the portfolio with the lowest risk for a given return, or the portfolio with the highest return for a given risk, but also the special cases of the global minimum risk portfolio and the portfolio with the highest return/risk ratio.

In Chapter 24 we explore the efficient frontier of mean-CVaR portfolios. We proceed in the same way as for the mean-variance portfolios. We consider the case of long-only, short, box, and group constrained efficient frontiers of mean-CVaR portfolios.

In **??** we present the solvers used for mean-CVaR portfolio optimization. When default settings are used, then the `Rglpk` solver from the GNU Linear Programming Kit (GLPK) is used. Interfaces to alternative solvers are available through the contributed R packages `Rsymphony`, `RlpsolveAPI`, and `Rlpsolve`.

Chapter 21
Mean-CVaR Portfolio Theory

In this chapter we formulate and solve the mean-CVaR portfolio model, where covariance risk is now replaced by the conditional Value at Risk as the risk measure. In contrast to the mean-variance portfolio optimization problem, we are no longer restrict the set of assets to have a multivariate elliptically contoured distribution.

We consider a portfolio of assets with random returns. We denote the portfolio vector of weights with w and the random events by the vector r. Let $f(w, r)$ denote the loss function when we choose the portfolio W from a set X of feasible portfolios and let r be the realization of the random events. We assume that the random vector r has a probability density function denoted by $p(r)$. For a fixed decision vector w, we compute the cumulative distribution function of the loss associated with that vector w.

$$\Psi(w, \gamma) = \int_{f(w,r)\leq\gamma} p(r)dr$$

Then, for a given confidence level α, the VaR_α associated with portfolio W is given as

$$\mathrm{VaR}_\alpha(w) = \min\{\gamma \in \mathfrak{R} : \Psi(w, \gamma) \geq \alpha\}$$

Similarly, we define the $CVaR_\alpha$ associated with portfolio W

$$\mathrm{CVaR}_\alpha(w) = \frac{1}{1-\alpha}\int_{f(w,r)\leq\mathrm{VaR}_\alpha(w)} f(w, r)p(r)dr$$

We then define the problem of mean-CVaR portfolio selection as follows:

$$\begin{aligned} \min_{w} \quad & \mathrm{CVaR}_\alpha(w) \\ s.t. \quad & \\ & w^T\hat{\mu} = \bar{r} \\ & w^T 1 = 1 \end{aligned}$$

21.1 Solution of the Mean-CVaR Portfolio

In general, minimizing $CVaR_\alpha$ and VaR_α are not equivalent. Since the definition of $CVaR_\alpha$ involves the VaR_α function explicitly, it is difficult to work with and optimize this function. Instead, we consider the following simpler auxiliary function:

$$F_\alpha(w,\gamma) = \gamma + \frac{1}{1-\alpha}\int_{f(w,r)\geq\gamma}(f(w,r)-\gamma)p(r)dr$$

Alternatively, we can write $\mathrm{F}_\alpha(w,\gamma)$ as follows:

$$F_\alpha(w,\gamma) = \gamma + \frac{1}{1-\alpha}\int(f(w,r)-\gamma)^+p(r)dr$$

where $z^+ = max(z,0)$. This final function of γ has the following important properties that make it useful for the computation of VaR_α and $CVaR_\alpha$:

- $F_\alpha(w,\gamma)$ is a convex function of γ,
- $\mathrm{VaR}_\alpha(w)$ is a minimizer of $F_\alpha(w,\gamma)$,
- the minimum value of the function $F_\alpha(w,\gamma))$ is $\mathrm{CVaR}_\alpha(w)$.

As a consequence, we deduce that CVaR_α can be optimized via optimization of the function $F_\alpha(w,\gamma)$ with respect to the weights w and VaR γ. If the loss function $f(w,r)$ is a convex function of the portfolio variables w, then $F_\alpha(w,\gamma)$ is also a convex function of w. In this case, provided the feasible portfolio set W is also convex, the optimization problems are smooth convex optimization problems that can be solved using well-known optimization techniques for such problems.

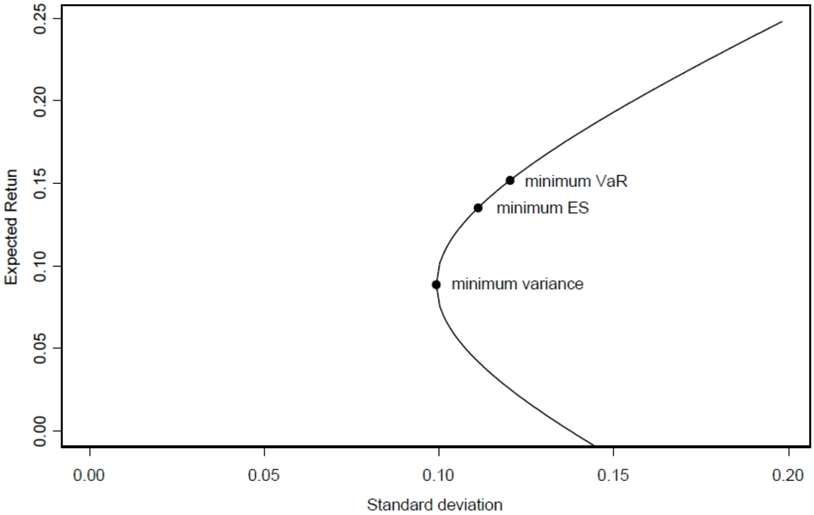


Figure 21.1 Efficient mean-variance frontier with the global minimum variance portfolio, the global minimum Value at Risk (5%) portfolio and the global minimum Conditional Value at Risk (5%) portfolio. The efficient frontiers under the various measures, are the subset of boundaries above the corresponding minimum global risk portfolios. We see that under 5% VaR and 5% CVaR the set of efficient portfolios is reduced with respect to the variance. *Source* *De Giorgi (2002)*.

21.2 Discretization

Often it is not possible or desirable to compute/determine the joint density function $p(r)$ of the random events in our formulation. Instead, we may have a number of scenarios, say r_s for $s = 1, \ldots, S$, which may represent some historical values of the returns. In this case, we obtain the following approximation to the function $F_\alpha(w, \gamma)$ by using the empirical distribution of the random returns based on the available scenarios:

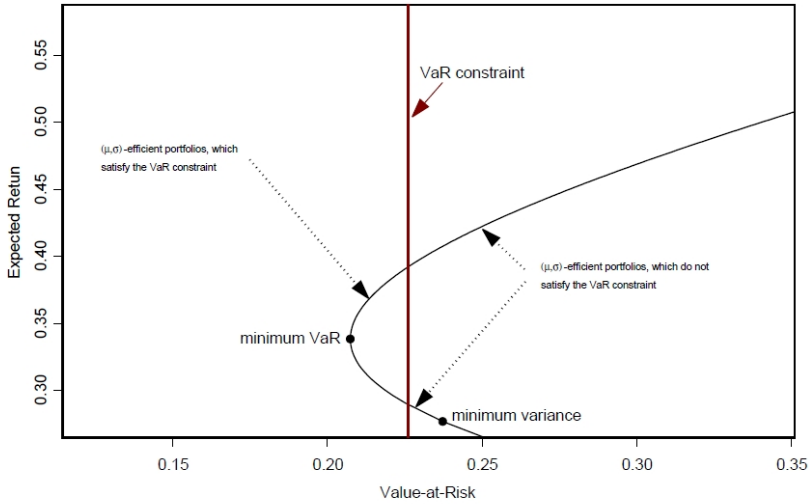


Figure 21.2 Mean-VaR(5%)-boundary with the global minimum variance portfolio. Portfolios on the mean-VaR(5%)-boundary between the global minimum VaR(5%) portfolio and the global minimum variance portfolio, are mean-variance efficient. The VaR constraint (vertical line) could force mean-variance investors with high variance to reduce the variance, and mean-variance investors with low variance to increase the variance, in order to be on the left side of the VaR constraint. *Source* De Giorgi *(2002)*.

$$\hat{F}_\alpha(w,\gamma) = \gamma + \frac{1}{(1-\alpha)S}\sum_{s=1}^{S}(f(w,r_s)-\gamma)^+ \ .$$

Now, the problem $\min_{w\in W} CVaR_\alpha(w)$ can be approximated by

$$\min_{w,\gamma} = \gamma + \frac{1}{(1-\alpha)S}\sum_{s=1}^{S}(f(w,r_s)-\gamma)^+ \ .$$

To solve this optimization problem, we introduce artificial variables z_s to replace $(f(w, r_s) - \gamma)^+$. This is achieved by imposing the constraints $z_s \geq f(w, r_s) - \gamma$ and $z_s \geq 0$

$$
\begin{aligned}
& \min \quad \gamma + \frac{1}{(1-\alpha)S} \sum_{s=1}^{S} z_s \\
s.t. & \\
& z_s \geq f(w, r_s) - \gamma \\
& z_s \geq 0 \; .
\end{aligned}
$$

Note that the constraints $z_s \geq f(w, r_s) - \gamma$ and $z_s \geq 0$ alone cannot ensure that $z_s = (f(w, r_s) - \gamma)^+$, since z_s can be larger than both right-hand term and still be feasible. However, since we are minimizing the objective function, which involves a positive multiple of z_s, it will never be optimal to assign z_s a value larger than the maximum of the two quantities $f(w, r_s) - \gamma$ and 0, and therefore, in an optimal solution z_s will be precisely $(f(w, r_s) - \gamma)^+$, thus justifying our substitution (Tütüncü, Toh & Todd, 2003).

21.3 Linearization

In the case that $f(w, r_s)$ is linear in w, all the expressions $z_s \geq f(w, r_s) - \gamma$ represent linear constraints and therefore the optimization problem becomes a linear programming problem that can be solved using the simplex method or alternative linear programming algorithms.

Chapter 22
Mean-CVaR Portfolio Settings

Required R package(s):

```
> library(fPortfolio)
```

Like all portfolios in Rmetrics, mean-CVaR portfolios are defined by the time series data set, the portfolio specification object, and the constraint strings. Specifying a mean-CVaR portfolio thus requires the three steps already familiar from the mean-variance portfolio approach.

22.1 Step 1: Portfolio Data

The input data for the portfolio is an S4 `"timeSeries"` object.

22.2 Step 2: Portfolio Specification

As in the case of the mean-variance portfolio, the portfolio specification manages all the settings which characterize the mean-CVaR portfolio.

It is important to note that in contrast to the mean-variance portfolio specification, the type of the portfolio always has to be specified in the case of CVaR portfolios. The significance level of α is 0.05 by default, but can be modified by the user. The default solver is the LP solver from the `GLPK`, `Rglpk()`. Alternative solvers are the solvers from the contributed R packages `lpSolveAPI` and `Rsymphony`. The following is an example of how to modify the default specifications to use them together with the mean-CVaR portfolios:

```
> cvarSpec <- portfolioSpec()
> setType(cvarSpec) = "CVaR"
> setAlpha(cvarSpec) = 0.05
> setSolver(cvarSpec) = "solveRglpk"
> print(cvarSpec)

Model List:
 Type:                      CVaR
 Optimize:                  minRisk
 Estimator:                 covEstimator
 Tail Risk:                 list()
 Params:                    alpha = 0.05 a = 1

Portfolio List:
 Target Weights:            NULL
 Target Return:             NULL
 Target Risk:               NULL
 Risk-Free Rate:            0
 Number of Frontier Points: 50
 Status:                    NA

Optim List:
 Solver:                    solveRglpk
 Objective:                 list()
 Options:                   meq = 2
 Control:                   list()
 Trace:                     FALSE

Message List:
 List:                      NULL
```

22.3 Step 3: Portfolio Constraints

In many cases we will work with long-only mean-CVaR portfolios. Specifying `constraints="LongOnly"` will force the lower and upper bounds for the weights to zero and one, respectively.

However, `fPortfolio` provides many alternative constraints. These include unlimited short-selling, lower and upper bounds, as well as linear equality and inequality constraints. The solver for dealing with these constraints has to be selected by the user and assigned by the function `setSolver()`.

Chapter 23
Mean-CVaR Portfolios

Required R package(s):

```
> library(fPortfolio)
```

The following examples show how to compute feasible mean-CVaR portfolios and efficient CVaR portfolios. These include not only the general cases, i.e. computing the portfolio with the lowest risk for a given return, or the portfolio with the highest return for a given risk, but also the special cases of the global minimum-risk portfolio and the portfolio with the highest return/risk ratio.

23.1 How to Compute a Feasible Mean-CVaR Portfolio

As a first example we consider the equal weights *feasible portfolio* with `"LongOnly"` constraints, which is the default case.

```
> lppData <- 100 * LPP2005.RET[, 1:6]
> cvarSpec <- portfolioSpec()
> setType(cvarSpec) <- "CVAR"
> nAssets <- ncol(lppData)
> setWeights(cvarSpec) <- rep(1/nAssets, times = nAssets)
> setSolver(cvarSpec) <- "solveRglpk"
> ewPortfolio <- feasiblePortfolio(
     data = lppData,
     spec = cvarSpec,
```

```
    constraints = "LongOnly")
> print(ewPortfolio)

Title:
 CVAR Feasible Portfolio
 Estimator:         covEstimator
 Solver:            solveRglpk
 Optimize:          minRisk
 Constraints:       LongOnly

Portfolio Weights:
   SBI    SPI    SII    LMI    MPI    ALT
0.1667 0.1667 0.1667 0.1667 0.1667 0.1667

Covariance Risk Budgets:
    SBI     SPI     SII     LMI     MPI     ALT
-0.0039  0.3526  0.0431 -0.0079  0.3523  0.2638

Target Return and Risks:
  mean     mu    Cov  Sigma   CVaR    VaR
0.0431 0.0431 0.3198 0.3198 0.7771 0.4472

Description:
 Thu May  7 14:00:38 2009 by user: Rmetrics
```

To display the results let us write a customized function to plot the weights, the performance attribution, and the risk attribution expressed by the covariance risk budgets.

```
> weightsPie(ewPortfolio, radius = 0.7)
> text <- "Equal Weights Man-CVaR Portfolio"
> mtext(text, side = 3, line = 1.5, font = 2, cex = 0.7, adj = 0)
> weightedReturnsPie(ewPortfolio, radius = 0.8, legend = FALSE)
> covRiskBudgetsPie(ewPortfolio, radius = 0.9, legend = FALSE)
```

The result is shown in Figure 23.1.

Now let us observe how the results change if we change the CVaR confidence level from $\alpha = 0.05$ to $\alpha = 0.10$

```
> setAlpha(cvarSpec) = 0.10
> ew10Portfolio <- feasiblePortfolio(
    data = lppData,
    spec = cvarSpec,
    constraints = "LongOnly")
> print(ew10Portfolio)
```

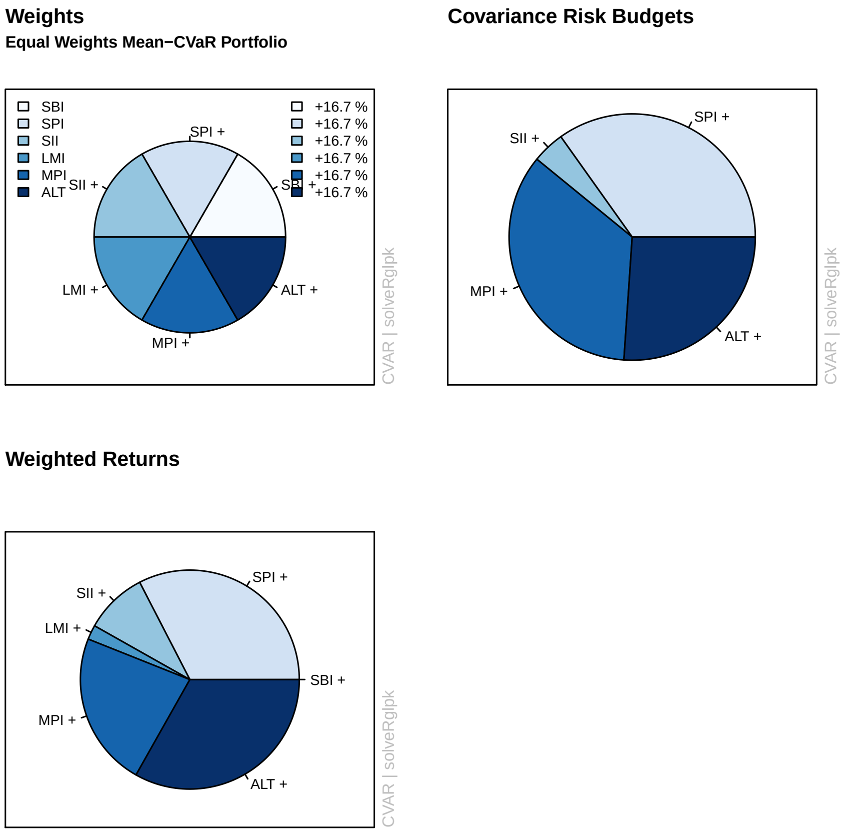


Figure 23.1 Weights plots for an equal-weights CVaR portfolio: Although we invest the same amount in each asset, the major contribution comes from the Swiss and foreign equities and alternative instruments. The same holds for the covariance risk budgets and the weighted returns.

```
Title:
 CVAR Feasible Portfolio
 Estimator:         covEstimator
 Solver:            solveRglpk
 Optimize:          minRisk
 Constraints:       LongOnly

Portfolio Weights:
   SBI    SPI    SII    LMI    MPI    ALT
0.1667 0.1667 0.1667 0.1667 0.1667 0.1667

Covariance Risk Budgets:
    SBI     SPI     SII     LMI     MPI     ALT
-0.0039  0.3526  0.0431 -0.0079  0.3523  0.2638

Target Return and Risks:
  mean     mu    Cov  Sigma   CVaR    VaR
0.0431 0.0431 0.3198 0.3198 0.5858 0.3215

Description:
 Thu May  7 14:00:39 2009 by user: Rmetrics
```

23.2 How to Compute the Mean-CVaR Portfolio with the Lowest Risk for a Given Return

Specifying the target return, we can compute an optimized efficient portfolio which has the lowest risk for a given return. In this example, we start from the equal weights portfolio, and search for a portfolio with the same returns, but a lower covariance risk.

```
> minriskSpec <- portfolioSpec()
> setType(minriskSpec) <- "CVaR"
> setAlpha(minriskSpec) <- 0.05
> setSolver(cvarSpec) <- "solveRglpk"
> setTargetReturn(minriskSpec) <- getTargetReturn(ewPortfolio@portfolio)["mean
     "]
> minriskPortfolio <- efficientPortfolio(data = lppData, spec = minriskSpec,
     constraints = "LongOnly")
> print(minriskPortfolio)

Title:
 CVaR Efficient Portfolio
```

```
 Estimator:         covEstimator
 Solver:            solveRglpk
 Optimize:          minRisk
 Constraints:       LongOnly
 VaR Alpha:         0.05

Portfolio Weights:
   SBI    SPI    SII    LMI    MPI    ALT
0.0000 0.0000 0.3848 0.2354 0.0000 0.3799

Covariance Risk Budgets:
    SBI     SPI     SII     LMI     MPI     ALT
 0.0000  0.0000  0.2425 -0.0102  0.0000  0.7677

Target Return and Risks:
  mean     mu    Cov  Sigma   CVaR    VaR
0.0431 0.0431 0.2484 0.2484 0.5101 0.3353

Description:
 Thu May  7 14:00:40 2009 by user: Rmetrics
```

The covariance risk of the optimized portfolio has been lowered from 0.32 to 0.25 for the same target return.

```
> weightsPie(minriskPortfolio, radius = 0.7)
> text <- "Minimum Risk CVaR Portfolio"
> mtext(text, side = 3, line = 1.5, font = 2, cex = 0.7, adj = 0)
> weightedReturnsPie(minriskPortfolio, radius = 0.8, legend = FALSE)
> covRiskBudgetsPie(minriskPortfolio, radius = 0.9, legend = FALSE)
```

The plots are shown in Figure 23.1.

23.3 How to Compute the Global Minimum Mean-CVaR Portfolio

The global *minimum risk portfolio* is the efficient portfolio with the lowest possible risk.

```
> globminSpec <- portfolioSpec()
> setType(globminSpec) <- "CVaR"
> setAlpha(globminSpec) <- 0.05
```

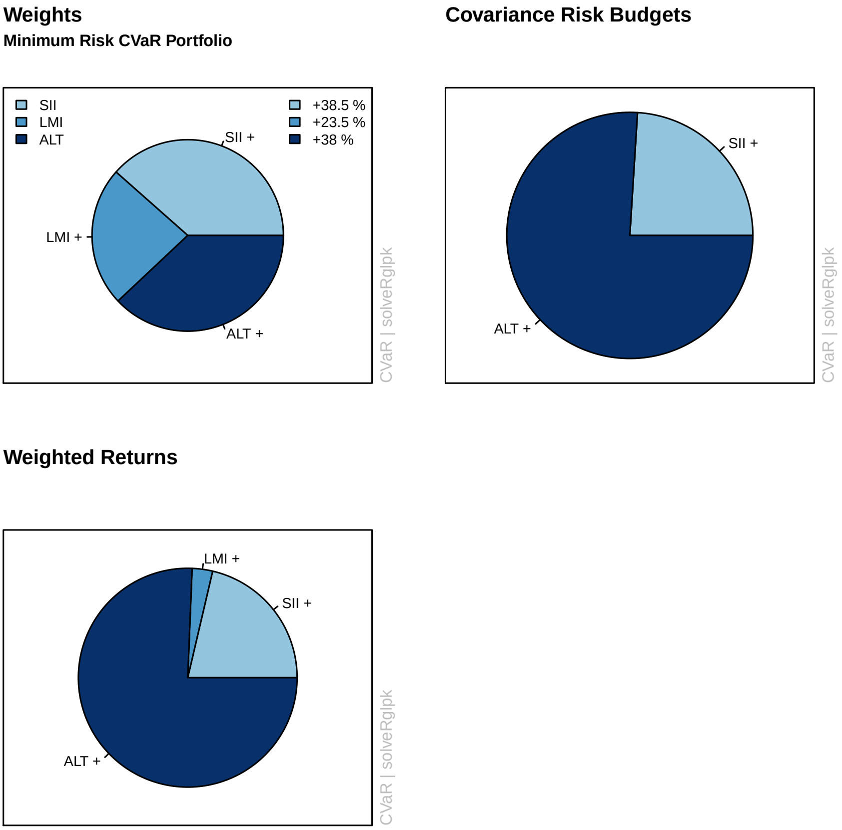


Figure 23.2 Weights plots for a minimum risk CVaR portfolio: Optimizing the risk for the target return of the equal weights portfolio leads to badly diversified portfolio, dominated by the risky alternative instruments.

```
> setSolver(globminSpec) <- "solveRglpk"
> setTargetReturn(globminSpec) <- getTargetReturn(ewPortfolio@portfolio)["mean
    "]
> globminPortfolio <- minriskPortfolio(data = lppData, spec = globminSpec,
    constraints = "LongOnly")
> print(globminPortfolio)

Title:
 CVaR Minimum Risk Portfolio
 Estimator:         covEstimator
 Solver:            solveRglpk
 Optimize:          minRisk
 Constraints:       LongOnly
 VaR Alpha:         0.05

Portfolio Weights:
   SBI    SPI    SII    LMI    MPI    ALT
0.1849 0.0000 0.1434 0.5945 0.0000 0.0772

Covariance Risk Budgets:
   SBI    SPI    SII    LMI    MPI    ALT
0.1438 0.0000 0.1991 0.5487 0.0000 0.1084

Target Return and Risks:
  mean     mu    Cov  Sigma   CVaR    VaR
0.0133 0.0133 0.1015 0.1015 0.1964 0.1524

Description:
 Thu May  7 14:00:42 2009 by user: Rmetrics
```

As expected, the portfolio is now dominated by the Swiss and foreign equities, which contribute 78% to the weights of the optimized portfolio.

Internally, the global minimum mean-CVaR portfolio is calculated by minimizing the efficient portfolio with respect to the target risk.

```
> weightsPie(globminPortfolio, radius = 0.7)
> text <- "Global Minimum Risk Portfolio"
> mtext(text, side = 3, line = 1.5, font = 2, cex = 0.7, adj = 0)
> weightedReturnsPie(globminPortfolio, radius = 0.8, legend = FALSE)
> covRiskBudgetsPie(globminPortfolio, radius = 0.9, legend = FALSE)
```

The plots are shown in Figure 23.3.

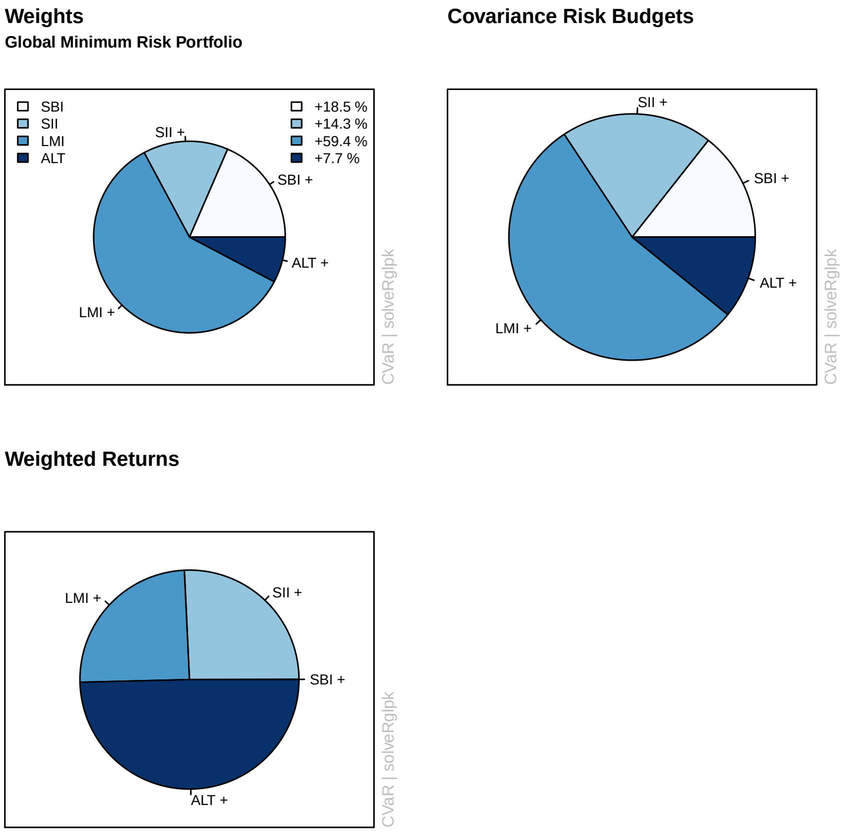


Figure 23.3 Weights plots for a global minimum risk CVaR portfolio: As expected, the global minimum risk portfolio is dominated by the low-risk Swiss and foreign bond assets.

How to Compute the Max Return/Risk Ratio Mean-CVaR Portfolio

The *Max Return/Risk portfolio* is calculated by minimization of the 'Sortino Ratio' for a given risk-free rate. The Sortino ratio is the ratio of the target return lowered by the risk-free rate and the CvaR risk. The risk-free rate in the default specification is zero and can be set to another value by using the function `setRiskFreeRate<-`.

```
> scData <- SMALLCAP.RET[, c("BKE", "GG", "GYMB", "KRON")]
> ratioSpec <- portfolioSpec()
> setType(ratioSpec) <- "CVaR"
> setAlpha(ratioSpec) <- 0.05
> setSolver(ratioSpec) <- "solveRglpk"
> setRiskFreeRate(ratioSpec) <- 0
> ratioPortfolio <- maxratioPortfolio(data = scData, spec = ratioSpec,
     constraints = "LongOnly")
> print(ratioPortfolio)
```

```
Title:
 CVaR Max Return/Risk Ratio Portfolio
 Estimator:         covEstimator
 Solver:            solveRglpk
 Optimize:          minRisk
 Constraints:       LongOnly
 VaR Alpha:         0.05

Portfolio Weights:
   BKE     GG   GYMB   KRON
0.3471 0.2546 0.0000 0.3983

Covariance Risk Budgets:
   BKE     GG   GYMB   KRON
0.3618 0.1377 0.0000 0.5006

Target Return and Risks:
  mean     mu    Cov  Sigma   CVaR    VaR
0.0294 0.0294 0.0984 0.0984 0.1269 0.1140

Description:
 Thu May  7 14:00:43 2009 by user: Rmetrics
```

```
> weightsPie(ratioPortfolio, radius = 0.7)
> text <- "Maximum Return/Risk Portfolio"
> mtext(text, side = 3, line = 1.5, font = 2, cex = 0.7, adj = 0)
> weightedReturnsPie(ratioPortfolio, radius = 0.8, legend = FALSE)
> covRiskBudgetsPie(ratioPortfolio, radius = 0.9, legend = FALSE)
```

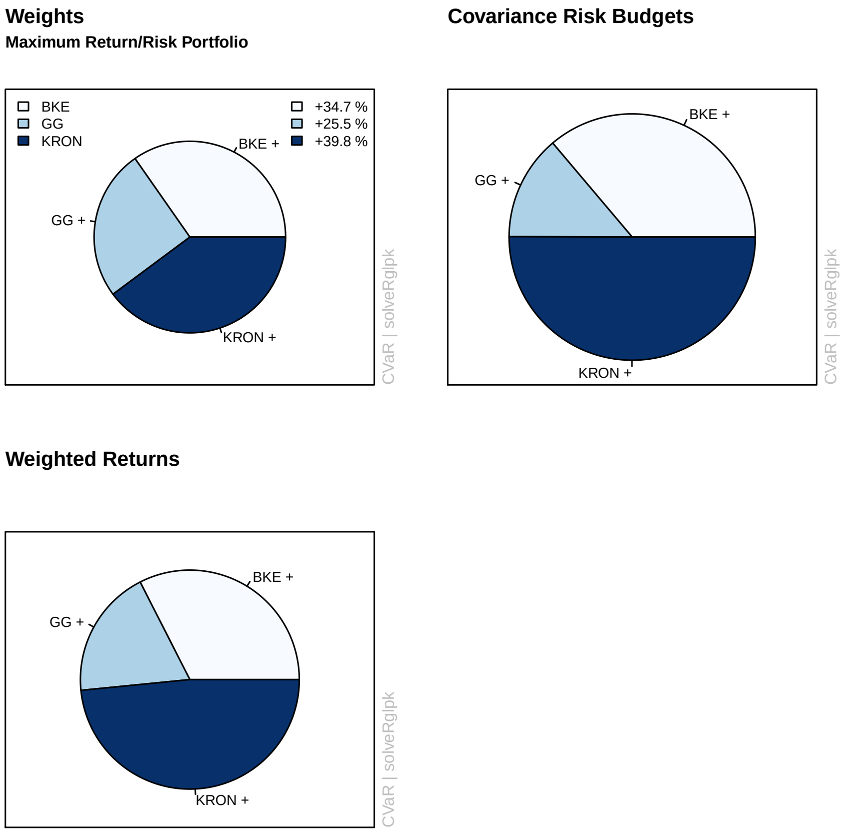


Figure 23.4 Weights, covariance risk budgets and weighetd returns plots for a max/risk ratio mean-CVaR portfolio:

The plots are shown in Figure 23.4.

Chapter 24
Mean-CVaR Portfolio Frontiers

Required R package(s):

```
> library(fPortfolio)
```

In this section we explore the efficient frontier, EF, and the minimum variance locus, MVL, of mean-CVaR portfolios. We proceed in the same way as for mean-variance portfolios: We select the two assets which lead to the smallest and largest returns and divide their range into equidistant parts which determine the target returns for which we try to find the efficient portfolios. We compute the global minimum risk portfolio and start from the closest returns to this point in both directions of the EF and the MVL. Note that only in the case of the long-only portfolio constraints do we reach both ends of the EF and the MVL. Usually, constraints will shorten the EF and MVL, and may even happen, that the constraints were so strong that do not find any solution at all.

In the following we compute and compare long-only, unlimited short, box, and group constrained efficient frontiers of mean-CVaR portfolios[1].

[1] Note that throughout this section we set the portfolio type to `CVaR` and the solver function to `solveRglpk()`.

24.1 The Long-only Portfolio Frontier

The long-only mean-variance portfolios. In this case all the weights are bounded between zero and one.

```
Functions:
portfolioFrontier       efficient portfolios on the frontier
frontierPoints          extracts risk/return frontier points
frontierPlot            creates an efficient frontier plot
  cmlPoints               adds market portfolio
  cmlLines                adds capital market line
  tangencyPoints          adds tangency portfolio point
  tangencyLines           adds tangency line
  equalWeightsPoints      adds point of equal weights portfolio
  singleAssetPoints       adds points of single asset portfolios
  twoAssetsLines          adds frontiers of two assets portfolios
  sharpeRatioLines        adds Sharpe ratio line
  monteCarloPoints        adds randomly feasible portfolios
weightsPlot             weights bar plot along the frontier
weightedReturnsPlot     weighted returns bar plot
covRiskBudgetsPlot      covariance risk budget bar plot
```

Listing 24.1 The table lists functions to compute the efficient frontier of linearly constrained mean-CVaR portfolios and to plot the results.

```
> lppData <- 100 * LPP2005.RET[, 1:6]
> longSpec <- portfolioSpec()
> setType(longSpec) <- "CVaR"
> setAlpha(longSpec) <- 0.05
> setNFrontierPoints(longSpec) <- 5
> setSolver(longSpec) <- "solveRglpk"
> longFrontier <- portfolioFrontier(data = lppData, spec = longSpec,
     constraints = "LongOnly")
> print(longFrontier)

Title:
 CVaR Portfolio Frontier
 Estimator:          covEstimator
 Solver:             solveRglpk
 Optimize:           minRisk
 Constraints:        LongOnly
```

```
 Portfolio Points:  5 of 5
 VaR Alpha:         0.05

Portfolio Weights:
     SBI    SPI    SII    LMI    MPI    ALT
1 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000
2 0.0000 0.0000 0.1988 0.6480 0.0000 0.1532
3 0.0000 0.0000 0.3835 0.2385 0.0000 0.3780
4 0.0000 0.0000 0.3464 0.0000 0.0000 0.6536
5 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000

Covariance Risk Budgets:
      SBI     SPI     SII     LMI     MPI     ALT
1  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000
2  0.0000  0.0000  0.2641  0.3126  0.0000  0.4233
3  0.0000  0.0000  0.2432 -0.0101  0.0000  0.7670
4  0.0000  0.0000  0.0884  0.0000  0.0000  0.9116
5  0.0000  0.0000  0.0000  0.0000  0.0000  1.0000

Target Return and Risks:
    mean     mu    Cov  Sigma   CVaR    VaR
1 0.0000 0.0000 0.1261 0.1261 0.2758 0.2177
2 0.0215 0.0215 0.1224 0.1224 0.2313 0.1747
3 0.0429 0.0429 0.2472 0.2472 0.5076 0.3337
4 0.0643 0.0643 0.3941 0.3941 0.8780 0.5830
5 0.0858 0.0858 0.5684 0.5684 1.3343 0.8978

Description:
 Thu Jun  4 14:05:21 2009 by user: Rmetrics
```

To shorten the output in the example above, we have lowered the number of frontier points to 5 points. The printout lists the weights, the covariance risk budgets and the target return and risk values along the minimum variance locus and the efficient frontier starting with the portfolio with the lowest return and ending with the portfolio with the highest achievable return at the end of the efficient frontier.

To plot the efficient frontier we repeat the optimization with 25 points at the frontier and plot the result using the function `tailoredFrontierPlot()`

```
> setNFrontierPoints(longSpec) <- 25
> longFrontier <- portfolioFrontier(data = lppData, spec = longSpec,
     constraints = "LongOnly")
```

```
> tailoredFrontierPlot(object = longFrontier, mText = "Mean-CVaR Portfolio -
     Long Only Constraints",
     risk = "CVaR")
```

The function `tailoredFrontierPlot()` displays, as the name says, a customized plot with fixed colour, font and symbol settings and several selected add-ons including, single assets points, tangency line, and Sharpe ratio line.

Figure 24.1 and Figure 24.2 show the results for the weights, the weighted returns and the covariance risk budgets along the minimum variance locus and the efficient frontier.

```
> weightsPlot(longFrontier, mtext = FALSE)
> text <- "Mean-CVaR Portfolio - Long Only Constraints"
> mtext(text, side = 3, line = 3, font = 2, cex = 0.9)
> weightedReturnsPlot(longFrontier, mtext = FALSE)
> covRiskBudgetsPlot(longFrontier, mtext = FALSE)
```

24.2 The Unlimited 'Short' Portfolio Frontier

When all weights are not restricted we have the case of unlimited short selling. Unlike in the mean-variance portfolio, we cannot optimize the portfolio analytically. To circumvent this we define box constraints with large lower and upper bounds.

```
> shortSpec <- portfolioSpec()
> setType(shortSpec) <- "CVaR"
> setAlpha(shortSpec) <- 0.05
> setNFrontierPoints(shortSpec) <- 5
> setSolver(shortSpec) <- "solveRglpk"
> shortConstraints <- c("minW[1:6]=-999", "maxW[1:6]=+999")
> shortFrontier <- portfolioFrontier(data = lppData, spec = shortSpec,
     constraints = shortConstraints)
> print(shortFrontier)
```

```
Title:
 CVaR Portfolio Frontier
 Estimator:         covEstimator
 Solver:            solveRglpk
 Optimize:          minRisk
 Constraints:       minW maxW
```

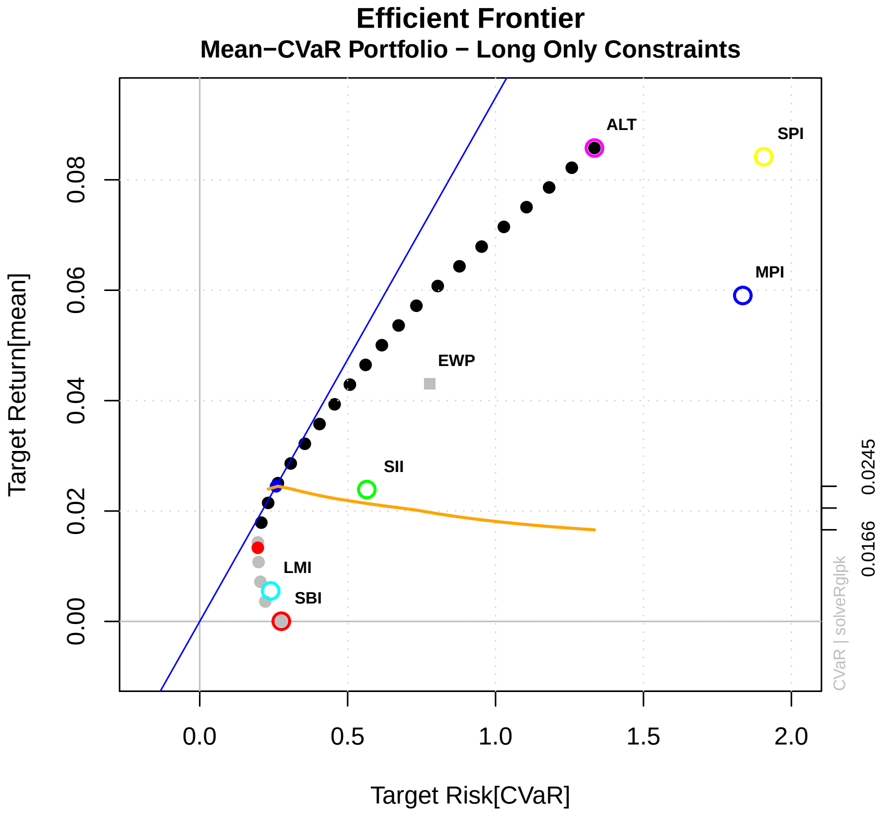


Figure 24.1 The graph shows for 25 equidistant return points the minimum variance locus and the efficient frontier. Added are the risk-return points for the individual assets and the equal weights portfolio. The line through the origin is the tangency line for a zero risk-free rate. The curved line with the maximum the tangency point is the Sharpe ration along the frontier.

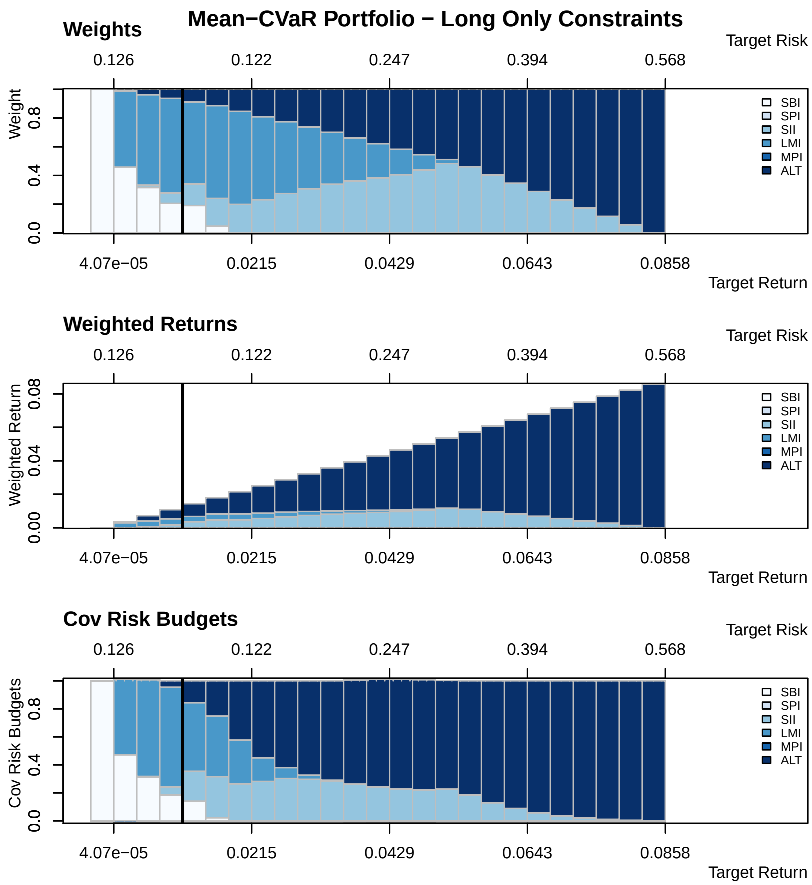


Figure 24.2 The graph shows for the weights, weighted returns, and covariance risk budgets 25 equidistant return points, along the minimum variance locus and the efficient frontier. Note that the strong separation line marks the position between the minimum variance locus and the efficient frontier. Target returns are increasing from left to right, whereas target risks, are increasing to the left and right with respect to the separation line.

```
 Portfolio Points:  5 of 5
 VaR Alpha:         0.05

Portfolio Weights:
      SBI     SPI     SII     LMI     MPI     ALT
1  0.4257 -0.0242  0.0228  0.5661  0.0913 -0.0816
2 -0.0201 -0.0101  0.1746  0.7134 -0.0752  0.2174
3 -0.3275 -0.0196  0.4318  0.6437 -0.2771  0.5486
4 -0.8113  0.0492  0.5704  0.8687 -0.5273  0.8503
5 -1.6975  0.0753  0.6305  1.5485 -0.6683  1.1115

Covariance Risk Budgets:
      SBI     SPI     SII     LMI     MPI     ALT
1  0.4056  0.0256  0.0062  0.5384 -0.0730  0.0972
2 -0.0080 -0.0173  0.2124  0.4559 -0.1256  0.4825
3  0.0054 -0.0204  0.3674  0.0592 -0.2787  0.8671
4  0.0572  0.0409  0.2901  0.0223 -0.2984  0.8880
5  0.1513  0.0510  0.1966  0.0215 -0.3235  0.9031

Target Return and Risks:
    mean     mu    Cov  Sigma   CVaR    VaR
1 0.0000 0.0000 0.1136 0.1136 0.2329 0.1859
2 0.0215 0.0215 0.1172 0.1172 0.2118 0.1733
3 0.0429 0.0429 0.2109 0.2109 0.3610 0.2923
4 0.0643 0.0643 0.3121 0.3121 0.5570 0.4175
5 0.0858 0.0858 0.4201 0.4201 0.7620 0.5745

Description:
 Thu Jun  4 14:06:06 2009 by user: Rmetrics
```

```
> setNFrontierPoints(shortSpec) <- 25
> shortFrontier <- portfolioFrontier(data = lppData, spec = shortSpec,
     constraints = shortConstraints)
> tailoredFrontierPlot(object = shortFrontier, mTText = "Mean-CVaR Portfolio -
     Short Constraints",
     risk = "CVaR")
```

```
> weightsPlot(shortFrontier, mtext = FALSE)
> text <- "Min-CVaR Portfolio - Short Constrained Portfolio"
> mtext(text, side = 3, line = 3, font = 2, cex = 0.9)
> weightedReturnsPlot(shortFrontier, mtext = FALSE)
> covRiskBudgetsPlot(shortFrontier, mtext = FALSE)
```

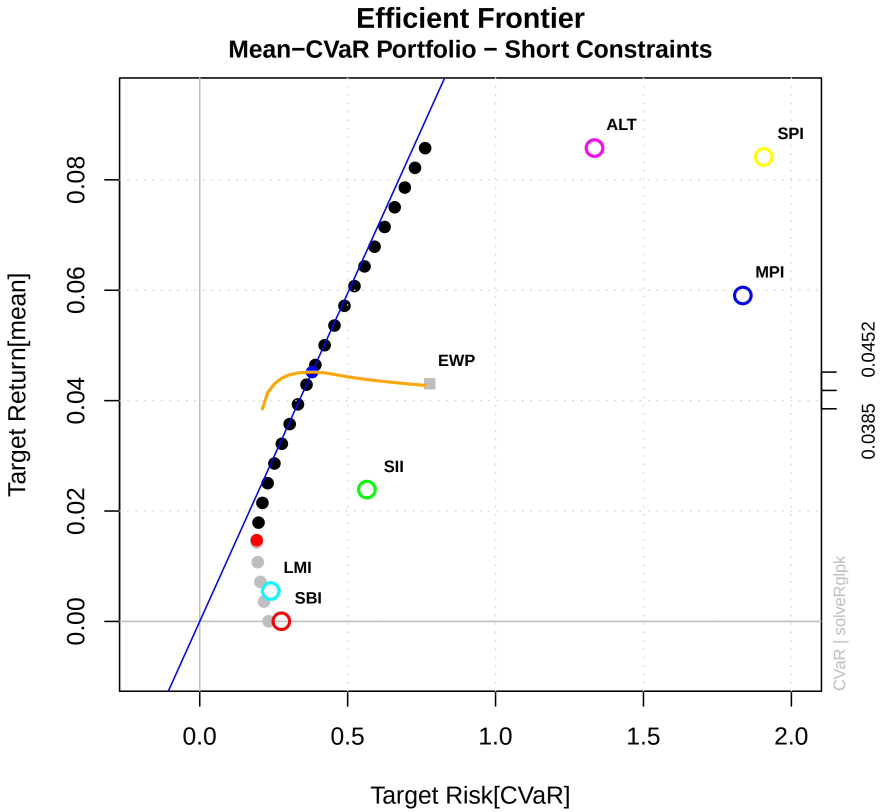


Figure 24.3 The graph shows for 25 equidistant return points the minimum variance locus and the efficient frontier when short selling is allowed. The major difference to the previous long-only is the fact, that MVL and EF do not end at the assets with the lowest and highest risks: For the same return the risk has lowered through short selling.

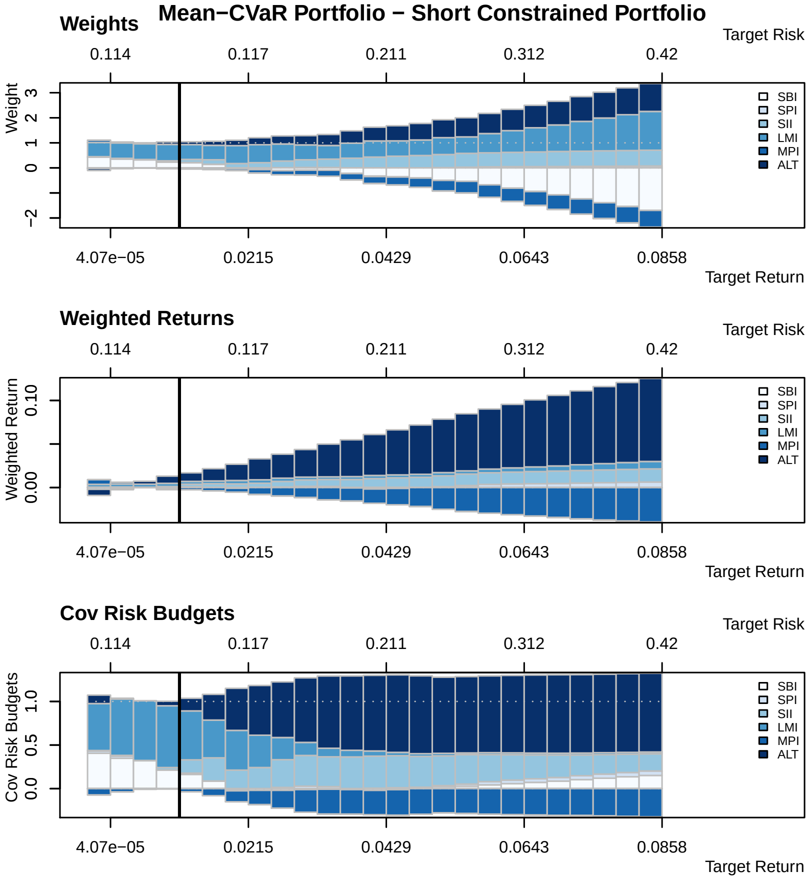


Figure 24.4 The graph shows for equidistant return points the weights, weighted returns, and covariance risk budgets, along the minimum variance locus and the efficient frontier.

24.3 The Box-Constrained Portfolio Frontier

A box-constrained portfolio is a portfolio where the weights are constrained by lower and upper bounds, e.g. we want to invest at least 10% and no more than 50% in each asset.

```
> boxSpec <- portfolioSpec()
> setType(boxSpec) <- "CVaR"
> setAlpha(boxSpec) <- 0.05
> setNFrontierPoints(boxSpec) <- 15
> setSolver(boxSpec) <- "solveRglpk"
> boxConstraints <- c("minW[1:6]=0.05", "maxW[1:6]=0.66")
> boxFrontier <- portfolioFrontier(data = lppData, spec = boxSpec,
     constraints = boxConstraints)
> print(boxFrontier)
```

```
Title:
 CVaR Portfolio Frontier
 Estimator:         covEstimator
 Solver:            solveRglpk
 Optimize:          minRisk
 Constraints:       minW maxW
 Portfolio Points:  5 of 9
 VaR Alpha:         0.05

Portfolio Weights:
     SBI    SPI    SII    LMI    MPI    ALT
1 0.0526 0.0500 0.1370 0.6600 0.0500 0.0504
3 0.0500 0.0500 0.2633 0.4127 0.0500 0.1739
5 0.0500 0.0500 0.4109 0.1463 0.0500 0.2928
7 0.0500 0.0500 0.3378 0.0500 0.0500 0.4622
9 0.0500 0.0501 0.1399 0.0500 0.0500 0.6600

Covariance Risk Budgets:
      SBI     SPI     SII     LMI     MPI     ALT
1  0.0189  0.1945  0.1435  0.3378  0.1742  0.1312
3  0.0023  0.1528  0.2209  0.0248  0.1582  0.4410
5 -0.0017  0.1042  0.2478 -0.0080  0.1136  0.5440
7 -0.0024  0.0823  0.1099 -0.0035  0.0931  0.7206
9 -0.0022  0.0650  0.0182 -0.0031  0.0752  0.8469

Target Return and Risks:
    mean     mu    Cov  Sigma   CVaR    VaR
1 0.0184 0.0184 0.1232 0.1232 0.2604 0.1913
3 0.0307 0.0307 0.1838 0.1838 0.3999 0.2651
```

```
5 0.0429 0.0429 0.2655 0.2655 0.5787 0.3654
7 0.0552 0.0552 0.3456 0.3456 0.7832 0.4818
9 0.0674 0.0674 0.4388 0.4388 1.0382 0.6675

Description:
 Thu Jun  4 14:06:56 2009 by user: Rmetrics

> setNFrontierPoints(boxSpec) <- 25
> boxFrontier <- portfolioFrontier(data = lppData, spec = boxSpec,
     constraints = boxConstraints)
> tailoredFrontierPlot(object = boxFrontier, mText = "Mean-CVaR Portfolio -
     Box Constraints",
     risk = "CVaR")


> weightsPlot(boxFrontier)
> text <- "Min-CVaR Portfolio - Box Constrained Portfolio"
> mtext(text, side = 3, line = 3, font = 2, cex = 0.9)
> weightedReturnsPlot(boxFrontier)
> covRiskBudgetsPlot(boxFrontier)
```

24.4 The Group-Constrained Portfolio Frontier

A group-constrained portfolio is a portfolio where the weights of groups of selected assets are constrained by lower and upper bounds for the total weights of the groups, e.g. we want to invest at least 30% in the group of bounds and not more than 50% in the groups of assets.

```
> groupSpec <- portfolioSpec()
> setType(groupSpec) <- "CVaR"
> setAlpha(groupSpec) <- 0.05
> setNFrontierPoints(groupSpec) <- 10
> setSolver(groupSpec) <- "solveRglpk"
> groupConstraints <- c("minsumW[c(1,4)]=0.3", "maxsumW[c(2:3,5:6)]=0.66")
> groupFrontier <- portfolioFrontier(data = lppData, spec = groupSpec,
     constraints = groupConstraints)
> print(groupFrontier)

Title:
 CVaR Portfolio Frontier
 Estimator:         covEstimator
```

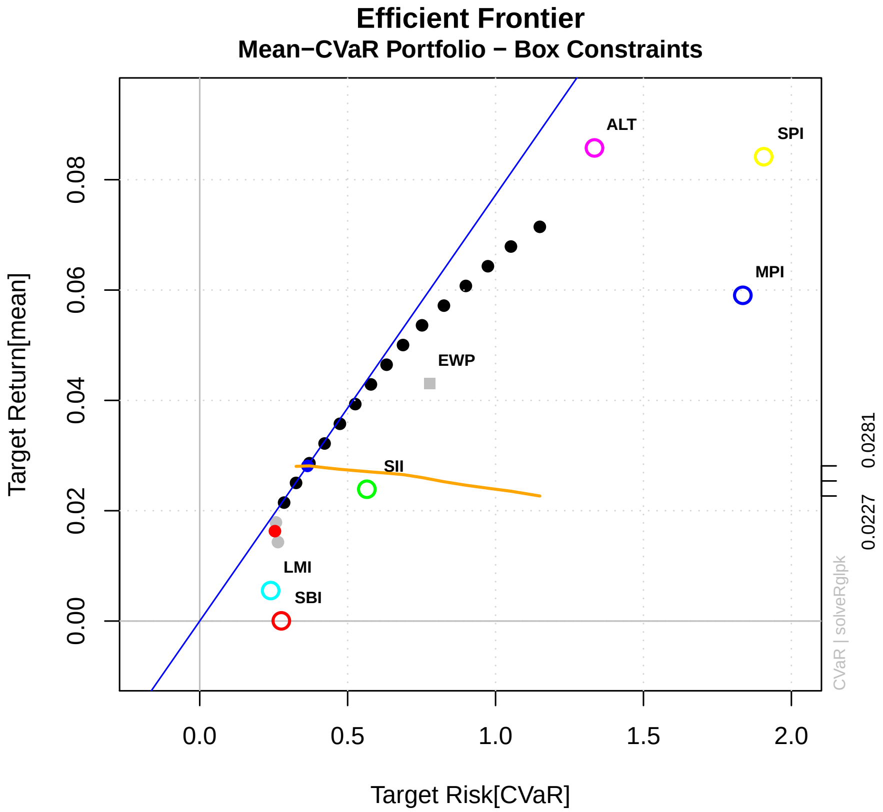


Figure 24.5 Box constrained Min-CVaR portfolio frontier plot.

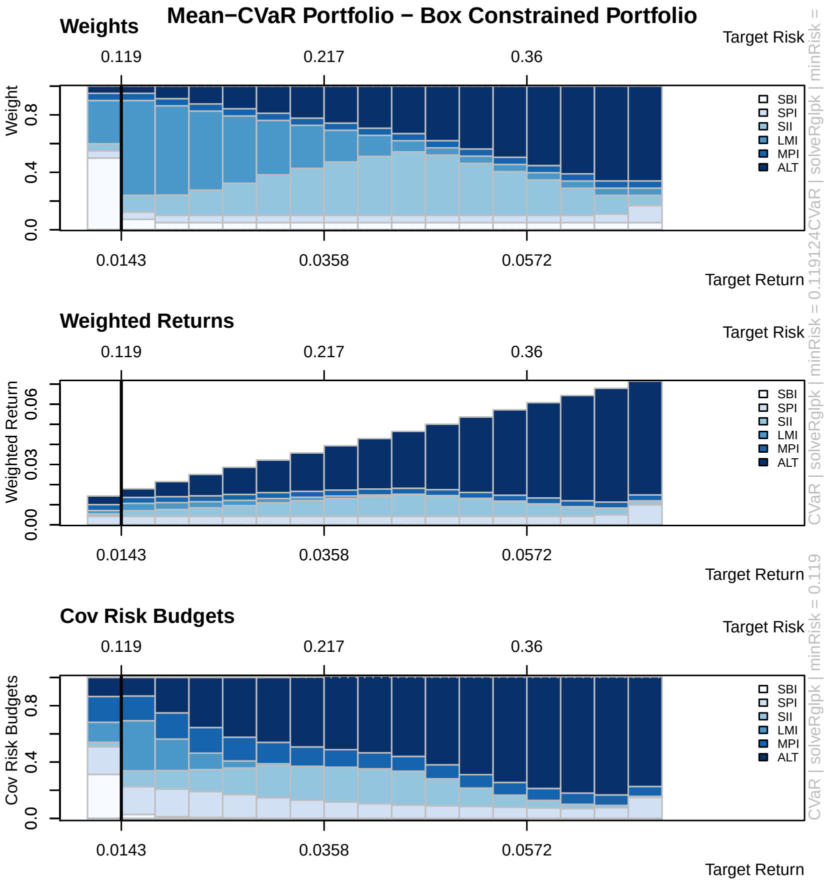


Figure 24.6 Mean-CVaR Portfolio - Box Constrained Weights Plot

```
 Solver:            solveRglpk
 Optimize:          minRisk
 Constraints:       minsumW maxsumW
 Portfolio Points:  5 of 7
 VaR Alpha:         0.05

Portfolio Weights:
     SBI    SPI    SII    LMI    MPI    ALT
1 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000
2 0.2440 0.0000 0.0410 0.6574 0.0000 0.0576
4 0.0000 0.0000 0.2744 0.5007 0.0000 0.2249
5 0.0000 0.0000 0.3288 0.3400 0.0000 0.3312
7 0.0000 0.0000 0.0209 0.3400 0.0000 0.6391

Covariance Risk Budgets:
      SBI     SPI     SII     LMI     MPI     ALT
1  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000
2  0.2294  0.0000  0.0218  0.7226  0.0000  0.0263
4  0.0000  0.0000  0.3024  0.0780  0.0000  0.6196
5  0.0000  0.0000  0.2388 -0.0024  0.0000  0.7636
7  0.0000  0.0000  0.0020 -0.0159  0.0000  1.0139

Target Return and Risks:
    mean     mu    Cov  Sigma   CVaR    VaR
1 0.0000 0.0000 0.1261 0.1261 0.2758 0.2177
2 0.0096 0.0096 0.1012 0.1012 0.2003 0.1622
4 0.0286 0.0286 0.1575 0.1575 0.3076 0.2178
5 0.0381 0.0381 0.2147 0.2147 0.4392 0.2783
7 0.0572 0.0572 0.3559 0.3559 0.8228 0.5517

Description:
 Thu Jun  4 14:07:42 2009 by user: Rmetrics
```

```
> setNFrontierPoints(groupSpec) <- 25
> groupFrontier <- portfolioFrontier(data = lppData, spec = groupSpec,
     constraints = groupConstraints)
> tailoredFrontierPlot(object = groupFrontier, mText = "Mean-CVaR Portfolio -
     Group Constraints",
     risk = "CVaR")
```

```
> weightsPlot(groupFrontier)
> text <- "Min-CVaR Portfolio - Group Constrained Portfolio"
> mtext(text, side = 3, line = 3, font = 2, cex = 0.9)
> weightedReturnsPlot(groupFrontier)
> covRiskBudgetsPlot(groupFrontier)
```

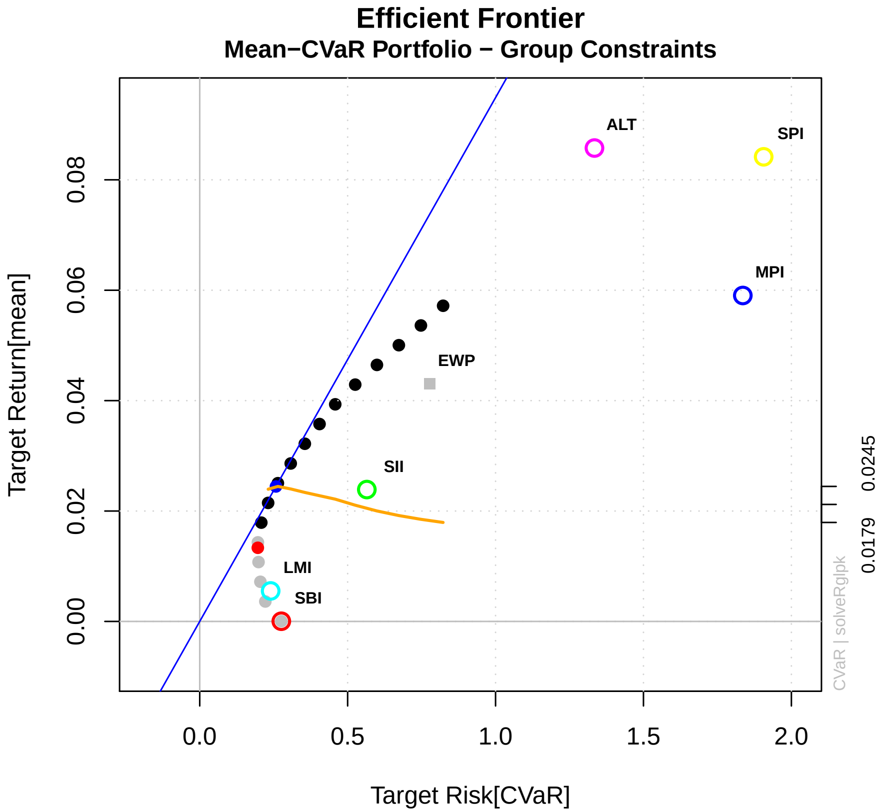


Figure 24.7 Group constrained Min-CVaR portfolio frontier plot.

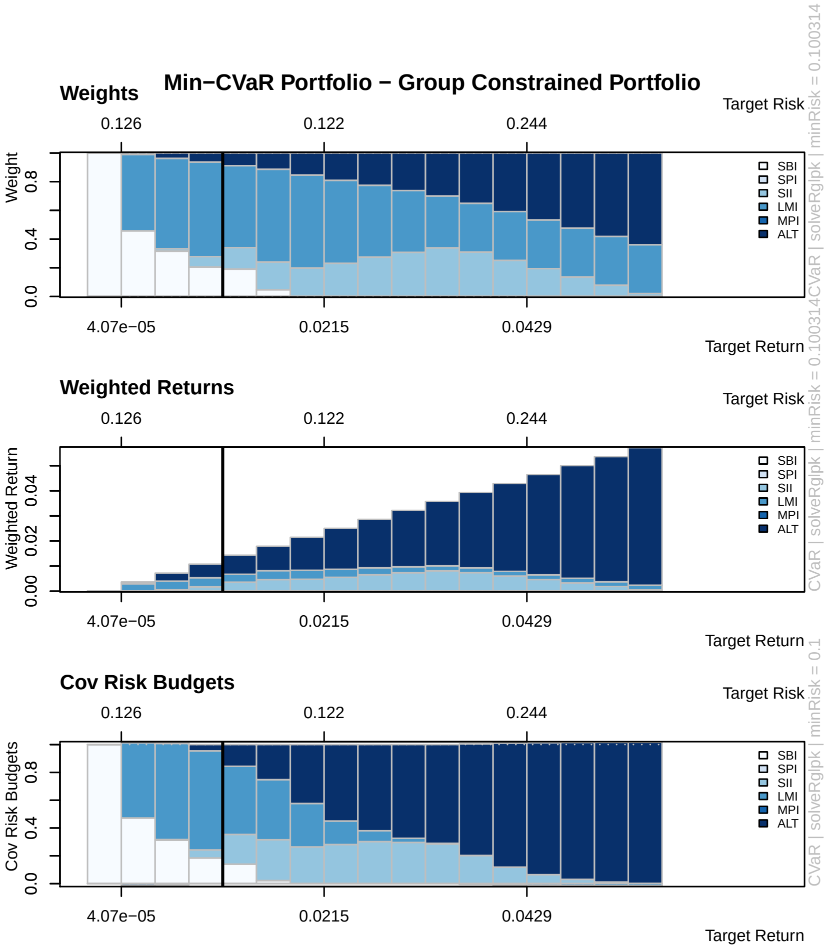


Figure 24.8 Mean-CVaR Portfolio - Group Constrained Weights Plot

24.5 The Box/Group-Constrained Portfolio Frontier

Box and group constraints can be combined

```
> boxgroupSpec <- portfolioSpec()
> setType(boxgroupSpec) <- "CVaR"
> setAlpha(boxgroupSpec) <- 0.05
> setNFrontierPoints(boxgroupSpec) <- 5
> setSolver(boxgroupSpec) <- "solveRglpk"
> boxgroupConstraints <- c(boxConstraints, groupConstraints)
> boxgroupFrontier <- portfolioFrontier(data = lppData, spec = boxgroupSpec,
     constraints = boxgroupConstraints)
> print(boxgroupFrontier)

Title:
 CVaR Portfolio Frontier
 Estimator:         covEstimator
 Solver:            solveRglpk
 Optimize:          minRisk
 Constraints:       minW maxW minsumW maxsumW
 Portfolio Points:  2 of 2
 VaR Alpha:         0.05

Portfolio Weights:
     SBI    SPI    SII    LMI    MPI    ALT
1 0.0500 0.0500 0.1421 0.6207 0.0500 0.0872
2 0.0500 0.0500 0.2245 0.2900 0.0500 0.3355

Covariance Risk Budgets:
      SBI     SPI     SII     LMI     MPI     ALT
1  0.0125  0.1951  0.1321  0.2234  0.1858  0.2510
2 -0.0013  0.1117  0.0906 -0.0112  0.1232  0.6871

Target Return and Risks:
    mean     mu    Cov  Sigma   CVaR    VaR
1 0.0215 0.0215 0.1347 0.1347 0.2852 0.2047
2 0.0429 0.0429 0.2609 0.2609 0.5953 0.3814

Description:
 Thu Jun  4 14:08:22 2009 by user: Rmetrics

> setNFrontierPoints(boxgroupSpec) <- 25
> boxgroupFrontier <- portfolioFrontier(data = lppData,
                                        spec = boxgroupSpec,
                                        constraints = boxgroupConstraints)
```

```
> tailoredFrontierPlot(object = boxgroupFrontier,
                      mText = "Mean-CVaR Portfolio - Box/Group Constraints",
                      risk = "CVaR")
```

```
> weightsPlot(boxgroupFrontier)
> text <- "Mean-CVaR Portfolio - Box/Group Constrained Portfolio"
> mtext(text, side = 3, line = 3, font = 2, cex = 0.9)
> weightedReturnsPlot(boxgroupFrontier)
> covRiskBudgetsPlot(boxgroupFrontier)
```

24.6 Other Constraints

Like in the case of the mean-variance portfolios quadratic and/or non-linear constraints complicate portfolio optimization. Those constraints will include for example quadratic covariance risk budget constraints and tail risk budget constraints, as well as non-linear function constraints, such as maximum drawdowns limit constraints or extension strategy constraints[2].

24.7 More About the Frontier Plot Tools

Note that the default axis type of the frontier plot is automatically taken from the portfolio specification, here the `"CVaR"` axis was selected and thus displayed. The reason for this is that the function `frontierPlot()` inspects the type of the portfolio and then decides what type of axis to display.

The function `frontierPlot()` returns a two column matrix with the target risk and target return to be plotted. For the target return we can extract either the `mean` or the `mu` values, for the target risk we can select from four choices, `"Cov"`, `"Sigma"`, `"CVaR"`, and `"VaR"`. Furthermore, we can overwrite the risk choice, and allow for an automated selection, `auto=TRUE`, which is the default. The `auto` selection does the following:

[2] These more complex constraints require quadratic and non-linear portfolio solvers which are considered in the ebook *Advanced Portfolio Optimization with R/Rmetrics*.

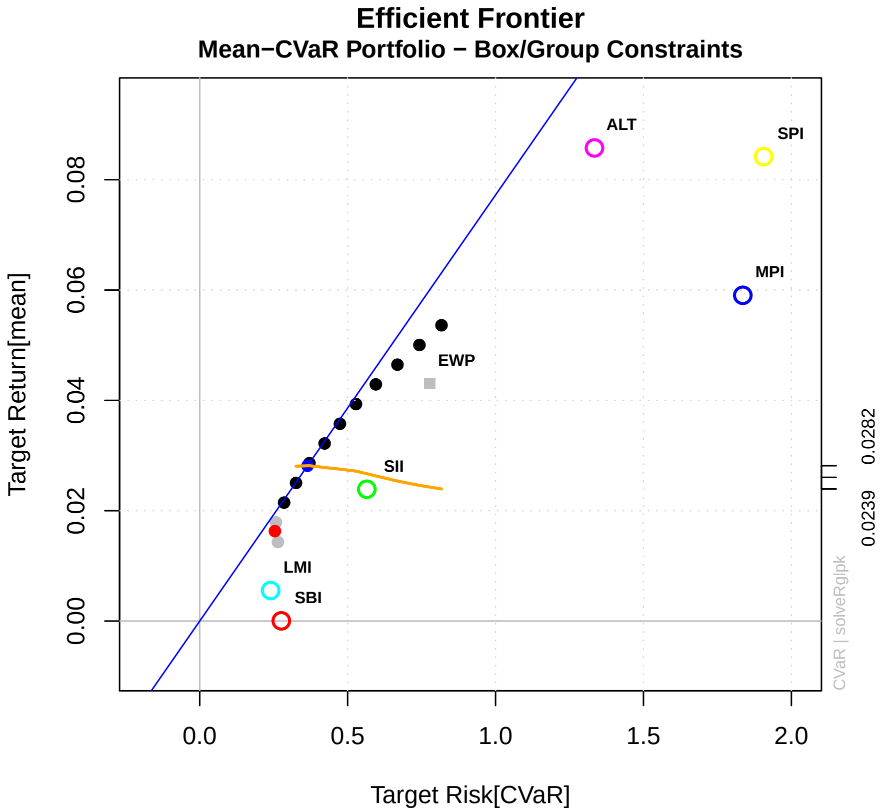


Figure 24.9 Box/Group constrained CVaR portfolio frontier plot.

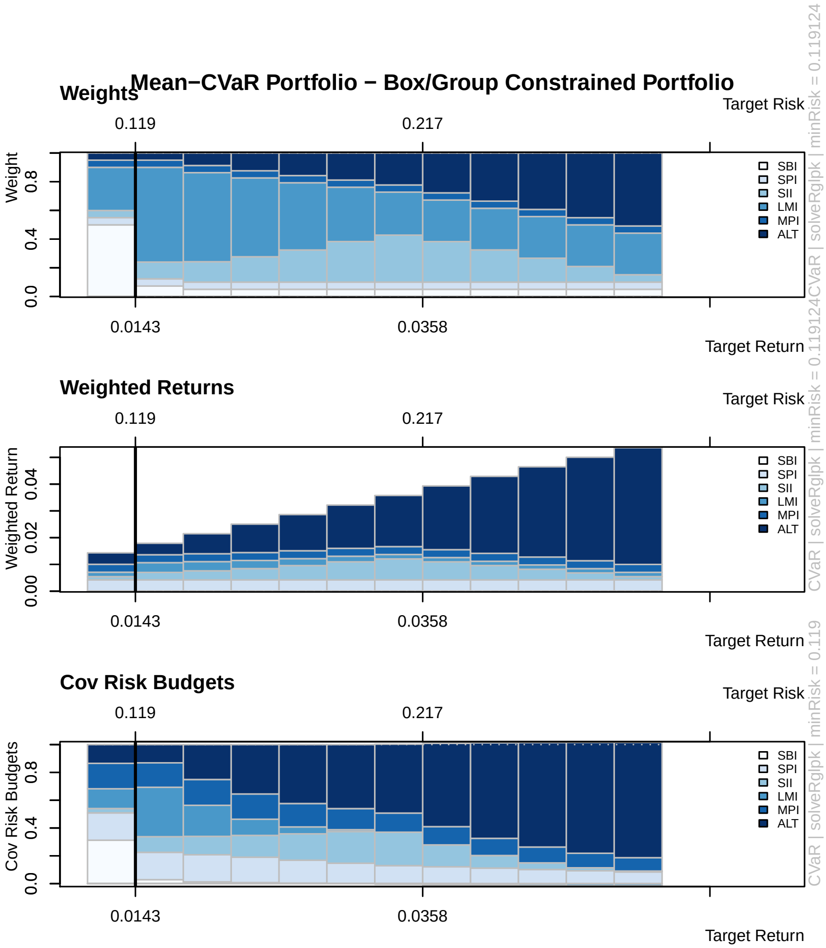


Figure 24.10 Mean-CVaR Portfolio - Box/Group Constrained Weights Plot

```
Type - Risk Relationships:
Type <- getType(object)
if (Type == "MV") risk = "Cov"
if (Type == "MV" & Estimator != "covEstimator") risk = "Sigma"
if (Type == "CVaR") risk = "CVaR"
```

Explicitly specifying the `risk` type in the function argument the function `frontierPlot()` allows us to display several views from the efficient frontier. Now let us plot the `"covariance"` frontier together with the `"CVaR"` frontier in the covariance risk view:

```
> longSpec <- portfolioSpec()
> setType(longSpec) <- "CVaR"
> setAlpha(longSpec) <- 0.1
> setNFrontierPoints(longSpec) <- 20
> setSolver(longSpec) <- "solveRglpk"
> longFrontier <- portfolioFrontier(data = lppData, spec = longSpec,
     constraints = "LongOnly")
> par(mfrow = c(2, 2))
> frontierPlot(longFrontier, pch = 16, type = "b", cex = 0.7)
> frontierPlot(longFrontier, risk = "Cov", auto = FALSE, pch = 16,
     type = "b", cex = 0.7)
> frontierPlot(longFrontier, risk = "VaR", auto = FALSE, pch = 16,
     type = "b", cex = 0.7)
```

The result is shown in Figure 24.11.

We can also compare the two risks and plot the covariance versus the CVaR:

```
> par(mfrow = c(1, 1))
> Cov <- frontierPoints(longFrontier, risk = "Cov", auto = FALSE)[,
     "targetRisk"]
> CVaR <- frontierPoints(longFrontier, risk = "CVaR", auto = FALSE)[,
     "targetRisk"]
> plot(Cov, CVaR, pch = 19, cex = 0.7)
> Cov <- frontierPoints(longFrontier, frontier = "lower", risk = "Cov",
     auto = FALSE)[, "targetRisk"]
> CVaR <- frontierPoints(longFrontier, frontier = "lower",
     risk = "CVaR", auto = FALSE)[, "targetRisk"]
```

The two branches belong to the efficient frontier and to the minimum variance locus, respectively.

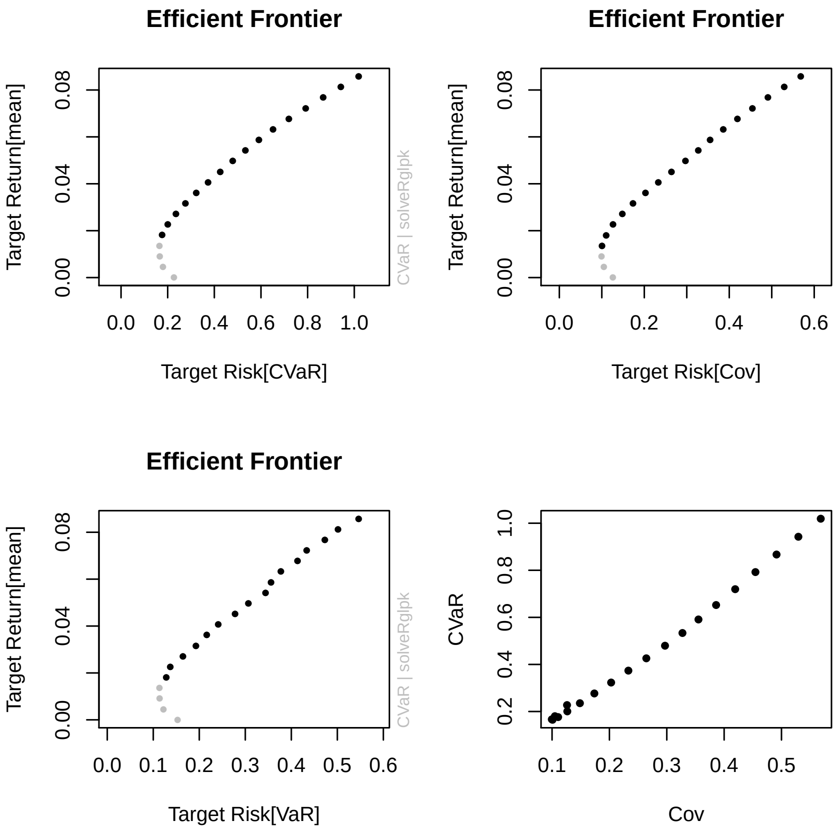


Figure 24.11 The first three charts show different views of return vs. risk plots. Here the risk is measured as covariance risk, as conditional value-at-risk, and as value-at-risk. Note the kinks in the VaR measure plot arise from the fact that VaR is not a coherent risk measure. The last graph plots the relationship between covariance risk and conditional value-at-risk.

Interactively plotting the efficient frontier

The generic `plot()` function allows you to interactively display the efficient frontier with several add-on plots

```
> plot(frontier)

Make a plot selection (or 0 to exit):

1:   Plot Efficient Frontier
2:   Add Minimum Risk Portfolio
3:   Add Tangency Portfolio
4:   Add Risk/Return of Single Assets
5:   Add Equal Weights Portfolio
6:   Add Two Asset Frontiers [0-1 PF Only]
7:   Add Wheel Pie of Weights
8:   Add Monte Carlo Portfolios
9:   Add Sharpe Ratio [MV PF Only]

Selection:
```

Part VI
Portfolio Backtesting

Introduction

Backtesting is a key component of portfolio management and it is often used to assess and compare different statistical models. Portfolio backtesting is accomplished by reconstructing, with historical data, trades that would have occurred in the past using rules defined by a given strategy. The underlying theory is that any strategy that worked well in the past is likely to work well in the future.

The backtesting results offer statistics that can be used to gauge the effectiveness of the strategy. However, the results achieved from the backtest are highly dependent on the movements of the tested period. Therefore, it is often a good idea to backtest over a long time frame that encompasses several different types of market conditions.

In Chapter 25 we introduce a new S4 object class `fPFOLIOBACKTEST`, which represents a portfolio backtest. The four slots keep all information on the rolling windows, on the investment strategy portfolio, on the smoother computing the weights for their re-balancing, and on optional messages. We describe in detail the rolling analysis technique used to run a portfolio backtest.

In Chapter 26 we illustrate the usage of the `fPortfolioBacktest` package with a realistic portfolio. The first example shows how to re-balance the Swiss Performance Sector Indexes over time to outperform the SPI market index.

In Chapter 27 we present a second example which shows us the backtest results for the MSCI GCC Index, a market index traded in the gulf region which covers the economies Bahrain, Kuwait, Oman, Qatar and the United Arab Emirates.

Chapter 25
S4 Portfolio Backtest Class

Required R package(s):

```
> library(fPortfolioBacktest)
```

In this chapter we introduce the S4 class fPFOLIOBACKTEST for performing a rolling optimization and performance analysis on portfolios over time. We present the components which allow such an analysis. This includes a discussion of how to create rolling windows, how to formulate portfolio strategies, and how to smooth and post-process the optimal portfolio weights.

25.1 Class Representation

All settings specifying the backtesting procedure are represented by an S4 class named fPFOLIOBACKTEST.

```
> showClass("fPFOLIOBACKTEST")

Class "fPFOLIOBACKTEST" [package "fPortfolioBacktest"]

Slots:

Name:   windows strategy smoother messages
Class:     list     list     list     list
```

An object of class `fPFOLIOBACKTEST` has four slots, named `@windows`, `@strategy`, `@smoother`, and `@messages`. The first slot, `@windows`, holds the rolling windows information, the second slot, `@strategy`, contains the portfolio strategy for backtesting, the `@smoother` slot contains information regarding weights smoothing and the last slot, named `@messages`, holds a list of optional messages.

How to create a portfolio backtest object

The function `portfolioBacktest()` allows the user to set backtest settings from scratch.

```
> formals(portfolioBacktest)

$windows
list(windows = "equidistWindows", params = list(horizon = "12m"))

$strategy
list(strategy = "tangencyStrategy", params = list())

$smoother
list(smoother = "emaSmoother", params = list(doubleSmoothing = TRUE,
    lambda = "3m", skip = 0, initialWeights = NULL))

$messages
list()
```

The default settings for rolling portfolios are composed of equidistant windows with a horizon of 12 months, a tangency portfolio strategy, and a double exponential moving average (EMA) smoother. The decay length, `lambda="3m"`, is three months and the computation starts with the first data point, i.e. `skip=0`. The `messages` list is empty.

To look inside the portfolio backtest structure you can call the function `str()`. This function compactly displays the internal structure of the portfolio backtest object. It can be considered as a diagnostic function and as a simple way to summarize the internal structure of the object. Let us create a new backtest object, and then print the structure for the default settings.

```
> backtest <- portfolioBacktest()
> str(backtest, width = 65, strict.width = "cut")

Formal class 'fPFOLIOBACKTEST' [package "fPortfolioBacktest"] w..
  ..@ windows :List of 2
```

```
.. ..$ windows: chr "equidistWindows"
.. ..$ params :List of 1
.. .. ..$ horizon: chr "12m"
..@ strategy:List of 2
.. ..$ strategy: chr "tangencyStrategy"
.. ..$ params  : list()
..@ smoother:List of 2
.. ..$ smoother: chr "emaSmoother"
.. ..$ params  :List of 4
.. .. ..$ doubleSmoothing: logi TRUE
.. .. ..$ lambda         : chr "3m"
.. .. ..$ skip           : num 0
.. .. ..$ initialWeights : NULL
..@ messages: list()
```

25.2 The Windows Slot

The `@windows` slot is a list with two named entries. The first entry named `windows` holds the name of the rolling windows function that defines the 'backtest' windows, and the second slot, entitled `params`, holds the parameters, such as the horizon of the windows. The slot and its entries can be extracted and modified by the user through extractor and constructor functions.

```
Extractor Functions:
getWindows          gets windows slot
getWindowsFun       gets windows function
getWindowsParams    gets windows specific parameters
getWindowsHorizon   gets windows horizon
```

Listing 25.1 Extractor functions for the `@windows` slot of an `fPFOLIOBACKTEST` object

To modify the settings from a portfolio backtest specifications we use the constructor functions.

```
Constructor Functions:
setWindowsFun       sets the name of the windows function
setWindowsParams    sets parameters for the windows function
```

```
setWindowsHorizon   sets the windows horizon
```

Listing 25.2 Constructor functions for the `@windows` slot of an `fPFOLIOBACKTEST` object

Note that you can write your own windows function, and if required, you can add additional parameters to the parameter list `params`. This is explained in Section 25.2. The extractor and constructor functions can also be used to set the new parameters.

How to inspect the default rolling windows

The default rolling window function in the `fPortfolioBacktest` package is called `equidistWindows()` and as the name of the function implies, this function generates windows with fixed equidistant horizons. Their value is set in the `params` list under the name `horizon`. The following code shows how to inspect this value for the default settings.

```
> defaultBacktest <- portfolioBacktest()
> getWindowsFun(defaultBacktest)

[1] "equidistWindows"

> getWindowsParams(defaultBacktest)

$horizon
[1] "12m"

> getWindowsHorizon(defaultBacktest)

[1] "12m"
```

Bear in mind that the horizon is specified as a span string with integer length, here `12`, and the unit, here `"m"` indicating that we measure spans in months. A windows functions has the two arguments:

```
> args(equidistWindows)

function (data, backtest = portfolioBacktest())
NULL
```

the data and the backtest object. Let us inspect the code of the function.

```
> equidistWindows
```

```
function (data, backtest = portfolioBacktest())
{
    horizon = getWindowsHorizon(backtest)
    ans = rollingWindows(x = data, period = horizon, by = "1m")
    ans
}
<environment: namespace:fPortfolioBacktest>
```

First, we use the function `getWindowsHorizon()` to extract the horizon, which is the length of the windows returned as a span value. Then we call the function `rollingWindows()` to create the windows. The result returned by the function `rollingWindows()` is a list with two entries named `from` and `to`. These two list entries give the start and the end date for each window. In addition, the `control` attribute holds information about the `start` and `end` dates of the whole series, the `period` length of the windows (also called horizon), and its regular time `shift`.

```
> swxData <- 100 * SWX.RET
> swxBacktest <- portfolioBacktest()
> setWindowsHorizon(swxBacktest) <- "24m"
> equidistWindows(data = swxData, backtest = swxBacktest)

$from
GMT
 [1] [2000-01-01] [2000-02-01] [2000-03-01] [2000-04-01] [2000-05-01]
 [6] [2000-06-01] [2000-07-01] [2000-08-01] [2000-09-01] [2000-10-01]
[11] [2000-11-01] [2000-12-01] [2001-01-01] [2001-02-01] [2001-03-01]
[16] [2001-04-01] [2001-05-01] [2001-06-01] [2001-07-01] [2001-08-01]
[21] [2001-09-01] [2001-10-01] [2001-11-01] [2001-12-01] [2002-01-01]
[26] [2002-02-01] [2002-03-01] [2002-04-01] [2002-05-01] [2002-06-01]
[31] [2002-07-01] [2002-08-01] [2002-09-01] [2002-10-01] [2002-11-01]
[36] [2002-12-01] [2003-01-01] [2003-02-01] [2003-03-01] [2003-04-01]
[41] [2003-05-01] [2003-06-01] [2003-07-01] [2003-08-01] [2003-09-01]
[46] [2003-10-01] [2003-11-01] [2003-12-01] [2004-01-01] [2004-02-01]
[51] [2004-03-01] [2004-04-01] [2004-05-01] [2004-06-01] [2004-07-01]
[56] [2004-08-01] [2004-09-01] [2004-10-01] [2004-11-01] [2004-12-01]
[61] [2005-01-01] [2005-02-01] [2005-03-01] [2005-04-01] [2005-05-01]
[66] [2005-06-01]

$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]

attr(,"control")
attr(,"control")$start
GMT
[1] [2000-01-04]

attr(,"control")$end
GMT
[1] [2007-05-08]

attr(,"control")$period
[1] "24m"

attr(,"control")$by
[1] "1m"
```

Currently, the backtest function is fully tested only for end-of-month windows with varying horizons, shifted monthly. We are working on implementing shifts of arbitrary lengths, so that one can create rolling windows which depend on market volatility or may even be triggered by trading decisions.

How to modify rolling window parameters

The list entry `params` from the `@windows` slot is a list with additional parameters used in different situations. If required, you can enhance this.

```
horizon             fixed horizon length used for the default
                    windows function
...                 your parameter settings for your
                    custom windows functions (if required)
```

Listing 25.3 List entry in the `@windows` slot an `fPFOLIOBACKTEST` object

By default the window size is fixed at 12 months (`horizon = "12m"`). This fixed horizon can be changed with the function `setWindowsHorizon()`.

```
> setWindowsHorizon(backtest) <- "24m"
```

The entire list of parameters can be extracted with the function `getWindowsParams()` and you can add and modify parameter settings with the function `setWindowsParams()`. Note that you must take care not to omit the `horizon` parameter when setting windows parameters with the `setWindowsParams()` function.

```
> getWindowsParams(backtest)

$horizon
[1] "24m"
```

How to write your own windows function

If you want to add your own rolling windows function you should proceed in the following way

```
# Set ANY additional windows parameters if required:
setWindowsParams(backtest) <- list(...)

# Template to create your own rolling windows function:
myRollingWindows <- function(data, backtest = portfolioBacktest())
{
    # Code:
    Params <- getWindowsParams(backtest)
    ...

    # Return:
    list(from = <...> , to = <...>)
}
```

Listing 25.4 Example of a rolling windows function

In this function template, `data` is a multivariate `timeSeries` object, and `backtest` the portfolio backtest object with class `fPFOLIOBACKTEST`. Additional parameters required by the function `myRollingWindows()` can be passed in through the list `@windows$params` of the backtest object. With the function `getWindowsParams` we can extract its parameter list. Note that `myRollingWindows` must at least return a named list, with two named entries `$from` and `$to`, which give the start and end dates for each backtest window.

As an example we want to create rolling windows which depend on the volatility of the underlying returns. In this case, if the volatility is low, we want to use longer window horizons, and if the volatility increases, then we shorten the window horizons. The result are windows of varying length, dependent on the volatility.

25.3 The Strategy Slot

The `@strategy` slot is a list with two named entries. The first entry, `strategy`, holds the name of the strategy function that defines the backtest portfolio strategy, and the second, `params`, holds their parameters if required. The slot and its entries can be extracted and modified by through extractor and constructor functions.

```
Extractor Functions:
getStrategy         gets strategy slot
getStrategyFun      gets the name of the strategy function
getStrategyParams   gets strategy specific parameters
```

Listing 25.5 Extractor functions for the `@strategy` slot of an `fPFOLIOBACKTEST` object

To modify the settings for a portfolio backtest strategy we use the constructor functions.

```
Constructor Functions:
setStrategyFun      sets the name of the strategy function
setStrategyParams   sets strategy specific parameters
```

Listing 25.6 Constructor functions for the `@strategy` slot of an `fPFOLIOBACKTEST` object

How to inspect the default portfolio strategy

The default rolling portfolio strategy provided by the `fPortfolioBacktest` package is the `tangencyStrategy`.

```
> args(tangencyStrategy)

function (data, spec = portfolioSpec(), constraints = "LongOnly",
    backtest = portfolioBacktest())
NULL
```

The first argument expects the data as a `timeSeries` object, the second the portfolio specification as an object of class `fPFOLIOSPEC`, constraints as a string vector, and backtest information from an object of class `fPFOLIOBACKTEST`.

Let us take a look at the code of the very simple portfolio investment strategy defined by the tangency strategy.

```
> tangencyStrategy

function (data, spec = portfolioSpec(), constraints = "LongOnly",
    backtest = portfolioBacktest())
{
    strategyPortfolio <- tangencyPortfolio(data, spec, constraints)
    Status = getStatus(strategyPortfolio)
    if (Status == 1)
        strategyPortfolio <- minvariancePortfolio(data, spec,
            constraints)
    strategyPortfolio
}
<environment: namespace:fPortfolioBacktest>
```

The `tangencyStrategy` invests in a portfolio that has the highest Sharpe ratio, and if such a portfolio does not exist, the minimum variance portfolio is taken instead. Strategy functions always have to return an object of class `fPORTFOLIO`.

The function name of the portfolio strategy can be extracted with the function `getStrategyFun()`.

```
> getStrategyFun(backtest)

[1] "tangencyStrategy"
```

and changed with the function `setStrategyFun()`.

```
> setStrategyFun(backtest) <- "tangencyStrategy"
```

where "tangencyStrategy" can be replaced with the name of the function you wish to use.

How to write your own strategy function

If you want to test your own portfolio strategies, you will need to write your own function. This is shown in the following example; here, we define a function called `myPortfolioStrategy()`.

```
> ## Add parameters needed in the function 'myPortfolioStrategy':
> setStrategyParams(backtest) <- list()
> ## Creating a new portfolio strategy function:
> myPortfolioStrategy <-
    function(data, spec, constraints, backtest)
  {
    ## Extract Parameters:
    Parameters <- getStrategyParams(backtest)

    ## Strategy Portfolio:
    strategyPortfolio <- tangencyPortfolio(data, spec, constraints)

    ## Return :
    strategyPortfolio
  }
```

Here, `data` is a multivariate time series object, `spec` the portfolio specification, `constraints` the string of portfolio constraints and `backtest` the portfolio backtest object. Additional parameters can be passed through the backtest object with the function `setStrategyParams()` and can be extracted within the portfolio strategy function with `getStrategyParams()`. Note that `myPortfolioStrategy()` function must return an S4 object of class `fPORTFOLIO`.

25.4 The Smoother Slot

The `@smoother` slot is a list with two named entries. The first entry named `smoother` holds the name of the smoother function that defines the backtest function to smooth the weights over time. The second, named `params`, holds the required parameters for the smoother function. The slot and its entries can be extracted and modified through extractor and constructor functions.

```
Extractor Functions:
getSmoother                  gets smoother slot
getSmootherFun               gets the name of the smoother function
getSmootherParams            gets parameters for strategy function
getSmootherLambda            gets smoothing parameter lambda
getSmootherDoubleSmoothing   gets setting for double smoothing
getSmootherInitialWeights    gets initial weights used in smoothing
getSmootherSkip              gets number of skipped months
```

Listing 25.7 Extractor functions for the `@smoother` slot of an `fPFOLIOBACKTEST` object

To modify the settings for a portfolio backtest strategy you can use the constructor functions.

```
Constructor Functions:
setSmootherFun               sets the name of the smoother function
setSmootherParams            sets parameters for strategy function
setSmootherLambda            sets smoothing parameter lambda
setSmootherDoubleSmoothing   sets setting for double smoothing
setSmootherInitialWeights    sets initial weights used in smoothing
setSmootherSkip              sets number of skipped months
```

Listing 25.8 Constructor functions for the `@smoother` slot of an `fPFOLIOBACKTEST` object

How to inspect the default smoother function

The default smoother function provided by the `fPortfolioBacktest` package is the `emaSmoother`.

```
> args(emaSmoother)

function (weights, spec, backtest)
NULL
```

The first argument expects the `weights`, the second the portfolio specification `spec` as an object of class `fPFOLIOSPEC`, and `backtest` information from an object of class `fPFOLIOBACKTEST`.

Let us examine the code of the exponential moving average (EMA) smoothing approach implemented in the `emaSmoother()` function.

```
> emaSmoother

function (weights, spec, backtest)
{
    ema <- function(x, lambda) {
        x = as.vector(x)
        lambda = 2/(lambda + 1)
        xlam = x * lambda
        xlam[1] = x[1]
        ema = filter(xlam, filter = (1 - lambda), method = "rec")
        ema[is.na(ema)] <- 0
        as.numeric(ema)
    }
    lambda = getSmootherLambda(backtest)
    lambdaLength = as.numeric(substr(lambda, 1, nchar(lambda) -
        1))
    lambdaUnit = substr(lambda, nchar(lambda), nchar(lambda))
    stopifnot(lambdaUnit == "m")
    lambda = lambdaLength
    nAssets = ncol(weights)
    initialWeights = getSmootherInitialWeights(backtest)
    if (!is.null(initialWeights))
        weights[1, ] = initialWeights
    smoothWeights1 = NULL
    for (i in 1:nAssets) {
        EMA = ema(weights[, i], lambda = lambda)
        smoothWeights1 = cbind(smoothWeights1, EMA)
    }
    doubleSmooth = getSmootherDoubleSmoothing(backtest)
    if (doubleSmooth) {
        smoothWeights = NULL
        for (i in 1:nAssets) {
            EMA = ema(smoothWeights1[, i], lambda = lambda)
            smoothWeights = cbind(smoothWeights, EMA)
        }
    }
    else {
        smoothWeights = smoothWeights1
    }
    rownames(smoothWeights) = rownames(weights)
    colnames(smoothWeights) = colnames(weights)
    smoothWeights
```

```
}
<environment: namespace:fPortfolioBacktest>
```

The `emaSmoother()` applies a single or double EMA filter to a vector of weights. First we define the internal smoother function `ema` and retrieve the decay parameter `lambda`. Then we apply single or double EMA smoothing to each series of asset weights. Finally, the smoothed weights are returned. Note that in each step you have to make sure that the weights add to one if you are fully invested.

How to modify rolling window parameters

The `emaSmoother()` function is controlled by four parameters, which are defined in the `params` entry of the `smoother` slot:

```
Smoother Control Parameters:
doubleSmoothing     logical, TRUE means the EMA filter is
                    applied twice
lambda              character, the amount of smoothing -
                    e.g. "3m", "6m", ...
skip                numeric value, number of months to skip -
                    e.g. 12, 18, ...
initialWeights      numeric vector containing the initial weights
```

Listing 25.9 Parameters of the emaSmoother function

The default settings are:

```
> getSmootherParams(backtest)

$doubleSmoothing
[1] TRUE

$lambda
[1] "3m"

$skip
[1] 0

$initialWeights
NULL
```

To modify these control parameters individually, we use the `setSmoother*` functions.

To change to single smoothing, type

```
> setSmootherDoubleSmoothing(backtest) <- FALSE
```

To modify the smoother's decay length, type

```
> setSmootherLambda(backtest) <- "12m"
```

To start rebalancing 12 months after the original start date, type

```
> setSmootherSkip(backtest) <- "12m"
```

To use equal weights as starting points, type

```
> nAssets <- 5
> setSmootherInitialWeights(backtest) <- rep(1/nAssets, nAssets)
```

After you have made changes, check if your settings are active.

```
> getSmootherParams(backtest)
```

```
$doubleSmoothing
[1] FALSE

$lambda
[1] "12m"

$skip
[1] "12m"

$initialWeights
[1] 0.2 0.2 0.2 0.2 0.2
```

How to write your own smoother function

If you want to apply your own weight-smoothing style, you will need to write a custom function. Note that additional smoothing parameters can be passed through the backtest object with the `setSmootherParams()` function, and they can be extracted within the function with a call to `getSmootherParams()`.

The following is an example of how to implement your own smoother function:

```
# Add additional parameters used by 'mySmoother':
setSmootherParams(backtest) <- list(...)

# Creating a new smoother function:
mySmoother <- function(weights, spec, backtest)
{
   # Code:
   Params <- getSmootherParams(backtest)
   ... Code to return smoothed Weights ...

   # Return:
   smoothedWeights
}
```

Listing 25.10 Custom smoother function

Here, the `weights` are a numeric vector of weights, `spec` is the portfolio specification and `backtest` is the portfolio backtest object.

25.5 Rolling Analysis

Rolling portfolio analysis is commonly used to backtest the outcome of portfolio optimization over time. A rolling backtest on historical data can give us insights into the stability and performance of a given strategy. The strategy can then be optimized and improved before applying it to real markets. The function available in the `fPortfolioBacktest` package for portfolio backtesting is `portfolioBacktesting()`.

How to backtest a portfolio

The `portfolioBacktesting()` function has the following arguments:

```
> args(portfolioBacktesting)

function (formula, data, spec = portfolioSpec(),
    constraints = "LongOnly", backtest = portfolioBacktest(),
    trace = TRUE)
NULL
```

portfolioBacktesting() requires a formula expression that tells us which assets from the data set have to be analyzed against a given benchmark. The second argument, data, requests a multivariate timeSeries object, which contains at least the columns referenced in the formula expression. Portfolio specifications are given by the argument spec and portfolio constraints are given by constraints. The last argument backtest takes a portfolio backtest object, as described above.

Suppose we want to backtest the simple tangency portfolio strategy with the following three assets: the SBI (Swiss Bond Index), the SPI (Swiss Performance Index) and the SII (Swiss Immofunds Index), and we want to backtest this strategy against the Pictet Benchmark Index LP40.

```
> swxData <- 100 * SWX.RET
> swxSpec <- portfolioSpec()
> swxConstraints <- "LongOnly"
> swxBacktest <- portfolioBacktest()
> setWindowsHorizon(swxBacktest) <- "18m"
> setSmootherLambda(swxBacktest) <- "6m"
> swxFormula <- LP40 ~ SBI + SPI + SII

> swxPortfolios <- portfolioBacktesting(formula = swxFormula,
     data = swxData, spec = swxSpec, constraints = swxConstraints,
     backtest = swxBacktest, trace = FALSE)
```

The output on the screen is too lengthy to show, so here is a limited trace:

```
Portfolio Backtesting:
Assets:              SBI SPI SII
Benchmark:           LP40
Start Series:        2000-01-04
End Series:          2007-05-08
Type:                MV
Cov Estimator:       covEstimator
Solver:              solveRquadprog
Portfolio Windows:   equidistWindows
  Horizon:           18m
Portfolio Strategy:  tangencyStrategy
Portfolio Smoother:  emaSmoother
  doubleSmoothing:   TRUE
  Lambda:            6m

Portfolio Optimization:
Optimization Period     Target  Benchmark  Weights
```

```
2000-01-01 2000-12-31   0.043   0.003      0.000   0.998   0.000   *  1
2000-02-01 2001-01-31   0.018   0.014      0.753   0.247   0.000   *  1
2000-03-01 2001-02-28   0.010   0.005      0.828   0.125   0.048   *  1
2000-04-01 2001-03-31   0.014  -0.010      0.987   0.013   0.000   *  1
2000-05-01 2001-04-30   0.011   0.000      0.921   0.018   0.061   *  1
2000-06-01 2001-05-31   0.013   0.006      0.904   0.013   0.082   *  1
2000-07-01 2001-06-30   0.013   0.002      0.906   0.005   0.089   *  1
2000-08-01 2001-07-31   0.015  -0.009      0.874   0.000   0.126   *  1
...
```

The `portfolioBacktesting()` function returns a list with the following named elements:

```
portfolioBacktesting Values:
formula             backtest formula expression
data                multivariate returns
spec                portfolio specifications
constraints         portfolio constraints
backtest            portfolio backtest specifications
benchmarkName       name of the benchmark
assetsNames         names of the assets
weights             a matrix of portfolio weights
strategyList        a list of the invested portfolios
```

Listing 25.11 Named elements of the list returned by `portfolioBacktesting()`

How to smooth the weights from a backtest

As you can see from the above R output, the portfolio weights may vary strongly as market conditions change from one optimization period to the next. It is often a reasonable idea to smooth the weights first before applying them to the portfolio performance analysis. The function provided in the `fPortfolioBacktest` package for weights smoothing is `portfolioSmoothing()`.

```
> args(portfolioSmoothing)

function (object, backtest, trace = TRUE)
NULL
```

The first argument `object` is simply the list returned by the `portfolioBacktesting()` function followed by `backtest`, a portfolio backtest object.

An advantage of separating the process of backtesting from that of smoothing is that once we have backtested a portfolio, we can investigate the effect of alternative weights smoothers on the performance of the portfolio, without backtesting again and again for different types of smoothers.

In `fPortfolioBacktest` the default smoother function is the `emaSmoother()`, which applies an exponential moving average, EMA, filter to the portfolio weights. There are several control parameters for the EMA smoother function, but for the following example we will use the default settings, except that we start with an equally weighted portfolio and set the decay parameter for the EMA to `lambda = "12m"`.

```
> setSmootherInitialWeights(backtest) <- rep(1/3, 3)
> setSmootherLambda(backtest) <- "12m"

> swxSmooth <- portfolioSmoothing(object = swxPortfolios, backtest =
     swxBacktest)
```

The `portfolioSmoothing()` function returns a list with the following named elements:

```
portfolioSmoothing - Values:
smoothWeights       matrix of smoothed portfolio weights
monthlyAssets       timeSeries of monthly asset returns
monthlyBenchmark    timeSeries of monthly benchmark returns
portfolioReturns    timeSeries of cumulated monthly portfolio returns
benchmarkReturns    timeSeries of cumulated monthly benchmark returns
P                   timeSeries of monthly portfolio returns
B                   timeSeries of monthly benchmark returns
stats               matrix of basic backtest statistics
```

Listing 25.12 Named elements of the list returned by `portfolioSmoothing()`

The list returned by `portfolioSmoothing()` also inherits some of the named elements from the list returned by `portfolioBacktesting()`.

How to plot backtesting results

The function `backtestPlot()` provides graphs and statistics to summarize the backtesting results:

```
> backtestPlot
```

```
function (object, which = "all", labels = TRUE, ...)
{
    if (any(which == "all"))
        par(mfrow = c(3, 2), mar = c(1.5, 4, 5, 2), oma = c(5,
            1, 0, 1))
    if (any(which == "1") || which == "all")
        backtestAssetsPlot(object, labels, ...)
    if (any(which == "2") || which == "all")
        backtestWeightsPlot(object, labels, ...)
    if (any(which == "3") || which == "all")
        backtestRebalancePlot(object, labels, ...)
    if (any(which == "4") || which == "all")
        backtestPortfolioPlot(object, labels, ...)
    if (any(which == "5") || which == "all")
        backtestDrawdownPlot(object, labels, ...)
    if (any(which == "6") || which == "all")
        backtestReportPlot(object, ...)
    invisible()
}
<environment: namespace:fPortfolioBacktest>
```

The first argument `object` is the list returned by the `portfolioSmoothing()` function followed by `which`, either a numeric vector from 1 to 6 or `"all"` which plots all 6 graphs. For a plot of the current example, see Figure Figure 25.1.

How to print backtest statistics

`backtestStats()` is a wrapper function that gathers rolling statistics over each of the backtest windows.

```
> args(backtestStats)

function (object, FUN = "rollingSigma", ...)
NULL
```

The function requires an `object` returned by the function `portfolioSmoothing()` and the name of the `stats()` function given as a character string that computes the statistics. By default, the portfolio risk, σ, is calculated with the function `rollingSigma()`.

```
> defaultStats <- backtestStats(swxSmooth)
> head(defaultStats)
```

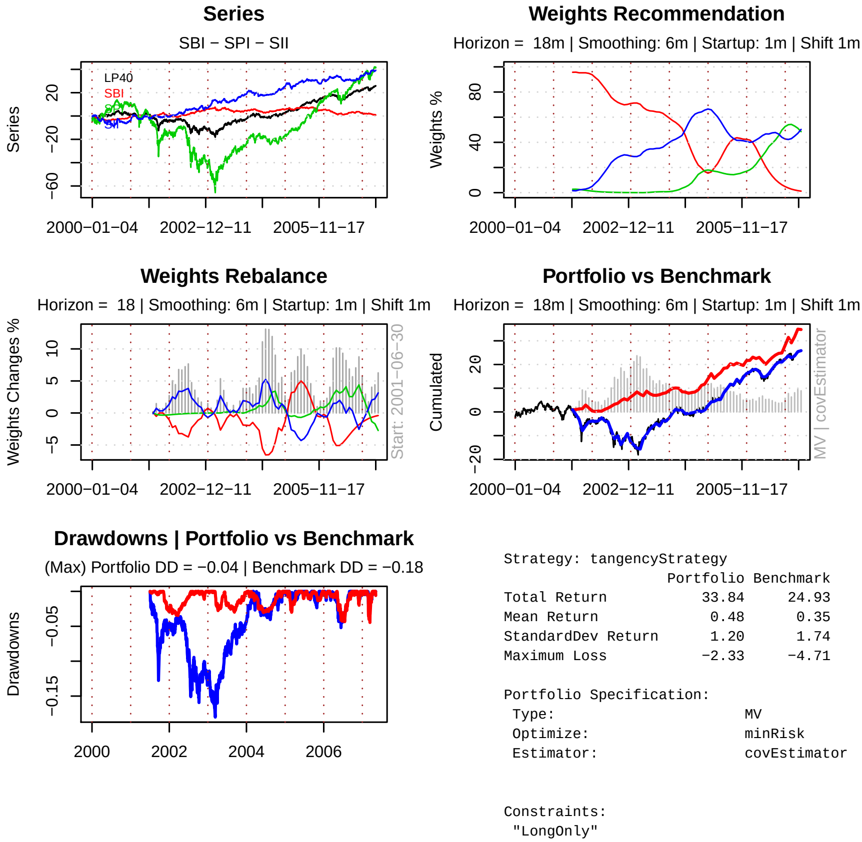


Figure 25.1 Portfolio backtesting for major Swiss indices: The backtesting is performed for a portfolio composed of three Swiss market indexes, equities, bonds and reits. The *Series* graph shows the time series indexes, the *Weights Recommendation* graph shows the smoothed recommended weights every month for the investment of the next month, the *Weights Rebalance* graph shows to which amount the weights were rebalanced, and the *Drawdowns* graph shows the drawdowns of the optimized portfolio at the end of each month in comparison to the benchmark. The table gives some characterizing numbers for the optimized portfolio and benchmark.

```
GMT
              Sigma
2001-06-29 0.095497
2001-07-31 0.093515
2001-08-31 0.094676
2001-09-28 0.093316
2001-10-31 0.085591
2001-11-30 0.092391
```

Bear in mind that there are other predefined rolling statistics functions available in `fPortfolioBacktest`:

```
rollingSigma         portfolio risk Sigma over time
rollingVaR           rolling Value at Risk
rollingCVaR          rolling Conditional Value at Risk
rollingDaR           rolling Drawdowns at Risk
rollingCDaR          rolling Conditional Drawdowns at Risk
rollingRiskBudgets   rolling Risk Budget
```

Listing 25.13 Predefined rolling statistics function in fPortfolioBacktest

How to write your own statistics functions

You can write your own `stats` functions to compute other statistics such as the Drawdown at Risk (DaR), Conditional Drawdown at Risk (CDaR), Shapiro-Wilk's test statistic, etc. Note that the function must take a list of `fPORTFOLIO` objects[1] as an argument, and additional arguments are passed in through the `...` argument in `backtestStats(object, stats, ...)`.

As an example, the following function, `rollingCDaR()`, returns a rolling estimate of the Conditional Drawdown at Risk.

```
> rollingCDaR

function (object)
{
    .cdar = function(x) {
        alpha = getAlpha(x)
```

[1] For example like the list returned by the `portfolioBacktesting()` function under the name `$strategyList`

```
        R = as.numeric(getSeries(x) %*% getWeights(x))
        dd = 100 * drawdowns(as.timeSeries(R)/100)
        z = quantile.default(dd, probs = alpha)
        mean(dd[dd <= z])
    }
    portfolios <- object$strategyList
    ans = sapply(portfolios, FUN = .cdar)
    dates = sapply(portfolios, function(x) rev(rownames(getSeries(x)))[1])
    alpha = getAlpha(portfolios[[1]])
    timeSeries(ans, charvec = dates, units = paste("CDaR", alpha,
        sep = "."))
}
<environment: namespace:fPortfolioBacktest>
```

The CDaR for the current example is:

```
> CDaRstats <- backtestStats(swxSmooth, FUN = "rollingCDaR")
> head(CDaRstats)

GMT
           CDaR.0.05
2001-06-29  -2.08980
2001-07-31  -1.77466
2001-08-31  -1.94771
2001-09-28  -1.57948
2001-10-31  -0.75097
2001-11-30  -0.53937
```

It can be very helpful to plot various risk measures; the covariance, Value-At-Risk, Conditional Value-At-Risk and Conditional Drawdowns-At-Risk are displayed in Figure 25.2.

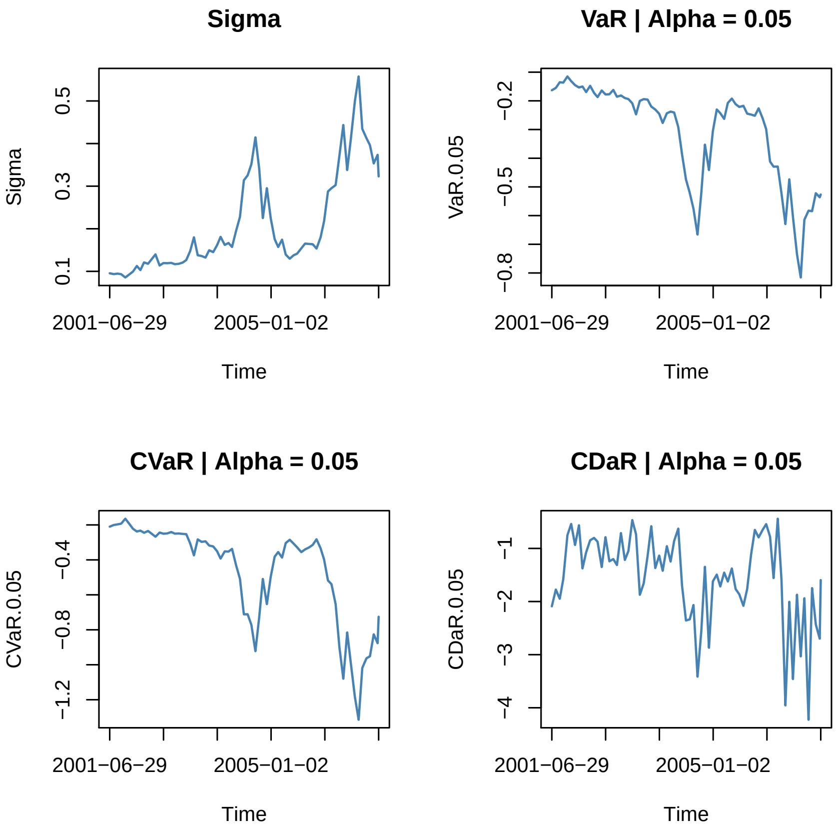


Figure 25.2 Rolling Analysis of the Swiss Performance Index: This plot shows the covariance, Value-At-Risk, Conditional Value-At-Risk and Conditional Drawdowns-At-Risk over time. The latter three all have a confidence level (α) of 0.05.

Chapter 26

Case Study: SPI Sector Rotation

Required R package(s):

```
> library(fPortfolioBacktest)
```

In this chapter we will demonstrate how to backtest portfolio strategies on realistic portfolios. The first case study concerns the sector rotation of Swiss equities.

26.1 SPI Portfolio Backtesting

The data we will be using for this case study are the returns from the Swiss Performance Index, SPI. This data set, `SPISECTOR.RET`, is comprised of 9 sectors; basic materials, industrials, consumer goods, health care, consumer service, telecommunications, utilities, finance and technology. For further details, see Section B.3.

To view the names of the individual sectors, type:

```
> colnames(SPISECTOR.RET)

 [1] "SPI"  "BASI" "INDU" "CONG" "HLTH" "CONS" "TELE" "UTIL" "FINA"
[10] "TECH"
```

The data set contains historical asset returns from January 2000 to June 2008, and the strategy we are going to backtest is to invest in the tangency portfolio

strategy with a fixed rolling window of 12 months shifted monthly. We call this investment type in the following *tangency strategy*. The portfolios will be optimized with the mean-variance Markowitz approach and we will use the sample covariance as the risk measure. First we specify the settings for the portfolio data, for the specification, for the constraints and finally for the portfolio backtest.

```
> spiData <- SPISECTOR.RET
> spiSpec <- portfolioSpec()
> spiConstraints <- "LongOnly"
> spiBacktest <- portfolioBacktest()
```

Then we specify the assets which should be used for backtesting.

```
> spiFormula <- SPI ~ BASI + INDU + CONG + HLTH + CONS + TELE +
    UTIL + FINA + TECH
```

Next, we optimize the rolling portfolios and perform the backtests.

```
> spiPortfolios <- portfolioBacktesting(formula = spiFormula,
    data = spiData, spec = spiSpec, constraints = spiConstraints,
    backtest = spiBacktest, trace = FALSE)
```

The weights of the first 12 months for the portfolios which are rebalanced every month are given by

```
> Weights <- round(100 * spiPortfolios$weights, 2)[1:12, ]
> Weights
```

```
           BASI INDU  CONG HLTH CONS  TELE  UTIL  FINA TECH
2000-12-31 0.00    0 45.71 0.00    0  0.00 29.16 17.25 7.87
2001-01-31 0.00    0 22.29 0.00    0  0.00 28.87 48.84 0.00
2001-02-28 0.00    0 29.15 3.24    0  0.00 33.50 34.12 0.00
2001-03-31 0.96    0 46.28 0.00    0  0.00 47.88  4.88 0.00
2001-04-30 0.00    0 38.39 0.00    0  0.00 46.53 15.08 0.00
2001-05-31 0.23    0 34.28 0.00    0  0.00 60.79  4.70 0.00
2001-06-30 2.13    0 45.85 0.00    0  0.00 52.02  0.00 0.00
2001-07-31 0.00    0 27.71 0.00    0  0.00 72.29  0.00 0.00
2001-08-31 0.00    0  4.14 0.00    0  8.15 87.70  0.00 0.00
2001-09-30 0.00    0  0.00 0.00    0 92.84  7.16  0.00 0.00
2001-10-31 0.00    0  0.00 0.00    0 84.80 15.20  0.00 0.00
2001-11-30 0.00    0  0.00 0.00    0 92.30  7.70  0.00 0.00
```

We see that the weights are fluctuating significantly and thus we smooth them to prevent to reduce a cost effective monthly rebalancing.

26.2 SPI Portfolio Weights Smoothing

We set the smoothing parameter `lambda` to 12 months to increase the smoothing effect and we have taken the recommended initial weights from the portfolio backtesting function as opposed to setting our own.

```
> setSmootherLambda(spiBacktest) <- "12m"
> spiSmoothPortfolios <- portfolioSmoothing(object = spiPortfolios,
     backtest = spiBacktest, trace = FALSE)
```

The weights during the first 12 months are now

```
> smoothWeights <- round(100 * spiSmoothPortfolios$smoothWeights,
     2)[1:12, ]
> smoothWeights
```

```
           BASI INDU  CONG HLTH CONS  TELE  UTIL  FINA TECH
2000-12-31 0.00    0 45.71 0.00    0  0.00 29.16 17.25 7.87
2001-01-31 0.00    0 45.16 0.00    0  0.00 29.15 18.00 7.69
2001-02-28 0.00    0 44.38 0.08    0  0.00 29.25 18.92 7.37
2001-03-31 0.02    0 43.87 0.13    0  0.00 29.76 19.24 6.97
2001-04-30 0.04    0 43.38 0.16    0  0.00 30.52 19.38 6.52
2001-05-31 0.05    0 42.81 0.19    0  0.00 31.79 19.12 6.04
2001-06-30 0.11    0 42.47 0.20    0  0.00 33.17 18.49 5.56
2001-07-31 0.15    0 41.88 0.20    0  0.00 35.08 17.60 5.08
2001-08-31 0.18    0 40.57 0.20    0  0.19 37.70 16.55 4.62
2001-09-30 0.19    0 38.66 0.19    0  2.52 38.85 15.40 4.17
2001-10-31 0.20    0 36.39 0.18    0  6.14 39.12 14.21 3.76
2001-11-30 0.20    0 33.90 0.17    0 10.77 38.56 13.03 3.37
```

Note that the process of the re-balancing now appears in a much smoother way.

26.3 SPI Portfolio Backtest Plots

```
> backtestPlot(spiSmoothPortfolios, cex = 0.6, font = 1, family = "mono")
```

Figure 26.1 summarizes the backtest results.

```
> backtestPlot(spiSmoothPortfolios, cex = 0.6, font = 1, family = "mono")
```

The function `backtestPlot()` shows five different views including the series, the recommended weights, the rebalancing of weights, the performance graphs and a drawdown plot.

26.4 SPI Performance Review

As we can see from Figure 26.1, if we had started the `tangency` strategy in January 2001, the total return for the portfolio would have been around 79.60% compared to 24.71% if we had invested in the SPI. Furthermore, during 2003 when the market was on a downward trend the portfolio was able to absorb the losses better than the SPI, with the maximum drawdown during that period for the portfolio being only -0.28 compared to -0.54 for the SPI. The amount of re-balancing seems reasonable as total weight changes per month were at most around 9%.

```
> netPerformance(spiSmoothPortfolios)

Net Performance % to 2008-10-31:
          1 mth 3 mths 6 mths  1 yr 3 yrs 5 yrs 3 yrs p.a. 5 yrs p.a.
Portfolio -0.20  -0.25  -0.31 -0.43 -0.26  0.46      -0.09       0.09
Benchmark -0.13  -0.17  -0.22 -0.38 -0.29  0.29      -0.10       0.06


Net Performance % Calendar Year:
           2001  2002 2003 2004 2005 2006 2007   YTD Total
Portfolio -0.10 -0.09 0.22 0.23 0.21 0.29 0.05 -0.39  0.42
Benchmark -0.25 -0.30 0.20 0.07 0.30 0.19 0.00 -0.32 -0.11
```

Over the last year the portfolio lost 13.04% in returns, 7% less than the SPI. For the past seven calendar years the portfolio strategy seemed to perform much better than the SPI, except for 2005. The portfolio strategy gave us relatively small losses in 2001 and 2002, and yielded significant gains in years 2003 to 2007.

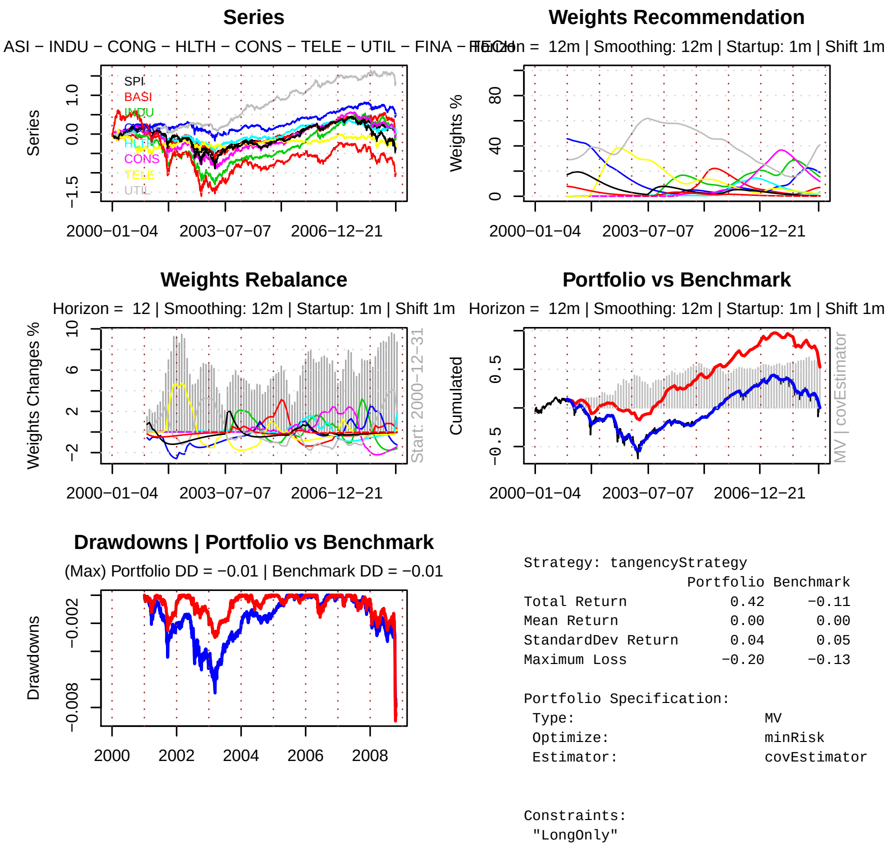


Figure 26.1 The five graphs show the results from portfolio backtesting with instruments from the SPI Subsectors. The graph to the upper left shows the subsector time series. To the upper right we see the weights recommendation over time. We have rolled a window with a 12 month horizon every month. The weights are smoothed with a double exponential EMA smoother with a time decay of 6 months. The end-of-month rebalancing is derived from the weights recommendation shown in the left graph in the middle of the graph sheets. The next graph to the right shows the development of the monthly returns over time. The last plot shows the rolling drawdowns for the optimized portfolio and the benchmark index.

Chapter 27

Case Study: GCC Index Rotation

Required R package(s):

```
> library(fPortfolioBacktest)
```

The MSCI GCC Countries Indices was launched in January 2006 to reflect growing investor interest in the Gulf region. The GCC index is a comprehensive family of equity markets traded in the GCC region, which covers Bahrain, Kuwait, Oman, Qatar and the United Arab Emirates. The index excludes Saudi Arabia because it is not open to foreign investment. In this case study, we will backtest the five indices open to foreign investment against the MSCI GCC index.

All of the indices have daily history going back to May 31, 2005 and the most recent data we have is to July 28, 2008. Again, the tangency strategy will be applied with a fixed rolling window of 12 months. However, this time the portfolios will be optimized with the Conditional Value at Risk (CVaR) approach with `alpha = 0.05`.

First, let us look at the column names of the GCC Index data set

```
> colnames(GCCINDEX.RET)

 [1] "BAHDSC"    "BAHSC"     "KUWDSC"    "OMADSC"    "OMASC"
 [6] "KSADSC"    "UAEDSC"    "UAESC"     "QATSC"     "GCCEXSASC"
[11] "GCCSC"
```

For further information on this data set, see Section B.5. Next, we define the specification

```
> gccData <- GCCINDEX.RET
> gccSpec <- portfolioSpec()
> setType(gccSpec) <- "CVaR"
> setSolver(gccSpec) <- "solveRglpk"
```

and then set the constraints.

```
> gccConstraints <- "LongOnly"
```

Let us use the default settings for the backtest, and select the instruments from which we build the portfolio, using the formula notation.

```
> gccBacktest <- portfolioBacktest()
> gccFormula <- GCCSC ~ BAHDSC + KUWDSC + OMADSC + UAEDSC +
    QATSC
```

Now, we are ready to combine all of the above:

```
> gccPortfolios <- portfolioBacktesting(formula = gccFormula,
    data = gccData, spec = gccSpec, constraints = gccConstraints,
    backtest = gccBacktest, trace = FALSE)
```

27.1 GCC Portfolio Weights Smoothing

Since we only have three years of historical data, we will change the smoothing parameter to `lambda="6m"`. Shortening the smoothing parameter (`lambda`) means that the portfolio is more responsive to changes in the market, which sounds like a good portfolio strategy but it does come with higher transaction costs because the portfolio is likely to be rebalanced whenever the market situation changes[1].

```
> setSmootherLambda(gccBacktest) <- "6m"
> gccSmooth <- portfolioSmoothing(object = gccPortfolios, backtest =
    gccBacktest,
    trace = FALSE)
```

[1] Note that the rolling windows are generated by the `equidistWindows()` function and that portfolios are re-balanced monthly

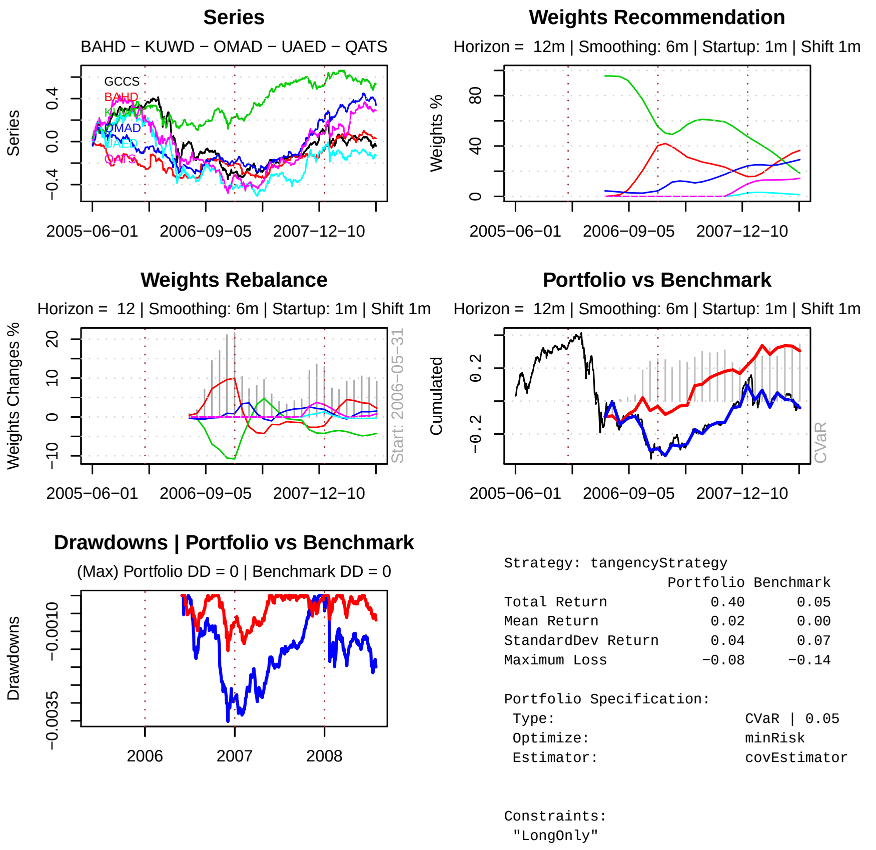


Figure 27.1 The five graphs show the results from portfolio backtesting with instruments from the GCC index. The graph to the upper left shows the five series which we have selected from the GCC indexes. To the upper right we see the weights recommendation over time. We have rolled a window with a 12 month horizon every month. The weights are smoothed with a double exponential EMA smoother with a time decay of 6 months. The end-of-month re-balancing is derived from the weights recommendation shown in the left graph in the middle of the graph sheets. The next graph to the right shows the development of the monthly returns over time. The last plot shows the rolling drawdowns for the optimized portfolio and the benchmark index, the GCC Market Index.

27.2 GCC Performance Review

The backtest plots suggest that the tangency strategy could also be an effective portfolio strategy for investing in the Gulf region. The results show that the total return for the portfolio over the two years was 43.07%, which is considerably greater than the benchmark at 5.39%. Maximum drawdown for the portfolio was about half that of the benchmark over the two years.

However, the backtest identified a few caveats with the strategy, the main one being diversification or the lack thereof, see Figure 27.1. The result of this poorly diversified portfolio is clustering of risk. In this case, the majority of the portfolio risk came from the Kuwait index. Furthermore, during the months where more than 15% of the portfolio was rebalanced, the transaction costs would reduce the actual return from the portfolio. Note that a risk-seeking investor may well justify these concerns with having higher returns, but a strategy that operates with a more diversified portfolio may be easier to market.

```
> netPerformance(gccSmooth)

Net Performance % to 2008-07-31:
          1 mth 3 mths 6 mths 1 yrs 2 yrs 2 yrs p.a.
Portfolio -0.03  -0.02   0.04  0.16  0.44       0.22
Benchmark -0.05  -0.09  -0.05  0.11  0.10       0.05


Net Performance % Calendar Year:
           2006 2007   YTD Total
Portfolio  0.06 0.25  0.09  0.40
Benchmark -0.19 0.38 -0.14  0.05
```

27.3 Alternative Strategy

An alternative strategy would be to start with an equally weighted portfolio. In order to reduce overall weight changes we increase `lambda` from `"6m"` to `"12m"`.

```
> setSmootherLambda(gccBacktest) <- "12m"
> setSmootherInitialWeights(gccBacktest) <- rep(1/5, 5)
```

```
> gccSmoothAlt <- portfolioSmoothing(object = gccPortfolios,
     backtest = gccBacktest)
```

Notice how overall weight changes have dropped to below 8% per month and the portfolio is much more diversified, see Figure 27.2. Total return over the two years is lower than the previous strategy (43.07% to 31.63%) but, surprisingly, it is still considerably greater than the benchmark.

```
> netPerformance(gccSmoothAlt)

Net Performance % to 2008-07-31:
          1 mth 3 mths 6 mths 1 yrs 2 yrs 2 yrs p.a.
Portfolio -0.03  -0.02   0.02  0.17  0.35       0.17
Benchmark -0.05  -0.09  -0.05  0.11  0.10       0.05


Net Performance % Calendar Year:
           2006 2007   YTD Total
Portfolio  0.01 0.24  0.07  0.32
Benchmark -0.19 0.38 -0.14  0.05
```

From the above output we can see that the defining point for this strategy was in 2006, where it avoided losses even when the majority of the market was in decline. The ability to minimize losses when the market conditions are poor seems to be a recurring feature of this tangency strategy. However, in saying that, we have only backtested this strategy with the two examples, further investigations are required to support this statement.

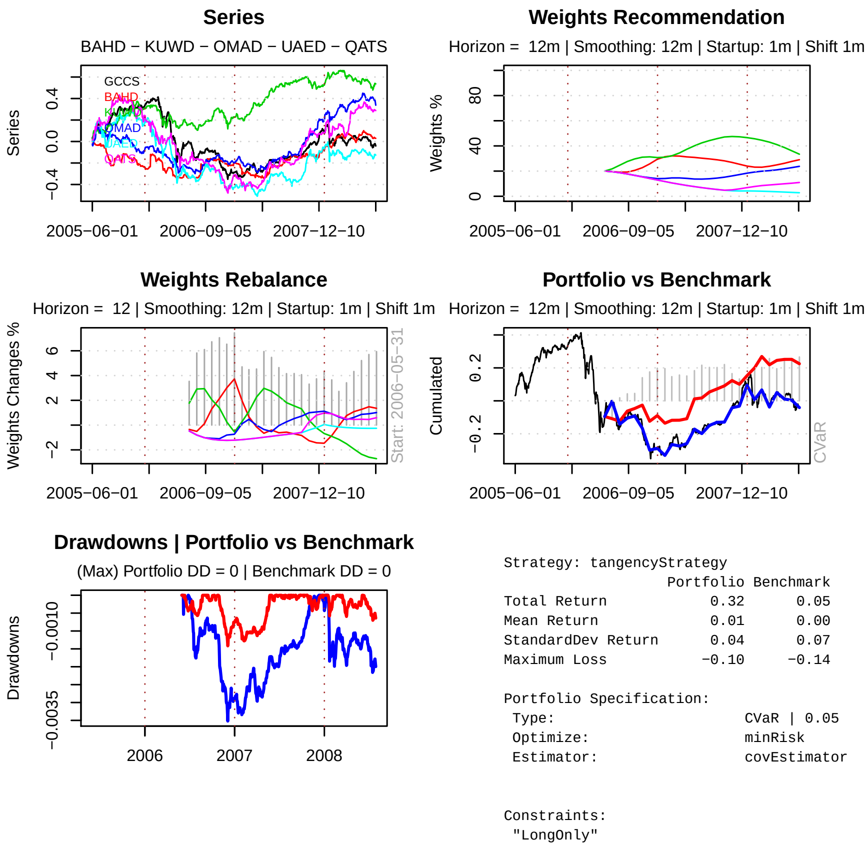


Figure 27.2 Backtesting for the GCC Index with alternative strategy: The graph to the upper left shows the five series which we have selected from the GCC indexes. To the upper right we see the weights recommendation over time. We have rolled a window with a 12 month horizon every month. The weights are smoothed with a double exponential EMA smoother with a time decay of 12 months, as opposed to 6 months in the previous figure. The end-of-month re-balancing is derived from the weights shown in the left graph in the middle row. The next graph to the right shows the development of the monthly returns over time. The last plot shows the rolling drawdowns for the optimized portfolio and the benchmark index, the GCC Market Index.

Part VII
Appendix

Appendix A
Packages Required for this Ebook

Required R package(s):

```
> library(fPortfolio)
```

In the following we briefly describe the packages required for this ebook. There are two major packages named `fPortfolio` and `fPortfolioBacktest`. The first package, `fPortfolio`, is the basic package which allows us to model mean-variance and mean-CVaR portfolios with linear constraints and to analyze the data sets of assets used in the portfolios. The second package, `fPortfolioBacktest`, adds additional functionality, including backtesting functions over rolling windows.

A.1 Rmetrics Package: fPortfolio

`fPortfolio` (Würtz & Chalabi, 2009a) contains the R functions for solving mean-variance and mean-CVaR portfolio problems with linear constraints. The package depends on the contributed R packages `quadprog` (Weingessel, 2004) for quadratic programming problems and `Rglpk` (Theussl & Hornik, 2009) with the appropriate solvers for quadratic and linear programming problems.

```
> listDescription(fPortfolio)
```

```
fPortfolio Description:

 Package:       fPortfolio
 Version:       2100.78
 Revision:      4093
 Date:          2009-04-19
 Title:         Rmetrics - Portfolio Selection and Optimization
 Author:        Diethelm Wuertz
 Depends:       R (>= 2.7.0), methods, MASS, timeDate, timeSeries,
               fBasics, fCopulae, fAssets (>= 2100.77), quadprog,
               Rglpk, Rsymphony, robustbase
 Suggests:      corpcor, covRobust, RUnit, tcltk
 Maintainer:    Rmetrics Core Team <Rmetrics-core@r-project.org>
 Description:   Environment for teaching "Financial Engineering and
               Computational Finance"
 NOTE:          SEVERAL PARTS ARE STILL PRELIMINARY AND MAY BE
               CHANGED IN THE FUTURE. THIS TYPICALLY INCLUDES
               FUNCTION AND ARGUMENT NAMES, AS WELL AS DEFAULTS
               FOR ARGUMENTS AND RETURN VALUES.
 LazyLoad:      yes
 LazyData:      yes
 License:       GPL (>= 2)
 URL:           http://www.rmetrics.org
 Built:         R 2.9.0; ; 2009-04-20 11:36:43 UTC; unix
```

A.2 Rmetrics Package: timeDate

`timeDate` (Würtz & Chalabi, 2009c) 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 centre, 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 centres 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)

 timeDate Description:

  Package:       timeDate
  Version:       290.85
  Revision:      4054
  Date:          2009-04-15
  Title:         Rmetrics - Chronological and Calendarical Objects
  Author:        Diethelm Wuertz and Yohan Chalabi
  Depends:       R (>= 2.6.0), graphics, utils, stats, methods
  Suggests:      RUnit
  Maintainer:    Rmetrics Core Team <Rmetrics-core@r-project.org>
  Description:   Environment for teaching "Financial Engineering and
               Computational Finance"
  NOTE:          SEVERAL PARTS ARE STILL PRELIMINARY AND MAY BE
               CHANGED IN THE FUTURE. THIS TYPICALLY INCLUDES
               FUNCTION AND ARGUMENT NAMES, AS WELL AS DEFAULTS
               FOR ARGUMENTS AND RETURN VALUES.
  LazyLoad:      yes
  LazyData:      yes
  License:       GPL (>= 2)
  URL:           http://www.rmetrics.org
  Built:         R 2.9.0; ; 2009-04-16 08:59:02 UTC; unix
```

A.3 Rmetrics Package: timeSeries

`timeSeries` (Würtz & Chalabi, 2009d) 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)

 timeSeries Description:

  Package:       timeSeries
  Version:       2100.84
  Revision:      4093
  Date:          2009-04-19
  Title:         Rmetrics - Financial Time Series Objects
  Author:        Diethelm Wuertz and Yohan Chalabi
  Depends:       R (>= 2.6.0), graphics, grDevices, methods, stats,
               utils, timeDate (>= 290.85)
  Suggests:      robustbase, RUnit
  Maintainer:    Rmetrics Core Team <Rmetrics-core@r-project.org>
  Description:   Environment for teaching "Financial Engineering and
               Computational Finance"
  NOTE:          SEVERAL PARTS ARE STILL PRELIMINARY AND MAY BE
               CHANGED IN THE FUTURE. THIS TYPICALLY INCLUDES
               FUNCTION AND ARGUMENT NAMES, AS WELL AS DEFAULTS
               FOR ARGUMENTS AND RETURN VALUES.
  LazyLoad:      yes
  LazyData:      yes
  License:       GPL (>= 2)
  URL:           http://www.rmetrics.org
  Built:         R 2.9.0; ; 2009-04-20 11:36:18 UTC; unix
```

A.4 Rmetrics Package: fBasics

`fBasics` (Würtz, 2009a) provides basic functions to analyze and to model data sets of financial asset returns.

```
> listDescription(fBasics)

 fBasics Description:
```

```
  Package:       fBasics
  Version:       2100.78
  Revision:      4093
  Date:          2009-04-19
  Title:         Rmetrics - Markets and Basic Statistics
  Author:        Diethelm Wuertz and many others, see the SOURCE
                file
  Depends:       R (>= 2.6.0), MASS, methods, timeDate, timeSeries
                (>= 2100.83)
  Suggests:      akima, spatial, RUnit, tcltk
  Maintainer:    Rmetrics Core Team <Rmetrics-core@r-project.org>
  Description:   Environment for teaching "Financial Engineering and
                Computational Finance"
  NOTE:          SEVERAL PARTS ARE STILL PRELIMINARY AND MAY BE
                CHANGED IN THE FUTURE. THIS TYPICALLY INCLUDES
                FUNCTION AND ARGUMENT NAMES, AS WELL AS DEFAULTS
                FOR ARGUMENTS AND RETURN VALUES.
  LazyLoad:      yes
  LazyData:      yes
  License:       GPL (>= 2)
  URL:           http://www.rmetrics.org
  Built:         R 2.9.0; universal-apple-darwin8.11.1; 2009-04-20
                11:37:31 UTC; unix
```

A.5 Rmetrics Package: fAssets

`fAssets` (Würtz, 2009a) provides functions to analyze and to model multivariate data sets of financial asset returns. The package depends on R's recommended packages `methods` and `MASS` (Venables & Ripley, 2008). It also depends on the contributed R packages `sn` (which depends on `mnormt`), and `robustbase`.

```
> listDescription(fAssets)

 fAssets Description:

  Package:       fAssets
  Version:       2100.78
  Revision:      4093
  Date:          2009-04-19
  Title:         Rmetrics - Assets Selection and Modelling
  Author:        Diethelm Wuertz and many others, see the SOURCE
```

```
                file
  Depends:        R (>= 2.6.0), methods, sn, MASS, robustbase,
                timeDate, timeSeries, fBasics, fCopulae (>=
                2100.76)
  Suggests:       RUnit
  Maintainer:     Rmetrics Core Team <Rmetrics-core@r-project.org>
  Description:    Environment for teaching "Financial Engineering and
                Computational Finance"
  NOTE:           SEVERAL PARTS ARE STILL PRELIMINARY AND MAY BE
                CHANGED IN THE FUTURE. THIS TYPICALLY INCLUDES
                FUNCTION AND ARGUMENT NAMES, AS WELL AS DEFAULTS
                FOR ARGUMENTS AND RETURN VALUES.
  LazyLoad:       yes
  LazyData:       yes
  License:        GPL (>= 2)
  URL:            http://www.rmetrics.org
  Built:          R 2.9.0; universal-apple-darwin8.11.1; 2009-04-20
                11:38:39 UTC; unix
```

A.6 Contributed R Package: quadprog

`quadprog` implements the dual method of Goldfarb & Idnani (1982, 1983) for solving quadratic programming problems with linear constraints. The original S package was written by Turlach, the R port was done by Weingessel (2004), who also maintains the package. The contributed R package `quadprog` is the default solver in Rmetrics for quadratic programming problems.

```
> listDescription(quadprog)

 quadprog Description:

  Package:        quadprog
  Version:        1.4-11
  Date:           2007-07-12
  Title:          Functions to solve Quadratic Programming Problems.
  Author:         S original by Berwin A. Turlach
                <berwin.turlach@anu.edu.au> R port by Andreas
                Weingessel <Andreas.Weingessel@ci.tuwien.ac.at>
  Maintainer:     Andreas Weingessel
                <Andreas.Weingessel@ci.tuwien.ac.at>
  Description:    This package contains routines and documentation
                for solving quadratic programming problems.
```

```
License:       GPL-2
Packaged:      Fri Jul 13 00:01:29 2007; hornik
Built:         R 2.9.0; universal-apple-darwin8.11.1; 2009-04-20
             18:48:46 UTC; unix
```

A.7 Contributed Package: Rglpk

Rglpk is the R interface to the GNU Linear Programing Kit, GLPK version 4.33, written and maintained by Makhorin (2008). GLPK is open source software for solving large-scale linear programming, mixed integer linear programming, and other related problems. The R port provides a high level interface to the low level C interface of the C solver. The interface was written by Theussl & Hornik (2009), the former author is also the maintainer of the package. The contributed R package Rglpk is Rmetrics' default solver for linear programming problems.

```
> listDescription(Rglpk)

 Rglpk Description:

  Package:                       Rglpk
  Version:                       0.2-9
  Date:                          2009-03-30
  Title:                         R/GNU Linear Programming Kit
                               Interface
  Author:                        Stefan Theussl and Kurt Hornik
  Maintainer:                    Stefan Theussl
                               <stefan.theussl@wu-wien.ac.at>
  Description:                   R interface to the GNU Linear
                               Programing Kit (GLPK version 4.37).
                               GLPK is open source software for
                               solving large-scale linear
                               programming (LP), mixed integer
                               linear programming (MILP) and other
                               related problems.
  Depends:                       R (>= 2.7.0)
  License:                       GPL-2
  URL:                           http://R-Forge.R-project.org/projects/rglp/,
                               http://www.gnu.org/software/glpk/
  Repository:                    CRAN
  Repository/R-Forge/Project:    rglp
```

```
Repository/R-Forge/Revision:   46
Publication/Date:              2009-03-30 13:48:12
Packaged:                      Tue Mar 31 04:56:59 2009; rforge
Date/Publication:              2009-03-31 08:55:52
Built:                         R 2.9.0;
                             universal-apple-darwin8.11.1;
                             2009-04-19 19:59:01 UTC; unix
```

A.8 Recommended Packages from base R

`methods` (R Development Core Team, 2009a), and `MASS` (Venables & Ripley, 2008) are used by Rmetrics. The two packages are recommended R packages, which means that they are installed with the base R environment.

A.9 Contributed RPackages

`sn` (Azzalini, Azzalini) and `robustbase` (Rousseeuw et al., 2008) are two contributed packages used by Rmetrics. The package `sn` comes with functions for manipulating skew-normal and skew-t probability distributions, and for fitting them to data, in the scalar and in the multivariate case. `sn` itself depends on the package `mnormt` (Azzalini, 2009), which provides functions for computing the density and the distribution function of, and for generating random vectors from, the multivariate normal and multivariate `t` distributions. The package `robustbase` provides 'essential' robust statistics. The goal of the package is to provide tools allowing to analyze data with robust methods. This includes regression methodology including model selections and multivariate statistics where the authors strive to cover the book 'Robust Statistics, Theory and Methods' by Maronna, Martin & Yohai (2006).

A.10 Rmetrics Package: fPortfolioBacktest

`fPortfolioBacktest` (Würtz & Chalabi, 2009b) is used to perform portfolio backtests together with a performance analysis for rolling portfolios.

```
> listDescription(fPortfolioBacktest)

fPortfolioBacktest Description:

 Package:       fPortfolioBacktest
 Version:       2100.3
 Revision:      4097
 Date:          2009-04-20
 Title:         Rmetrics - Portfolio Backtesting
 Author:        Diethelm Wuertz, Yohan Chalabi, William Chen
 Depends:       R (>= 2.6.0), methods, timeDate, timeSeries,
              fBasics, fAssets, fPortfolio (>= 2100.77)
 Maintainer:    Rmetrics Core Team <Rmetrics-core@r-project.org>
 Description:   Environment for teaching "Financial Engineering and
              Computational Finance"
 NOTE:          SEVERAL PARTS ARE STILL PRELIMINARY AND MAY BE
              CHANGED IN THE FUTURE. THIS TYPICALLY INCLUDES
              FUNCTION AND ARGUMENT NAMES, AS WELL AS DEFAULTS
              FOR ARGUMENTS AND RETURN VALUES.
 LazyLoad:      yes
 LazyData:      yes
 License:       GPL (>= 2)
 URL:           http://www.rmetrics.org
 Built:         R 2.9.0; ; 2009-04-23 09:48:16 UTC; unix
```

Appendix B

Description of Data Sets

Required R package(s):

```
> library(fPortfolio)
```

In the following be briefly describe the data sets used in this ebook.

B.1 Data Set: SWX

SWX stands for the Swiss Exchange in Zurich. The SWX provides downloads for historical time series including equities, bonds, reits, their indices, and many other financial instruments. The data set `SWX`, which we provide here, contains three daily SWX market indices, the Swiss Performance Index `SPI`, the Swiss Bond Index `SBI`, and the Swiss Immofunds Index `SII`. In addition, the data set contains Pictet's Pension Fund Indices `LP25`, `LP40`, `LP60` from the `LPP2000` index family.

```
> colnames(SWX)

[1] "SBI"  "SPI"  "SII"  "LP25" "LP40" "LP60"

> range(time(SWX))

GMT
[1] [2000-01-03] [2007-05-08]

> nrow(SWX)
```

```
[1] 1917
```

B.2 Data Set: LPP2005

Pictet is one of Switzerland's largest private banks. The bank is well known for its Swiss Pension Fund Benchmarks `LPP2000` and `LPP2005`. The family of Pictet LPP indices was created in 1985 with the introduction of new regulations in Switzerland governing the investment of pension fund assets. Since then it has established itself as the authoritative pension fund index for Switzerland. In 2000, a family of three indices, called `LP25`, `LP40`, `LP60`, where the number denotes increasing risk profiles, was introduced to provide a suitable benchmark for Swiss pension funds. During the last years, new investment instruments have become available for alternative asset classes. With Pictet's LPP2005 indices the bank took this new situation into consideration. The `LPP2005` keeps the family of the three indices now named `LPP25`, `LPP40`, and `LPP60` by adding 'plus' to the name represented by the second `P` in the index names.

```
> colnames(LPP2005)

[1] "SBI"   "SPI"   "SII"   "LMI"   "MPI"   "ALT"   "LPP25" "LPP40" "LPP60"

> range(time(LPP2005))

GMT
[1] [2005-11-01] [2007-04-11]

> nrow(LPP2005)

[1] 377
```

B.3 Data Set: SPISECTOR

The `SPISECTOR` data set also provides data from the Swiss exchange. It covers the *Swiss Performance Index*, SPI, and nine of its sectors. The sectors include basic materials, industrials, consumer goods, health care, consumer services, telecommunications, utilities, financial, and technology. The oil and gas sector index is not included since it was introduced later than the others.

```
> colnames(SPISECTOR)

 [1] "SPI"  "BASI" "INDU" "CONG" "HLTH" "CONS" "TELE" "UTIL" "FINA" "TECH"

> range(time(SPISECTOR))

GMT
[1] [1999-12-30] [2008-10-17]

> nrow(SPISECTOR)

[1] 2216
```

B.4 Data Set: SMALLCAP

Scherer & Martin (2005) comes with several CRSP (Center for Research in Security Prices) data sets used in the book in examples. These data sets contain monthly data records of market-cap-weighted equities recorded between 1997 and 2001. One of the data sets, with 20 small cap equities, the MARKET index and the T90 rates are made available in the Rmetrics data fileSMALLCAP[1].

```
> colnames(SMALLCAP)

 [1] "MODI"   "MGF"    "MEE"    "FCEL"   "OII"    "SEB"    "RML"
 [8] "AEOS"   "BRC"    "CTC"    "TNL"    "IBC"    "KWD"    "TOPP"
[15] "RARE"   "HAR"    "BKE"    "GG"     "GYMB"   "KRON"   "MARKET"
[22] "T90"

> range(time(SMALLCAP))

GMT
[1] [1997-01-31] [2001-12-31]

> nrow(SMALLCAP)

[1] 60
```

[1] Please note that the data provided in the Rmetrics data file are not those from the CRSP data base. The data records were obtained from free sources, such as Yahoo, amongst others. Therefore, the SMALLCAP data records are exactly the same as those from the CRSP database.

B.5 Data Set: GCCINDEX

The Gulf Cooperation Council, GCC, is an organization of six Arab states which share many social and economic objectives. These states are Saudi Arabia, Bahrain, Oman, Qatar, United Arab Emirate, and Kuwait. The index was launched by MSCI Barra in January 2006 to reflect growing investor interest in this region. The GCC Countries Indices offer broad coverage (up to 99MSCI Barra maintains two series indices for the GCC and Arabian Markets. One is applicable to international investors, while the domestic series is aimed at investors not constrained by foreign ownership limits). The indices have daily history back to May 31, 2002.

```
> colnames(GCCINDEX)

 [1] "BAHDSC"    "BAHSC"     "KUWDSC"    "OMADSC"    "OMASC"
 [6] "KSADSC"    "UAEDSC"    "UAESC"     "QATSC"     "GCCEXSASC"
[11] "GCCSC"

> range(time(GCCINDEX))

GMT
[1] [2005-05-31] [2008-07-28]

> nrow(GCCINDEX)

[1] 825
```

Bibliography

Aragon, T. (2008). *EpiTools: R Package for Epidemiologic Data and Graphics*. cran.r-project.org. 104

Azzalini, A. *The sn package: The skew-normal and skew-t distributions*. cran.r-project.org. 396

Azzalini, A. (2009). *The mnormt package: The multivariate normal and t distributions*. cran.r-project.org. 396

Azzalini, A. & Capitanio, A. (1999). Statistical applications of the multivariate skew-normal distribution. *J.Roy.Statist.Soc*, *61*, 579–602. 121

Azzalini, A. & Capitanio, A. (2003). Distributions generated by perturbation of symmetry with emphasis on a multivariate skew-t distribution. *J.Roy.Statist.Soc*, *65*, 367–389. 121

Azzalini, A. & Dalla Valle, A. (1996). The multivariate skew-normal distribution. *Biometrika*, *83*(4), 715–726. 121

Bacon, C. R. (2008). *Practical Portfolio Performance Measurement and Attribution* (2 ed.). John Wiley & Sons. 32

Carl, P. & Peterson, B. G. (2008). *PerformanceAnalytics*. cran.r-project.org. 34, 109

Chambers, J. M., Cleveland, W. S., Kleiner, B., & Tukey, P. A. (1983). *Graphical Methods for Data Analysis*. Wadsworth & Brooks/Cole. 83, 139

Cleveland, W. S. (1979). Robust locally weighted regression and smoothing scatterplots. *J. Amer. Statist. Assoc.*, *74*, 829–836. 36

Cleveland, W. S. (1981). Lowess: A program for smoothing scatterplots by robust locally weighted regression. *The American Statistician*, *35*, 54. 36

De Giorgi, E. G. (2002). A note on portfolio selection under various risk measures. Working Paper 122. 305, 306

Filzmoser, P., Garrett, R. G., & Reimann, C. (2005). Multivariate outlier detection in exploration geochemistry. *Computers & Geosciences*, *31*(5), 579–587. 64

Forgy, E. W. (1965). Cluster analysis of multivariate data: Efficiency vs interpretability of classifications. *Biometrics*, *21*, 768–769. 131

Friendly, M. (2002). Corrgrams: Exploratory displays for correlation matrices. *The American Statistician*, *56*, 316–324. working paper. 152, 153

Gnanadesikan, R. & Kettenring, J. R. (1972). Robust estimates, residuals and outlier detection with multiresponse data. *Biometrics*, *28*, 81–124. 59, 288

Goldfarb, D. & Idnani, A. (1982). Dual and primal-dual methods for solving strictly convex quadratic programs. In J. Hennart (Ed.), *Numerical Analysis* (pp. 226–239). Springer. 185, 227, 394

Goldfarb, D. & Idnani, A. (1983). A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, (27), 1–33. 227, 394

Gschwandtner, M. & Filzmoser, P. (2009). *mvoutlier: Multivariate Outlier Detection Based on Robust Methods*. cran.r-project.org. 63

Hartigan, J. A. & Wong, M. A. (1979). A k-means clustering algorithm. *Applied Statistics*, *28*, 100–108. 131

Hyndman, R. J. & Fan, Y. (1996). Sample quantiles in statistical packages. *The American Statistician*, *50*, 361–365. 47

Ibanez, F., Grosjean, P., & Etienne, M. (2009). *pastecs: Package for Analysis of Space-Time Ecological Series*. cran.r-project.org. 37

Jarek, S. (2009). *mvnormtest: Normality test for multivariate variables*. cran.r-project.org. 117

Kaufman, L. & Rousseeuw, P. J. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*. New York: Wiley. 125

Keitt, T. (2007). *colorRamps*. cran.r-project.org. 107

Kotsiantis, S. & Pintelas, P. (2004). Combining bagging and boosting. *International Journal of Computational Intelligence*, *1*(4), 324–333. 177

Lloyd, S. P. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, *28*, 128–137. 131

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, (pp. 281–297)., Berkeley. University of California Press. 131

Maechler, M., Rousseeuw, P., Struyf, A., & Hubert, M. (2009). *The cluster Package*. cran.r-project.org. 133

Makhorin, A. (2008). *GNU Linear Programming Kit Reference Manual*. www.gnu.org/software/glpk. 185, 395

Marazzi, A. (1993). *Algorithms, Routines and S Functions for Robust Statistics* (2 ed.). Chapman & Hall. 277

Markowitz, H. (1952). Portfolio selection. *Journal of Finance*, *7*, 77–91. 221, 223

Maronna, R. & Zamar, R. (2002). Robust estimates of location and dispersion of high-dimensional datasets. *Technometrics*, *44*(4), 307–317. 59, 288

Maronna, R. A., Martin, D. R., & Yohai, V. J. (2006). *Robust Statistics: Theory and Method* (1 ed.). Wiley. 396

Neuwirth, E. (2007). *RColorBrewer*. cran.r-project.org. 109

Opgen-Rhein, R. & Strimmer, K. (2007). Accurate ranking of differentially expressed genes by a distribution-free shrinkage approach. *Statistical Applications in Genetics and Molecular Biology*, *6*(1). 177

Pflug, G. (2000). Some remarks on the value-at-risk and the conditional value-at-risk. In S. Uryasev (Ed.), *Probabilistic Constrained Optimization*, volume 38 (pp. 272–281). Dordrecht: Kluwer. 301

R Development Core Team (2009a). *methods*. cran.r-project.org. 396

R Development Core Team (2009b). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. xv

Rizzo, M. L. (2002). *A New Rotation Invariant Goodness-of-Fit Test*. PhD thesis, Bowling Green State University. 117

Rizzo, M. L. & Szekely, G. J. (2008). *energy: E-statistics*. cran.r-project.org. 118, 119

Rockafellar, R. & Uryasev, S. (2000). Optimization of conditional value-at-risk. *The Journal of Risk*, *2*(3), 21–41. 2, 301

Rousseeuw, P., Croux, C., Todorov, V., Ruckstuhl, A., Salibian-Barrera, M., Verbeke, T., & Maechler, M. (2008). *The robustbase package: Basic Robust Statistics*. cran.r-project.org. 57, 59, 176, 279, 396

Rousseeuw, P. & Van Driessen, K. (1999). A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, *41*, 212–223. 55, 279

Rousseeuw, P. J. (1985). Multivariate estimation with high breakdown point. In W. e. a. Grossmann (Ed.), *Mathematical Statistics and Applications*, volume B (pp. 283–297). Budapest: Akademiai Kiado. 55

Rousseeuw, P. J. & Leroy, A. M. (1987). *Robust Regression and Outlier Detection*. Wiley. 281

Royston, P. (1982). An extension of shapiro and wilk's w test for normality to large samples. *Applied Statistics*, *31*, 115–124. 117

Ryan, J. A. (2008). *IBrokers: R API to Interactive Brokers Trader Workstation*. cran.r-project.org. 32

Sams, R. (2009). *RBloomberg*. cran.r-project.org. 32

Schaefer, J., Opgen-Rhein, R., & Strimmer, K. (2008). *The corpcor package: Efficient Estimation of Covariance and (Partial) Correlation*. cran.r-project.org. 61, 176, 292

Schäfer, J. & Strimmer, K. (2005). A shrinkage approach to large-scale covariance matrix estimation and implications for functional genomics. *Statistical Applications in Genetics and Molecular Biology*, *4*(1). 176, 292

Scherer, B. & Martin, R. D. (2005). *Introduction to Modern Portfolio optimization with NUOPT and S-PLUS* (1 ed.). New York: Springer New York. 228, 401

Sharpe, W. F. (1994). The sharpe ratio. *Journal of Portfolio Management*, *21*(1), 49–58. 227

Szekely, G. & Rizzo, M. (2005). A new test for multivariate normality. *Journal of Multivariate Analysis*, *93*(1), 58–80. 117

Szekely, G. J. (1989). Potential and kinetic energy in statistics. Lecture notes, Budapest Institute of Technology (Technical University). 117

Szekely, G. J., Rizzo, M. L., & Bakirov, N. K. (2007). Measuring and testing independence by correlation of distances. *Annals of Statistics*, *35*(6), 2769–2794. 118

Theussl, S. & Hornik, K. (2009). *The Rglpk package: R/GNU Linear Programming Kit Interface*. cran.r-project.org. 389, 395

Tukey, J. W. (1977). *Exploratory Data Analysis*. Addison-Wesley. 85

Tütüncü, R. H., Toh, K. C., & Todd, M. J. (2003). Solving semidefinite-quadratic-linear programs using sdpt3. *Mathematical Programming*, *95*(2), 189–217. 307

Vanini, P. & Vignola, L. (2001). Markowitz model. 224

Venables, W. N. & Ripley, B. D. (2008). *The MASS Package*. cran.r-project.org. 54, 55, 176, 393, 396

Wang, N. & Raftery, A. (2002). Nearest neighbor variance estimation (nnve): Robust covariance estimation via nearest neighbor cleaning. *Journal of the American Statistical Association*, *97*, 994–1019. 59, 177

Wang, N., Raftery, A., & Fraley, C. (2008). *The covRobust package: Robust Covariance Estimation via Nearest Neighbor Cleaning*. cran.r-project.org. 59, 177

Weingessel, A. (2004). *quadprog - Functions to Solve Quadratic Programming Problems*. cran.r-project.org. 227, 389, 394

Wright, K. (2009). *corrgram: Plot a correlogram*. cran.r-project.org. 152

Würtz, D. (2009a). *The fBasics Package*. cran.r-project.org. 392, 393

Würtz, D. (2009b). *fCopulae: Rmetrics - Dependence Structures with Copulae*. cran.r-project.org. 121

Würtz, D. (2009c). *The fImport Package*. cran.r-project.org. 11, 32

Würtz, D. & Chalabi, Y. (2009a). *The fPortfolio Package*. cran.r-project.org. 389

Würtz, D. & Chalabi, Y. (2009b). *The fPortfolioBacktest Package*. cran.r-project.org. 397

Würtz, D. & Chalabi, Y. (2009c). *The timeDate Package*. cran.r-project.org. 390

Würtz, D. & Chalabi, Y. (2009d). *The timeSeries Package*. cran.r-project.org. 391

Index

D

E

F

G

H

I

K

About the Authors

Diethelm Würtz is private lecturer at the Institute for Theoretical Physics, ITP, and for the Curriculum Computational Science and Engineering, CSE, at the Swiss Federal Institute of Technology in Zurich. He teaches Econophysics at ITP and supervises seminars in Financial Engineering at CSE. Diethelm is senior partner of Finance Online, an ETH spin-off company in Zurich, and co-founder of the Rmetrics Association.

Yohan Chalabi has a master in Physics from the Swiss Federal Institute of Technology in Lausanne. He is now a PhD student in the Econophysics group at ETH Zurich at the Institute for Theoretical Physics. Yohan is a co-maintainer of the Rmetrics packages.

William Chen has a master in statistics from University of Auckland in New Zealand. In the summer of 2008, he did a Student Internship in the Econophysics group at ETH Zurich at the Institute for Theoretical Physics. During his three months internship, William contributed to the portfolio backtest package.

Andrew Ellis read neuroscience and mathematics at the University in Zurich. He is working for Finance Online and is currently doing a 6 month Student Internship in the Econophysics group at ETH Zurich at the Institute for Theoretical Physics. Andrew is working on the Rmetrics documentation project and co-authored this ebook on portfolio optimization with Rmetrics.