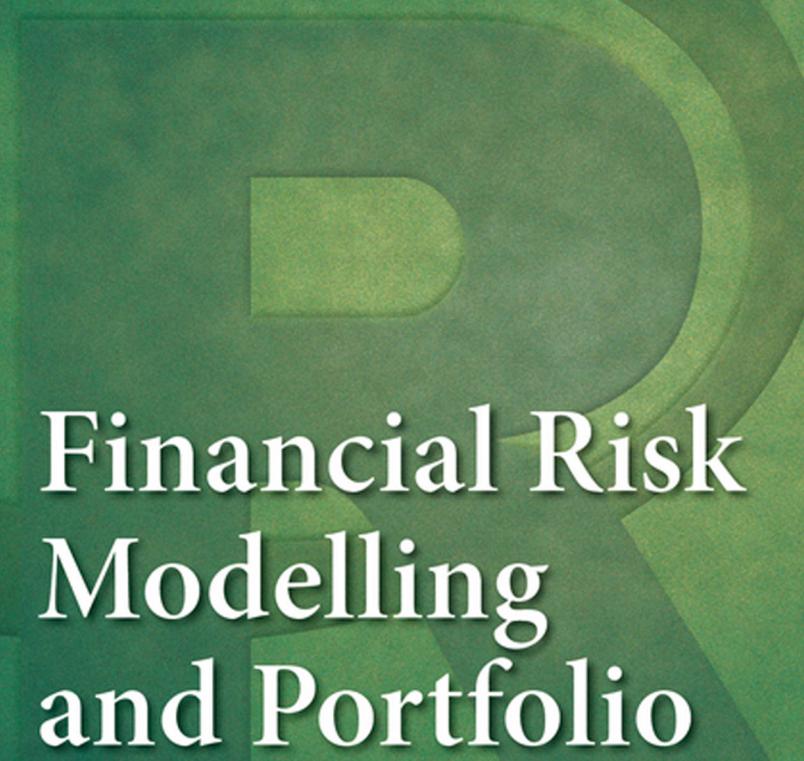


BERNHARD PFAFF



Financial Risk Modelling and Portfolio Optimization with R

SECOND EDITION



WILEY

Financial Risk Modelling and Portfolio Optimization with R

Financial Risk Modelling and Portfolio Optimization with R

Second Edition

Bernhard Pfaff

WILEY

This edition first published 2016
© 2016, John Wiley & Sons, Ltd
First Edition published in 2013

Registered office
John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom

For details of our global editorial offices, for customer services and for information about how to apply for permission to reuse the copyright material in this book please see our website at www.wiley.com.

The right of the author to be identified as the author of this work has been asserted in accordance with the Copyright, Designs and Patents Act 1988.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, except as permitted by the UK Copyright, Designs and Patents Act 1988, without the prior permission of the publisher.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Designations used by companies to distinguish their products are often claimed as trademarks. All brand names and product names used in this book are trade names, service marks, trademarks or registered trademarks of their respective owners. The publisher is not associated with any product or vendor mentioned in this book

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. It is sold on the understanding that the publisher is not engaged in rendering professional services and neither the publisher nor the author shall be liable for damages arising herefrom. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

Library of Congress Cataloging-in-Publication Data applied for

ISBN : 9781119119661

A catalogue record for this book is available from the British Library.

Cover Image: R logo © 2016 The R Foundation. Creative Commons Attribution-ShareAlike 4.0 International license (CC-BY-SA 4.0).

Set in 10/12pt, TimesLTStd by SPi Global, Chennai, India.

1 2016

Contents

Preface to the Second Edition	xi
Preface	xiii
Abbreviations	xv
About the Companion Website	xix
PART I MOTIVATION	1
1 Introduction	3
Reference	5
2 A brief course in R	6
2.1 Origin and development	6
2.2 Getting help	7
2.3 Working with R	10
2.4 Classes, methods, and functions	12
2.5 The accompanying package FRAPO	22
References	28
3 Financial market data	29
3.1 Stylized facts of financial market returns	29
3.1.1 Stylized facts for univariate series	29
3.1.2 Stylized facts for multivariate series	32
3.2 Implications for risk models	35
References	36
4 Measuring risks	37
4.1 Introduction	37
4.2 Synopsis of risk measures	37
4.3 Portfolio risk concepts	42
References	44
5 Modern portfolio theory	46
5.1 Introduction	46

5.2	Markowitz portfolios	47
5.3	Empirical mean-variance portfolios	50
	References	52
PART II RISK MODELLING		55
6	Suitable distributions for returns	57
6.1	Preliminaries	57
6.2	The generalized hyperbolic distribution	57
6.3	The generalized lambda distribution	60
6.4	Synopsis of R packages for GHD	66
6.4.1	The package fBasics	66
6.4.2	The package GeneralizedHyperbolic	67
6.4.3	The package ghyp	69
6.4.4	The package QRM	70
6.4.5	The package SkewHyperbolic	70
6.4.6	The package VarianceGamma	71
6.5	Synopsis of R packages for GLD	71
6.5.1	The package Davies	71
6.5.2	The package fBasics	72
6.5.3	The package gld	73
6.5.4	The package lmomco	73
6.6	Applications of the GHD to risk modelling	74
6.6.1	Fitting stock returns to the GHD	74
6.6.2	Risk assessment with the GHD	77
6.6.3	Stylized facts revisited	80
6.7	Applications of the GLD to risk modelling and data analysis	82
6.7.1	VaR for a single stock	82
6.7.2	Shape triangle for FTSE 100 constituents	84
	References	86
7	Extreme value theory	89
7.1	Preliminaries	89
7.2	Extreme value methods and models	90
7.2.1	The block maxima approach	90
7.2.2	The r th largest order models	91
7.2.3	The peaks-over-threshold approach	92
7.3	Synopsis of R packages	94
7.3.1	The package evd	94
7.3.2	The package evdbayes	95
7.3.3	The package evir	96
7.3.4	The packages extRemes and in2extRemes	98

7.3.5	The package fExtremes	99
7.3.6	The package ismev	101
7.3.7	The package QRM	101
7.3.8	The packages Renext and RenextGUI	102
7.4	Empirical applications of EVT	103
7.4.1	Section outline	103
7.4.2	Block maxima model for Siemens	103
7.4.3	r -block maxima for BMW	107
7.4.4	POT method for Boeing	110
	References	115
8	Modelling volatility	116
8.1	Preliminaries	116
8.2	The class of ARCH models	116
8.3	Synopsis of R packages	120
8.3.1	The package bayesGARCH	120
8.3.2	The package ccgarch	121
8.3.3	The package fGarch	122
8.3.4	The package GEVStableGarch	122
8.3.5	The package gogarch	123
8.3.6	The package lgarch	123
8.3.7	The packages rugarch and rmgarch	125
8.3.8	The package tseries	127
8.4	Empirical application of volatility models	128
	References	130
9	Modelling dependence	133
9.1	Overview	133
9.2	Correlation, dependence, and distributions	133
9.3	Copulae	136
9.3.1	Motivation	136
9.3.2	Correlations and dependence revisited	137
9.3.3	Classification of copulae	139
9.4	Synopsis of R packages	142
9.4.1	The package BLCOP	142
9.4.2	The package copula	144
9.4.3	The package fCopulae	146
9.4.4	The package gumbel	147
9.4.5	The package QRM	148
9.5	Empirical applications of copulae	148
9.5.1	GARCH–copula model	148
9.5.2	Mixed copula approaches	155
	References	157

PART III PORTFOLIO OPTIMIZATION APPROACHES 161

10 Robust portfolio optimization	163
10.1 Overview	163
10.2 Robust statistics	164
10.2.1 Motivation	164
10.2.2 Selected robust estimators	165
10.3 Robust optimization	168
10.3.1 Motivation	168
10.3.2 Uncertainty sets and problem formulation	168
10.4 Synopsis of R packages	174
10.4.1 The package covRobust	174
10.4.2 The package fPortfolio	174
10.4.3 The package MASS	175
10.4.4 The package robustbase	176
10.4.5 The package robust	176
10.4.6 The package rrcov	178
10.4.7 Packages for solving SOCPs	179
10.5 Empirical applications	180
10.5.1 Portfolio simulation: robust versus classical statistics	180
10.5.2 Portfolio back test: robust versus classical statistics	186
10.5.3 Portfolio back-test: robust optimization	190
References	195
11 Diversification reconsidered	198
11.1 Introduction	198
11.2 Most-diversified portfolio	199
11.3 Risk contribution constrained portfolios	201
11.4 Optimal tail-dependent portfolios	204
11.5 Synopsis of R packages	207
11.5.1 The package cccp	207
11.5.2 The packages DEoptim , DEoptimR , and RcppDE	207
11.5.3 The package FRAPO	210
11.5.4 The package PortfolioAnalytics	211
11.6 Empirical applications	212
11.6.1 Comparison of approaches	212
11.6.2 Optimal tail-dependent portfolio against benchmark	216
11.6.3 Limiting contributions to expected shortfall	221
References	226
12 Risk-optimal portfolios	228
12.1 Overview	228
12.2 Mean-VaR portfolios	229
12.3 Optimal CVaR portfolios	234
12.4 Optimal draw-down portfolios	238

12.5	Synopsis of R packages	241
12.5.1	The package fPortfolio	241
12.5.2	The package FRAPO	243
12.5.3	Packages for linear programming	245
12.5.4	The package PerformanceAnalytics	249
12.6	Empirical applications	251
12.6.1	Minimum-CVaR versus minimum-variance portfolios	251
12.6.2	Draw-down constrained portfolios	254
12.6.3	Back-test comparison for stock portfolio	260
12.6.4	Risk surface plots	265
	References	272
13	Tactical asset allocation	274
13.1	Overview	274
13.2	Survey of selected time series models	275
13.2.1	Univariate time series models	275
13.2.2	Multivariate time series models	281
13.3	The Black–Litterman approach	289
13.4	Copula opinion and entropy pooling	292
13.4.1	Introduction	292
13.4.2	The COP model	292
13.4.3	The EP model	293
13.5	Synopsis of R packages	295
13.5.1	The package BLCOP	295
13.5.2	The package dse	297
13.5.3	The package fArma	300
13.5.4	The package forecast	301
13.5.5	The package MSBVAR	302
13.5.6	The package PortfolioAnalytics	304
13.5.7	The packages urca and vars	304
13.6	Empirical applications	307
13.6.1	Black–Litterman portfolio optimization	307
13.6.2	Copula opinion pooling	313
13.6.3	Entropy pooling	318
13.6.4	Protection strategies	324
	References	334
14	Probabilistic utility	339
14.1	Overview	339
14.2	The concept of probabilistic utility	340
14.3	Markov chain Monte Carlo	342
14.3.1	Introduction	342
14.3.2	Monte Carlo approaches	343
14.3.3	Markov chains	347
14.3.4	Metropolis–Hastings algorithm	349

14.4	Synopsis of R packages	354
14.4.1	Packages for conducting MCMC	354
14.4.2	Packages for analyzing MCMC	358
14.5	Empirical application	362
14.5.1	Exemplary utility function	362
14.5.2	Probabilistic versus maximized expected utility	366
14.5.3	Simulation of asset allocations	369
	References	375
	Appendix A Package overview	378
A.1	Packages in alphabetical order	378
A.2	Packages ordered by topic	382
	References	386
	Appendix B Time series data	391
B.1	Date/time classes	391
B.2	The <code>ts</code> class in the base package <code>stats</code>	395
B.3	Irregularly spaced time series	395
B.4	The package <code>timeSeries</code>	397
B.5	The package <code>zoo</code>	399
B.6	The packages <code>tframe</code> and <code>xts</code>	401
	References	404
	Appendix C Back-testing and reporting of portfolio strategies	406
C.1	R packages for back-testing	406
C.2	R facilities for reporting	407
C.3	Interfacing with databases	407
	References	408
	Appendix D Technicalities	411
	Reference	411
	Index	413

Preface to the Second Edition

Roughly three years have passed since the first edition, during which episodes of higher risk environments in the financial market could be observed. Instances thereof are, for example, due to the abandoning of the Swiss franc currency ceiling with respect to the euro, the decrease in Chinese stock prices, and the Greek debt crisis; and these all happened just during the first three quarters of 2015. Hence, the need for a knowledge base of statistical techniques and portfolio optimization approaches for addressing financial market risk appropriately has not abated.

This revised and enlarged edition was also driven by a need to update certain **R** code listings to keep pace with the latest package releases. Furthermore, topics such as the concept of reference classes in **R** (see Section 2.4), risk surface plots (see Section 12.6.4), and the concept of probabilistic utility optimization (see Chapter 14) have been added, though the majority of the book and its chapters remain unchanged. That is, in each chapter certain methods and/or optimization techniques are introduced formally, followed by a synopsis of relevant **R** packages, and finally the techniques are elucidated by a number of examples.

Of course, the book's accompanying package **FRAPO** has also been refurbished (version $\geq 0.4.0$). Not only have the **R** code examples been updated, but the routines for portfolio optimization cast with a quadratic objective function now utilize the facilities of the **cccp** package. The package is made available on CRAN. Furthermore, the URL of the book's accompanying website remains unchanged and can be accessed from www.pfaffikus.de.

Bernhard Pfaff
Kronberg im Taunus

Preface

The project for this book commenced in mid-2010. At that time, financial markets were in distress and far from operating smoothly. The impact of the US real-estate crisis could still be felt and the sovereign debt crisis in some European countries was beginning to emerge. Major central banks implemented measures to avoid a collapse of the inter-bank market by providing liquidity. Given the massive financial book and real losses sustained by investors, it was also a time when quantitatively managed funds were in jeopardy and investors questioned the suitability of quantitative methods for protecting their wealth from the severe losses they had made in the past.

Two years later not much has changed, though the debate on whether quantitative techniques *per se* are limited has ceased. Hence, the modelling of financial risks and the adequate allocation of wealth is still as important as it always has been, and these topics have gained in importance, driven by experiences since the financial crisis started in the latter part of the previous decade.

The content of the book is aimed at these two topics by acquainting and familiarizing the reader with market risk models and portfolio optimization techniques that have been proposed in the literature. These more recently proposed methods are elucidated by code examples written in the **R** language, a freely available software environment for statistical computing.

This book certainly could not have been written without the public provision of such a superb piece of software as **R**, and the numerous package authors who have greatly enriched this software environment. I therefore wish to express my sincere appreciation and thanks to the **R** Core team members and all the contributors and maintainers of the packages cited and utilized in this book. By the same token, I would like to apologize to those authors whose packages I have not mentioned. This can only be ascribed to my ignorance of their existence. Second, I would like to thank John Wiley & Sons Ltd for the opportunity to write on this topic, in particular Ilaria Meliconi who initiated this book project in the first place and Heather Kay and Richard Davies for their careful editorial work. Special thanks belongs to Richard Leigh for his meticulous and mindful copy-editing. Needless to say, any errors and omissions are entirely my responsibility. Finally, I owe a debt of profound gratitude

to my beloved wife, Antonia, who while bearing the burden of many hours of solitude during the writing of this book remained a constant source of support.

This book includes an accompanying website. Please visit www.wiley.com/go/financial_risk.

Bernhard Pfaff
Kronberg im Taunus

Abbreviations

ACF	Autocorrelation function
ADF	Augmented Dickey–Fuller
AIC	Akaike information criterion
AMPL	A modelling language for mathematical programming
ANSI	American National Standards Institute
APARCH	Asymmetric power ARCH
API	Application programming interface
ARCH	Autoregressive conditional heteroskedastic
AvDD	Average draw-down
BFGS	Broyden–Fletcher–Goldfarb–Shanno algorithm
BL	Black–Litterman
BP	Break point
CDaR	Conditional draw-down at risk
CLI	Command line interface
CLT	Central limit theorem
CML	Capital market line
COM	Component object model
COP	Copula opinion pooling
CPPI	Constant proportion portfolio insurance
CRAN	Comprehensive R archive network
CVaR	Conditional value at risk
DBMS	Database management system
DE	Differential evolution
DGP	Data-generation process
DR	Diversification ratio
EDA	Exploratory data analysis
EGARCH	Exponential GARCH
EP	Entropy pooling
ERS	Elliott–Rothenberg–Stock
ES	Expected shortfall
EVT	Extreme value theory
FIML	Full-information maximum likelihood
GARCH	Generalized autoregressive conditional heteroskedastic
GEV	Generalized extreme values

GHD	Generalized hyperbolic distribution
GIG	Generalized inverse Gaussian
GLD	Generalized lambda distribution
GLPK	GNU Linear Programming Kit
GMPL	GNU MathProg modelling language
GMV	Global minimum variance
GOGARCH	Generalized orthogonal GARCH
GPD	Generalized Pareto distribution
GPL	GNU Public License
GUI	Graphical user interface
HYP	Hyperbolic
IDE	Integrated development environment
iid	independently, identically distributed
JDBC	Java database connectivity
LP	Linear program
MaxDD	Maximum draw-down
MCD	Minimum covariance determinant
MCMC	Markov chain Monte Carlo
MDA	Maximum domain of attraction
mES	Modified expected shortfall
MILP	Mixed integer linear program
ML	Maximum likelihood
MPS	Mathematical programming system
MRC	Marginal risk contributions
MRL	Mean residual life
MSR	Maximum Sharpe ratio
mVaR	Modified value at risk
MVE	Minimum volume ellipsoid
NIG	Normal inverse Gaussian
NN	Nearest neighbour
OBPI	Option-based portfolio insurance
ODBC	Open database connectivity
OGK	Orthogonalized Gnanadesikan–Kettenring
OLS	Ordinary least squares
OO	Object-oriented
PACF	Partial autocorrelation function
POT	Peaks over threshold
PWM	Probability-weighted moments
QMLE	Quasi-maximum-likelihood estimation
RDBMS	Relational database management system
RE	Relative efficiency
RPC	Remote procedure call
SDE	Stahel–Donoho estimator
SIG	Special interest group
SMEM	Structural multiple equation model
SPI	Swiss performance index

SVAR	Structural vector autoregressive model
SVEC	Structural vector error correction model
TAA	Tactical asset allocation
TDC	Tail dependence coefficient
VAR	Vector autoregressive model
VaR	Value at risk
VECM	Vector error correction model
XML	Extensible markup language

Unless otherwise stated, the following notation, symbols, and variables are used.

Notation

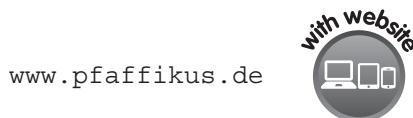
Lower case in bold: $\mathbf{y}, \boldsymbol{\alpha}$	Vectors
Upper case: Y, Σ	Matrices
Greek letters: α, β, γ	Scalars
Greek letters with $\hat{\cdot}$ or \sim or $\bar{\cdot}$	Sample values (estimates or estimators)

Symbols and variables

$ \cdot $	Absolute value of an expression
\sim	Distributed according to
\otimes	Kronecker product of two matrices
$\arg \max$	Maximum value of an argument
$\arg \min$	Minimum value of an argument
\perp	Complement of a matrix
\mathcal{C}, c	Copula
COR	Correlation(s) of an expression
COV	Covariance of an expression
\mathbb{D}	Draw-down
\det	Determinant of a matrix
\mathbb{E}	Expectation operator
\mathcal{I}	Information set
$I(d)$	Integrated of order d
\mathbb{L}	Lag operator
\mathfrak{L}	(Log-)likelihood function
μ	Expected value
\mathcal{N}	Normal distribution
$\boldsymbol{\omega}$	Weight vector
\mathbb{P}	Portfolio problem specification
\mathbb{P}	Probability expression
Σ	Variance-covariance matrix
σ	Standard deviation
σ^2	Variance
\mathcal{U}	Uncertainty set
VAR	Variance of an expression

About the Companion Website

Don't forget to visit the companion website for this book:



There you will find valuable material designed to enhance your learning, including:

- All **R** code examples
- The **FRAPO** **R** package.

Scan this QR code to visit the companion website.



Part I

MOTIVATION

1

Introduction

The period since the late 1990s has been marked by financial crises—the Asian crisis of 1997, the Russian debt crisis of 1998, the bursting of the dot-com bubble in 2000, the crises following the attack on the World Trade Center in 2001 and the invasion of Iraq in 2003, the sub-prime mortgage crisis of 2007, and European sovereign debt crisis since 2009 being the most prominent. All of these crises had a tremendous impact on the financial markets, in particular an upsurge in observed volatility and massive destruction of financial wealth. During most of these episodes the stability of the financial system was in jeopardy and the major central banks were more or less obliged to take countermeasures, as were the governments of the relevant countries. Of course, this is not to say that the time prior to the late 1990s was tranquil—in this context we may mention the European Currency Unit crisis in 1992–1993 and the crash on Wall Street in 1987, known as Black Monday. However, it is fair to say that the frequency of occurrence of crises has increased during the last 15 years.

Given this rise in the frequency of crises, the modelling and measurement of financial market risk have gained tremendously in importance and the focus of portfolio allocation has shifted from the μ side of the (μ, σ) coin to the σ side. Hence, it has become necessary to devise and employ methods and techniques that are better able to cope with the empirically observed extreme fluctuations in the financial markets. The hitherto fundamental assumption of independent and identically normally distributed financial market returns is no longer sacrosanct, having been challenged by statistical models and concepts that take the occurrence of extreme events more adequately into account than the Gaussian model assumption does. As will be shown in the following chapters, the more recently proposed methods of and approaches to wealth allocation are not of a revolutionary kind, but can be seen as an evolutionary development: a recombination and application of already existing statistical concepts to solve finance-related problems. Sixty years after Markowitz's seminal paper “Modern

Portfolio Theory,” the key (μ, σ) paradigm must still be considered as the anchor for portfolio optimization. What has been changed by the more recently advocated approaches, however, is how the riskiness of an asset is assessed and how portfolio diversification, that is, the dependencies between financial instruments, is measured, and the definition of the portfolio’s objective per se.

The purpose of this book is to acquaint the reader with some of these recently proposed approaches. Given the length of the book this synopsis must be selective, but the topics chosen are intended to cover a broad spectrum. In order to foster the reader’s understanding of these advances, all the concepts introduced are elucidated by practical examples. This is accomplished by means of the **R** language, a free statistical computing environment (see R Core Team 2016). Therefore, almost regardless of the reader’s computer facilities in terms of hardware and operating system, all the code examples can be replicated at the reader’s desk and s/he is encouraged not only to do so, but also to adapt the code examples to her/his own needs. This book is aimed at the quantitatively inclined reader with a background in finance, statistics, and mathematics at upper undergraduate/graduate level. The text can also be used as an accompanying source in a computer lab class, where the modelling of financial risks and/or portfolio optimization are of interest.

The book is divided into three parts. The chapters of this first part are primarily intended to provide an overview of the topics covered in later chapters and serve as motivation for applying techniques beyond those commonly encountered in assessing financial market risks and/or portfolio optimization. Chapter 2 provides a brief course in the **R** language and presents the **FRAPO** package that accompanies the book. For the reader completely unacquainted with **R**, this chapter cannot replace a more dedicated course of study of the language itself, but it is rather intended to provide a broad overview of **R** and how to obtain help. Because in the book’s examples quite a few **R** packages will be presented and utilized, a section on the existing classes and methods is included that will ease the reader’s comprehension of these two frameworks. In Chapter 3, stylized facts of univariate and multivariate financial market data are presented. The exposition of these empirical characteristics serves as motivation for the methods and models presented in Part II. Definitions used in the measurement of financial market risks at the single-asset and portfolio level are the topic of the Chapter 4. In the final chapter of Part I (Chapter 5), the Markowitz portfolio framework is described and empirical artifacts of the accordingly optimized portfolios are presented. The latter serve as motivation for the alternative portfolio optimization techniques presented in Part III.

In Part II, alternatives to the normal distribution assumption for modelling and measuring financial market risks are presented. This part commences with an exposition of the generalized hyperbolic and generalized lambda distributions for modelling returns of financial instruments. In Chapter 7, the extreme value theory is introduced as a means of modelling and capturing severe financial losses. Here, the block-maxima and peaks-over-threshold approaches are described and applied to stock losses. Both Chapters 6 and 7 have the unconditional modelling of financial losses in common. The conditional modelling and measurement of financial market risks is presented in the form of GARCH models—defined in the broader sense—in

Chapter 8. Part II concludes with a chapter on copulae as a means of modelling the dependencies between assets.

Part III commences by introducing robust portfolio optimization techniques as a remedy to the outlier sensitivity encountered by plain Markowitz optimization. In Chapter 10 it is shown how robust estimators for the first and second moments can be used as well as portfolio optimization methods that directly facilitate the inclusion of parameter uncertainty. In Chapter 11 the concept of portfolio diversification is reconsidered. In this chapter the portfolio concepts of the most diversified, equal risk contributed and minimum tail-dependent portfolios are described. In Chapter 12 the focus shifts to downside-related risk measures, such as the conditional value at risk and the draw-down of a portfolio. Chapter 13 is devoted to tactical asset allocation (TAA). Aside from the original Black–Litterman approach, the concept of copula opinion pooling and the construction of a wealth protection strategy are described. The latter is a synthesis between the topics presented in Part II and TAA-related portfolio optimization.

In Appendix A all the **R** packages cited and used are listed by name and topic. Due to alternative means of handling longitudinal data in **R**, a separate chapter (Appendix B) is dedicated to the presentation of the available classes and methods. Appendix C shows how **R** can be invoked and employed on a regular basis for producing back-tests, utilized for generating or updating reports, and/or embedded in an existing IT infrastructure for risk assessment/portfolio rebalancing. Because all of these topics are highly application-specific, only pointers to the **R** facilities are provided. A section on the technicalities concludes the book.

The chapters in Parts II and III adhere to a common structure. First, the methods and/or models are presented from a theoretical viewpoint only. The following section is reserved for the presentation of **R** packages, and the last section in each chapter contains applications of the concepts and methods previously presented. The **R** code examples provided are written at an intermediate language level and are intended to be digestible and easy to follow. Each code example could certainly be improved in terms of profiling and the accomplishment of certain computations, but at the risk of too cryptic a code design. It is left to the reader as an exercise to adapt and/or improve the examples to her/his own needs and preferences.

All in all, the aim of this book is to enable the reader to go beyond the ordinarily encountered standard tools and techniques and provide some guidance on when to choose among them. Each quantitative model certainly has its strengths and drawbacks and it is still a subjective matter whether the former outweigh the latter when it comes to employing the model in managing financial market risks and/or allocating wealth at hand. That said, it is better to have a larger set of tools available than to be forced to rely on a more restricted set of methods.

Reference

R Core Team 2016 *R: A Language and Environment for Statistical Computing* R Foundation for Statistical Computing Vienna, Austria.

2

A brief course in **R**

2.1 Origin and development

R is mainly a programming environment for conducting statistical computations and producing high-level graphics (see R Core Team 2016). These two areas of application should be interpreted widely, and indeed many tasks that one would not normally directly subsume under these topics can be accomplished with the **R** language. The website of the **R** project is <http://www.r-project.org>. The source code of the software is published as free software under the terms of the GNU General Public License (GPL; see <http://www.gnu.org/licenses/gpl.html>).

The language **R** is a dialect of the **S** language, which was developed by John Chambers and colleagues at Bell Labs in the mid-1970s.¹ At that time the software was implemented as FORTRAN libraries. A major advancement of the **S** language took place in 1988, following which the system was rewritten in C and functions for conducting statistical analysis were added. This was version 3 of the **S** language, referred to as **S3** (see Becker et al. 1988; Chambers and Hastie 1992). At that stage in the development of **S**, the **R** story commences (see Gentleman and Ihaka 1997). In August 1993 Ross Ihaka and Robert Gentleman, both affiliated with the University of Auckland, New Zealand, released a binary copy of **R** on Statlib, announcing it on the **s-news** mailing list. This first **R** binary was based on a Scheme interpreter with an **S**-like syntax (see Ihaka and Gentleman 1996). The name of **R** traces back to the initials of the first names of Ihaka and Gentleman, and is by coincidence a one-letter abbreviation to the language in the same manner as **S**. The announcement

¹ A detailed account of the history of the **S** language is accessible at <http://ect.bell-labs.com/s1/S/>.

by Ihaka and Gentleman did not go unnoticed and credit is due to Martin Mächler from ETH Zürich, who persistently advocated the release of R under GNU's GPL. This happened in June 1995. Interest in the language grew by word of mouth, and as a first means of communication and coordination a mailing list was established in March 1996 which was then replaced a year later by the electronic mail facilities that still exist today. The growing interest in the project led to the need for a powerful distribution channel for the software. This was accomplished by Kurt Hornik, at that time affiliated to TU Vienna. The master repository for the software (known as the "Comprehensive R Archive Network") is still located in Vienna, albeit now at the Wirtschaftsuniversität and with mirror servers spread all over the globe. In order to keep pace with changes requested by users and the fixing of bugs in a timely manner, a core group of R developers was set up in mid-1997. This established framework and infrastructure is probably the reason why R has since made such tremendous further progress. Users can contribute packages to solve specific problems or tasks and hence advances in statistical methods and/or computations can be swiftly disseminated. A detailed analysis and synopsis of the social organization and development of R is provided by Fox (2009). The next milestone in the history of the language was in 1998, when John Chambers introduced a more formal class and method framework for the S language (version 4), which was then adopted in R (see Chambers 1998, 2008). This evolution explains the coexistence of S3- and S4-like structures in the R language, and the user will meet them both in Section 2.4. More recent advancements are the inclusion of support for high-performance computations and a byte code compiler for R. From these humble beginnings, R has become the *lingua franca* for statistical computing.

2.2 Getting help

It is beyond the scope of this book to provide the reader with an introduction to the R language itself. Those who are completely new to R are referred to the manual *An Introduction to R*, available on the project's website under "Manuals." The purpose of this section is rather to provide the reader with some pointers on obtaining help and retrieving the relevant information for solving a particular problem.

As already indicated in the previous paragraph, the first resort for obtaining help is to read the R manuals. These manuals cover different aspects of R and the one mentioned above provides a useful introduction to R. The following R manuals are available, and their titles are self-explanatory:

- *An Introduction to R*
- *The R Language Definition*
- *Writing R Extensions*
- *R Data Import/Export*
- *R Installation and Administration*

- *R Internals*
- *The R Reference Index*

These manuals can either be accessed from the project’s website or invoked from an R session by typing

```
> help.start()
```

This function will load an HTML index file into the user’s web browser and local links to these manuals appear at the top. Note that a link to the “Frequently Asked Questions” is included, as well as a “Windows FAQ” if R has been installed under Microsoft Windows.

Incidentally, in addition to these R manuals, many complementary tutorials and related material can be accessed from <http://www.r-project.org/other-docs.html> and an annotated listing of more than 100 books on R is available at <http://www.r-project.org/doc/bib/R-books.html>. The reader is also pointed to the *The R Journal* (formerly *R News*), which is a biannual publication of user-contributed articles covering the latest developments in R.

Let us return to the subject of invoking help within R itself. As shown above, the function `help.start()` as invoked from the R prompt is one of the in-built help facilities that R offers. Other means of accessing help are:

```
> ## invoking the manual page of help() itself
> help()
> ## help on how to search in the help system
> help("help.search")
> ## help on search by partial matching
> help("apropos")
> ## Displaying available demo files
> demo()
> demo(scoping)
> ## Displaying available package vignettes
> ?vignette
> vignette()
> vignette("parallel")
```

The first command will invoke the help page for `help()` itself; its usage is described therein and pointers given to other help facilities. Among these other facilities are `help.search()`, `apropos()`, and `demo()`. If the latter is executed without arguments, the available demonstration files are displayed and `demo(scoping)` then runs the R code for familiarizing the user with the concept of lexical scoping in R, for instance. More advanced help is provided in vignettes associated with packages. The purpose of these documents is to show the user how the functions and facilities of a package can be employed. These documents can be opened in either a PDF reader or a web browser. In the last code line, the vignette contained in the **parallel** package is opened and the user is given a detailed description of how parallel computations can be carried out with R.

A limitation of these help facilities is that with these functions only local searches are conducted, so that the results returned depend on the **R** installation itself and the contributed packages installed. To conduct an online search the function **RSiteSearch()** is available which includes searches in the **R** mailing lists (mailing lists will be covered as another means of getting help in due course).

```
> ## Online search facilities
> ?RSiteSearch
> RSiteSearch("Portfolio")
> ## The CRAN package sos
> ## 1. Installation
> install.package("sos")
> ## 2. Loading
> library(sos)
> ## 3. Getting an overview of the content
> help(package = sos)
> ## 4. Opening the package's vignette
> vignette("sos")
> ## 5. Getting help on findFn
> ?findFn
> ## 6. Searching online for "Portfolio"
> findFn("Portfolio")
```

A very powerful tool for conducting online searches is the **sos** package (see Graves et al. 2013). If the reader has not installed this contributed package by now, s/he is recommended to do so. The cornerstone function is **findFn()**, which conducts online searches. In the example above, all relevant entries with respect to the keyword “Portfolio” are returned in a browser window and the rightmost column contains a description of the entries with a direct web link.

As shown above, **findFn()** can be used for answering questions of the form “Can this be achieved with **R**?” or “Has this already been implemented in **R**?” In this respect, given that at the time of writing more than 6300 packages are available on CRAN (not to speak of R-Forge), the “Task View” concept is beneficial.² CRAN packages that fit into a certain category, say “Finance,” are grouped together and each is briefly described by the maintainer(s) of the task view in question. Hence, the burden of searching the archive for a certain package with which a problem or task can be solved has been greatly reduced. Not only do the task views provide a good overview of what is available, but with the CRAN package **ctv** (see Zeileis 2005) the user can choose to install either the complete set of packages in a task view along with their dependencies or just those considered to be core packages. A listing of the task views can be found at <http://cran.r-project.org/web/views>.

```
> install.packages("ctv")
> library(ctv)
> install.views("Finance")
```

² To put matters into perspective: whence the first edition of this book was printed, the count of CRAN packages was only 3700.

As mentioned above, mailing lists are available, where users can post their problem/question to a wide audience. An overview of those available is provided at <http://www.r-project.org/mail.html>. Probably of most interest are R-help and R-SIG-Finance. The former is a high-traffic list dedicated to general questions about R, and the latter is focused on finance-related problems. In either case, before submitting to these lists the user should adhere to the posting guidelines, which can be found at <http://www.r-project.org/posting-guide.html>.

This section concludes with an overview of R conferences that have taken place in the past and will most likely come around again in the future.

- **useR!** This is an international R user conference and consists of keynote lectures and user-contributed presentations which are grouped together by topic. Finance-related sessions are ordinarily among these topics. The conference started in 2004 on a biannual schedule in Vienna, but now takes place every year at a different location. For more information, see the announcement at <http://www.r-project.org>.
- **R/Rmetrics Summer Workshop** This annual conference started in 2007 and is solely dedicated to finance-related subjects. The conference has recently been organized as a workshop with tutorial sessions in the morning and user presentations in the afternoon. The venue has previously been at Meielisalp, Lake Thune, Switzerland, but now takes place usually during the third week of June at different locations. More information is provided at <https://www.rmetrics.org>.
- **R in Finance** Akin to the R/Rmetrics Workshop, this conference is also solely dedicated to finance-related topics. It is a two-day event held annually during spring in Chicago at the University of Illinois. Optional pre-conference tutorials are given and the main conference consists of keynote speeches and user-contributed presentations (see <http://www.rinfinance.com> for more information).

2.3 Working with R

By default, R is provided with a command line interface (CLI). At first sight, this might be perceived as a limitation and as an antiquated software design. This perception might be intensified for novice users of R. However, the CLI is a very powerful tool that gives the user direct control over calculations. The dilemma is that probably only experienced users of R with a good command of the language might share this view on working with R, but how do you become a proficient R user in the first place? In order to solve this puzzle and ease the new user's way on this learning path, several graphical user interfaces (GUIs) and/or integrated development environments (IDEs) are available. Incidentally, it is possible to make this rather rich set of eye-catching GUIs and IDEs available because R is provided with a CLI in the first place, and all of them are factored around it.

In this section some of the platform-independent GUIs and IDEs are presented, acknowledging the fact that R is shipped with a GUI on the Microsoft Windows operating system only. The listing below is in alphabetical order and does not advocate the usage of one GUI/IDE framework over another, for good reasons. Deciding which system to use is a matter of personal preference and taste. Therefore, the reader is invited to inspect each of the GUIs and IDEs presented and then choose whichever is to her/his liking.

1. **Eclipse** Eclipse is a Java-based IDE and was first designed as an IDE for this language. More information about Eclipse is available on the project's website at <http://www.eclipse.org>. Since then many modules/plugins for other languages have been made available. The plugin for R is called **StatET** and is distributed via <http://www.walware.de/goto/statet>. Instructions for installing and configuring this module into Eclipse can be found on this website. Further online guidance is available elsewhere.
2. **Emacs/ESS** GNU Emacs is an extensible and customizable text editor, which at its core is an interpreter for the Emacs Lisp language. The project's website is <http://www.gnu.org/software/emacs>. Derived from this editor are the distributions XEmacs (<http://www.xemacs.org>), where the "X" indicates the X Windows system of Unix/Linux platforms, and Aquamacs (<http://aquamacs.org>) for Mac OS X only. Similar to Eclipse, the connection between this editor and R is established by means of a module, ESS, which stands for "Emacs Speaks Statistics". The project's website is <http://ess.r-project.org>, where this Lisp module can be downloaded and installation instructions are available. A strength of ESS is that other statistical packages such as S-PLUS, SAS, Stata, OpenBUGS, and JAGS are also supported. A dedicated mailing list for ESS is available in the "Getting help" section of the website cited above. Users working in Microsoft Windows might be interested in the prepackaged Emacs/ESS version made available by Vincent Goulet: <http://vgoulet.act.ulaval.ca/en/emacs>.
3. **JGR** In contrast to Eclipse and Emacs/ESS, JGR is a GUI rather than an IDE for R. Like Eclipse, it is based on Java, and "JGR" stands for "Java Gui for R." The project's website is <http://rforge.net/JGR>, where installation instructions can be found for Microsoft Windows, Mac OS X, and Linux platforms.
4. **RStudio** The latest addition to platform-independent GUIs/IDEs is RStudio. The software is hosted at <http://www.rstudio.com> and is distributed under the AGPLv3 license. A feature of this IDE is that it can either be installed as a desktop application or run on a server, where users can access RStudio via a web browser.
5. **Vim** Last, but not least, there is an R plugin for the Vim editor available. The Vim editor itself has been available for more than 20 years. The software is

hosted at <http://www.vim.org> and is based on the Unix vi editor. The R plugin is contained in the section “Scripts.”

Further information about R GUIs and IDEs can be found at <http://www.sciviews.org>, where a synopsis of available GUIs and IDEs is provided, some of which are platform dependent. This is quite an extensive listing, and software solutions that might not appear as a GUI, such as a web service, are also included. Furthermore, the reader can subscribe to a special interest group mailing list, RSIG-GUI, by following the instructions at <https://stat.ethz.ch/mailman/listinfo/r-sig-gui>.

2.4 Classes, methods, and functions

In this section, a concise introduction to the three flavors of class and method definitions in R is provided. The first class and method mechanism is referred to as S3, the second as S4, and the third as reference class (RC). Because the S4 and reference class mechanisms were included in R at a later stage in its development cycle (made available since versions 1.4.0 and 2.12.0, respectively), S3 classes and methods are sometimes also called old-style classes and methods. Detailed and elaborate accounts of these class and method schemes are provided in Becker et al. (1988) and Chambers and Hastie (1992) for S3, and Chambers (1998, 2008) for S4. Aside from these sources, the manual pages for S3 and S4 classes and methods can be inspected by `?Classes` and `?Methods`, respectively. Reference classes are documented in their associated help page `?ReferenceClasses`. The evolution of and distinction between these three object-oriented (OO) programming incarnations is described in Chambers (2014, 2016). The need to familiarize oneself with these class and method concepts is motivated by the fact that nowadays contributed R packages utilize either one or the other concept and in some packages a link between the class and method definitions of either kind is established. It should be noted that there are also R packages in which neither object-oriented programming concept has been employed at all in the sense of new class and/or method definitions, and such packages can be viewed as collections of functions only.

Before each of the three concepts is described, the reader will recall that everything in R is an object, that a class is the definition of an object, and that a method is a function by which a predefined calculation/manipulation on the object is conducted. Furthermore, there are generic functions for S3 and S4 objects which have the sole purpose of determining the class of the object and associating the appropriate method with it. If no specific method can be associated with an object, a default method will be called as a fallback. Generic functions for S3 and S4 objects can therefore be viewed as an umbrella under which all available class-specific methods are collected. The difference between S3 and S4 classes/methods lies in how a certain class is associated with an object and how a method is dispatched to it.

Because R is a dialect of the S language, S3 objects, classes, and methods have been available since the very beginning of R, almost 20 years ago. The assignment

of a class name and the method dispatch in this scheme are rather informal and hence very simple to implement. No formal requirements on the structure of an S3 class object are necessary, it is just a matter of adding a class attribute to an object. How swiftly such an attribute can be added to an object and/or changed is shown in the following in-line example:

```
> x <- 1:5
> x
[1] 1 2 3 4 5
> class(x)
[1] "integer"
> xchar <- x
> class(xchar) <- "character"
> class(xchar)
[1] "character"
> xchar
[1] "1" "2" "3" "4" "5"
```

Noteworthy in this example are the different shapes when `x` and `xchar` are printed. In the former case the object is printed as a numeric vector, and in the latter as a character vector indicated by quotation marks. This observation directly leads on to how method dispatching is accomplished within the S3 framework. Here, a simple naming convention is followed: `foo()` methods for objects of class `bar` are called `foo.bar()`. When such a function is not defined, the S3 method dispatch mechanism searches for a function `foo.default()`. The available methods for computing the mean of an object can be taken as an example:

```
> mean
function (x, ...)
UseMethod("mean")
<bytecode: 0x34956f0>
<environment: namespace:base>
> methods("mean")
[1] mean.Date           mean.default
[3] mean.difftime       mean.POSIXct
[5] mean.POSIXlt        mean.simple_sparse_array*
[7] mean.simple_triplet_matrix*
see "?methods" for accessing help and source code
```

Here, `mean()` is a generic function and the defined methods for `mean.bar()` are returned by the `methods()` function. As one can see from the output, apart from the default, methods for computing the mean of quite a few other classes of objects have been defined. By now, it should be apparent that S3 classes and methods can best be described as a naming convention, but fall short of what one assumes under the rubric of a mature object-oriented programming approach. Hence, in Chambers (2014) S3 should be viewed as an object-*based* functional programming scheme rather than an object-*oriented* one. A major pitfall of S3 classes and methods is that no validation process exists for assessing whether an object claimed to be of a certain class really

belongs to it or not. Sticking to our previous example, this is exhibited by trying to compute the means for `x` and `xchar`:

```
> mean(x)
[1] 3
> mean(xchar)
[1] NA
```

Given that `x` is of type `integer`, the default method for computing the mean is invoked. Because, there is no method `mean.character()`, the default method is also called for `xchar`. However, this default method tests whether the argument is either numeric or logical, and because this test fails for `xchar`, an `NA` value is returned and the associated warning is pretty indicative of why such a value has been returned. Just for exemplary purposes, one could define a `mean()` method for objects of class `character` in the sense that the average count of characters in the strings is returned, as shown next:

```
> mean.character <- function(x, ...) {
+   ans <- mean(nchar(x, ...))
+   ans
+ }
> mean(xchar)
[1] 1
```

However, its simplicity and quite powerful applicability are points in favor of the `S3` system.

`S4` classes offer a rigorous definition of an object by demanding that any valid object must be compliant with the specified class structure. Recall that for `S3` classes no formal testing of the correct contents of an object belonging to a certain class is required. The introduction of a more formal class mechanism is, however, associated with a cost: complexity. Now it is no longer sufficient to assign a certain object with a class attribute and define methods by adhering to the `foo.bar()` naming convention, but rather the handling of `S4` classes and methods is accomplished by a set of functions contained in the **methods** package, which is included in the base `R` installation. The most commonly encountered ones are:

- `setClass()` for defining a new `S4` class;
- `new()` for creating an object of a certain class;
- `setGeneric()` for defining a function as generic;
- `setMethods()` for defining a method for a certain class;
- `as()` and `setAs()` for coercing an object of one class to another class;
- `setValidity()` and `validObject()` for validating the appropriateness of an object belonging to a certain class;

- `showClass()`, `getClass()`, `showMethods()`, `findMethods()`, and `getMethods()` for displaying the definition/availability of S4 classes and methods;
- `slot()`, `getSlots()`, `@` for extracting elements from an object.

The following in-line examples show (i) how a class for portfolio weights can be defined, (ii) how these objects can be validated, and (iii) how methods can be created for objects of this kind. A more elaborate definition can certainly be designed, but the purpose of these code examples is to give the reader an impression of how S4 classes and methods are handled.

First, a class `PortWgt` is created:

```
> setClass("PortWgt",
+           representation(Weights = "numeric",
+                           Name = "character",
+                           Date = "character",
+                           Leveraged = "logical",
+                           LongOnly = "logical"))
> showClass("PortWgt")
Class "PortWgt" [in ".GlobalEnv"]

Slots:
Name:      Weights      Name      Date Leveraged  LongOnly
Class:  numeric character character  logical  logical
```

The portfolio weight class is defined in terms of a numeric vector `Weights` that will contain the portfolio weights, a character string `Name` for naming the portfolio associated with this weight vector, as well as a date reference, `Date`. In addition to these slots, the kind of portfolio is characterized: whether it is of the long-only kind and/or whether leverage is allowed or not. This is accomplished by including the two logical slots `Leveraged` and `LongOnly`, respectively.

At this stage, objects of class `PortWgt` could in principle already be created by utilizing `new()`:

```
> P1 <- new("PortWgt", Weights = rep(0.2, 5),
+           Name = "Equal Weighted",
+           Date = "2001-03-31",
+           LongOnly = TRUE,
+           Leveraged = FALSE)
```

However, a constructor function is ordinarily provided for creating these objects. Within the function body some measures for safeguarding the appropriateness of the user-provided input can already be taken into account, but this can also be implemented by means of a specific validation function.

```
> PortWgt <- function(Weights, Name, Date = NULL,
+                           LongOnly = TRUE, Leveraged = FALSE) {
+   Weights <- as.numeric(Weights)
```

```

+   Name <- as.character(Name)
+   if(is.null(Date)) Date <- as.character(Sys.Date())
+   ans <- new("PortWgt", Weights = Weights, Name = Name,
+             Date = Date, LongOnly = LongOnly,
+             Leveraged = Leveraged)
+   ans
+ }
> P2 <- PortWgt(Weights = rep(0.2, 5),
+                 Name = "Equal Weighted")

```

One of the strengths of S4 is its validation mechanism. In the above example, for instance, an object of class `PortWgt` could have been created for a long-only portfolio whereby some of the weights could have been negative, or a portfolio that should not be leveraged but whose absolute weight sum could be greater than unity. In order to check whether the arguments supplied in the constructor function do not violate the class specification from a content point of view, the following validation function is specified, mostly for elucidating this concept:

```

> validPortWgt <- function(object) {
+   if(object@LongOnly) {
+     if(any(object@Weights < 0)) {
+       return("\nNegative weights for long-only.")
+     }
+   }
+   if(!object@Leveraged) {
+     if(sum(abs(object@Weights)) > 1) {
+       return("\nAbsolute sum of weights greater than one.")
+     }
+   }
+   TRUE
+ }
> setValidity("PortWgt", validPortWgt)
Class "PortWgt" [in ".GlobalEnv"]

```

Slots:

Name:	Weights	Name	Date	Leveraged	LongOnly
Class:	numeric	character	character	logical	logical

This function returns `TRUE` if the supplied information is valid and in accordance with the class specification, or an informative message otherwise:

```

> PortWgt(Weights = rep(-0.2, 5),
+           Name = "Equal Weighted", LongOnly = TRUE)

```

```
Error in validObject(.Object) : invalid class
Negative weights for long-only.
```

In the above in-line statement the erroneous creation of a `PortWgt` object was tried for a long-only portfolio, but with negative weights. An error message is returned

and the user is alerted that at least one weight is negative and hence in conflict with the long-only characteristic. Similarly, in the following example the sum of weights is greater than 1 for a nonleveraged portfolio and hence the object is not created, but an informative error message as defined in `validPortWgt()` is returned:

```
> PortWgt(Weights = rep(0.3, 5),
+           Name = "Equal Weighted", Leveraged = FALSE)

Error in validObject(.Object) : invalid class
Absolute sum of weights greater than one.
```

So far, an S4 class for portfolio weights, `PortWgt`, has been defined and a constructor function `PortWgt()` has been created along with a function for validating the user-supplied arguments. The rest of this section shows how S4 methods can be defined. First, a `show()` method for nicely displaying the portfolio weights is created by means of `setMethod()`:

```
> setMethod("show", signature = "PortWgt",
+   function(object){
+     if(is.null(names(object@Weights))){
+       N <- length(object@Weights)
+       names(object@Weights) <- paste("Asset", 1:N)
+     }
+     cat(paste("Portfolio:", object@Name))
+     cat("\n")
+     cat(paste("Long-Only:", object@LongOnly))
+     cat("\n")
+     cat(paste("Leveraged:", object@Leveraged))
+     cat("\n")
+     cat("Weights:\n")
+     print(object@Weights)
+     cat("\n")
+   })
[1] "show"
```

If the supplied weight vector has been passed to the creation of the object without names, a generic character vector is created first. In the rest of the body of the `show()` method are calls to the function `cat()` by which the content of the `PortWgt` object will be displayed. The result will then look like this:

```
> P2
Portfolio: Equal Weighted
Long-Only: TRUE
Leveraged: FALSE
Weights:
Asset 1 Asset 2 Asset 3 Asset 4 Asset 5
  0.2      0.2      0.2      0.2      0.2
```

It might make sense to define a `summary()` method for producing descriptive statistics of the weight vector, which is accomplished by:

```
> setMethod("summary", "PortWgt", function(object, ...) {
+   summary(object@Weights, ...)
+ })
[1] "summary"
> summary(P2)
  Min. 1st Qu. Median     Mean 3rd Qu.     Max.
  0.2      0.2      0.2      0.2      0.2      0.2
```

In this method definition the already existing `summary()` method for objects of class `numeric` is directly applied to the slot `Weights`. Similarly, a `length()` method can be defined, which returns the count of assets as the length of this vector:

```
> setMethod("length", "PortWgt", function(x)
+   length(x@Weights)
+   )
[1] "length"
> length(P2)
[1] 5
```

The reader might wonder why, in the first instance, the function's definition is in terms of `function(object, ...)` and in the second `function(x)` only. The reason lies in the differing specifications of the “generic” function. This specification is displayed by invoking `getMethod("foo")` for method `foo`. Incidentally, a skeleton of a method definition for a particular class `bar` is created by `method.skeleton("foo", "bar")`.

The next in-line example shows the creation of a generic function `weights()` and an associated method for objects of class `PortWgt`. First, the generic function is defined without a default method and the method definition for `PortWgt` objects follows next:

```
> setGeneric("weights", function(object)
+   standardGeneric("weights")
+   )
[1] "weights"
> setMethod("weights", "PortWgt", function(object)
+   object@Weights
+   )
[1] "weights"
> weights(P2)
[1] 0.2 0.2 0.2 0.2 0.2
```

It would be handy if this method could also be used for assigning new values to the slot `Weights`. For this, one proceeds likewise by defining:

```
> setGeneric("weights<-", function(x, ..., value)
+   standardGeneric("weights<-")
+   )
```

```
[1] "weights<-
> setReplaceMethod("weights", "PortWgt",
+                     function(x, ..., value) {
+                         x@Weights <- value
+                         x
+                     })
[1] "weights<-
> weights(P2) <- rep(0.25, 4)
> P2
Portfolio: Equal Weighted
Long-Only: TRUE
Leveraged: FALSE
Weights:
Asset 1 Asset 2 Asset 3 Asset 4
  0.25    0.25    0.25    0.25
```

This time, because `weights()` is used as a replacement method, the function `setReplaceMethod()` has been invoked in its definition.

A final example shows how a `coerce()` method can be created by utilizing the `setAs()` function of the **methods** package:

```
> setAs(from = "PortWgt", to = "data.frame", function(from) {
+   anames <- names(from@Weights)
+   if(is.null(anames)){
+     N <- length(from)
+     anames <- paste("Asset", 1:N)
+   }
+   ans <- data.frame(from@Date, t(weights(from)))
+   colnames(ans) <- c("Date", anames)
+   ans
+ })
> as(P2, "data.frame")
      Date Asset 1 Asset 2 Asset 3 Asset 4
1 2015-10-09    0.25    0.25    0.25    0.25
```

In this call an object of class `PortWgt` is coerced into a `data.frame` object by combining the date stamp and the portfolio weight vector. In this definition the previously defined `length()` and `weights()` methods have been used. Note how this coercing method is invoked: the target class is included as a character string in the call to `as()`. This scheme is different than the S3-style coercing methods, such as `as.data.frame()`. Of course, this old-style method can also be defined for objects of class `PortWgt`, but this is left as an exercise for the reader.

The concept of a reference class is very different from the S3 and S4 paradigms. The latter are best described as flavors of a functional object-oriented programming implementation, whereas a reference class is an instance of an encapsulated object-oriented programming style. In other words, for S3 and S4 classes, methods belong to functions, but are members for reference class objects. Furthermore, RC objects are mutable such that the ordinary R-like behavior of “copy-on-modify,” that is, a copy of the local reference is enforced when a computation might alter a nonlocal reference, does not apply. As such, RC behaves more like object-oriented

programming paradigms as encountered in languages such as Python, Java, and/or C++. This behavior is accomplished by embedding RC objects in an environment slot of an S4 class.

In the following in-line statements the class `PortWgt` is reengineered as reference class `PortWgtRC`:

```
> PortWgtRC <- setRefClass("PortWgtRC",
+                             fields = list(
+                               Weights = "numeric",
+                               Name = "character",
+                               Date = "character",
+                               Leveraged = "logical",
+                               LongOnly = "logical"))
```

The call to `setRefClass()` is akin to the function `setClass()` introduced above for defining S4 classes. In a RC object a “field” is similar to a “slot” in S4 classes. New object instances of `PortWgtRC` are created by invoking the `$new()` method on RC instances:

```
> P3 <- PortWgtRC$new()
> P3
Reference class object of class "PortWgtRC"
Field "Weights":
numeric(0)
Field "Name":
character(0)
Field "Date":
character(0)
Field "Leveraged":
logical(0)
Field "LongOnly":
logical(0)
```

Because no `$initialize()` method has been defined, all fields are prefilled with their default (“empty”) values. Assignment of field values can either be accomplished in the call to the `$new()` method or in statements of the form `foo$fieldname`:

```
> P3$Weights <- rep(0.2, 5)
> P3$Name <- "Equal Weighted"
> P3$Date <- "2001-03-31"
> P3$LongOnly <- TRUE
> P3$Leveraged <- FALSE
```

The mutability of RC objects is illustrated by trying to “copy” an object and changing the content of one of its fields:

```
> P4RC <- P3
> P4RC$LongOnly <- FALSE
> P3$LongOnly
[1] FALSE
```

This behavior is probably not expected by an unwary user and in contrast to the S4 class implementation:

```
> P4$S4 <- P1
> P4$S4@LongOnly <- FALSE
> P1@LongOnly
[1] TRUE
```

All reference classes inherit from the class `envRefClass` which provides some useful methods, such as `$copy()`. Hence, if one would like to copy a reference class object, one should pursue the following route:

```
> P3$LongOnly <- TRUE
> PortWgtRC$methods()
[1] "callSuper"      "copy"          "export"
[4] "field"          "getClass"       "getRefClass"
[7] "import"          "initFields"     ".objectPackage"
[10] ".objectParent" "show"          "trace"
[13] "untrace"        "usingMethods"
> P4RC <- P3$copy()
> P4RC$LongOnly <- FALSE
> P3$LongOnly
[1] TRUE
```

Methods for a reference class can either be defined within the call to `setRefClass()` as a named list `methods` or through the inherited `$methods()` creator, as in the following example in which the `$show()` method is redefined:

```
> PortWgtRC$methods(show = function() {
+   if(is.null(names(Weights))) {
+     N <- length(Weights)
+     names(Weights) <- paste("Asset", 1:N)
+   }
+   cat(paste("Portfolio:", Name))
+   cat("\n")
+   cat(paste("Long-Only:", LongOnly))
+   cat("\n")
+   cat(paste("Leveraged:", Leveraged))
+   cat("\n")
+   cat("Weights:\n")
+   print(Weights)
+   cat("\n")
+ })
> P4 <- PortWgtRC$new(Weights = rep(0.2, 5),
+                       Name = "Equal Weighted",
+                       Date = "2001-03-31",
+                       LongOnly = TRUE,
+                       Leveraged = FALSE)
> P4
Portfolio: Equal Weighted
Long-Only: TRUE
Leveraged: FALSE
```

Weights:

Asset 1	Asset 2	Asset 3	Asset 4	Asset 5
0.2	0.2	0.2	0.2	0.2

Two points contained in the above method definition are noteworthy. First, in contrast to the functional OO style of S4 methods, members of a reference class object can be accessed directly in the function body of the method definition, for example, `Weights` instead of `objectWeights`. Second, members can only be altered by using the nonlocal assignment operator `<--`.

This has been a very concise introduction to the available OO styles in R. First-time R users are likely to be irritated by these three complementary class/method schemes, but might return to the literature and manual references provided for an in-depth discussion. As a novice R user progresses on the learning curve, s/he will probably appreciate the ability to choose from three distinct class/method incarnations: one informal functional OO style (S3), a more elaborate implementation (S4), and an encapsulated one (RC). Each of these facilities has its strength and weakness. For instance, the S4 class/method scheme can be utilized for the formulation of portfolio problems and finding an optimal solution; reference class objects might come in handy for back-testing portfolio strategies and thereby exploiting the mutability of the portfolio weights through time; and, last but not least, the S3 scheme can be fruitfully exploited for fast prototyping of R code and/or statistical model/distribution fitting.

2.5 The accompanying package **FRAPO**

A package accompanying this book, **FRAPO**, is available on CRAN. It can also be downloaded from the author's website at www.pfaffikus.de. Within the package, S4 classes and methods are employed. The purpose of this package is to provide:

- the examples in this book;
- the data sets that are used in the R code listings;
- classes, methods, and functions for portfolio optimization techniques and approaches that have not been covered in other contributed R packages;
- utility functions, such as the computation of returns, trend/momentum-based indicators, and descriptive portfolio statistics.

Listing 2.1 shows first how the package can swiftly be installed, where it is assumed that the user has access to the Internet. Additional packages are automatically installed as required. Next, the package is loaded into the workspace. An overview of the package in terms of the help topics covered is returned to the console by executing line 6. The description part of the package is followed by an index of the help topics. The first entry is "BookEx": "Utility functions for handling book examples." The help page for this entry is then opened by executing the command on line 8, where

R code 2.1 The package FRAPO.

```

## Installation of the book's accompanying package          1
install.packages("FRAPO")                                2
## Loading of the package                                3
library(FRAPO)                                         4
## Overview of the help topics                         5
help(package = FRAPO)                                    6
## Displaying help for the examples                   7
?BookEx                                                 8
## Showing the listing of R code examples            9
listEx()                                                 10
## Returning the content of this example            11
showEx("C3R1")                                         12
## Executing this script                           13
runEx("C3R1", echo = TRUE)                           14

```

the shortcut `?` for the `help()` function is used. The functions implemented to handle the R code examples covered in this book are listed in the usage section. That is, `listEx()` will return a character vector of the names of the R code examples, the function `showEx()` will display an R code example on the standard output (i.e., the console), or it can be directly executed by calling `runEx()`, a copy of an example can be created in the working directory by means of the function `saveEx()`, and can be edited by calling `editEx()`. The latter function comes in handy if the reader wishes to “play around” with the code or to modify it in some way (s/he is encouraged to do so). To reiterate, a copy of the R code example shipped with the package is first created in the working directory, and editing this file and saving the changes will not affect the original R code as listed in this book, unless R is started in the package subdirectory “BookEx,” which is not recommended for the above reason. In the remaining code lines, the handling of the R code examples is elucidated.

Given the scarcity of publicly available financial time series, some data sets are included in the package, as listed below. Some of these data sets can be considered as benchmark data.

- EESCBFX: ESCB FX Reference Rates
- EuroStoxx50: EURO STOXX 50
- FTSE100: FTSE 100
- INDTRACK1: Hang Seng Index and Constituents
- INDTRACK2: DAX 100 Index and Constituents
- INDTRACK3: FTSE 100 Index and Constituents
- INDTRACK4: Standard & Poor's 100 Index and Constituents

- INDTRACK5: Nikkei 225 Index and Constituents
- INDTRACK6: Standard & Poor's 500 Index and Constituents
- MIBTEL: Milano Indice Borsa Telematica
- MultiAsset: Multi Asset Index Data
- NASDAQ: NASDAQ
- SP500: Standard & Poor's 500
- StockIndex: Stock Index Data
- StockIndexAdj and StockIndexAdjD: Stock Index Data, month-end and daily.

The data sets EuroStoxx50, FTSE100, MIBTEL, NASDAQ, and SP500 are used in Cesarone et al. (2011) and can be retrieved from <http://host.uniroma3.it/docenti/cesarone/DataSets.htm>. These data sets comprise weekly observations of the index constituents starting on 3 March 2003 and ending on 24 March 2008. The authors adjusted the price data for dividends and removed stocks if two or more consecutive missing values were found. In the remaining cases the NA entries were replaced by interpolated values.

The series of data objects INDTRACK* are part of the OR library (see <http://people.brunel.ac.uk/~mastjjb/jeb/info.html>) and are used in Beasley et al. (2003) and Canakgoz and Beasley (2008). Similar to the data sets described above, these objects hold weekly observations of the index and its constituents. Stocks with missing values during the sample period have been discarded. The data was downloaded from DATASTREAM and made anonymous. The first column refers to the index data itself. The data license for these time series is included in the package as file BeasleyLicence and can also be found at <http://people.brunel.ac.uk/~mastjjb/jeb/orlib/legal.html>.

The ESCB FX data set consists of daily spot exchange rates for major currencies against the euro. The sample starts on 1 April 1999 and ends on 4 April 2012, giving a total of 3427 observations. The currencies are AUD, CAD, CHF, GBP, HKD, JPY, and USD.

The source of the remaining data sets MultiAsset, StockIndex, StockIndexAdj, and StockIndexAdjD is Yahoo! Finance (see <http://finance.yahoo.com>). The un/adjusted month-end/daily prices for the major stock and/or bond markets as well as gold are provided. The sample of MultiAsset starts in November 2004 and ends in November 2011. The sample period for data sets covering the major stock markets is larger, starting in July 1991 and ending in June 2011.

With respect to portfolio optimization, which is thoroughly covered in Chapter 5 and Part III of this book, the following approaches are available (in alphabetical order):

- `PAveDD()`: Portfolio optimization with average draw-down constraint;
- `PCDaR()`: Portfolio optimization with conditional draw-down at risk constraint;
- `PERC()`: equal risk contributed portfolios;
- `PGMV()`: global minimum variance portfolio;
- `PMD()`: most diversified portfolio;
- `PMTD()`: minimum tail-dependent portfolio;
- `PMaXDD()`: portfolio optimization with maximum draw-down constraint;
- `PMiNCDaR()`: portfolio optimization for minimum conditional draw-down at risk.

These are constructor functions for objects of the S4 class `PortSol` defined in **FRAPO**. In order to foster the reader's comprehension of S4 classes and methods as introduced in Section 2.4, the handling of these objects is elucidated in the following R code in-line statements. As an example, the solution of a global minimum-variance (GMV) portfolio is determined for the major stock indexes as contained in the `StockIndexAdj` data set:

```
> data(StockIndexAdj)
> R <- returnseries(StockIndexAdj, method = "discrete",
+                      trim = TRUE)
> P <- PGMV(R, optctrl = ctrl(trace = FALSE))
```

After the data set has been loaded into the workspace, the discrete returns of the price series are computed and assigned to the object R. The result of calling `PGMV` is then stored in the object P. The structure of an unknown object can be investigated with the function `str()`, but here we will query the class of P:

```
> class(P)
[1] "PortSol"
attr(,"package")
[1] "FRAPO"
```

The structure of this class can then be returned:

```
> showClass("PortSol")
Class "PortSol" [package "FRAPO"]

Slots:
Name:   weights      opt      type      call
Class:  numeric      list    character    call
```

```
Known Subclasses: "PortCdd", "PortAdd", "PortMdd"
```

In the output it is indicated that this class is defined in the package **FRAPO** and contains the slots `weights`, `opt`, `type`, and `call`. In the second line of the slots section, the classes of these entities are listed. Thus, the class of the portfolio weights is `numeric`, the outcome of the optimization is a `list` object, the type of the portfolio is `character`, and the call to the function by which the object has been created is of class `call`. The last line of the output indicates which other classes inherit from `PortSol`. The manual page of the `PortSol` class is displayed by `help("PortSol-class")`. The methods defined for `PortSol` objects are displayed by calling `showMethods()` with the class name is passed as argument `classes`.

```
> showMethods(classes = "PortSol", inherited = FALSE)
Function "complete":
<not an S4 generic function>

Function "coredata":
<not an S4 generic function>

Function "coredata<-":
<not an S4 generic function>

Function ".DollarNames":
<not an S4 generic function>

Function "formals<-":
<not an S4 generic function>

Function "functions":
<not an S4 generic function>

Function "prompt":
<not an S4 generic function>

Function "scale":
<not an S4 generic function>
Function: show (package methods)
object="PortSol"

Function: Solution (package FRAPO)
object="PortSol"

Function: update (package stats)
object="PortSol"

Function: Weights (package FRAPO)
object="PortSol"
```

This says that a `show()` method is available, which is executed automatically when the object is returned. The generic function for this method is available in the **methods** package. The `Solution()` method for retrieving the outcome of the optimizer is defined in the package **FRAPO** itself, as is the method `Weights()` for extracting the optimal weight vector.

```

> P
Optimal weights for portfolio of type:
Global Minimum Variance

  SP500      N225  FTSE100      CAC40      GDAX      HSI
36.6719 13.9499 49.3782  0.0000  0.0000  0.0000

> Weights(P)
      SP500          N225          FTSE100          CAC40
3.667185e+01 1.394991e+01 4.937824e+01 -5.526138e-07
      GDAX          HSI
-6.809098e-07 -7.415764e-07

```

An `update()` method is available, too. This comes in handy and saves typing if one wishes to update an existing object by altering the values passed to the generating function's arguments. For instance, portfolio back-testing whereby the fund is rebalanced regularly can be carried out by utilizing the `update()` method and incrementally increasing the underlying data sample.

In order to see the actual source code of a method definition, rather than entering the name of the method one can use `selectMethod()`:

```

> showMethods("Weights", inherited = FALSE)
Function: Weights (package FRAPO)
object="PortSol"
> selectMethod(f = "Weights", signature = "PortSol")
Method Definition:

function (object)
{
  ans <- slot(object, "weights")
  return(ans)
}
<environment: namespace:FRAPO>

Signatures:
  object
target "PortSol"
defined "PortSol"

```

As can be seen from the function's body, alternative means for retrieving the slot `weights` are:

```

> Weights(P)
      SP500          N225          FTSE100          CAC40
3.667185e+01 1.394991e+01 4.937824e+01 -5.526138e-07
      GDAX          HSI
-6.809098e-07 -7.415764e-07
> slot(P, "weights")
      SP500          N225          FTSE100          CAC40
3.667185e+01 1.394991e+01 4.937824e+01 -5.526138e-07
      GDAX          HSI
-6.809098e-07 -7.415764e-07
> P@weights
      SP500          N225          FTSE100          CAC40

```

```
3.667185e+01 1.394991e+01 4.937824e+01 -5.526138e-07
GDAX HSI
-6.809098e-07 -7.415764e-07
```

With these last pointers and examples, the reader should hopefully have some insight into working with the formal class and method scheme in R.

References

- Beasley J., Meade N., and Chang T. 2003 An evolutionary heuristic for the index tracking problem. *European Journal of Operational Research* **148**, 621–643.
- Becker R., Chambers J., and Wilks A. 1988 *The New S Language*. Chapman & Hall, London.
- Canakgoz N. and Beasley J. 2008 Mixed-integer programming approaches for index tracking and enhanced indexation. *European Journal of Operational Research* **196**, 384–399.
- Cesarone F., Scozzari A., and Tardella F. 2011 Portfolio selection problems in practice: a comparison between linear and quadratic optimization models. Quantitative Finance Papers 1105.3594, arXiv.org.
- Chambers J. 1998 *Programming with Data*. Springer-Verlag, New York.
- Chambers J. 2008 *Software for Data Analysis: Programming with R*. Springer-Verlag, New York.
- Chambers J. 2014 Object-oriented programming, functional programming in R. *Statistical Science* **29**(2), 167–180.
- Chambers J. 2016 *Extending R*. CRC Press, Taylor & Francis Group, Boca Raton.
- Chambers J. and Hastie T. 1992 *Statistical Models in S*. Chapman & Hall, London.
- Fox J. 2009 Aspects of the social organization and trajectory of the R project. *The R Journal* **1**(2), 5–13.
- Gentleman R. and Ihaka R. 1997 The R language In *Proceedings of the 28th Symposium on the Interface* (ed. Billard L. and Fisher N.) The Interface Foundation of North America.
- Graves S., Dorai-Raj S., and Francois R. 2013 *sos: Search contributed R packages, sort by package*. R package version 1.3-8.
- Ihaka R. and Gentleman R. 1996 R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics* **5**, 299–314.
- R Core Team 2016 *R: A Language and Environment for Statistical Computing* R Foundation for Statistical Computing Vienna, Austria.
- Zeileis A. 2005 CRAN task views. *R News* **5**(1), 39–40.

3

Financial market data

3.1 Stylized facts of financial market returns

3.1.1 Stylized facts for univariate series

Before we turn to the topic of modelling financial market risks, it is worthwhile to consider and review typical characteristics of financial market data. These are summarized in the literature as “stylized facts” (see Campbell et al. 1997; McNeil et al. 2005). These observed properties have important implications for assessing whether the risk model chosen is appropriate or not. Put differently, a risk model that does not capture the time series characteristics of financial market data adequately will also not be useful for deriving risk measures. For observed financial market data, the following stylized facts can be stated:

- Time series data of returns, in particular daily return series, are in general not independent and identically distributed (iid). This fact is not jeopardized by low absolute values of the first-order autocorrelation coefficient.
- The volatility of return processes is not constant with respect to time.
- The absolute or squared returns are highly autocorrelated.
- The distribution of financial market returns is leptokurtic. The occurrence of extreme events is more likely compared to the normal distribution.
- Extreme returns are observed closely in time (volatility clustering).
- The empirical distribution of returns is skewed to the left; negative returns are more likely to occur than positive ones.

R code 3.1 Stylized facts on the returns for Siemens.

```

library(fBasics)                                     1
library(evir)                                       2
data(siemens)                                      3
SieDates <- as.character(format(as.POSIXct(attr(siemens,      4
                                              "times")), "%Y-%m-%d"))
SieRet <- timeSeries(siemens * 100, charvec = SieDates)  5
colnames(SieRet) <- "SieRet"                         6
## Stylised Facts I                                7
par(mfrow = c(2, 2))                                8
seriesPlot(SieRet, title = FALSE, main = "Daily Returns of  9
           Siemens", col = "blue")                   10
boxPlot(SieRet, title = FALSE, main = "Box plot of Returns", 11
        col = "blue", cex = 0.5, pch = 19)           12
acf(SieRet, main = "ACF of Returns", lag.max = 20, ylab = "", 13
    xlab = "", col = "blue", ci.col = "red")        14
pacf(SieRet, main = "PACF of Returns", lag.max = 20,      15
      ylab = "", xlab = "", col = "blue", ci.col = "red") 16
## Stylised Facts II                                17
SieRetAbs <- abs(SieRet)                            18
SieRet100 <- tail(sort(abs(series(SieRet))), 100)[1] 19
idx <- which(series(SieRetAbs) > SieRet100, arr.ind = TRUE) 20
SieRetAbs100 <- timeSeries(rep(0, length(SieRet)),      21
                           charvec = time(SieRet))        22
SieRetAbs100[idx, 1] <- SieRetAbs[idx]               23
acf(SieRetAbs, main = "ACF of Absolute Returns", lag.max = 20, 24
    ylab = "", xlab = "", col = "blue", ci.col = "red") 25
pacf(SieRetAbs, main = "PACF of Absolute Returns",      26
      lag.max = 20, ylab = "", xlab = "", col = "blue", 27
      ci.col = "red")                                28
qqnormPlot(SieRet, main = "QQ-Plot of Returns", title = FALSE, 29
           col = "blue", cex = 0.5, pch = 19)           30
plot(SieRetAbs100, type = "h", main = "Volatility Clustering", 31
      ylab = "", xlab = "", col = "blue")             32
                                              33

```

As an example, we will now check whether these stylized facts are applicable to the returns of Siemens stock. The data set of daily returns is contained in the package **evir** (see Pfaff and McNeil 2012). This series starts on 2 January 1973 and ends on 23 July 1996, and comprises 6146 observations.

In Listing 3.1 the necessary packages **fBasics** (see Würtz et al. 2014) and **evir** are loaded first. Functions contained in the former package will be utilized to produce some of the graphs. Next, the **siemens** data set is loaded and converted into an object of class **timeSeries** with the function of the same name. The time series plot of the percentage returns, a box plot thereof, and the autocorrelation and partial autocorrelation are produced next and exhibited in Figure 3.1. As can be deduced from the time series plot, volatility clustering does exist. This is more pronounced

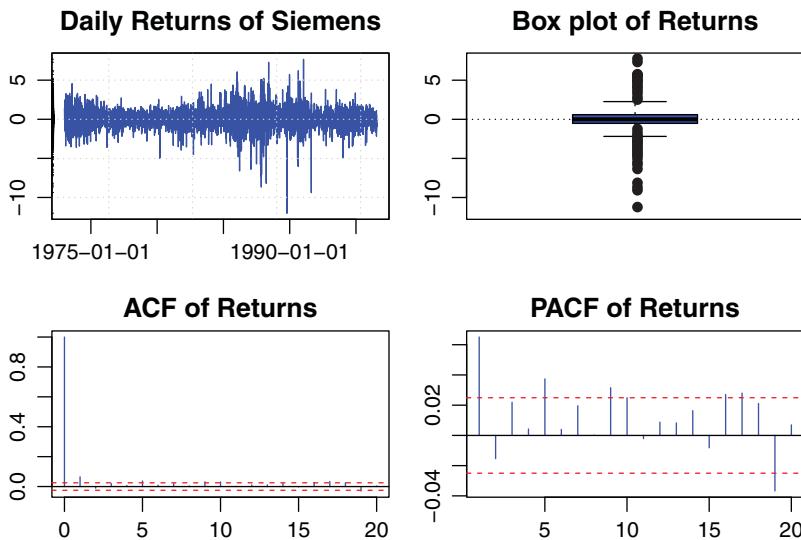


Figure 3.1 Stylized facts for Siemens, part one.

in the second half of the sample period. Furthermore, by mere eyeball econometrics the returns are skewed to the left and heavy tails are evident, as can be seen from the box plot (upper right panel). The largest loss occurred on 16 October 1989, at -12.01% . The highest return of 7.67% occurred on 17 January 1991. The skewness is -0.52 and the excess kurtosis 7.74 , clearly indicating heavy tails. The autocorrelation function (ACF) and partial autocorrelation function (PACF) hint at a slight first-order autocorrelation. Incidentally, the series shows some systematic variation on the weekly frequency; though significant, it is much less pronounced than the daily autocorrelation.

Figure 3.2 further investigates whether the stylized facts about financial market returns hold in the case of Siemens. In the upper panels of this figure, the autocorrelations and partial autocorrelations of the absolute returns are plotted. Clearly, these are significantly different from zero and taper off only slowly. In the lower left panel a quantile–quantile (QQ) plot compared to the normal distribution is produced. The negative skew and heavy tails are mirrored from their quantitative values in this graph. Finally, in Listing 3.1 the 100 largest absolute returns have been retrieved from the object `SieRet`. These values are shown in the lower right-hand panel. This time series plot vindicates more clearly what could already be deduced from the upper left-hand panel in Figure 3.1: first, the existence of volatility clustering; and second, that the returns become more volatile in the second half of the sample period.

Although these stylized facts have only been exemplified by the stock returns of Siemens, they not only hold for basically all stock returns, but also are applicable to other asset classes, such as bonds, currencies, and commodity futures.

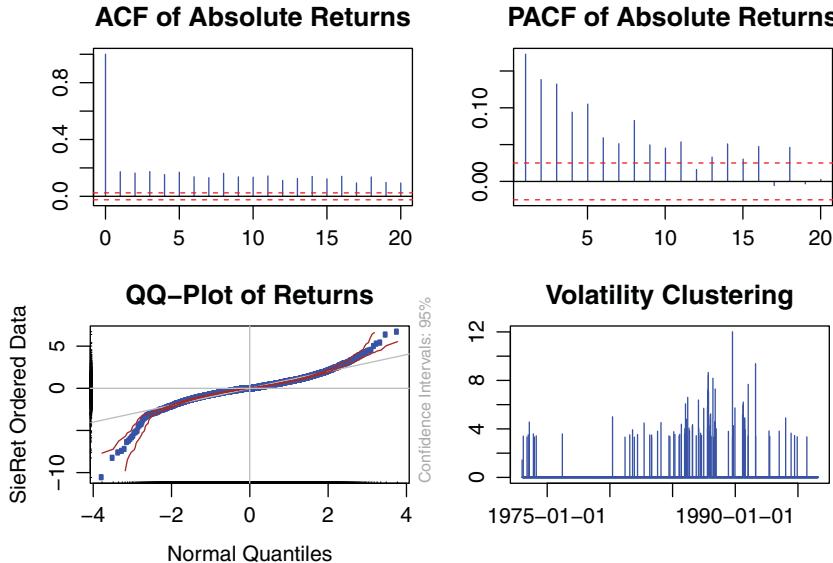


Figure 3.2 Stylized facts for Siemens, part two.

3.1.2 Stylized facts for multivariate series

The previous subsection presented the stylized facts for univariate financial market returns. From the portfolio point of view the characteristics of multivariate return series are of interest. Here we will focus on these stylized facts:

- The absolute value of cross-correlations between return series is less pronounced and contemporaneous correlations are in general the strongest.
- In contrast, the absolute or squared returns do show high cross-correlations. This empirical finding is similar to the univariate case.
- Contemporaneous correlations are not constant over time.
- Extreme observations in one return series are often accompanied by extremes in the other return series.

These stylized facts are elucidated in Listing 3.2. Here, daily European stock market data for France, Germany, and the United Kingdom are employed. This data set is first loaded into R and converted from an object with class attribute `mts` into a `zoo` object. The package `zoo` (see Zeileis and Grothendieck 2005) is an alternative to the package `timeSeries` which was utilized in the previous listing. Next, a time series chart of the index data is produced.

The co-movement between these three European equity markets is quite apparent from Figure 3.3. The co-movement with respect to the volatility between the three continuous return series (not shown here) is similar.

R code 3.2 Stylized facts on the European equity market.

```

library (zoo)                                     1
data (EuStockMarkets)                         2
## Time Series plot of Levels                 3
EuStockLevel <- as.zoo(EuStockMarkets) [, c("DAX", "CAC", 4
                                         "FTSE")]
                                                 5
plot(EuStockLevel, xlab = "", main = "")       6
## Percentage returns                         7
EuStockRet <- diff(log(EuStockLevel)) * 100   8
plot(EuStockRet, xlab = "", main = "")          9
## Cross correlations                         10
layout(matrix(1:6, nrow = 3, ncol = 2, byrow = TRUE)) 11
ccf(EuStockRet[, 1], EuStockRet[, 2], ylab = "", xlab = "", 12
     lag.max = 20, main = "Returns DAX vs CAC") 13
ccf(abs(EuStockRet)[, 1], abs(EuStockRet)[, 2], ylab = "", 14
     xlab = "", lag.max = 20, main = "Absolute returns DAX vs 15
     CAC")
ccf(EuStockRet[, 1], EuStockRet[, 3], ylab = "", xlab = "", 16
     lag.max = 20, main = "Returns DAX vs FTSE") 17
ccf(abs(EuStockRet)[, 1], abs(EuStockRet)[, 3], ylab = "", 18
     xlab = "", lag.max = 20, main = "Absolute returns DAX vs 19
     FTSE")
ccf(EuStockRet[, 2], EuStockRet[, 3], ylab = "", xlab = "", 20
     lag.max = 20, main = "Returns CAC vs FTSE") 21
ccf(abs(EuStockRet)[, 2], abs(EuStockRet)[, 3], ylab = "", 22
     xlab = "", lag.max = 20, main = "Absolute returns CAC vs 23
     FTSE")
## Rolling correlations                      24
rollc <- function(x){                      25
  dim <- ncol(x)                           26
  rcor <- cor(x)[lower.tri(diag(dim), diag = FALSE)] 27
  return(rcor)                            28
}
rcor <- rollapply(EuStockRet, width = 250, rollc, 29
                  align = "right", by.column = FALSE) 30
colnames(rcor) <- c("DAX & CAC", "DAX & FTSE", "CAC & FTSE") 31
plot(rcor, main = "", xlab = "")            32
                                                 33
                                                 34
                                                 35
                                                 36

```

This artifact is mirrored when one views the cross-correlations of the absolute returns in Figure 3.4 (graphs shown to the right). As pointed out earlier, the returns themselves are barely cross-correlated between markets and taper off fairly quickly (graphs shown to the left). However, significant cross-correlations are evident for their absolute counterpart. This artifact is most pronounced for the German vis-à-vis the British stock market.

Finally, the correlations based upon a moving window of 250 observations are exhibited in Figure 3.5. In order to calculate these rolling correlations, an auxiliary

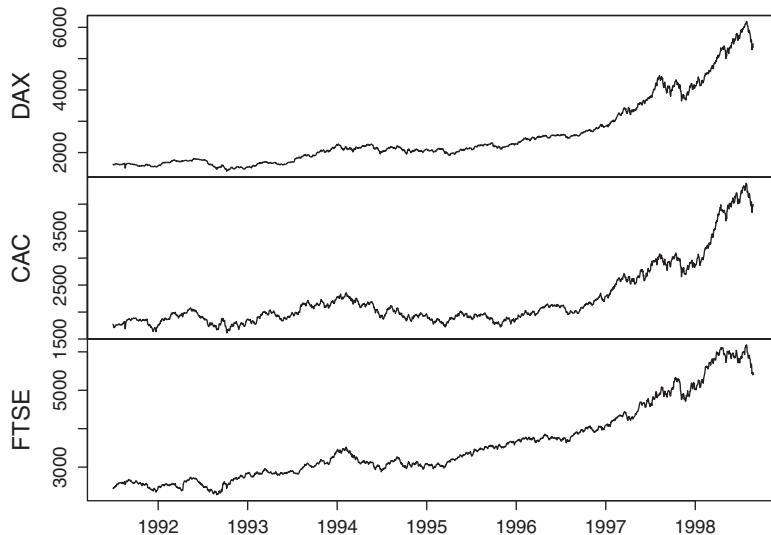


Figure 3.3 European stock market data.

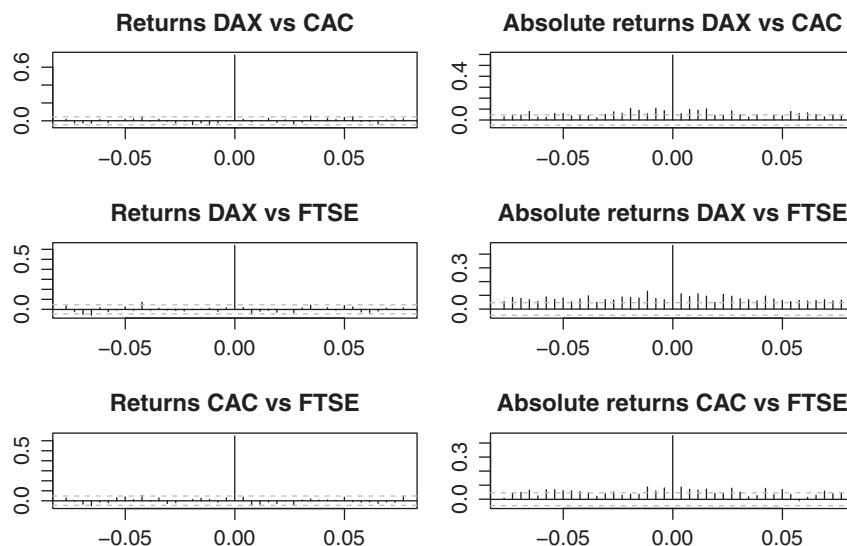


Figure 3.4 Cross-correlations between European stock market returns.

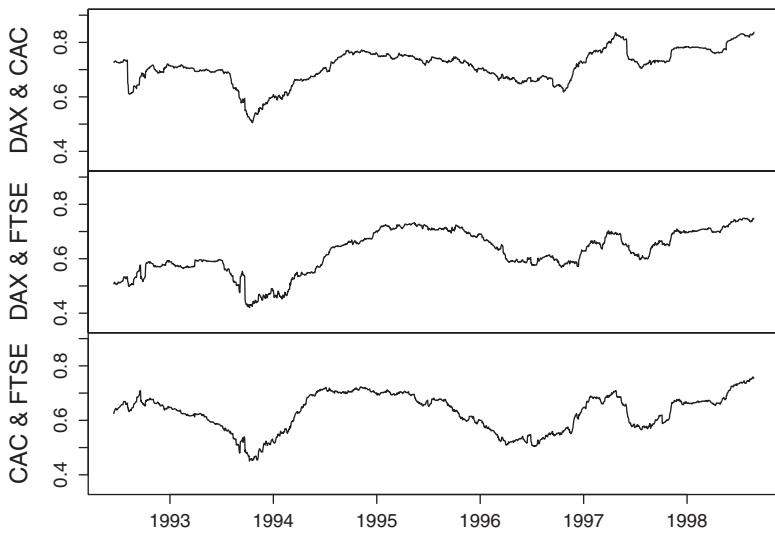


Figure 3.5 Rolling correlations of European stock markets.

function `rollc()` is defined first. Within this function the correlation matrix is calculated and, due to its symmetry, the upper triangular part thereof is extracted. The computation of the rolling correlations is then performed with the function `rollapply()` contained in `zoo`. The plots of these correlations are fairly similar for all pairs portrayed, as are their ranges. For the DAX/CAC the correlation ranges between 0.505 and 0.838, for the DAX/FTSE the values are between 0.42 and 0.749, and lastly for the CAC/FTSE the correlation is in the interval 0.451 to 0.76.

3.2 Implications for risk models

The stylized facts for univariate and multivariate financial returns have been given in the previous section. With respect to risk models and the risk measures derived from them the following normative requirements can be deduced so far:

- Risk models which assume iid processes for the losses are not adequate during all market episodes.
- Risk models that are based on the normal distribution will fall short in predicting the frequency of extreme events (losses).
- Risk models should be able to encompass and address the different volatility regimes. This means that the derived risk measures should be adaptive to changing environments of low and high volatility.
- In the portfolio context, the model employed should be flexible enough to allow for changing dependencies between the assets; in particular, the co-movement of losses should be taken care of.

Part II of this book will focus on these issues. Statistical methods and techniques will be presented which either in whole or in part address these stylized facts.

References

- Campbell J., Lo A., and MacKinlay A. 1997 *The Econometrics of Financial Markets* 2nd printing with corrections edn. Princeton University Press, Princeton, NJ.
- McNeil A., Frey R., and Embrechts P. 2005 *Quantitative Risk Management: Concepts, Techniques and Tools*. Princeton University Press, Princeton, NJ.
- Pfaff B. and McNeil A. 2012 *evir: Extreme Values in R*. R package version 1.7-3.
- Würtz D., Setz T., and Chalabi Y. 2014 *fBasics: Rmetrics – Markets and Basic Statistics*. R package version 3011.87.
- Zeileis A. and Grothendieck G. 2005 *zoo: S3 infrastructure for regular and irregular time series*. *Journal of Statistical Software* **14**(6), 1–27.

4

Measuring risks

4.1 Introduction

Toward the end of the first decade of this century, the focus of investors shifted, not least because of the financial crisis, to the need to adequately measure risks. But it is also worth being able to capture potential losses correctly during market episodes that are more tranquil. If an investor were too conservative with respect to risk during such phases, s/he would jeopardize potential investment opportunities. Furthermore, there are regulatory reasons and economic arguments which necessitate a proper risk assessment. With respect to the former the stricter requirements imposed by the Basel Accords are worthy of mention, and for the latter the normative presumption of an efficient resource allocation between risky assets. If the riskiness of a financial instrument is not captured correctly, its price would in general be misleading and would therefore cause an inefficient allocation of funds. Finally, a proper assessment of market risks by its participants helps to ensure smooth functioning of the financial system. For example, if market participants have to revise their risk estimates on a larger scale, as was the case during the sub-prime mortgage crisis, and hence want or have to re-balance their funds, a stampede-like reaction will quickly dry up market liquidity and aggravate the potential losses, due to the lack of counter-parties.

4.2 Synopsis of risk measures

The risk measures introduced in this section are based upon a probability model for the potential losses an investor would face. The investor's wealth $W(t)$ is viewed as a random variable, and the realization thereof at the time point t is assumed to be

known. The value of wealth at this specific point in time is dependent on the vector of risk factors z_t , which exert an influence on the investor's wealth position:

$$W_t = f(t, z_t). \quad (4.1)$$

The prices of financial instruments, exchange rates, and/or interest rates can be modelled as risk factors, for instance. It is assumed that these are known at the time t . In contrast, the future value of wealth after a time span is unknown. This could be the wealth position in 1 or 10 days' time. The loss that results after a time is denoted by $L_{t,t+\Delta}$, and this is just the difference in wealth positions at $t + \Delta$ and t . It is conventional that losses are expressed as positive numbers:

$$L_{t,t+\Delta} := -(W_{t+\Delta} - W_t). \quad (4.2)$$

Substituting (4.1) in (4.2), it is immediately evident that the losses are determined by the changes in the risk factors:

$$L_{t+\Delta} = -(f(t + \Delta, z_t + x_{t+\Delta}) - f(t, z_t)), \quad (4.3)$$

where $x_t = z_t - z_{t-\Delta}$.

Because wealth $W(t)$ is a random variable, the loss function, as the difference between two wealth positions in time, is also a random variable. As such it has a probability distribution, which will henceforth be termed the *loss distribution*. This distribution can be made dependent upon the information available at the time point t . In this case, a conditional distribution would result. If, on the other hand, the distribution is time-independent, the distribution is unconditional with respect to time.

As a first step, only the modelling of market risk for a single financial instrument is considered. Therefore, the price of the asset is the risk factor and the loss depends on the time span and the price change within this period. We will further confine the analysis to the unconditional distribution of losses and the risk measures that can be deduced from it.

In practice, the most commonly encountered risk measure is the *value at risk* (VaR). This concept was introduced by JP Morgan in the first version of their publication *RiskMetrics* and was then covered thoroughly in the academic literature (see RiskMetrics Group 1994).¹ Textbook expositions are included, for example, in Jorion (2001) and McNeil et al. (2005). For a given confidence level $\alpha \in (0, 1)$, the VaR is defined as the smallest number l such that the probability of a loss L is not higher than $(1 - \alpha)$ for losses greater than l . This value corresponds to a quantile of the loss distribution and can be formally expressed as

$$\text{VaR}_\alpha = \inf \{l \in \mathbb{R} : P(L > l) \leq 1 - \alpha\} = \inf \{l \in \mathbb{R} : F_L(l) \geq \alpha\}, \quad (4.4)$$

¹ Incidentally, as stressed in Alexander and Baptista (2002), this sort of risk measure was advocated as early as 1963 by Baumol and was referred to as the expected gain-confidence limit criterion (see Baumol 1963).

where F_L is the distribution function of the losses. For the sake of completeness the concept of the mean VaR, $\text{VaR}_\alpha^{\text{mean}}$, is also introduced. This risk measure is defined as the difference between VaR_α and the expected return μ . If the chosen time period is 1 day, this measure is also referred to as the *daily earnings at risk*.

One criticism against the use of VaR as a measure of risk is that it is inconclusive about the size of the loss if it is greater than that implied by the chosen confidence level. Put differently, if there are 95 sunny days and 5 rainy days, one is typically not interested in the least amount of rain to expect on these five rainy days, but rather would like to have an estimate of the average rainfall in these cases. The *expected shortfall* (ES) risk measure, introduced by Artzner et al. (1997) and Artzner (1999), directly addresses this issue. This measure provides hindsight about the size of the expected loss if the VaR has been violated for a given level of confidence. It is defined for a Type I error α as

$$\text{ES}_\alpha = \frac{1}{1 - \alpha} \int_\alpha^1 q_u(F_L) du, \quad (4.5)$$

where $q_u(F_L)$ is the quantile function of the loss distribution F_L . The ES can therefore be expressed in terms of the VaR as

$$\text{ES}_\alpha = \frac{1}{1 - \alpha} \int_\alpha^1 \text{VaR}_u(L) du, \quad (4.6)$$

and can be interpreted as the average VaR in the interval $(1 - \alpha, 1)$. To better understand these concepts, the VaR and ES risk measures and the expected loss are shown in Figure 4.1.

Hitherto, no explicit assumption has been made about the kind of loss distribution. The VaR and ES can be computed based upon the empirical distribution for a given sample. In this approach to a historic simulation one sorts the losses with respect to their size. As an example, take a sample size of 1000 loss observations. Then the VaR at the 99% confidence level would correspond to the 10th largest loss and the ES can be determined as the mean or median of the 10 largest losses. In contrast to determining the risk measures from historic data alone, one could also assume that the losses follow some distribution. If one assumes that the losses are normally distributed, then these risk measures can be computed in closed form as

$$\text{VaR}_\alpha = \sigma \Phi^{-1}(\alpha) - \mu, \quad (4.7)$$

$$\text{ES}_\alpha = \sigma \frac{\phi(\Phi^{-1}(\alpha))}{1 - \alpha} - \mu, \quad (4.8)$$

where ϕ denotes the density function of the standard normal distribution.

Although we have assumed so far that the losses are iid, it is worth stressing that by choosing an ill-founded assumption with respect to the chosen distribution from which the risk measures are derived, a severe modelling error can result. Recall from the stylized facts of the previous chapter that one cannot ordinarily assume normally distributed returns. The empirical distribution of returns possesses more probability mass in the tails compared to the Gaussian distribution. The implication is then that risk measures that are derived from the normal assumption generally

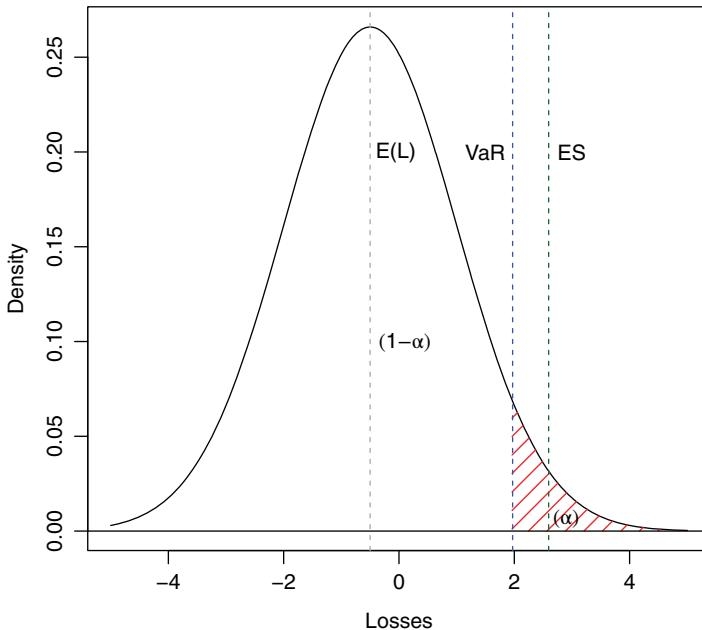


Figure 4.1 Density of losses with VaR and ES.

underestimate the riskiness of a position in a single financial instrument. The modelling error is—*cum grano salis*—lessened by assuming, for instance, a Student's t distribution with fewer than 30 degrees of freedom, due to a greater probability mass in the tails compared to the normal distribution. But a modelling error may still result. It should further be stressed that these two distributions are symmetric. Hence, this characteristic would imply a modelling error in cases of skewed losses, that is, the losses are distributed asymmetrically. This error can in principle be rectified by employing partial moment estimators for the dispersion of losses.

The stylized facts of skewed and fat-tailed return/loss distributions can also be reflected directly in the calculation of the VaR. Zangari (1996) proposed the modified VaR (mVaR), which explicitly takes the higher moments into account. Here, the true but unknown quantile function is approximated by a second-order Cornish–Fisher expansion based upon the quantile function of the normal distribution (see Cornish and Fisher 1937; Jaschke 2001). Hence, the mVaR is also referred to as Cornish–Fisher VaR and is defined as

$$mVaR_\alpha = VaR_\alpha + \frac{(q_\alpha^2 - 1)S}{6} + \frac{(q_\alpha^3 - 3q_\alpha)K}{24} - \frac{(2q_\alpha^3 - 5q_\alpha)S^2}{36}, \quad (4.9)$$

where S denotes the skewness, K the excess kurtosis, and q_α the quantile of a standard normal random variable with level α . In the case of a normally distributed random variable, $mVaR_\alpha = VaR_\alpha$ because the skewness and excess kurtosis are zero. The Cornish–Fisher VaR produces a more conservative risk estimate if the return

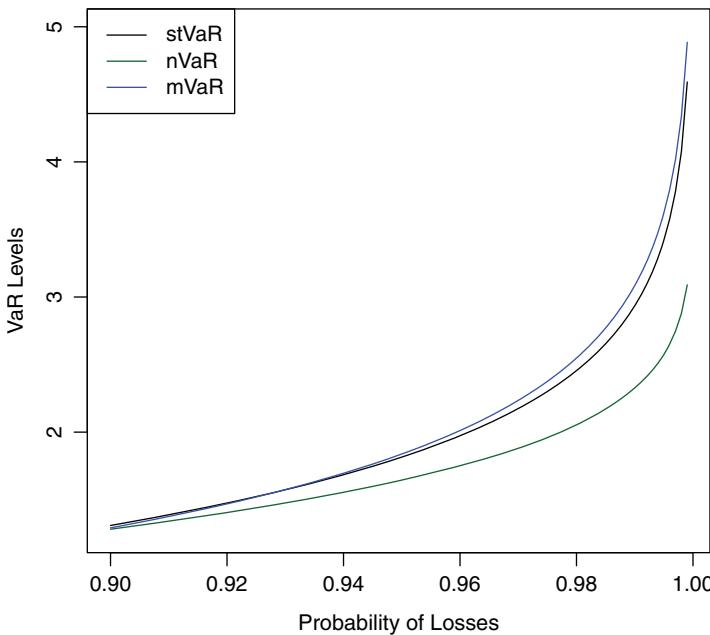


Figure 4.2 Gaussian VaR, mVaR, and VaR of a skewed Student's t distribution.

distribution is skewed to the left (losses are skewed to the right) and/or the excess kurtosis is greater than zero. It should be noted that the mVaR yields less conservative risk estimates than the Gaussian VaR if these conditions are not met.

A comparison between the mVaR and the Gaussian VaR for confidence levels in the interval [90%, 100%) is shown in Figure 4.2. Here the losses have been computed from a skewed Student's t distribution with 10 degrees of freedom and a skew parameter of $\xi = 1.5$. The skewness parameter $\xi > 0$ is defined as the square root of the ratio for probabilities above and below the mean. If $\xi < 1$ a left-skewed distribution results, if $\xi = 1$ the distribution is symmetric, and if $\xi > 1$ a right-skewed distribution occurs (see Lambert and Laurent 2001). It can be concluded from this figure that for the chosen parameter constellation the actual VaR is pretty well captured by the mVaR, and that if one were to use the Gaussian VaR instead the risk would be markedly underestimated.

In Boudt et al. (2008) a modified ES (mES) was derived from the Cornish–Fisher VaR, whereby a second-order Edgeworth expansion was utilized. Its computation is confined to the left tail of the distribution only. The mES thus derived can, however, be smaller than the mVaR for $\alpha \rightarrow 0$. This artifact is due to the nature of the Edgeworth expansion. The authors suggest in these instances choosing the more conservative risk estimate of the two as a pragmatic solution.

More direct approaches and promising alternatives in terms of modelling market risks while adhering to the stylized facts of financial market returns will be covered in Part II.

4.3 Portfolio risk concepts

Needless to say, the risk measures introduced in the previous section can be computed for portfolio return data too. But in a portfolio context an investor is often interested in the risk contributions of single positions or a group thereof. Furthermore, s/he might be interested in how the portfolio risk is affected if the weight of a position is increased by one unit, that is, the marginal contribution to risk. The concept of component VaR or component ES addresses the former question. This notion of breaking down the overall risk into single positions held in a portfolio can also be applied to the mVaR and mES (see Boudt et al. 2008).

We will now assume that a portfolio consists of N financial instruments with weight vector ω . The portfolio return equals $r_p = \omega' \mu$ and the portfolio variance is computed as $m_2 = \omega' \Sigma \omega$, where μ denotes the return vector of the instruments and Σ is the variance-covariance matrix of the portfolio's assets. Under the assumption of normality the risk measures VaR and ES can be computed as

$$\text{VaR}_\alpha = -\omega' \mu - \sqrt{m_2} \Phi^{-1}(\alpha), \quad (4.10)$$

$$\text{ES}_\alpha = -\omega' \mu + \sqrt{m_2} \frac{1}{\alpha} \phi[\Phi^{-1}(\alpha)], \quad (4.11)$$

where $\phi(\cdot)$ is the density and $\Phi^{-1}(\cdot)$ the quantile function of the standard normal distribution.

These two risk measures depend only on the first and second central moments and are linear homogeneous in ω . For risk measures of this kind (i.e., $f(\omega)$), a componentwise breakdown can be achieved by considering the partial derivatives

$$f(\omega) = \sum_{i=1}^N \omega_i \delta_i f(\omega), \quad (4.12)$$

where $\delta_i f(\omega) = \delta f(\omega)/\delta \omega_i$ is the partial derivative for the portfolio weight of the i th asset and $i = 1, \dots, N$.

The risk contribution of the i th portfolio position is therefore given by $C_i f(\omega) = \omega_i \delta_i f(\omega)$. This can also be expressed in relative terms (i.e., as a percentage), by using $\%C_i f(\omega) = C_i f(\omega)/f(\omega) \times 100$. The risk contributions for each of the portfolio's components can therefore be computed as

$$\delta_i \text{VaR}(\alpha) = -\mu_i - \frac{\delta_i m_2}{2\sqrt{m_2}} \Phi^{-1}(\alpha), \quad (4.13)$$

$$\delta_i \text{ES}(\alpha) = -\mu_i + \frac{\delta_i m_2}{2\sqrt{m_2}} \frac{1}{\alpha} \phi[\Phi^{-1}(\alpha)], \quad (4.14)$$

where $\delta_i m_2 = 2(\Sigma \omega)_i$. It should be noted that, in the case of the normal distribution, the partial derivatives with respect to the portfolio weights can be computed rather easily and that this is not the case if one assumes alternative distributions and/or

more complex measures of risk. In these cases, the risk contributions can be obtained heuristically by a Monte Carlo analysis, for instance.

As shown above, the third and fourth moments are required for computing the modified risk measures mVaR and mES. In the case of multivariate random variables, these moments are defined by the $(N \times N^2)$ matrix

$$M_3 = \mathbb{E}[(\mathbf{r} - \boldsymbol{\mu})(\mathbf{r} - \boldsymbol{\mu})' \otimes (\mathbf{r} - \boldsymbol{\mu})] \quad (4.15)$$

for the skewness and by the $N \times N^3$ matrix

$$M_4 = \mathbb{E}[(\mathbf{r} - \boldsymbol{\mu})(\mathbf{r} - \boldsymbol{\mu})' \otimes (\mathbf{r} - \boldsymbol{\mu})' \otimes (\mathbf{r} - \boldsymbol{\mu})'] \quad (4.16)$$

for the kurtosis. Here, the operator \otimes denotes the Kronecker product between two matrices. It follows that the third moment of a portfolio is given by $m_3 = \boldsymbol{\omega}' M_3 (\boldsymbol{\omega} \otimes \boldsymbol{\omega})$ and the fourth moment can be computed as $m_4 = \boldsymbol{\omega}' M_4 (\boldsymbol{\omega} \otimes \boldsymbol{\omega} \otimes \boldsymbol{\omega})$, where the theoretical moments are expressed by their respective estimates. The partial derivatives for these moments are $\delta_i m_3 = 3(M_3(\boldsymbol{\omega} \otimes \boldsymbol{\omega}))_i$ and $\delta_i m_4 = 4(M_4(\boldsymbol{\omega} \otimes \boldsymbol{\omega} \otimes \boldsymbol{\omega}))_i$. Finally, the skewness of a portfolio can be determined as $s_p = m_3/M_2^{3/2}$ and the excess kurtosis by evaluating the expression $k_p = m_4/M_2^2 - 3$. Boudt et al. (2008) provide a detailed derivation of these results as well as the partial derivatives for determining the risk contributions. These can be stated similarly as was done for the non-modified risk measure (see (4.13) and (4.14) above).

Having introduced the most commonly encountered measures of market risk, namely VaR, ES, and their modifications. we will next address the issue of their appropriateness in the portfolio context. Artzner et al. (1996) and Artzner (1999); Artzner et al. (1997) defined four axioms and derived from these the definition of a coherent measure of risk. The axioms are termed:

- monotonicity
- translation invariance
- positive homogeneity
- sub-additivity.

These four axioms will now be briefly explained. Let ρ denote a risk measure and $\rho(L)$ the risk value of a portfolio, where the loss L is a random variable.

The axiom of monotonicity requires, for two given losses L_1 and L_2 with $L_1 \leq L_2$, that this relation is also reflected when the risk measures are calculated for each loss. Therefore a risk measure must fulfill the condition $\rho(L_1) \leq \rho(L_2)$ in order to be considered as a monotone measure of risk.

The axiom of translation invariance ensures that the risk measure is defined in the same units as the losses and is formally written as $\rho(L + l) = \rho(L) + l$ with $l \in \mathbb{R}$.

A risk measure satisfies the axiom of positive homogeneity if $\rho(\lambda L) = \lambda \rho(L)$ with $\lambda > 0$. This requirement ensures the scaling of the risk measures with respect to the size of a position. This axiom would be violated if the size of a portfolio position

directly influenced its riskiness. As an example, suppose that the price of a stock is 100 monetary units and the associated risk measure is 3% for a holding period of 1 day. Then the risk should be scalable with respect to the size of the position. If one owned one stock the risk would be 3 monetary units, and if one owned 10 stocks then the corresponding risk would be tenfold, namely 30 monetary units. This axiom should not be mistaken for liquidity risk. When one needs to sell certain assets the exercise price can of course be a function of the trade size.

Finally, a risk measure is said to be sub-additive if $\rho(L_1 + L_2) \leq \rho(L_1) + \rho(L_2)$. Intuitively, this axiom states that the portfolio risk shall be less than or equal to the sum of the single risk measures of the assets contained in the portfolio. Put differently, due to diversification effects the holding of a portfolio is less risky.

A risk measure ρ is said to be coherent if it satisfies all four axioms. Because the VaR is a quantile, the first three axioms are fulfilled, but not the axiom of sub-additivity. Hence, VaR is not a coherent measure of risk in the portfolio context. This implies that the VaR of a portfolio can be greater than the sum of the individual risks. Further, if the VaR is employed for portfolio optimization, ordinarily a more concentrated portfolio results. For an exposition of this the reader is referred to Frey and McNeil (2002) and McNeil et al. (2005, Chapter 6). It should be noted that the standard deviation is also not a coherent measure of risk because the axiom of monotonicity is violated. Incidentally, the semi-standard deviation is coherent, because when this partial moment is calculated only the values above the average value are used. Finally, the ES derived from a continuous density function qualifies as a coherent measure of risk (see Acerbi and Tasche 2002).

References

- Acerbi C. and Tasche D. 2002 On the coherence of expected shortfall. *Journal of Banking and Finance* **26**(7), 1487–1503.
- Alexander G. and Baptista A. 2002 Economic implications of using a mean-VaR model for portfolio selection: A comparison with mean-variance analysis. *Journal of Economic Dynamics & Control* **26**, 1159–1193.
- Artzner P. 1999 Coherent measures of risk. *Mathematical Finance* **9**, 203–228.
- Artzner P., Delbaen F., Eber J., and Heath D. 1996 A Characterization of Measures of Risk. Technical Report 1186, School of Operations Research and Industrial Engineering, College of Engineering, Cornell University, Ithaca, NY.
- Artzner P., Delbaen F., Eber J., and Heath D. 1997 Thinking coherently. *Risk* **10**(11), 68–71.
- Baumol W. 1963 An expected gain-confidence limit criterion for portfolio selection. *Management Science* **10**, 174–182.
- Boudt K., Peterson B., and Croux C. 2008 Estimation and decomposition of downside risk for portfolios with non-normal returns. *The Journal of Risk* **11**(2), 79–103.
- Cornish E. and Fisher R. 1937 Moments and cumulants in the specification of distributions. *Revue de l'Institut International de Statistique* **5**(4), 307–320.
- Frey R. and McNeil A. 2002 VaR and expected shortfall in portfolios of dependent credit risks: Conceptual and practical insights. *Journal of Banking and Finance* **26**, 1317–1334.

- Jaschke S. 2001 The Cornish–Fisher-expansion in the context of delta-gamma-normal approximations. Technical Report 54, Sonderforschungsbereich 373: Quantification and Simulation of Economic Processes, Humboldt-Universität, Wirtschaftswissenschaftliche Fakultät, Berlin.
- Jorion P. 2001 *Value at Risk: The New Benchmark for Measuring Financial Risk* 2nd edn. McGraw-Hill, New York.
- Lambert P. and Laurent S. 2001 *Modelling financial time series using GARCH-type models and a skewed Student density* Université de Liège.
- McNeil A., Frey R., and Embrechts P. 2005 *Quantitative Risk Management: Concepts, Techniques and Tools*. Princeton University Press, Princeton, NJ.
- RiskMetrics Group 1994 RiskMetrics technical document. Technical report, J.P. Morgan, New York.
- Zangari P. 1996 A VaR methodology for portfolios that include options. *RiskMetrics Monitor First Quarter*, 4–12. JP Morgan-Reuters.

5

Modern portfolio theory

5.1 Introduction

More than 60 years have passed since Harry Markowitz's groundbreaking article "Portfolio selection" was published (see Markowitz 1952). Because the more recently advocated approaches to portfolio optimization are still based on this approach, Markowitz's work on modern portfolio theory will be reviewed in this chapter. In the next section the approach itself will be discussed, and subsequently the problems encountered in practice are highlighted. Within this last section the path is laid for the topics covered in Part III of the book, where portfolio optimization techniques designed to cope with the problems of modern portfolio theory and/or tailor-made for certain investors' demands will be covered.

Given the turbulence in the financial markets witnessed during the first decade of this century, the focus of academia and practitioners alike has again shifted to the Markowitz approach for selecting assets in a portfolio, in particular minimum variance portfolios. Thus, the concluding remark of Rubinstein's article on the occasion of the fiftieth anniversary of Markowitz's seminal paper remains true:

Near the end of his reign in 14 AD, the Roman emperor Augustus could boast that he had found Rome a city of brick and left it a city of marble. Markowitz can boast that he found the field of finance awash in the imprecision of English and left it with the scientific precision and insight made possible only by mathematics. (Rubinstein 2002)

5.2 Markowitz portfolios

In this section the portfolio selection approach proposed by Markowitz and the notation used in the chapters of Part III will be introduced. Given the scope of this book, a more detailed derivation of the results is omitted and the reader is referred to such works as Ingersoll (1987), Huang and Litzenberger (1988), Markowitz (1991), Elton et al. (2007), and Scherer (2010).

The groundbreaking insight of Markowitz was that the risk/return profiles of single assets should not be viewed separately but in their portfolio context. In this respect, portfolios are considered to be efficient if they are either risk minimal for a given return level or have the maximum return for a given level of risk. Even though both views of efficient portfolios are equivalent, the kind of portfolio optimization does differ for these two cases. The former is a quadratic optimization with linear constraints, whereas in the latter the objective function is linear and the constraints are quadratic.

In the following it is assumed that there are N assets and that they are infinitely divisible. The returns of these assets are jointly normally distributed. The portfolio return \bar{r} is defined by the scalar product of the $(N \times 1)$ weight and return vectors ω and μ . The portfolio risk is measured by the portfolio variance $\sigma_W^2 = \omega' \Sigma \omega$, where Σ denotes the positive semi-definite variance-covariance matrix of the assets' returns. For the case of minimal variance portfolios for a given portfolio return, \bar{r} , the optimization problem can be stated as:

$$\begin{aligned} P &= \arg \min_{\omega} \sigma_W^2 = \omega' \Sigma \omega, \\ \omega' \mu &= \bar{r}, \\ \omega' i &= 1, \end{aligned} \tag{5.1}$$

where i is the $(N \times 1)$ vector of ones.

In the same year as Markowitz published his seminal paper, the function for determining efficient portfolios was derived by Roy (1952), although the paper by Merton (1972) is cited in the literature more frequently. According to this function, the weight vector for a minimal variance portfolio and a given target return is given by

$$\omega^* = \bar{r}\omega_0^* + \omega_1^*, \tag{5.2}$$

with

$$\begin{aligned} \omega_0^* &= \frac{1}{d}(c\Sigma^{-1}\mu - b\Sigma^{-1}i), \\ \omega_1^* &= -\frac{1}{d}(b\Sigma^{-1}\mu - a\Sigma^{-1}i). \end{aligned}$$

The portfolio standard deviation is given by

$$\sigma = \sqrt{\frac{1}{d}(c\bar{r}^2 - 2b\bar{r} + a)}, \tag{5.3}$$

with $a = \mu' \Sigma^{-1} \mu$, $b = \mu' \Sigma^{-1} i$, $c = i' \Sigma^{-1} i$, and $d = ac - b^2$. Equation (5.2) results from a Lagrange optimization with the constraints for a given target return and weights summing to one. A detailed exposition is contained in Merton (1972). It can be concluded from this equation that the portfolio weights are a linear function of the expected returns. Furthermore, it can be shown that each efficient portfolio can be generated as a linear combination of two other efficient portfolios. In particular, the risk/return profile of an efficient portfolio can be expressed in terms of a linear combination between the global minimal variance (GMV) portfolio and any other efficient portfolio. The covariance between these two portfolios equals the variance of the minimum variance portfolio. Though it might not be evident at first glance, it should be stressed that the only constraint with respect to the portfolio weights is that their sum equals one. Hence, neither the existence of negative weights (short positions) nor weights greater than one (leveraged positions) can be ruled out, *per se*.

Equation (5.3) describes a hyperbola for efficient mean-variance portfolios. The hyperbola is enclosed by the asymptotes $\bar{r} = b/c \pm \sqrt{d/c}\sigma$. The locus of the GMV portfolio is the apex of the hyperbola with weights given by $\omega_{\text{GMV}}^* = \Sigma^{-1} i / i' \Sigma^{-1} i$ (see Figure 5.1).

In contrast to the mean-variance portfolios, the weight vector of the global minimum variance portfolio does not depend on the expected returns of the assets. The upper branch of the hyperbola is the geometric location of all efficient mean-variance portfolios. The marginal risk contributions of the assets contained in these kinds of portfolios are all the same and the weights correspond to the percentage risk contributions. Hence, these weights are Pareto-efficient. Intuitively this makes sense, because in the case of differing marginal contributions to risk, an overall reduction in risk would be feasible and this would violate the minimum variance characteristic for

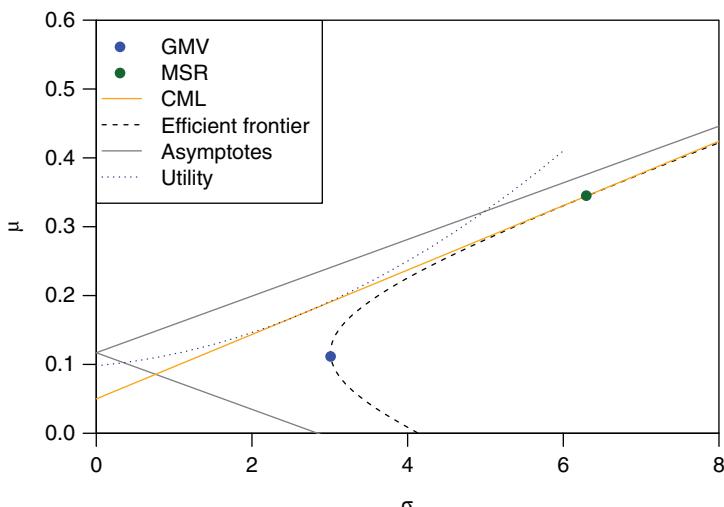


Figure 5.1 Global minimum variance and maximum Sharpe ratio portfolios.

these kinds of portfolios. The risk/return points that are enclosed by the hyperbola are referred to as the feasible portfolios, although these are sub-optimal. In other words, portfolios exist that have either a higher return for a given level of risk or are less risky for certain portfolio return. Both instances would yield a higher utility for the investor.

So far, it has been assumed that the portfolio holdings of an investor are entirely in risky assets. We will now depart from the Markowitz model in the strict sense and allow the holding of a riskless asset with a return of r_f . The question that now arises is how high the optimal holding of this asset in a portfolio should be. This depends on the risk aversion of the investor. A risk averse investor tries to maximize his end-of-period wealth (his expected utility), whereby the decision as to the shape of the portfolio has to be taken at the beginning of the period:¹

$$\max \mathbb{E}[U(W_{t+1})], \quad (5.4)$$

where \mathbb{E} denotes the expectation operator. The utility function can be approximated by a Taylor series expansion and it is further assumed that this function is twice differentiable. After a neutral expansion $U(W_{t+1}) = U(W_{t+1} + \mathbb{E}[W_{t+1}] - \mathbb{E}[W_{t+1}])$, the utility function to be maximized can be written as

$$\begin{aligned} \mathbb{E}[U(W_{t+1})] = & U(\mathbb{E}[W_{t+1}]) + \frac{U'(\mathbb{E}[W_{t+1}])}{1!} \mathbb{E}[W_{t+1} - \mathbb{E}[W_{t+1}]] \\ & + \frac{U''(\mathbb{E}[W_{t+1}])}{2!} \mathbb{E}[W_{t+1} - \mathbb{E}[W_{t+1}]]^2 \\ & + \sum_{i=3}^{\infty} \frac{U^{(i)}(\mathbb{E}[W_{t+1}])}{i!} \mathbb{E}[W_{t+1} - \mathbb{E}[W_{t+1}]]^i. \end{aligned} \quad (5.5)$$

It is worth noting that so far no assumptions have been made about how wealth is distributed. Therefore, (5.5) is defined for a broad class of distribution functions. Further, utility is a function of the higher moments of the wealth distribution. If wealth is assumed to be normally distributed, then the above expression simplifies to

$$\begin{aligned} \mathbb{E}[U(W_{t+1})] = & U(\mathbb{E}[W_{t+1}]) + \frac{U'(\mathbb{E}[W_{t+1}])}{1!} \mathbb{E}[W_{t+1} - \mathbb{E}[W_{t+1}]] \\ & + \frac{U''(\mathbb{E}[W_{t+1}])}{2!} \mathbb{E}[W_{t+1} - \mathbb{E}[W_{t+1}]]^2. \end{aligned} \quad (5.6)$$

Hence, the investor's utility only depends on the first two moments. It should be stressed that even in the case of non-normality, the above optimization approach can be used for quadratic utility functions of the form $U(W) = W - \frac{\lambda}{2}W^2$. Here the parameter λ is a measure of the risk aversion of an investor.

¹ The following exposition draws on Huang and Litzenberger (1988).

If one assumes quadratic utility, then the weight vector is given by $\omega_U = (1/\lambda)\Sigma^{-1}\mu$. The greater the risk aversion is, the smaller the sum of the weights. The expected total return R from the risky assets and the riskless asset is given by

$$\begin{aligned}\mathbb{E}[R] &= (1 - \gamma)r_f + \gamma\bar{r} \\ &= r_f + \gamma(\bar{r} - r_f),\end{aligned}\tag{5.7}$$

where $\gamma = i'\omega$ is the share of risky assets compared to the total wealth. The standard deviation for this portfolio is $\sigma(R) = \gamma\sigma_W$. From this, the capital market line (CML) can be derived—a linear relationship in the (μ, σ) plane—as

$$\mathbb{E}[R] = r_f + \frac{\bar{r} - r_f}{\sigma_W}\sigma(R).\tag{5.8}$$

The optimal portfolio is located at the tangency point of this line and the upper branch of the efficient frontier. This is given when the slope is greatest and hence the Sharpe ratio is at its maximum. The portfolio that is characterized at this tangency point is therefore also referred to as the maximum Sharpe ratio (MSR) portfolio. In the case of the MSR portfolio the investor holds only risky assets. The marginal contributions of the selected assets to the Sharpe ratio are all the same. The investment grade of an investor is determined by the tangency point of his utility function and the CML. This point lies southwest of the MSR portfolio, and the higher the risk aversion is, the closer it will be located to the ordinate.

5.3 Empirical mean-variance portfolios

The theoretical portfolio concepts outlined in the previous section are unfortunately not directly applicable in practice. So far the population moments have been employed in the analysis, but these entities are unknown. In empirical applications these unknown parameters must be replaced by estimates. The locus of the set for the feasible portfolios is below the efficient frontier. At first glance, the sample mean and the unbiased estimator of the variance-covariance matrix of the returns seem to be appropriate candidates for replacing the population moments. In practice, however, potential estimation errors exert a direct impact on the portfolio weights such that, for instance, the desired properties of an efficient and/or a minimum variance portfolio are no longer valid, in general. Ultimately, estimation errors are mirrored by a higher portfolio risk compared to the case of population moments. This is regardless of whether the estimates have been generated from historic data or *ex ante* forecasts for these parameters are employed. For portfolio optimizations that are based on estimators for the expected returns and the variance-covariance matrix, these should have a greater effect compared to optimization approaches that only rest on estimates for the return dispersion, *ceteris paribus* (see Chopra and Ziemba 1993; Merton 1980). Hence, mean-variance portfolio optimizations should suffer more severely from estimation error than minimum variance ones. In empirical simulations and studies it was found that the weights of the former kind of portfolio optimizations are characterized

by wide spans and erratic behavior over time (see, for example, DeMiguel et al. 2007; Frahm 2008; Jagannathan and Ma 2003; Kempf and Memmel 2006; Ledoit and Wolf 2003). From a normative point of view both characteristics are undesired. The effect of “haphazardly” behaving weights is ameliorated for minimum variance portfolios, and hence this portfolio design is to be preferred with respect to the potential impact of estimation errors. The sensitivity of the optimal solutions for mean-variance portfolios with respect to the utilized expected returns is *per se* not a flaw of the approach proposed by Markowitz, but rather an artifact of the quadratic optimization for deriving the portfolio weights.

The errors of the estimates for the expected returns and the variance-covariance matrix could be quantified heuristically beforehand by means of Monte Carlo simulations. This portfolio resampling was proposed by Michaud (1998), and a detailed description with a critique is given in Scherer (2002) and Scherer (2010, Chapter 4).

In a first step, the estimates $\hat{\mu}_0$ and $\hat{\Sigma}_0$ for the theoretical moments μ and Σ for given sample size T are calculated and m points on the empirical efficient frontier are computed. Next, K random samples of dimension $(T \times N)$ are generated and from these the sample moments are determined, giving in total K pairs $(\hat{\mu}_i, \hat{\Sigma}_i)$, $i = 1, \dots, K$. Each of these pairs is then used to compute m points on the respective efficient frontiers. The locus of these simulated efficient frontiers is below the efficient frontier for $\hat{\mu}_0$ and $\hat{\Sigma}_0$. To assess the impact of the estimation error with respect to the mean, the above procedure is repeated, but now the random pairs $(\hat{\mu}_i, \hat{\Sigma}_0)$, $i = 1, \dots, K$, are used. That is, the estimation error is confined to the expected returns. Likewise, the impact of the estimation error for the return dispersion can be evaluated by generating random pairs $(\hat{\mu}_0, \hat{\Sigma}_i)$, $i = 1, \dots, K$. To conclude the exposition of portfolio resampling, it should be noted that randomized efficient portfolios can be retrieved by averaging the weights over m and K . This approach should not be viewed as a panacea for coping with estimation errors. The main critique against this approach is the propagation of errors. The initial values $(\hat{\mu}_0, \hat{\Sigma}_0)$ from which the random samples are generated are estimates themselves and are hence contaminated by estimation errors. Therefore, the initial errors are replicated and mirrored by applying the Monte Carlo analysis. Furthermore, for the case of constrained portfolio optimizations the above procedure can yield unintuitive results (see Scherer 2010, Chapter 4).

Further, recall from Chapter 3 that asset returns are in general not multivariate normally distributed. This implies that, in addition to estimation errors, a model error often exists. A non-stationary return process would be modelled according to a distribution for stationary processes. Hence, there is a trade-off between using a distribution assumption for stationary processes on the one hand, thereby committing a model error, and using a longer sample span by which the stationarity assumption is more likely to be violated but the estimation error diminishes.

The above-stated consequences of estimation errors could in principle be ameliorated beforehand by imposing restrictions on the weights. The following example elucidates the effect of placing constraints on the portfolio weights. Consider two independent investment opportunities A and B with an expected return of 3% and a volatility of 10%. A return-maximizing agent would be indifferent to all linear combinations between these two assets. However, an estimation error as high as one basis

point would result in a very different outcome. Suppose that the estimates for the expected returns are $\hat{\mu}_A = 3.01$ and $\hat{\mu}_B = 2.99$, respectively. This would imply an infinitely high long position in asset A that is financed by an equal-sized short position in asset B . Matters are rather different if long-only and/or bound constraints are included in the optimization. It was found that these kinds of restrictions yield a favorable out-of-sample performance (see, for instance, Frost and Savarino 1988) or are associated with a reduced portfolio risk (see, for instance, Gupta and Eichhorn 1998; Jagannathan and Ma 2003). Both of these empirical findings can be traced back to a smaller implied estimation error if restrictions are imposed on the weights. It is worth mentioning that the locus of portfolios in the (μ, σ) plane are inferior to efficient portfolios to a greater degree as the restrictions become more binding. However, in general an investor is eager to achieve a portfolio allocation that comes as close as possible to the efficient frontier. But the implementation of long-only constraints is undesirable for other reasons. For instance, the implementation of most of the hedge-fund type strategies requires short positioning to be allowed. In summary, the imposition of constraints is not a panacea for all kinds of portfolio strategies and optimizations. Part III of this book will address these issues in more detail and also offer examples of how these more recent advances in portfolio construction can be explored with R.

References

- Chopra V. and Ziemba W. 1993 The effect of errors in means, variances, and covariances on optimal portfolio choice. *Journal of Portfolio Management* **19**, 6–11.
- DeMiguel V., Garlappi L., and Uppal R. 2007 Optimal versus naive diversification: How inefficient is the $1/n$ portfolio strategy?. *Review of Financial Studies* **22**(5), 1915–1953.
- Elton E., Gruber M., Brown S., and Goetzmann W. 2007 *Modern Portfolio Theory and Investment Analysis* 7th edn. John Wiley & Sons, New York, NY.
- Frahm G. 2008 Linear statistical inference for global and local minimum variance portfolios. *Statistical Papers* **51**(4), 789–812.
- Frost P. and Savarino J. 1988 For better performance: Constrain portfolio weights. *Journal of Portfolio Management* **15**(1), 29–34.
- Gupta F. and Eichhorn D. 1998 Mean-variance optimization for practitioners of asset allocation In *Handbook of Portfolio Management* (ed. Fabozzi F.) John Wiley & Sons Chichester, England chapter 4, pp. 57–74.
- Huang C. and Litzenberger R. 1988 *Foundations for Financial Economics*. Elsevier Science Publishing, Amsterdam.
- Ingersoll J. 1987 *Theory of Financial Decision Making*. Rowman & Littlefield, Savage, MD.
- Jagannathan R. and Ma T. 2003 Risk reduction in large portfolios: Why imposing wrong constraints helps. *Journal of Finance* **58**, 1651–1683.
- Kempf A. and Memmel C. 2006 Estimating the global minimum variance portfolio. *Schmalenbach Business Review* **58**, 332–348.
- Ledoit O. and Wolf M. 2003 Improved estimation of the covariance matrix of stock returns with an application to portfolio selection. *Journal of Empirical Finance* **10**, 603–621.

- Markowitz H. 1952 Portfolio selection. *The Journal of Finance* **7**(1), 77–91.
- Markowitz H. 1991 *Portfolio Selection: Efficient Diversification of Investments* 2nd edn. Basil Blackwell, Cambridge, MA.
- Merton R. 1972 An analytical derivation of the efficient portfolio frontier. *Journal of Financial and Quantitative Analysis* **7**, 1851–1872.
- Merton R. 1980 On estimating the expected return on the market: An exploratory investigation. *Journal of Financial Economics* **8**, 323–361.
- Michaud R. 1998 *Efficient Asset Management: A Practical Guide to Stock Portfolio Optimization and Asset Allocation*. Oxford University Press, New York.
- Roy A. 1952 Safety first and the holding of assets. *Econometrica* **20**, 431–449.
- Rubinstein M. 2002 Markowitz's "portfolio selection": A fifty-year retrospective. *The Journal of Finance* **57**(3), 1041–1045.
- Scherer B. 2002 Portfolio resampling: Review and critique. *Financial Analysts Journal* **58**(6), 98–109.
- Scherer B. 2010 *Portfolio Construction and Risk Budgeting* 4th edn. Risk Books, London.

Part II

RISK MODELLING

6

Suitable distributions for returns

6.1 Preliminaries

It was shown in Chapter 4 that risk measures like VaR and ES are quantile values located in the left tail of a distribution. Given the stylized facts of empirical return series, it would therefore suffice to capture the tail probabilities adequately. This is the subject of extreme value theory, which will be covered in the next chapter. However, the need often arises to model not just the tail behavior of the losses, but the entire return distribution. This need arises when, for example, returns have to be sampled for Monte Carlo type applications. Therefore, the topic of this chapter is the presentation of distribution classes that allow returns to be modelled in their entirety, thereby acknowledging the stylized facts. Such a distribution should be capable of mirroring not only heavy-tail behavior but also asymmetries. In particular, the classes of the generalized hyperbolic distribution (GHD) and its special cases, namely the hyperbolic (HYP) and normal inverse Gaussian (NIG) distributions, as well as the generalized lambda distribution (GLD) will be introduced in Sections 6.2 and 6.3. A synopsis of available R packages follows in Sections 6.4 and 6.5, and the chapter ends with applications of the GHD, HYP, NIG, and GLD to financial market data.

6.2 The generalized hyperbolic distribution

The GHD was introduced into the literature by Barndorff-Nielsen (1977). The application of this distribution to the increments of financial market price processes

was probably first proposed by Eberlein and Keller (1995). Further contributions followed in which this distribution class was applied to financial market data. The work of Prause (1997), Barndorff-Nielsen (1997), Barndorff-Nielsen (1998), Eberlein and Prause (1998), and Prause (1999) paved the way for this distribution class to become more widely known in the financial community. The generalized hyperbolic distribution owes its name to the fact that the logarithm of the density function is of hyperbolic shape, whereas the logarithmic values of the normal distribution are parabolic.

The density of the GHD is given by

$$\begin{aligned} \text{gh}(x; \lambda, \alpha, \beta, \delta, \mu) &= a(\lambda, \alpha, \beta, \delta)(\delta^2 + (x - \mu)^2)^{(\lambda - \frac{1}{2})/2} \\ &\quad \times K_{\lambda - \frac{1}{2}}(\alpha \sqrt{\delta^2 + (x - \mu)^2}) \exp(\beta(x - \mu)), \end{aligned} \quad (6.1)$$

where $a(\lambda, \alpha, \beta, \delta)$ is defined as

$$a(\lambda, \alpha, \beta, \delta) = \frac{(\alpha^2 - \beta^2)^{\lambda/2}}{\sqrt{2\pi} \alpha^{\lambda-1/2} \delta^\lambda K_\lambda(\delta \sqrt{\alpha^2 - \beta^2})} \quad (6.2)$$

and K_ν denotes a modified third-order Bessel function with index value ν . The density is defined for $x \in \mathbb{R}$ and encompasses five parameters: $\lambda, \alpha, \beta, \delta, \mu$. The allowable parameter space is defined as $\lambda, \mu \in \mathbb{R}$, $\delta > 0$, $\delta > 0$, and $0 \leq |\beta| < \alpha$. The parameter λ can be interpreted as a class-defining parameter, whereas μ and δ are location and scale parameters.

Three reparameterizations of the GHD can also be found in the literature:

$$\begin{aligned} \zeta &= \delta \sqrt{\alpha^2 - \beta^2}, \rho = \beta/\alpha, \\ \xi &= (1 + \zeta)^{-1/2}, \chi = \xi/\rho, \\ \bar{\alpha} &= \alpha\delta, \bar{\beta} = \beta\delta. \end{aligned} \quad (6.3)$$

These reparameterizations have in common a lack of location and scale parameters. Put differently, these parameterizations do not change for affine transformations of the random variable x . Figure 6.1 shows the densities for different parameter constellations. As is evident from this graph, it is possible to capture not only semi-strong tails (i.e., with a kurtosis greater than 3), but also skewed distributions. From an intuitive point of view it should be reasonable to expect that the following continuous distributions can be derived from the GHD: the hyperbolic, hyperboloid, normal inverse Gaussian, normal reciprocal inverse Gaussian, normal, variance gamma, Student's t , Cauchy, generalized inverse Gaussian (GIG), and skewed Laplace distributions.

The HYP is a special case of the GHD. If the parameter $\lambda = 1$, then the following density results:

$$\text{hyp}(x; \alpha, \beta, \delta, \mu) = \frac{\sqrt{\alpha^2 - \beta^2}}{2\delta\alpha K_1(\delta \sqrt{\alpha^2 - \beta^2})} \exp(-\alpha \sqrt{\delta^2 + (x - \mu)^2} + \beta(x - \mu)), \quad (6.4)$$

where $x, \mu \in \mathbb{R}$, $0 \leq \delta$, and $|\beta| < \alpha$.

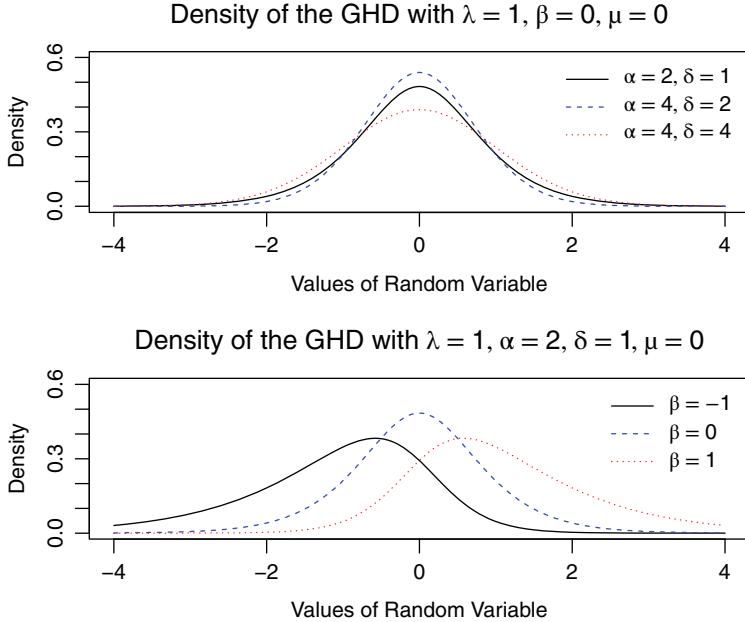


Figure 6.1 Density function of the GHD class.

As mentioned above, λ can be interpreted as a class-selecting parameter. Analogously to the reparameterization of the GHD, the parameters of the HYP can be expressed in these specifications. Here, the reparameterization in the form of (ξ, χ) is of particular interest, since the defined range is given by $0 \leq |\chi| < \xi < 1$. This relation describes a triangle, the so-called shape triangle. Asymptotically, the parameters reflect the third and fourth moments of the distribution (skewness and kurtosis). The HYP can itself be viewed as a general class of distributions which encompasses the following distributions at the limit: for $\xi \rightarrow 0$ a normal distribution results; for $\xi \rightarrow 1$ one obtains symmetric and asymmetric Laplace distributions; for $\chi \rightarrow \pm\xi$ the HYP converges to a generalized inverse Gaussian distribution; and for $|\chi| \rightarrow 1$ an exponential distribution results. The shape triangle can therefore be used as a graphical means of assessing whether a return process can be approximated by one of these distributions.

The NIG distribution can be derived from the GHD if the class-selecting parameter is set to $\lambda = -1/2$. The density of the NIG is given by

$$\text{nig}(x; \alpha, \beta, \delta, \mu) = \frac{\alpha\delta}{\pi} \exp(\delta\sqrt{\alpha^2 - \beta^2} + \beta(x - \mu)) \frac{K_1(\alpha\sqrt{\delta^2 + (x - \mu)^2})}{\sqrt{\delta^2 + (x - \mu)^2}}, \quad (6.5)$$

where the parameter space is defined as $x, \mu \in \mathbb{R}$, $0 \leq \delta$, and $0 \leq |\beta| \leq \alpha$. For instance, this distribution was employed by Barndorff-Nielsen (1998) to model financial market time series.

The unknown parameters can be estimated by the maximum likelihood (ML) principle for a given sample. However, closed-form estimators cannot be derived and hence the negative log-likelihood has to be minimized numerically.

6.3 The generalized lambda distribution

The GLD is an extension of the family of lambda distributions proposed by Tukey (see Tukey 1962). The latter family is defined by the quantile function $Q(u)$ with $u \in [0, 1]$, that is, the inverse of the distribution function:

$$Q(u) = \begin{cases} \frac{u^\lambda - (1-u)^\lambda}{\lambda} & \lambda \neq 0, \\ \frac{\log u}{1-u} & \lambda = 0. \end{cases} \quad (6.6)$$

The parameter λ is referred to as a shape parameter and $Q(u)$ is symmetric. It should be noted that this quantile function does not have a simple closed form for any parameter values λ , except for $\lambda = 0$, and hence the values of the density and distribution function have to be computed numerically. Incidentally, the density function can be expressed parametrically for all values of λ in terms of the quantile function, as in the equation above, and the reciprocal of the quantile density function, that is, the first derivative of (6.6). Tukey's lambda distribution is termed a family of distributions, because many other statistical distributions can be approximated by it. For instance, if $\lambda = -1$ then $Q(u)$ behaves approximately as a Cauchy distribution, and if $\lambda = 1$ a uniform $[-1, 1]$ distribution results. Indeed, an approximate distribution for a given data series can be discerned by plotting the probability plot correlation coefficient.

By splitting the parameter λ in (6.6) into distinct parameters, one obtains the GLD. Here, different parameterizations have been proposed in the literature. A four-parameter extension of the quantile function is due to Ramberg and Schmeiser (1974):

$$Q(u)_{\text{RS}} = \lambda_1 + \frac{u^{\lambda_3} - (1-u)^{\lambda_4}}{\lambda_2}. \quad (6.7)$$

Tukey's lambda distribution is recovered from this specification when $\lambda_1 = 0$ and $\lambda_2 = \lambda_3 = \lambda_4 = \lambda$. The four parameters represent the location (λ_1), the scale (λ_2), and the shape characteristics (λ_3 and λ_4) of a distribution. A symmetric distribution is given for $\lambda_3 = \lambda_4$. The characteristics of this specification have been extensively investigated by Ramberg and Schmeiser (1974), Ramberg et al. (1979), King and MacGillivray (1999), and Karian and Dudewicz (2000), among others. As it turns out, not all parameter combinations yield a valid density/distribution function. The probability density function of the GLD at the point $x = Q(u)$ is given by

$$f(x) = f(Q(u)) = \frac{\lambda_2}{\lambda_3 u^{\lambda_3-1} + \lambda_4 (1-u)^{\lambda_4-1}}. \quad (6.8)$$

Valid parameter combinations for λ must yield the following, such that (6.8) qualifies as a density function:

$$f(x) \geq 0 \quad (6.9)$$

and

$$\int f(x)dx = 1. \quad (6.10)$$

Originally, only four regions of valid parameter constellations for λ_3 and λ_4 were identified by Ramberg and Schmeiser (1974). In Karian et al. (1996) this scheme was amended by additional regions which are labeled “5” and “6” in Figure 6.2. The distributions pertinent to these regions share the same boundaries as for adjacent regions. The parameter constellations for the four/six regions and the implied support boundaries of the GLD are replicated in Table 6.1.

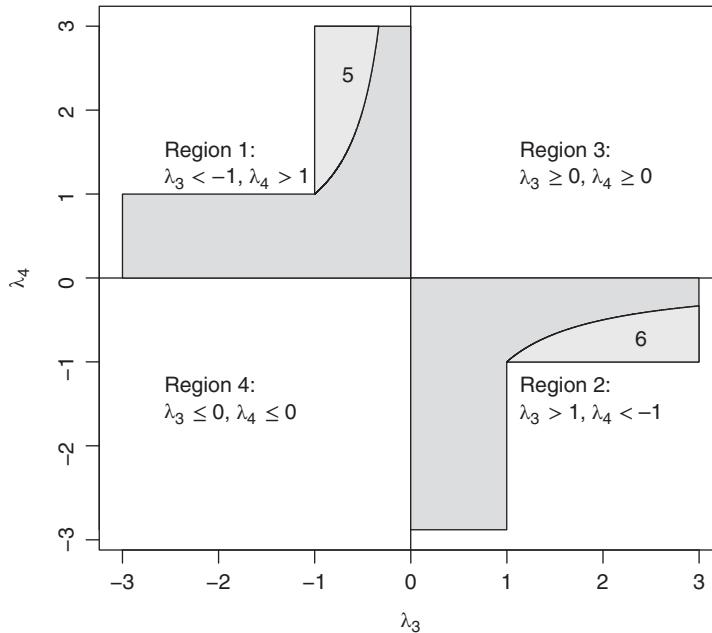


Figure 6.2 GLD: valid parameter combinations of λ_3 and λ_4 in non-shaded areas.

Table 6.1 Range of valid GLD parameter combinations.

Region	λ_1	λ_2	λ_3	λ_4	Minimum	Maximum
1 and 5	all	< 0	< -1	> 1	$-\infty$	$\lambda_1 + (1/\lambda_2)$
2 and 6	all	< 0	> 1	< -1	$\lambda_1 - (1/\lambda_2)$	∞
	all	> 0	> 0	> 0	$\lambda_1 - (1/\lambda_2)$	$\lambda_1 + (1/\lambda_2)$
3	all	> 0	$= 0$	> 0	λ_1	$\lambda_1 + (1/\lambda_2)$
	all	> 0	> 0	$= 0$	$\lambda_1 - (1/\lambda_2)$	λ_1
	all	< 0	< 0	< 0	$-\infty$	∞
4	all	< 0	$= 0$	< 0	λ_1	∞
	all	< 0	< 0	$= 0$	$-\infty$	λ_1

Two observations can be made from Table 6.1. First, the support of the GLD distribution can change quite abruptly for slightly different parameter constellations. Second, parameter constellations that fall in the third quadrant imply skewed and heavy-tailed distributions. These characteristics are part of the stylized facts about financial return series that were stated earlier. Recalling that the market risk measures VaR and ES are quantile values, the GLD seems to be an ideal candidate for computing these measures. This will be shown further below.

In order to avoid the problem that the GLD is confined to certain parameter constellations for λ_3 and λ_4 , Freimer et al. (1988) proposed a different specification:

$$Q(u)_{\text{FMKL}} = \lambda_1 + \frac{\frac{u^{\lambda_3}-1}{\lambda_3} - \frac{(1-u)^{\lambda_4}}{\lambda_4}}{\lambda_2}. \quad (6.11)$$

This specification yields valid density functions over the entire (λ_3, λ_4) plane. The distribution given by this specification will have finite k th-order moment if $\min(\lambda_3, \lambda_4) > -1/k$.

Recently, Chalabi et al. (2010, 2011) proposed a respecification of the GLD. This proposed approach is a combination of using robust estimators for location, scale, skewness, and kurtosis, and expressing the tail exponents λ_3 and λ_4 by more intuitive steepness and asymmetric parameters ξ and χ . The new parameterization takes the following form:

$$\hat{\mu} = u_{0.5}, \quad (6.12)$$

$$\hat{\sigma} = u_{0.75} - u_{0.25}, \quad (6.13)$$

$$\chi = \frac{\lambda_3 - \lambda_4}{\sqrt{1 + (\lambda_3 - \lambda_4)^2}}, \quad (6.14)$$

$$\xi = \frac{1}{2} - \frac{\lambda_3 + \lambda_4}{2 \times \sqrt{1 + (\lambda_3 + \lambda_4)^2}}. \quad (6.15)$$

Here, the bounds for χ and ξ are $-1 < \chi < 1$ and $0 \leq \xi < 1$, respectively. The quantile function of the GLD can then be written as

$$Q(u|\hat{\mu}, \hat{\sigma}, \chi, \xi) = \hat{\mu} + \hat{\sigma} \frac{\hat{S}(u|\chi, \xi) - \hat{S}(0.5|\chi, \xi)}{\hat{S}(0.75|\chi, \xi) - \hat{S}(0.25|\chi, \xi)}. \quad (6.16)$$

The function $\hat{S}(u|\chi, \xi)$ is defined for the following cases:

$$\hat{S}(u|\chi, \xi) = \begin{cases} \log(u) - \log(1-u) & \chi = 0, \xi = 0.5, \\ \log(u) - \frac{(1-u)^{2\alpha}-1}{2\alpha} & \chi = 2\xi - 1, \\ \frac{u^{2\alpha}-1}{2\alpha} - \log(1-u) & \chi = 1 - 2\xi, \\ \frac{u^{\alpha-\beta}-1}{\alpha-\beta} - \frac{(1-u)^{\alpha-\beta}-1}{\alpha-\beta} & \text{otherwise,} \end{cases} \quad (6.17)$$

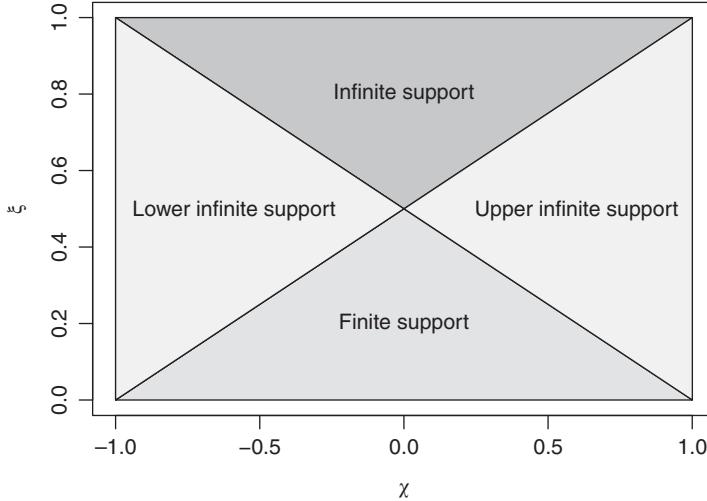


Figure 6.3 GLD shape plot.

where

$$\alpha = \frac{1}{2} \frac{0.5 - \xi}{\sqrt{\xi(1 - \xi)}}, \quad (6.18)$$

$$\beta = \frac{1}{2} \frac{\chi}{\sqrt{1 - \chi^2}}. \quad (6.19)$$

In this parameterization the GLD has infinite support if the condition $(|\chi| + 1)/2 \leq \xi$ is met. Akin to the shape triangle of the HYP, one can now construct a triangle which starts at $\chi = 0$ and has as corners $(\xi = 1, \chi = -1)$ and $(\xi = 1, \chi = 1)$. All parameter combinations of χ and ξ would thus give a distribution with infinite support. The GLD shape plot is shown in Figure 6.3.

As already hinted, the computation of VaR and ES can easily be achieved when a return series has been fitted to the GLD. Here, the formulas are expressed in terms of returns and not for the losses, which are expressed as positive numbers. The VaR for a given probability of error is given by

$$\begin{aligned} \text{VaR}_\alpha &= Q(u|\lambda) \\ &= \lambda_1 + \frac{\alpha^{\lambda_3} - (1 - \alpha)^{\lambda_4}}{\lambda_2}, \end{aligned} \quad (6.20)$$

and the formula for computing the ES for a given probability of error can be expressed as

$$\begin{aligned} \text{ES}_\alpha &= \int_{-\infty}^{\text{VaR}} xf(x|\lambda)dx = \int_{-\infty}^{\alpha} Q(u|\lambda)du \\ &= \lambda_1 + \frac{1}{\lambda_2(\lambda_3 + 1)} \alpha^{\lambda_3 + 1} + \frac{1}{\lambda_2(\lambda_4 + 1)} [(1 - \alpha)^{\lambda_4 + 1} - 1]. \end{aligned} \quad (6.21)$$

Various estimation methods for finding optimal values for the parameter vector λ have been proposed in the literature. Among these are

- the moment-matching approach
- the percentile-based approach
- the histogram-based approach
- the goodness-of-fit approach
- maximum likelihood and maximum product spacing.

The method of moment matching was suggested in the seminal papers of Ramberg and Schmeiser (1974) and Ramberg et al. (1979). The first four moments are matched to the distribution parameters $\lambda_1, \dots, \lambda_4$. For $\lambda_1 = 0$, the k th moment of the GLD is defined as

$$\mathbb{E}(X^k) = \lambda_2^{-k} \sum_{i=0}^k \binom{k}{i} (-1)^i \beta(\alpha, \gamma), \quad (6.22)$$

where $\beta(\alpha, \gamma)$ denotes the beta function evaluated at $\alpha = \lambda_3(k - i) + 1$ and $\gamma = \lambda_4 + 1$. A nonlinear system of four equations in four unknowns results, and has to be solved. Incidentally, this can be accomplished sequentially, by first determining estimates for λ_3 and λ_4 and then solving for the two remaining parameters (see Ramberg and Schmeiser 1974, Section 3). This approach is only valid in the parameter regions for which these moments exist (region 4), and the condition $\min(\lambda_3, \lambda_4) < -1/4$ must be met. As an aside, because the estimates for the skewness and the kurtosis of a data set are very sensitive to outliers, the resulting parameter vector λ is affected likewise. In order to achieve robust estimation results with respect to outlier sensitivity, Chalabi et al. (2010) suggested replacing the moment estimators by their robust counterparts. The robust moments for location, scale, skewness, and kurtosis are defined as (see, for instance, Kim and White 2004):

$$\mu_r = \pi_{1/2}, \quad (6.23)$$

$$\sigma_r = \pi_{3/4} - \pi_{1/4}, \quad (6.24)$$

$$s_r = \frac{\pi_{3/4} + \pi_{1/4} - 2\pi_{1/2}}{\pi_{3/4} - \pi_{1/4}}, \quad (6.25)$$

$$k_r = \frac{\pi_{7/8} - \pi_{5/8} + \pi_{3/8} - \pi_{1/8}}{\pi_{6/8} - \pi_{2/8}}. \quad (6.26)$$

These statistics can be estimated by inserting the empirical quantiles p_q . It is shown in Chalabi et al. (2010) that the higher robust skewness and kurtosis moments only depend on λ_3 and λ_4 . Hence, a nonlinear system of two equations in two unknowns results, which has to be solved. The robust estimation of the GLD parameters has the further advantage of deriving a standardized distribution characterized by a zero median, a unit interquartile range, and the two shape parameters λ_1 and λ_2 :

$$\begin{aligned}
Q(u|\lambda_3, \lambda_4) &= Q(u|\lambda_1^*, \lambda_2^*, \lambda_3, \lambda_4), \\
\lambda_2^* &= S_{\lambda_3, \lambda_4}(3/4) - S_{\lambda_3, \lambda_4}(1/4), \\
\lambda_1^* &= -S_{\lambda_3, \lambda_4}(1/2)/\lambda_2^*, \tag{6.27}
\end{aligned}$$

where $S_{\lambda_3, \lambda_4}(u)$ is equal to the numerator in (6.7).

Karian and Dudewicz (1999) proposed an estimation approach based on the empirical percentiles of the data. From the order statistics $\hat{\pi}_p$ of the data the following four percentiles are defined, where $u \in (0, 0.25)$:

$$\hat{p}_1 = \hat{\pi}_{0.5}, \tag{6.28}$$

$$\hat{p}_2 = \hat{\pi}_{1-u} - \hat{\pi}_u, \tag{6.29}$$

$$\hat{p}_3 = \frac{\hat{\pi}_{0.5} - \hat{\pi}_u}{\hat{\pi}_{1-u} - \hat{\pi}_{0.5}}, \tag{6.30}$$

$$\hat{p}_4 = \frac{\hat{\pi}_{0.75} - \hat{\pi}_{0.25}}{\hat{p}_2}. \tag{6.31}$$

For $u = 0.1$ these percentiles refer to the sample median (\hat{p}_1), the interdecile range (\hat{p}_2), the left-right tail weight ratio (\hat{p}_3), and a measure of relative tail weights of the left tail to the right tail (\hat{p}_4). These quantiles correspond to the following quantiles of the GLD:

$$p_1 = Q(0.5) = \lambda_1 + \frac{0.5^{\lambda_3} - 0.5^{\lambda_4}}{\lambda_2}, \tag{6.32}$$

$$p_2 = Q(1-u) - Q(u) = \frac{(1-u)^{\lambda_3} - u^{\lambda_4} + (1-u)^{\lambda_4} - u^{\lambda_3}}{\lambda_2}, \tag{6.33}$$

$$p_3 = \frac{Q(0.5) - Q(u)}{Q(1-u) - Q(0.5)} = \frac{(1-u)^{\lambda_4} - u^{\lambda_3} + 0.5^{\lambda_3} - 0.5^{\lambda_4}}{(1-u)^{\lambda_3} - u^{\lambda_4} + 0.5^{\lambda_4} - 0.5^{\lambda_3}}, \tag{6.34}$$

$$p_4 = \frac{Q(0.75) - Q(0.25)}{p_2} = \frac{0.75^{\lambda_3} - 0.25^{\lambda_4} + 0.75^{\lambda_4} - 0.25^{\lambda_3}}{(1-u)^{\lambda_3} - u^{\lambda_4} + (1-u)^{\lambda_4} - u^{\lambda_3}}. \tag{6.35}$$

This nonlinear system of four equations in four unknowns has to be solved. Similar to the moment-matching method, a sequential approach by first solving only the subsystem consisting of $\hat{p}_3 = p_3$ and $\hat{p}_4 = p_4$ for λ_3 and λ_4 can be applied.

Deriving estimates for λ from histograms was proposed by Su (2005, 2007). Within this approach the empirical probabilities are binned in a histogram and the resulting midpoint probabilities are fitted to the true GLD density. A drawback of this method is that the resultant estimates are dependent on the chosen number of bins.

The fourth kind of estimation method is based on goodness-of-fit statistics, such as the Kolmogorov–Smirnov, Cramér–von Mises, or Anderson–Darling statistics. These statistics measure the discrepancy between the hypothetical GLD and the empirical distribution, which is derived from the order statistics of the data in question. Parameter estimates can be employed when these statistics are minimized with respect to the parameters of the GLD. The determination of the parameter values can be achieved

with the starship method as proposed by Owen (1988) and adapted to the fitting of the GLD by King and MacGillivray (1999). It consists of the following four steps:

1. Compute the pseudo-uniform variables of the data set.
2. Specify a valid range of values for $\lambda_1, \dots, \lambda_4$ and generate a four-dimensional grid of values that obey these bounds.
3. Calculate the goodness-of-fit statistics compared to the uniform $(0, 1)$ distribution.
4. Choose the grid point $(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$ that minimizes the goodness-of-fit statistic as the estimate for λ .

Finally, the GLD parameters could also be estimated by the ML principle and/or the method of maximum product spacing. The latter method was proposed separately by Cheng and Amin (1983) and Ranneby (1984). This method is also based on the order statistics $\{x_{(1)}, x_{(2)}, \dots, x_{(N)}\}$ of the sample $\{x_1, x_2, \dots, x_N\}$ of size N . Next, the spacings between adjacent points are defined as $D(x_{(i)}|\lambda) = F(x_{(i)}|\lambda) - F(x_{(i-1)}|\lambda)$ for $i = 2, \dots, N$. The objective is the maximization of the sum of the logarithmic spacings. Compared to the ML method, the maximum product spacing method has the advantage of not breaking down when the support of the distribution is not warranted for a given parameter combination.

6.4 Synopsis of R packages for GHD

6.4.1 The package **fBasics**

The package **fBasics** is part of the Rmetrics suite of packages (see Würtz et al. 2014). The primary purpose of this package is to provide basic tools for the statistical analysis of financial market data. Within the package S4 classes and methods are utilized. The package is considered a core package in the CRAN “Finance” Task View and is also listed in the “Distributions” Task View. The package has dependencies on other packages contained in the Rmetrics suite. With respect to the modelling, fitting, and inferences drawn from the GHD, quite a few functions have been directly ported and/or included from the package **GeneralizedHyperbolic** to this package. The latter package will be presented in the next subsection.

With respect to the topic of this chapter, the following distributions are addressed in this package: the generalized hyperbolic, the generalized hyperbolic Student’s t , the hyperbolic, and the normal inverse Gaussian, as well as the standardized versions of the GHD and NIG distributions. For each of these distributions, functions for calculating the value of the density, the probabilities, the quantiles, and the generation of random numbers are available. The R naming convention for the density (prefix **d**), distribution (prefix **p**), quantile function (prefix **q**), and the generation of random numbers (prefix **r**) is followed. In addition, routines for fitting and for calculation of the mode and moments are included in the package. In particular, the mean, variance, skewness, and kurtosis are implemented for all distributions. It is

further possible to return their robust counterparts, namely, the median, the interquartile range, and skewness or kurtosis measures that are derived from these quartiles. The shape of the density for the GHD, HYP, and NIG can be plotted interactively for various parameter combinations through a `tcl/Tk` interface. These functions are termed `ghSlider()`, `hypSlider()`, and `nigSlider()`, respectively, and are intended primarily for illustrative purposes.

The functions that relate to the GHD have `gh` in their names, while those for the generalized hyperbolic Student's t have `ght`, those for the HYP have `hyp`, and those for the NIG have `nig`. Thus, the routines for fitting these distributions to financial market return data are termed `fooFit()` where `foo` is replaced by one of these abbreviations. Similarly, the mode is returned by the function `fooMode()` and the routines for the moments are `fooMean()`, `fooVar()`, `fooSkew()`, and `fooKurt()` for the mean, the variance, the skewness, and the kurtosis, respectively. Analogously, the functions `fooMED()`, `fooIQR()`, `fooSKEW()`, and `fooKURT()` relate to the robust counterparts, namely the median, the interquartile range, and the robust definitions of the skewness and the kurtosis.

By default, the unknown parameters of the distributions are estimated by applying the ML principle. Here, the negative log-likelihood is minimized with the function `nlminb()`. It should be noted that the ellipsis argument in the function `fooFit()` is passed down to the plotting of the fitted distribution, hence the user cannot pass arguments directly to the optimization routine. With respect to fitting the NIG, in addition to the ML principle, the parameters can be estimated by the generalized methods of moments, maximum product spacing, or minimum variance product spacing. It is also possible to produce a shape triangle for fitted objects with the routine `nigShapeTriangle()`. All `fit()` methods return an object of formal class `FDISTFIT`, for which a `show()` method is defined.

6.4.2 The package `GeneralizedHyperbolic`

This package offers functions not only for the GHD, but also for the derived distributions HYP, GIG, and skew Laplace (see Scott 2015). The package is written purely in R. A NAMESPACE file is included in the package's source that contains the export directives for the functions and S3 methods pertinent to the above-mentioned distributions. Some of the routines contained in this package have been ported to **fBasics**. Routines for fitting the hyperbolic, normal inverse Gaussian, and generalized inverse Gaussian distributions to data have been implemented. Six data sets from the fields of geology, engineering, and finance are provided. The package is contained in the CRAN Task View "Distributions."

The functions that relate to the generalized hyperbolic have the acronym `ghyp` in their names. Routines for the density, the distribution, the quantile function, and the generation of random numbers are implemented, and the `dpqr` naming convention as introduced in Section 6.4.1 is followed. Furthermore, the first derivative of the density function has been implemented as function `ddghyp()`. Quantile-quantile and percentile-percentile plots for a given set of parameters and an observed time series have been implemented as functions `qqghyp()` and `ppghyp()`,

respectively. It is further possible to return the ranges on the real line where the probability mass is small for a given parameter set (`ghypCalcRange()`). Whether certain parameters are in the allowable set of values can be checked with the function `ghypCheckPars()`. Furthermore, with the routine `ghypChangePars()` the user can calculate the parameters for the alternative specifications of the GHD. The moments and mode of the GHD distribution can be computed with the functions `ghyperbMean()`, `ghyperbVar()`, `ghyperbSkew()`, `ghyperbKurt()`, and `ghyperbMode()` for the mean, variance, skewness, kurtosis, and mode, respectively. The moments of the GHD can also be computed with the function `ghypMom()`.

The chosen acronym for the hyperbolic distribution in this package is `hyperb`. The `dpqr` naming convention is followed as in the case of the generalized hyperbolic distribution, and the first derivative of the density function has been implemented as function `ddhyperb()`. The two-sided Hessian matrix for a given data vector and parameter set can be calculated by calling `hyperbHessian()`. The points on the real line for which the probability mass for a given set of parameters is negligibly small can be determined by means of the routine `hyperbCalcRange()`. Conversion from one HYP parameterization to another is accomplished by invoking `hyperbChangePars()`. A similar group of functions to the case of the GHYP for recovering the moments is also made available, now prefixed by `hyperb` instead of `ghyp`. The fitting of the HYP distribution is achieved with the function `hyperbFit()`. Suitable starting parameters can be determined with the function `hyperbFitStart()`. The user has the option to choose between six different numerical optimization algorithms, namely Nelder–Mead, BFGS (Broyden–Fletcher–Goldfarb–Shanno), and L-BFGS-B (all implemented in `optim()`), nonlinear minimization (`nlm()`), or constrained optimization (`constrOptim()`). Arguments to control the behavior of the optimizer can be specified in the call to `hyperbFit()`. This function returns an object with class attributes `hyperbFit`, `distFit`. For objects of this kind, `print()`, `summary()`, and `plot()` methods have been defined. In addition, there are `coef()` and `vcov()` methods for retrieving the estimated parameters and their variance-covariance matrix. The fit can be assessed in the form of a quantile–quantile (QQ) plot (`qqhyperb()`) or probability–probability (PP) plot (`pphyperb()`). Furthermore, a Cramér–von Mises goodness-of-fit test is implemented as function `hyperbCvMTest()`. This function returns an object with class attribute `hyperbCvMTest` for which a `print()` method has been defined. Finally, a linear model with hyperbolic errors can be fitted by utilizing `hyperblm()`.

Basically, the same set of functions that have been defined for the HYP are made available for the generalized inverse Gaussian (acronym `gig`). In addition, the moments and raw moments for a given parameter specification are computed by the functions `gigMom()` and `gigRawMom()`, respectively. The moments of its special cases, namely the gamma and inverse gamma distributions, can be invoked by `gammaRawMom()`. Similarly, the same set of functions that have been defined for the GHD are made available for the GIG. In addition, QQ and PP plot routines are included in the package for this distribution.

In addition, the functions that directly relate to the skew Laplace distribution (acronym `skewlap`) are available: the density, the distribution, the quantile function, and the generation of random numbers, as well as methods for producing QQ and PP plots.

6.4.3 The package `ghyp`

In contrast to the previous package, `ghyp` provides functions for fitting not only the univariate HYP, but also the GHD, NIG, VG, Student’s *t*, and Gaussian distributions for the univariate and multivariate cases (see Luethi and Breymann 2013). The package utilizes S4 classes and methods and is shipped with a NAMESPACE file. It is contained in the CRAN “Distributions” and “Finance” Task Views. In addition to the package’s help files, a vignette is available.

The functions that relate to the GHD or the GIG are the density, quantile, probability, and random variates routines. The `dpqr` naming convention is followed and the chosen acronyms are `ghyp` and `gig` for the GHD and the GIG, respectively. A feature of this package is the inclusion of routines for calculating the expected shortfall for these distributions. The functions are named `ESghyp()` and `ESgig()`. Furthermore, the package offers a function for portfolio optimization (`portfolio.optimize()`). The user can choose the risk measure employed, namely the standard deviation, the VaR, or the ES, and whether the portfolio should be a minimum risk, a tangency, or a target return portfolio. These portfolios are derived from a multivariate GHD.

To estimate the unknown coefficients of the GHD and its special cases, the ML principle is employed. The function names are made up of the prefix `fit.` followed by the acronym of the desired distribution, followed by either `uv` or `mv` for fitting univariate or multivariate data. The objects returned are of formal class `mle.ghyp`. For objects of this kind, `show()` and `summary()` methods are defined as well as methods for extracting the Akaike information criterion (AIC) and the value of the log-likelihood. Furthermore, a routine for model selection using the AIC is implemented as function `stepAIC.ghyp()`. In addition, a function for discriminating between models in the form of a likelihood ratio test is implemented (see routine `lik.ratio.test()`). The estimated parameters can be extracted with the `coef()` method. The moments of objects that inherit from the class `ghyp` can be computed with the function `mean()` for the mean, with the function `vcov()` for the variance in the univariate and the variance-covariance matrix in the multivariate case, and with the functions `ghyp.skewness()` and `ghyp.kurtosis()` for the skewness and kurtosis, respectively. In general, central and non-central moments can be computed with the function `ghyp.moment()`. By default, the skewness and the kurtosis are returned.

For plotting purposes, a QQ plot, a histogram view, and a pairs plot for the graphical display of multivariate QQ plots, as well as plotting the density or superimposing the density on existing plot devices, are available. The functions are termed `qqghyp()`, `hist()`, `pairs()`, `plot()`, and `lines()`, in that order.

The package comes with two data sets. The first, `indices`, contains monthly returns for five asset classes between August 1999 and October 2008. The second data set (`smi.stocks`) contains daily return data for the Swiss equity market and equity returns of selected Swiss companies from 5 January 2000 to 10 January 2007.

6.4.4 The package **QRM**

Most of the examples contained in McNeil et al. (2005) can be replicated with the functions contained in the package **QRM** (see Pfaff and McNeil 2016). These were originally written in the S-PLUS language by A. McNeil and distributed as package **QRMlib**. An initial **R** port was accomplished by S. Ulman and is still available from the CRAN archive (see McNeil and Ulman 2011). The package **QRM** is based on this initial **R** port. It has dependencies on the CRAN packages **gsl**, **mvtnorm**, **numDeriv**, and **timeSeries**. Within **QRM** partial use of **S3** classes and methods is made. The more burdensome computations are interfaced from **C** routines. In addition, 14 financial data sets are available.

With respect to the GHD, functions for fitting data to its special cases, namely the NIG and HYP, are included in this package. These are termed `fit.NH()` for univariate and `fit.mNH()` for multivariate data. The `case` argument of these functions controls whether the negative log-likelihood of the NIG or HYP is minimized. Both routines return a list object without a class attribute. Hence, no further methods are available.

In addition, the moment and log moment of the GIG can be computed with the functions `EGIG()` and `ElogGIG()`, respectively. Random variates of this distribution can be generated with the function `rGIG()` that interfaces to a routine written in **C**.

6.4.5 The package **SkewHyperbolic**

The package **SkewHyperbolic** is dedicated solely to the modelling and fitting of the skew hyperbolic Student's *t* distribution (see Scott and Grimson 2015). The package is written purely in **R**, and **S3** classes and methods are used. It is shipped with a **NAMESPACE** file, and some underlying utility functions are imported from the packages **GeneralizedHyperbolic** and **DistributionUtils**. As well as the functions that primarily deal with the skew hyperbolic distribution, three data sets are included in the package.

With respect to the distribution itself, routines for its density, distribution, and quantile functions as well as for the generation of random variates are included. The `dpqr` naming convention is followed, and the chosen acronym for this distribution is `skewhyp`. In addition to these functions there is a routine for returning the first derivative of the density function (`ddskewhyp()`). Similar to the package **GeneralizedHyperbolic**, a function for determining ranges for which the probability mass is small is available (`skewhypCalcRange()`). The coherence of a parameter set can be checked with the function `skewhypCheckPars()`.

The included routines `skewhypMean()`, `skewhypVar()`, `skewhypSkew()`, and `skewhypKurt()` are used for calculating the mean, variance, skewness, and

kurtosis, respectively. The mode of the distribution for a given parameter set can be computed with `skewhypMode()`. Similarly to the package **GeneralizedHyperbolic**, the central and non-central moments of any order can be calculated with the routine `skewhypMom()`.

The fitting of data to the skew hyperbolic Student's t distribution is accomplished by the function `skewhypFit()`. Suitable starting values can be determined with the routine `skewhypFitStart()`. The parameters are determined numerically by applying the ML principle. The negative log-likelihood is minimized by employing either the general purpose optimizer `optim()` or the function `n1m()`. For the former the user can use either the BFGS or Nelder–Mead algorithm. The function `skewhypFit()` returns an object of informal class `skewhypFit`. For objects of this kind, `print()`, `plot()`, and `summary()` methods are available. Goodness of fit can be inspected graphically by means of a QQ and/or PP plot. The relevant functions are termed `qqskewhyp()` and `ppskewhyp()`, respectively. In addition, a function for producing a tail plot line (`skewhypTailPlotLine()`) for a given data set and parameter specification is provided.

6.4.6 The package **VarianceGamma**

The package **VarianceGamma** can be considered as a twin package to the **SkewHyperbolic** package discussed in the previous subsection, but its focus is on the variance gamma distribution (see Scott and Dong 2015). As its twin, the package is contained in the CRAN “Distributions” Task View. Within the package S3 classes and methods are employed and the package is shipped with a `NAMESPACE` file in which import directives for the utility functions contained in the packages **GeneralizedHyperbolic** and **DistributionUtils** are included. Basically, all functionalities contained in the package **SkewHyperbolic** have been mirrored in this package, and the `dpqr` naming convention is followed. The acronym `vg` is used for the variance gamma distribution. Hence, the discussion of the functions, methods, and classes in Section 6.4.5 carries over to these instances, too.

6.5 Synopsis of R packages for GLD

6.5.1 The package **Davies**

Even though the focus of the package **Davies** is an implementation of the Davies quantile function (see Hankin and Lee 2006), R routines that address the GLD distribution are also included. The package is listed in the CRAN “Distributions” Task View. The package is shipped with a `NAMESPACE` file, but neither S3 nor S4 classes/methods are employed. Hence, in addition to two data sets, the package offers a collection of functions for dealing with these two kinds of distributions—no more and no less.

With respect to the GLD, the functions are based on the Ramberg–Schmeiser (RS) specification. The density, distribution, and quantile functions of the GLD have been implemented, as well as a function for generating random variates, and the `dpqr`

naming convention is followed for naming these routines, (e.g., `dglld()`). Furthermore, the routine `dglld.p()` is an implementation of the density function expressed in terms of the quantile.

The expected value of the GLD for a given parameterization can be retrieved either as an exact value with the routine `expected.gld()` or as an approximation with `expected.gld.approx()` . Within both functions the values are determined as the sum of a constant (λ_1) and two Davies quantile functions.

6.5.2 The package **fBasics**

A general description of the package **fBasics** has already been provided in Section 6.4.1. Hence, in the following description the focus will be on the **R** routines for handling the GLD.

The density, distribution, and quantile functions and a routine for obtaining random variates have been implemented as **R** routines, and the `dpqr` naming convention is followed (e.g., `dglld()`). These functions are wrappers for the bodies of the functions of the routines with the same name contained in the package **gld** by King et al. (2016)—see Section 6.5.3 for a discussion of this package. However, these wrapper functions are limited to the RS specification, and only parameter values for $\lambda_1, \dots, \lambda_4$ pertinent to region 4 of the parameter space can be supplied as arguments to these functions, otherwise an error is returned.

The fitting of data to the GLD is provided by the function `gldFit()` . Similar to the above functions, the GLD is expressed in the RS specification and the optimization is carried out for the parameter space pertinent to region 4. Apart from the data argument `x` and the initial parameter values `lambda1` [234] , the function has an argument `method` by which the estimation method can be set. The available estimation procedures are: maximization of the log-likelihood ("mle"), the method of maximum product spacing ("mps"), robust moment matching ("rob"), goodness of fit ("gof"), and histogram binning ("hist"). If one of the latter two methods is chosen, the user can set the type of goodness-of-fit statistic or the binning method via the function's argument `type` . This argument is passed as an ellipsis argument to either of the user-hidden functions `.gldFit.gof()` or `.gldFit.hist()` , respectively. For estimation methods based on goodness of fit this can be the Anderson–Darling ("ad"), Cramér–von Mises ("cvm"), or Kolmogorov–Smirnov ("ks") statistic. If the histogram approach is chosen, the count of bins can be determined by the Freedman–Diaconis binning ("fd"), Scott's histogram binning ("scott"), or Sturges binning approach ("sturges") . The function returns an S4 object `fDISTFIT` . The estimates, the value of the objective, and the convergence code of the `nlminb()` optimizer are returned as a list in the slot `fit` of objects of this kind. By default, a plot of the estimated density is produced, which can be suppressed by setting `doplot = FALSE` .

The mode of the GLD can be computed for given parameter values of $\lambda_1, \dots, \lambda_4$ with the function `gldMode()` . Robust estimates for location, dispersion, skewness, and kurtosis can be computed for given parameter values with the functions `gldMED()` , `gldIQR()` , `gldSKEW()` , and `gldKURT()` , respectively.

6.5.3 The package **gld**

The package **gld** is, to the author's knowledge, the only one that implements all three GLD specifications: RS, FMKL, and FM5 (see King et al. 2016). The latter is an extension of the FMKL version in which a fifth parameter is included in order to explicitly capture the skewness of the data. The FM5 specification is derived from the modification of the RS specification by Gilchrist (2000).

The package is included in CRAN "Distributions" Task View. S3 classes and methods have been utilized, and the package contains a NAMESPACE file. The distribution functions of the GLD specifications are interfaced from routines written in the C language.

The density, quantile density, distribution, and quantile distribution functions are implemented as R routines `dgl()`, `dqgl()`, `pgl()`, and `qdg1()`, respectively. Random variates of the GLD can be generated with the function `rgl()`. With respect to parameter estimation, the starship method has been implemented as function `starship()`. Here, the initial values are determined according to an adaptive grid search (`starship.adaptivegrid()`) and then used in the call to `optim()`. The objective function itself is included in the package as function `starship.obj()`. The function `starship()` returns an object of informal class `starship` for which `plot()`, `print()`, and `summary()` methods are made available. The validity of the estimated λ parameters can be checked with the function `g1.check.lambda()`. As a means of assessing the goodness of fit graphically, the function `qqgl()` produces a QQ plot. Finally, the density of the GLD can be depicted with the function `plotgl()`.

6.5.4 The package **lmomco**

Estimation methods based on L-moments for various distributions are implemented in the package **lmomco** (see Asquith 2016). Here we will concentrate on those tools that directly address the GLD. The package is considered to be a core package in the CRAN "Distributions" Task View. It is written purely in R and is shipped with a NAMESPACE file with export directives for all relevant user functions. The package is quite huge, judged by the size of its manual, which runs to more than 500 pages. It is worth mentioning that, in addition to estimation methods based on L-moments and their extensions, probability-weighted moment (PWM) estimators are available.

In order to estimate the parameters of the GLD, the L-moments for univariate sample data must be determined first. This can be achieved with the function `lmom.ub()` for unbiased L-moment estimates, with the function `TLmom()` for trimmed L-moments, or with the function `pwm.ub()` for unbiased sample PWMs. If the latter route is chosen, these PWM estimates can be converted to L-moment estimates with the function `pwm2lmom()`. Having estimated the L-moments, the resulting object can be used in the call to the functions `pargld()` and/or `parTLgld()` to estimate the parameters of the GLD by L-moments or trimmed L-moments, respectively. The package offers routines for checking the validity of the estimated parameters and/or L-moments (functions `are.par.valid()`,

`are.pargld.valid()`, and `are.lmom.valid()`) as well as means of converting between parameter estimates and the associated L-moments for a given distribution (functions `vec2par()`, `lmom2par()`, `par2lmom()`, and `lmomgld()`).

The R functions that directly relate to the GLD are `cdfgld()` for the cumulative distribution function, `quagld()` for the quantile function, and `pdfgld()` for the density function. Random variates for a given parameterization of the GLD can be generated with the function `rlmomco()`. The correctness of an empirically determined probability or density function can be assessed with the functions `check.fs()` and `check.pdf()`, respectively.

6.6 Applications of the GHD to risk modelling

6.6.1 Fitting stock returns to the GHD

In this subsection the daily returns of Hewlett Packard (HWP) stock are fitted to the GHD and its special cases, the HYP and NIG. The R code is shown in Listing 6.1. The sample runs from 31 December 1990 to 2 January 2001 and consists of 2529 observations. The following analysis has been conducted with the functions contained in the package `ghyp`. In the listing this package is loaded into the workspace first. The package `fBasics` contains the data set `DowJones30`, which includes the HWP stock price. This series is converted into a `timeSeries` object and the continuous percentage returns are then computed. For comparison of the fitted distributions, the empirical distribution (EDF) is first retrieved from the data with the function `ef()`. Then the returns are fitted to GHD, HYP, and NIG distributions. In each case, possible asymmetries in the data are allowed (i.e., non-zero skewness). In the next chunk of code the shapes of the estimated densities are computed, along with a Gaussian distribution which serves as the benchmark. A plot of the empirical and fitted densities is then produced (see Figure 6.4).

The rather poor description of the empirical return distribution for the Gaussian case is immediately evident from this plot. The normal distribution falls short of capturing the excess kurtosis of 4.811. Matters are different for the class of generalized hyperbolic distributions. In these instances the empirical distribution function is tracked rather well. The fitted HYP and NIG models almost coincide, and from this plot these two distributions cannot be discerned. The fitted GHD seems to mirror the returns slightly better. In particular, the values of the density are closer to their empirical counterparts around the median of the EDF. *Ceteris paribus*, this implies higher probability masses in the tails of the distribution compared to the λ -restricted HYP and NIG distributions.

As a second means of graphically comparing the fitted distributions, QQ plots are produced in the ensuing code lines of Listing 6.1. These are shown in Figure 6.5. For clarity the marks of the fitted normal distribution have been omitted from the plot. The reader is encouraged to adopt the plot accordingly. What has already been concluded from the density becomes even more evident when the QQ plot is examined.

R code 6.1 Fitting HPW returns to the GHD.

```

library(ghyp)                                1
library(timeSeries)                           2
library(fBasics)                            3
## Return calculation                      4
data(DowJones30)                            5
y <- timeSeries(DowJones30[, "HWP"], charvec = 6
                  as.character(DowJones30[, 1]))
yret <- na.omit(diff(log(y)) * 100)          7
## Fitting                                    8
ef <- density(yret)                         9
ghdfit <- fit.ghypuv(yret, symmetric = FALSE, 10
                      control = list(maxit = 1000)) 11
hypfit <- fit.hypuv(yret, symmetric = FALSE, 12
                      control = list(maxit = 1000)) 13
nigfit <- fit.NIGuv(yret, symmetric = FALSE, 14
                      control = list(maxit = 1000)) 15
## Densities                                 16
ghddens <- dghyp(ef$x, ghdfit)              17
hypdens <- dghyp(ef$x, hypfit)               18
nigdens <- dghyp(ef$x, nigfit)               19
nordens <- dnorm(ef$x, mean = mean(yret), sd = 20
                  sd(c(yret[, 1])))
col.def <- c("black", "red", "blue", "green", "orange") 21
plot(ef, xlab = "", ylab = expression(f(x)), ylim = c(0, 0.25)) 22
lines(ef$x, ghddens, col = "red")            23
lines(ef$x, hypdens, col = "blue")            24
lines(ef$x, nigdens, col = "green")           25
lines(ef$x, nordens, col = "orange")          26
legend("topleft", 27
       legend = c("empirical", "GHD", "HYP", "NIG", "NORM"), 28
       col = col.def, lty = 1)                      29
## QQ-Plots                                    30
qqghyp(ghdfit, line = TRUE, ghyp.col = "red", 31
        plot.legend = FALSE, gaussian = FALSE, 32
        main = "", cex = 0.8)                      33
qqhyp(hypfit, add = TRUE, ghyp.pch = 2, ghyp.col = "blue", 34
       gaussian = FALSE, line = FALSE, cex = 0.8) 35
qqhyp(nigfit, add = TRUE, ghyp.pch = 3, ghyp.col = "green", 36
       gaussian = FALSE, line = FALSE, cex = 0.8) 37
legend("topleft", legend = c("GHD", "HYP", "NIG"), 38
       col = col.def[-c(1,5)], pch = 1:3)        39
## Diagnostics                                40
AIC <- stepAIC.ghyp(yret, dist = c("ghyp", "hyp", "NIG"), 41
                      symmetric = FALSE, 42
                      control = list(maxit = 1000)) 43
LRghdnig <- lik.ratio.test(ghdfit, nigfit)    44
LRghdhyp <- lik.ratio.test(ghdfit, hypfit)    45

```

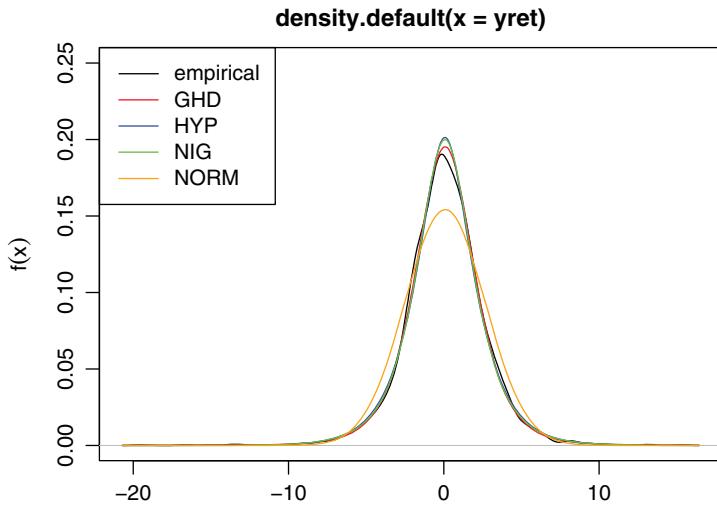


Figure 6.4 Fitted densities for HWP returns.

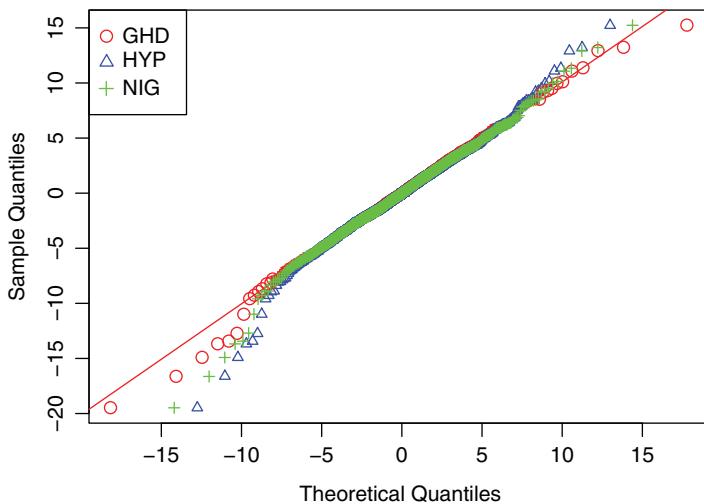


Figure 6.5 QQ plot of fitted GHD for HWP returns.

The daily returns can be tracked better with the GHD than with the HYP and NIG distributions, especially in the tails. Furthermore—this conclusion was less clear from the density plot—the returns can be slightly better explained by the NIG than by the HYP distribution.

In the last three lines of the listing, diagnostic measures for the three models are produced. First, the function `stepAIC.ghyp()` is utilized to determine with which distribution the data can be explained best in terms of the AIC. This function returns a

Table 6.2 Results of distributions fitted to HWP returns.

Distribution	AIC	LLH	λ	$\bar{\alpha}$	μ	σ	γ
GHD	11684.07	-5837.04	-2.270	0.051	0.093	2.584	-0.013
NIG	11691.90	-5841.95	-0.500	1.110	0.078	2.552	0.003
HYP	11704.41	-5848.21	1.000	0.903	0.069	2.528	0.011

list object with three elements: `best.model`, `all.models`, and `fit.table`. The latter is of most interest because it not only provides information about the AICs and the values of the log-likelihood (LLH), but also returns the estimates of the distribution parameters, whether a symmetric distribution has been fitted or not, whether the optimizer achieved convergence, and the number of iterations required. An excerpt from these results is provided in Table 6.2.

The conclusions drawn from the graphical inspection of the results are mirrored by their quantitative counterparts. Clearly, a GHD-based model is favored over the NIG and HYP distributions according to the AIC. However, the differences between the AIC and/or the log-likelihood of the GHD and NIG are rather small. A cross-comparison to the values of the HYP model would yield a preference for the NIG, if one had to choose between the restricted distributions. The reason for this is primarily that the unrestricted estimate of $\hat{\lambda}$ is -2.27 closer to the parameter restriction for λ of the NIG than that of the HYP. Whether the differences in the values for the log-likelihoods are significantly different from zero can be tested by means of a likelihood ratio test. These tests are carried out in the last two lines of the R code listing. First, it is checked whether the GHD can be replaced by the NIG. The value of the test statistic is 0.007 and the p -value is 0.002. Hence, the null hypothesis that the explanatory power of the two distributions is equal must be rejected at a confidence level of 95%. The corresponding value of the test statistic for comparing the GHD with the HYP is 0 and the implied p -value is 0. Here, the null hypothesis must be rejected even more clearly, which is plausible given the ordering of the three log-likelihood values.

6.6.2 Risk assessment with the GHD

In this subsection the behavior of the VaR and ES risk measures according to each of the models is investigated. In particular, the two risks measures are derived from the fitted GHD, HYP, and NIG distributions for the HWP returns from the previous subsection. These measures are calculated over a span from the 95.0% to the 99.9% levels. The resulting trajectories of the VaR and ES are then compared to their empirical counterparts. For the ES the mean of the lower quintile values is used.

The relevant code is provided in Listing 6.2. First, the sequence of probabilities is created for which the VaR and ES are to be computed. Because we are dealing with returns instead of losses, these are defined for the left tail of the distribution. In the next lines the VaR for these levels is computed by utilizing the quantile function for the GHD. By convention, losses are expressed as positive numbers, and hence the

R code 6.2 VaR and ES derived from the GHD, HYP, and NIG.

```

## Probabilities
p <- seq(0.001, 0.05, 0.001)
## VaR
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49

ghd.VaR <- abs(qghyp(p, ghdfit))
hyp.VaR <- abs(qghyp(p, hypfit))
nig.VaR <- abs(qghyp(p, nigfit))
nor.VaR <- abs(qnorm(p, mean = mean(yret),
sd = sd(c(yret[, 1]))))
emp.VaR <- abs(quantile(x = yret, probs = p))
# Plot of VaR
plot(emp.VaR, type = "l", xlab = "", ylab = "VaR",
axes = FALSE, ylim = range(c(hyp.VaR, nig.VaR, ghd.VaR,
nor.VaR, emp.VaR)))
box()
axis(1, at = seq(along = p), labels = names(emp.VaR),
tick = FALSE)
axis(2, at = pretty(range(emp.VaR, ghd.VaR, hyp.VaR,
nig.VaR, nor.VaR)))
lines(seq(along = p), ghd.VaR, col = "red")
lines(seq(along = p), hyp.VaR, col = "blue")
lines(seq(along = p), nig.VaR, col = "green")
lines(seq(along = p), nor.VaR, col = "orange")
legend("topright",
legend = c("Empirical", "GHD", "HYP", "NIG", "Normal"),
col = col.def, lty = 1)
## ES
ghd.ES <- abs(ESghyp(p, ghdfit))
hyp.ES <- abs(ESghyp(p, hypfit))
nig.ES <- abs(ESghyp(p, nigfit))
nor.ES <- abs(mean(yret) - sd(c(yret[, 1])) *
dnorm(qnorm(1 - p)) / p)
obs.p <- ceiling(p * length(yret))
emp.ES <- sapply(obs.p, function(x) abs(mean(sort(c(yret))[
1:x])))
## Plot of ES
plot(emp.ES, type = "l", xlab = "", ylab = "ES", axes = FALSE,
ylim = range(c(hyp.ES, nig.ES, ghd.ES, nor.ES, emp.ES)))
box()
axis(1, at = 1:length(p), labels = names(emp.VaR),
tick = FALSE)
axis(2, at = pretty(range(emp.ES, ghd.ES, hyp.ES, nig.ES,
nor.ES)))
lines(1:length(p), ghd.ES, col = "red")
lines(1:length(p), hyp.ES, col = "blue")
lines(1:length(p), nig.ES, col = "green")
lines(1:length(p), nor.ES, col = "orange")
legend("topright",
legend = c("Empirical", "GHD", "HYP", "NIG", "Normal"),
col = col.def, lty = 1)

```

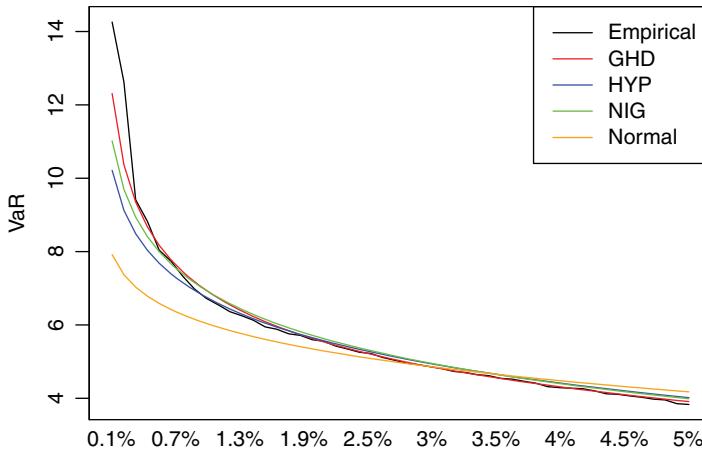


Figure 6.6 Progression of VaR based upon GHD, HYP, and NIG models.

absolute values of the quantiles returned by the function are used. The VaR based on the normal distribution can be computed by providing the necessary estimates for the location and scale. The VaR values thus determined are compared to their empirical counterparts, which are determined by the `quantile()` function.

The development of these risk measures is plotted in Figure 6.6. The quantiles derived from the GHD and its special cases track the associated empirical loss levels fairly closely. Only in the extreme confidence region of 99.0% or greater is the risk slightly underestimated by these models. The ordering of the goodness of fit for the three distributions can also be concluded from this graph: the fitted GHD tracks the data in that region very well, whereas the risk is underestimated by the NIG and HYP models. Two conclusions can be drawn from the VaR based on the normal distribution. First, as expected, the normal distribution falls short of capturing extreme risk events (i.e., above the 97.5% level). Second, the riskiness of holding a position in the HWP stock is overestimated for the confidence region between 95.0% and 97.5%. Put differently, for these levels the VaR derived from the normal distribution is consistently too conservative and hence an investor could be disadvantaged by not being allowed to take a larger position in that stock.

Next in Listing 6.2, the ES is calculated for the fitted model objects. As mentioned in Section 6.4.3, this risk measure can be computed with the function `ESghyp()`. The expected shortfall for the normal distribution is determined by (4.5) in Section 4.2. Similarly to the VaR, the trajectory of the ES for alternative confidence levels is compared to its empirical counterpart. Here, the mean of the values smaller than the quantile is employed. The risk measures thus determined are shown in Figure 6.7. In contrast to the VaR, now the risk measures derived from all models consistently underestimate the expected loss in the case of such an event. However, this underestimation is less severe for the GHD-based models and for the less conservative levels. The ES derived from the normal distribution fares worst. Overall, the reason why

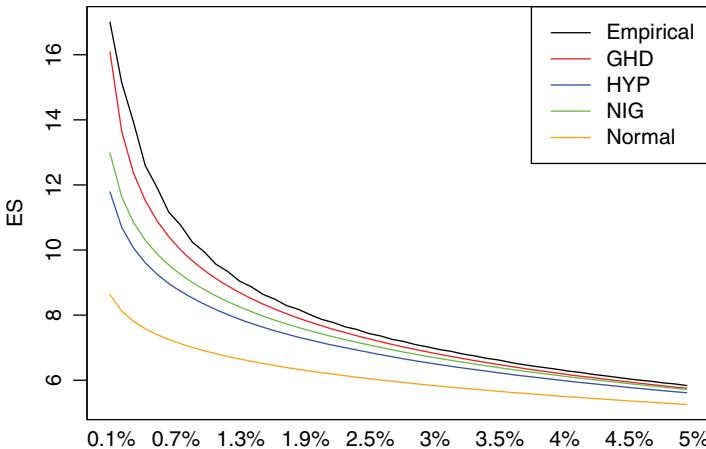


Figure 6.7 ES trajectory based upon GHD, HYP, and NIG models.

the risk is consistently underestimated, regardless of the model chosen, is primarily that all distributions underestimate the probability of extreme losses and hence these errors accumulate when the ES is calculated.

6.6.3 Stylized facts revisited

In this subsection the stylized facts of financial returns are reconsidered by employing the shape triangle of the HYP distribution. Ordinarily, the empirical distribution of financial market returns is characterized by excess kurtosis and negative skewness. These characteristics are most pronounced for higher-frequency returns (i.e., intra-daily or daily). Therefore, the need arises to utilize models/distributions which acknowledge these stylized facts. One might also be interested in whether these more complicated (compared to the normal distribution) models are indeed necessary if the focus is shifted to lower-frequency return processes (i.e., weekly, monthly, or bi-monthly). As discussed in Section 6.2, the exponential, Laplace, left- or right-skewed hyperbolic, and normal distributions are contained within the HYP distribution as limit cases. Whether the HYP can be approximated by one of these distributions can be graphically inspected by the shape triangle. This tool can also be employed to assess whether the stylized facts are still applicable for lower-frequency return processes. This kind of analysis is conducted in the R code in Listing 6.3.

The first line of the listing defines the return days used. These correspond to daily, weekly, biweekly, monthly, and bimonthly returns. The next line computes the returns for these frequencies. This can be achieved most easily with the `lapply()` function. The resulting `list` object is then converted to a matrix and `NA` values are omitted. The package `ghd` does not provide a function for parameterizing the HYP in the (ξ, χ) space. Hence, in the subsequent lines the function `xichi()` is specified for this purpose. Then the unknown parameters of the HYP distribution are estimated by means of the function `fit.hypuv()` for each of the columns in

R code 6.3 Shape triangle for HYP distribution.

```

rd <- c(1, 5, 10, 20, 40)                                1
yrets <- na.omit(matrix(unlist(lapply(rd,                      2
  function(x) diff(log(y), lag = x))),                      3
  ncol = 5))                                              4
## Function for xi/chi coefficients                      5
xichi <- function(x){                                     6
  param <- coef(x, type = "alpha.delta")                7
  rho <- param[["beta"]]/param[["alpha"]]]               8
  zeta <- param[["delta"]]*sqrt(param[["alpha"]]^2 -      9
    param[["beta"]]^2)                                    10
  xi <- 1/sqrt(1+zeta)                                    11
  chi <- xi * rho                                       12
  result <- c(chi, xi)                                    13
  names(result) <- c("chi", "xi")                         14
  return(result)                                         15
}
## HYP Fitting                                         16
hypfits <- apply(yrets, 2, fit.hypuv, symmetric = FALSE) 17
points <- matrix(unlist(lapply(hypfits, xichi)),           18
  ncol = 2, byrow = TRUE)                                19
## Shape triangle                                         20
col.def <- c("black", "blue", "red", "green", "orange") 21
leg.def <- paste(rd, rep("day return", 5))              22
plot(points, ylim = c(-0.2, 1.2), xlim = c(-1.2, 1.2), 23
  col = col.def, pch = 16, ylab = expression(xi),        24
  xlab = expression(chi))                                25
lines(x = c(0, -1), y = c(0, 1))                         26
lines(x = c(0, 1), y = c(0, 1))                         27
lines(x = c(-1, 1), y = c(1, 1))                         28
legend("bottomright", legend = leg.def, col = col.def, pch = 16) 29
text(x = 0.0, y = 1.05, label = "Laplace", srt = 0)      30
text(x = -1.0, y = 1.05, label = "Exponential", srt = 0) 31
text(x = 1.0, y = 1.05, label = "Exponential", srt = 0) 32
text(x = 0.0, y = -0.1, label = "Normal", srt = 0)        33
text(x = -0.6, y = 0.5, label = "Hyperbolic, left skewed", 34
  srt = 302)
text(x = 0.6, y = 0.5, label = "Hyperbolic, right skewed", 35
  srt = 57)

```

`yrets` by employing the function `apply()`. The returned `list` object `hypfits`, which contains the fitted distributions, is then submitted to `lapply()` in order to extract the $(\hat{\xi}, \hat{\chi})$ pairs. These points are plotted in the final lines of the listing and the resulting shape triangle is provided in Figure 6.8.

A clear pattern can be detected from this shape triangle: the lower the frequency of the return, the more it approaches the south of the triangle, hence the HYP distribution could in principle be approximated by a normal distribution for these lower-frequency

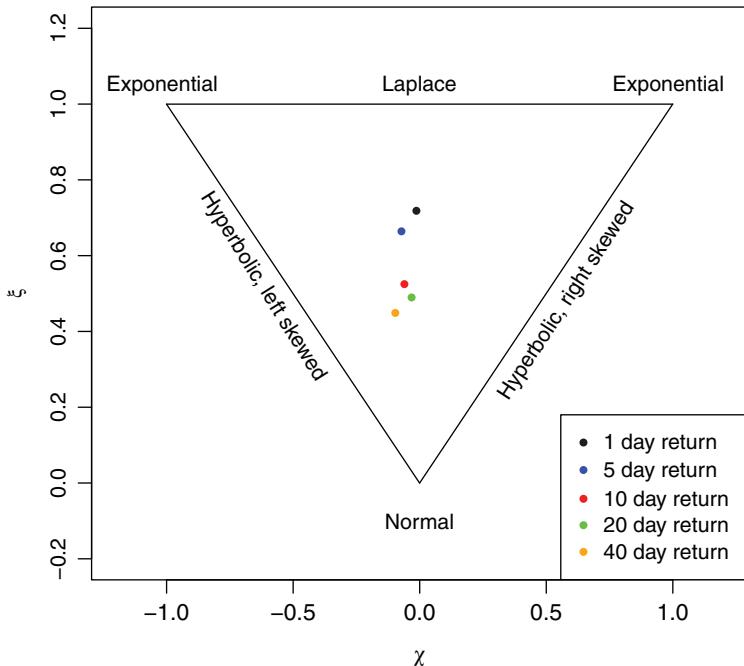


Figure 6.8 Shape triangle for HWP returns.

returns. Overall, however, the point cloud remains fairly close to the center of the triangle, so this approximation may not work well. Furthermore, a comparison of the $(\hat{\xi}, \hat{\chi})$ pairs indicates that the bimonthly returns show the greatest negative skewness in absolute terms. Surprisingly, even the weekly returns are more heavily skewed to the left than their daily counterparts.

6.7 Applications of the GLD to risk modelling and data analysis

6.7.1 VaR for a single stock

In the first application of the GLD a back-test is conducted for the 99% VaR of the weekly losses of the QCOM stock contained in the S&P 500 Index. The data is provided in the **R** package **FRAPO**. The sample covers the period from 2003 to 2008 and comprises 265 observations. The back-test is expressed in terms of the unconditional VaR implied by the GLD and the normal distribution. The **R** code for conducting the back-test is exhibited in Listing 6.4.

First, the necessary packages are loaded into memory. The fitting of the loss series to the GLD is accomplished with the functions of the package **lmomco** and, as stated above, the data set is contained in the package **FRAPO**. The `data.frame` object

R code 6.4 VaR of QCOM stock: comparison of GLD and normal distribution.

```

## Loading of packages
library(lmomco)                                     1
library(FRAPO)                                      2
## Data loading
data(SP500)                                         3
Idx <- SP500[, "QCOM"]                             4
L <- -1 * returnseries(Idx, method = "discrete", trim = TRUE) 5
## Computing VaR (Normal & GLD) 99%, moving window
ep <- 104:length(L)                                6
sp <- 1:length(ep)                                 7
level <- 0.99                                       8
VaR <- matrix(NA, ncol = 2, nrow = length(ep))    9
for(i in 1:length(sp)){                           10
  x <- L[sp[i]:ep[i]]                            11
  lmom <- lmom.ub(x)                            12
  fit <- pargld(lmom)                           13
  VaRGld <- quagld(level, fit)                  14
  VaRNor <- qnorm(level, mean(x), sd(x))       15
  VaR[i, ] <- c(VaRGld, VaRNor)                16
  print(paste("Result for", ep[i], ":", VaRGld, "and", VaRNor)) 17
}
## Summarising results
Res <- cbind(L[105:length(L)], VaR[-nrow(VaR), ]) 18
colnames(Res) <- c("Loss", "VaRGld", "VaRNor")    19
## Plot of backtest results
plot(Res[, "Loss"], type = "p", xlab = "Time Index", 20
      ylab = "Losses in percent", pch = 19, cex = 0.5, 21
      ylim = c(-15, max(Res)))
abline(h = 0, col = "grey")                         22
lines(Res[, "VaRGld"], col = "blue", lwd = 2)       23
lines(Res[, "VaRNor"], col = "red", lwd = 2)        24
legend("bottomright", legend = c("Losses", "VaR GLD", 25
                                  "VaR Normal"), 26
       col = c("black", "blue", "red"), 27
       lty = c(NA, 1, 1), pch = c(19, NA, NA), bty = "n") 28

```

SP500 is loaded next and the weekly percentage losses of QCOM are stored in the object `L`. Then, the shape of the back-test is defined. A moving window approach with a window size of 104 observations, equivalent to a time span of two years, has been chosen. The objects `ep` and `sp` define the relevant end and start points, respectively. The 99% confident level is assigned to the object `level`. At line 12 a `matrix` object is initialized with row dimension equal to the number of back-testing periods and two columns in which the VaR of the GLD and the normal distributions will be written. One might argue that a `for` loop is not R-like, but to correct a common prejudice, as long as the proper memory slot of the object `VaR` is allotted before the

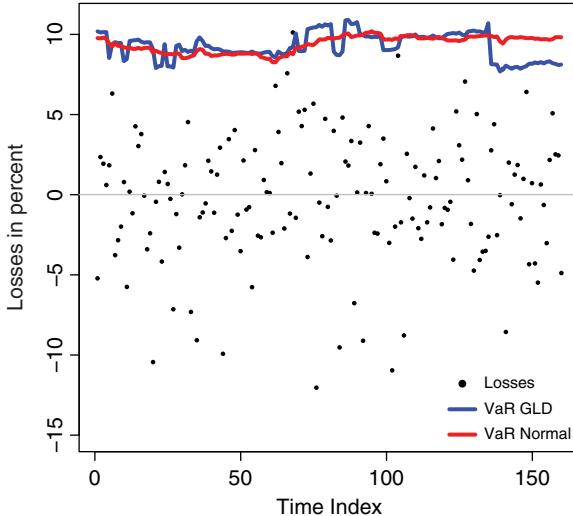


Figure 6.9 Back-test: VaR (99%) and losses for QCOM stock.

loop is executed, one is not penalized by a worse execution time. Within the loop, the relevant data window is extracted and fitted to the GLD, and the VaR measures are calculated for the two distributions and stored in the i th row of VaR . The last line in the loop informs the user about the progress of the loop execution by printing the time index and the values of the VaRs. The results are then aggregated into the object Res , which has the losses as the first column and the VaR measures lagged by one week in the next two columns. The trajectory of the two unconditional VaR measures is then plotted together with losses as points in the ensuing block of plot statements.

The plot is shown in Figure 6.9. The VaR trajectory according to the GLD model is more volatile than for the normal distribution. Notice the sharp drop of the GLD VaR toward the end of the back-test period, which goes hand in hand with a lower loss volatility, but also reflects the sensitivity of moment estimates with respect to outliers. During this episode, the VaR measures according to the normal distribution are too conservative. Both models violate the actual losses only once and hence the size of the risk measure at the 99% confidence level is not violated—that is, given 160 back-test observations one would expect at most two violations.

6.7.2 Shape triangle for FTSE 100 constituents

In the second example, the characteristics of FTSE 100 stocks are analyzed by means of a shape triangle for the standardized GLD as in (6.23)–(6.27). This kind of shape triangle was proposed in Chalabi et al. (2010) and applied to the constituent stocks of the NASDAQ 100. The shape triangle is depicted in the $\delta = \lambda_4 - \lambda_3$ and $\beta = \lambda_3 + \lambda_4$ plane. The R code is shown in Listing 6.5.

The robust estimation of the GLD parameters is covered in the package **fBasics** and the weekly price data of the FTSE 100 constituents is part of the package **FRAPO**,

R code 6.5 FTSE 100 stocks: shape triangle of standardized GLD.

```

library(FRAPO) 1
library(fBasics) 2
## Loading of data 3
data(INDTRACK3) 4
P <- INDTRACK3[, -1] 5
R <- returnseries(P, method = "discret", trim = TRUE) 6
## Fitting and calculating beta and lambda 7
Fit <- apply(R, 2, gldFit, method = "rob", doplot = FALSE, 8
  trace = FALSE)
DeltaBetaParam <- matrix(unlist(lapply(Fit, function(x){ 9
  l <- x@fit$estimate[c(3, 4)] 10
  res <- c(l[2] - l[1], l[1] + l[2]) 11
  res})), ncol = 2, byrow = TRUE) 12
## Shape triangle 13
plot(DeltaBetaParam, xlim = c(-2, 2), ylim = c(-2, 0), 14
  xlab = expression(delta == lambda[4] - lambda[3]), 15
  ylab = expression(beta == lambda[3] + lambda[4]), 16
  pch = 19, cex = 0.5) 17
segments(x0 = -2, y0 = -2, x1 = 0, y1 = 0, 18
  col = "grey", lwd = 0.8, lty = 2) 19
segments(x0 = 2, y0 = -2, x1 = 0, y1 = 0, 20
  col = "grey", lwd = 0.8, lty = 2) 21
segments(x0 = 0, y0 = -2, x1 = 0, y1 = 0, col = "blue", 22
  lwd = 0.8, lty = 2) 23
segments(x0 = -0.5, y0 = -0.5, x1 = 0.5, y1 = -0.5, 24
  col = "red", lwd = 0.8, lty = 2) 25
segments(x0 = -1.0, y0 = -1.0, x1 = 1.0, y1 = -1.0, 26
  col = "red", lwd = 0.8, lty = 2) 27

```

hence these two packages are loaded into the workspace first. Next, the data object `INDTRACK3` is loaded and its first column—representing the FTSE 100 index—is omitted from further analysis. The percentage returns of the stocks are assigned to the object `R`, which is then used to fit each of its columns to the GLD with the function `gldFit()`. This task is swiftly accomplished by utilizing the `apply()` function. The object `Fit` is a list with the returned objects of `gldFit`. In lines 10–13 a small function is defined that returns the (δ, β) parameter pairs, which are then plotted in the shape triangle. The output is shown in Figure 6.10.

The x -axis represents the difference between the right- and left-tail shape parameters, and the y -axis their sum. There are a total of six regions discernible in this triangle. The light gray dashed line discriminates between stocks that are characterized by either a left-skewed or a right-skewed distribution. Points on that line refer to a symmetric distribution. As can easily be seen, the majority of stock returns are characterized by being skewed to the left, thus confirming a stylized fact of financial returns. Points in the top part of the triangle represent return distributions with finite variance and kurtosis, and points in the middle region, between the -0.5 and -1.0

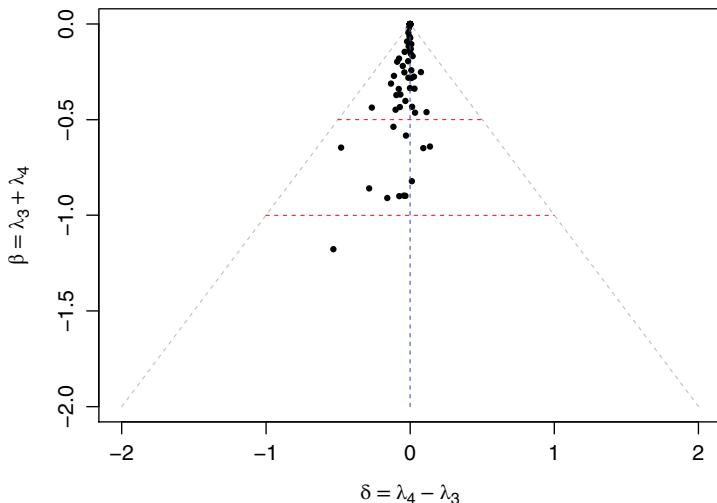


Figure 6.10 Shape triangle for FTSE 100 stock returns.

dark gray dashed lines, refer to return distributions with infinite kurtosis but finite variance. The (δ, β) pairs below the -1 line represent return processes where these moments are infinite, which is the case for one of the FTSE 100 constituents stocks.

References

- Asquith W. 2016 *lmomco—L-moments, censored L-moments, trimmed L-moments, L-comoments, and many distributions*. R package version 2.2.2.
- Barndorff-Nielsen O. 1977 Exponential decreasing distributions for the logarithm of particle size *Proceedings of the Royal Society London A*, vol. 353, pp. 401–419.
- Barndorff-Nielsen O. 1997 Normal inverse Gaussian distributions and stochastic volatility modelling. *Scandinavian Journal of Statistics* **24**, 1–13.
- Barndorff-Nielsen O. 1998 Processes of normal inverse Gaussian type. *Finance & Stochastics* **2**, 41–68.
- Chalabi Y., Scott D., and Würtz D. 2010 The generalized lambda distribution as an alternative to model financial returns. Working paper, Eigenössische Technische Hochschule and University of Auckland, Zürich and Auckland.
- Chalabi Y., Scott D., and Würtz D. 2011 A more intuitive parameterization of the generalized lambda distribution R/Rmetrics Meielisalp Workshop, Lake Thune, Switzerland.
- Cheng R. and Amin N. 1983 Estimating parameters in continuous univariate distributions with a shifted origin. *Journal of the Royal Statistical Society, Series B (Methodological)* **45**(3), 394–403.
- Eberlein E. and Keller U. 1995 Hyperbolic distributions in finance. *Bernoulli* **1**, 281–299.
- Eberlein E. and Prause K. 1998 The generalized hyperbolic model: financial derivatives and risk measures FDM Preprint 56, University of Freiburg.

- Freimer M., Mudholkar G., Kollia G., and Lin C. 1988 A study of the generalised Tukey lambda family. *Communications in Statistics – Theory and Methods* **17**, 3547–3567.
- Gilchrist W. 2000 *Statistical Modelling with Quantile Functions*. Chapman & Hall, London.
- Hankin R. and Lee A. 2006 A new family of non-negative distributions. *Australia and New Zealand Journal of Statistics* **48**, 67–78.
- Karian Z. and Dudewicz E. 1999 Fitting the generalized lambda distribution to data: a method based on percentiles. *Communications in Statistics – Simulation and Computation* **28**(3), 793–819.
- Karian Z. and Dudewicz E. 2000 *Fitting Statistical Distributions: The Generalized Lambda Distribution and Generalised Bootstrap Methods*. Chapman & Hall, New York.
- Karian Z., Dudewicz E., and McDonald P. 1996 The extended generalized lambda distribution system for fitting distributions to data: history, completion of theory, tables, applications, the ‘final word’ on moment fits. *Communications in Statistics – Simulation and Computation* **25**, 611–642.
- Kim T. and White H. 2004 On more robust estimation of skewness and kurtosis. *Finance Research Letters* **1**(1), 56–73.
- King R. and MacGillivray H. 1999 A starship estimation method for the generalised lambda distributions. *Australia and New Zealand Journal of Statistics* **41**(3), 353–374.
- King R., Dean B., and Klinke S. 2016 gld: Estimation and use of the generalised (Tukey) lambda distribution. R package version 2.3.3.
- Luethi D. and Breymann W. 2013 ghyp: A package on the generalized hyperbolic distribution and its special cases. R package version 1.5.6.
- McNeil A. and Ulman S. 2011 QRMLib: Provides R-language code to examine Quantitative Risk Management concepts. R package version 1.4.5.1.
- McNeil A., Frey R., and Embrechts P. 2005 *Quantitative Risk Management: Concepts, Techniques and Tools*. Princeton University Press, Princeton, NJ.
- Owen D. 1988 The starship. *Communications in Statistics – Simulation and Computation* **17**, 315–323.
- Pfaff B. and McNeil A. 2016 QRM: *Provides R-language Code to Examine Quantitative Risk Management Concepts*. R package version 0.4-13.
- Prause K. 1997 Modelling financial data using generalized hyperbolic distributions FDM Preprint 48, University of Freiburg.
- Prause K. 1999 How to use NIG laws to measure market risk FDM Preprint 65, University of Freiburg.
- Ramberg J. and Schmeiser B. 1974 An approximate method for generating asymmetric random variables. *Communications of the Association for Computing Machinery* **17**, 78–82.
- Ramberg J., Pandu R., Dudewicz E., and Mykytka E. 1979 A probability distribution and its uses in fitting data. *Technometrics* **21**(2), 201–214.
- Ranneby B. 1984 The maximum spacing method: An estimation method related to the maximum likelihood method. *Scandinavian Journal of Statistics* **11**(2), 93–112.
- Scott D. 2015 GeneralizedHyperbolic: *The Generalized Hyperbolic Distribution*. R package version 0.8-1.
- Scott D. and Dong C. Y. 2015 VarianceGamma: *The Variance Gamma Distribution*. R package version 0.3-1.

- Scott D. and Grimson F. 2015 *SkewHyperbolic: The Skew Hyperbolic Student t-Distribution*. R package version 0.3-2.
- Su S. 2005 A discretized approach to flexibly fit generalized lambda distributions to data. *Journal of Modern Applied Statistical Methods* **4**(2), 408–424.
- Su S. 2007 Fitting single and mixture of generalized lambda distributions to data via discretized and maximum likelihood methods: GLDEX in R. *Journal of Statistical Software* **21**(9), 1–17.
- Tukey J. 1962 The future of data analysis. *The Annals of Mathematical Statistics* **33**(1), 1–67.
- Würtz D., Setz T., and Chalabi Y. 2014 *fBasics: Rmetrics – Markets and Basic Statistics*. R package version 3011.87.

Extreme value theory

7.1 Preliminaries

Extreme value theory (EVT) is a branch of statistics. Its subject is the modelling of large deviations from the median of a distribution. EVT is by no means a young discipline—its roots can be traced back to the 1920s. However, interest in EVT and its application to modelling financial market risks has increased recently among practitioners; the reason for this development might well be the increasing number of episodes of financial market turmoil.

This chapter begins with the theoretical underpinnings of EVT. Three approaches to modelling extreme values are presented: the block maxima method, the peaks-over-threshold method, and Poisson processes. This exposition is by no means exhaustive, and its purpose is only to help the reader gain a better idea of how these methods are implemented in R. For a more thorough treatment of the subject the reader is referred to the monographs of Coles (2001), Embrechts et al. (1997), Leadbetter et al. (1983), and McNeil et al. (2005, Chapter 7).

Section 7.3 provides a synopsis of the currently available packages in R. Although there is quite an overlap between the various EVT models and their implementations in R, a comparison of the various packages reveals subtle differences, weaknesses, and strengths. Hopefully, the user will gain an insight into what is already implemented and how. Having done so, s/he can utilize the package(s) as desired. All packages presented are available on CRAN and can be swiftly installed by executing `install.packages ("foo")` for package `foo`.

The chapter concludes with some empirical applications of EVT, in which the various methods are applied to financial market data. The focus is on how EVT fares in contrast to the assumption of normally distributed losses.

7.2 Extreme value methods and models

7.2.1 The block maxima approach

The focus of EVT is on the modelling and inference of maxima. Assume that a sequence of iid random variables X_1, \dots, X_n over a time span of n periods is given. Ordinarily, the time span is a calendar period such as a month, quarter, half year, or year, and the observed data within this period is of daily frequency. With respect to EVT, the question arises as to which distribution the maximum of these random variables follows, or, to put it more precisely, is asymptotically best approximated by.

$$M_n = \max\{X_1, \dots, X_n\} \quad (7.1)$$

In principle, if the distribution function for the X_i is assumed to be known, then the distribution of M_n could be derived as:

$$\begin{aligned} \mathbb{P}\{M_n \leq z\} &= \mathbb{P}\{X_1 \leq z, \dots, X_n \leq z\} \\ &= \mathbb{P}\{X_1 \leq z\} \times \dots \times \mathbb{P}\{X_n \leq z\} \\ &= \{F(z)\}^n. \end{aligned} \quad (7.2)$$

In practice, this approach is not feasible for various reasons. First, the distribution function F is, in general, unknown. This could be rectified by either estimating a kernel density or assuming that the X_i are governed by a particular distribution. If this route is taken, the next obstacle is that estimation errors are raised to the power of n , leading to quite divergent results. An alternative route would be to seek a family of distributions F^n that can be used to approximate any kind of F . Therefore, the characteristics and properties of F^n for $n \rightarrow \infty$ need to be investigated. However, this asymptotic reasoning would imply that the values of the distribution function for z less than z_+ approach zero, whereby z_+ denotes the upper right point. Put differently, the mass of the distribution would not collapse over the point z_+ . This artifact can be circumvented by a linear transformation M_n^* of M_n :

$$M_n^* = \frac{M_n - b_n}{a_n}, \quad (7.3)$$

where $a_n > 0$ and b_n are sequences of constants. The purpose of these constants is to straighten out M_n , such that the probability mass would collapse over a single point. Under the assumption that the sequences a_n and b_n exist, it can be shown that the probability expression

$$\mathbb{P} \left\{ M_n^* = \frac{M_n - b_n}{a_n} \leq z \right\} \rightarrow G(z) \text{ for } n \rightarrow \infty \quad (7.4)$$

converges to a non-degenerate distribution $G(z)$, which belongs to one the following distribution families: Gumbel, Fréchet, or Weibull. All three distributions have a location and scale parameter. The Fréchet and Weibull distributions also have a shape

parameter. The three distributions can be subsumed into the generalized extreme value (GEV) distribution,

$$G(z) = \exp \left\{ -1 \left[1 + \xi \left(\frac{z - \mu}{\sigma} \right) \right]^{-1/\xi} \right\}. \quad (7.5)$$

The GEV is a three-parameter distribution where μ is the location, σ the scale, and ξ the shape parameter. For the limit $\xi \rightarrow 0$ the Gumbel distribution is obtained, for $\xi > 0$ the Fréchet, and for $\xi < 0$ the Weibull. The Weibull has a finite right point, whereas z_+ is infinity for the other two distributions. The density is exponential in the case of Gumbel and polynomial for the Fréchet distribution. Hence, the characteristics and properties of the GEV can be deduced from the value of the shape parameter.

7.2.2 The r th largest order models

A problem that arises when the block maxima method is applied to financial series is the lack of a sufficiently long data history. In these instances, the unknown distribution parameters will be estimated with greater uncertainty. To circumvent this problem the r th largest order model has been suggested. Not only is the maximal loss in a given block used as a data point for fitting the GEV, but also the r largest loss observations are employed. The data selection pertinent to the block maxima and the r largest orders is shown in Figure 7.1.

In this figure, 100 losses have been randomly generated and subdivided into 10 blocks of equal size. The maximum losses in each block are indicated by squares. These data points would be used in the block maxima approach. Further, the second largest losses in each of the 10 blocks are marked by triangles. The sets of these first- and second-order losses would be utilized if one were to fit a second largest order model. The variance of the parameter estimates will certainly be reduced by

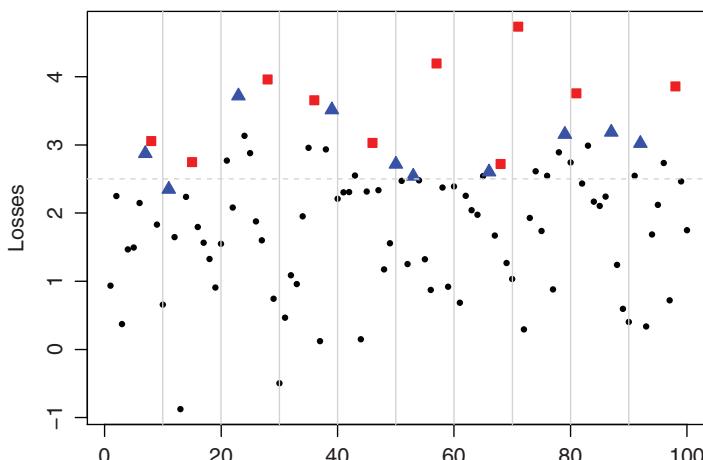


Figure 7.1 Block maxima, r largest orders, and peaks-over-threshold.

increasing the sample size. However, one runs the risk of having a biased sample in the sense that the second highest losses possibly would not qualify as extreme events or be considered as such. Finally, a dashed horizontal line has been drawn on the figure at the ordinate value of 2.5. This threshold value leads directly into the topic of the next subsection.

7.2.3 The peaks-over-threshold approach

With regard to the application of the block maxima method and the r th largest order model in the context of financial market data, the following issues arise:

- For a given financial instrument a sufficiently long data history is often not available. This will result in wide confidence bands for the unknown distribution parameters. As a consequence, the derived risk measures should be used with great caution.
- Not all observations that should be considered as extremes are exploited in estimating the distribution parameters, although this issue is ameliorated in the case of the r th largest order statistics. Hence, the available information about extreme events is not fully taken into account.
- Given the stylized facts for univariate return series, in particular volatility clustering, data points during tranquil periods could be selected as block maxima values, which they are not. Hence, an estimation bias would result in these instances.

Due to these issues the peaks-over-threshold (POT) method is more widely encountered when financial risks are of interest. As the name of this method suggests, neither the block maxima data points nor the r largest values within a block are considered as extreme observations, but rather all observations above a certain threshold value. This can be summarized for a given threshold u by the following probability expression:

$$P\{X > u + y | X > u\} = \frac{1 - F(u + y)}{1 - F(u)}, \quad y > 0. \quad (7.6)$$

It is evident from this equation that when the distribution function F is known, the expression can be solved for the exceedances $y > u$. In practice, however, the distribution function F is generally unknown, and hence, similarly to the derivation of the GEV, one needs an approximating distribution for sufficiently large threshold values. It can be shown that the exceedances $(X - u)$ are distributed according to the generalized Pareto distribution (GPD),

$$H(y) = 1 - \left(1 + \frac{\xi y}{\tilde{\sigma}}\right)^{-1/\xi}, \quad (7.7)$$

for $y : y > 0$ and $\tilde{\sigma} = \sigma + \xi(u - \mu)$. This means, that if the distribution of the block maxima for an iid sample can be approximated by the GEV, then the exceedances can

be approximated by the GPD for a sufficiently large threshold value u . By comparison of the GEV and GPD, it can be concluded that the shape parameter ξ is identical for both and hence independent of the chosen number of block maxima. It can then be deduced further that the distribution for $\xi < 0$ possesses an upper bound of $u - \tilde{\sigma}/\xi$ and is unbounded to the right for parameter values satisfying $\xi > 0$. At the limit of $\xi \rightarrow 0$ the GPD converges to an exponential distribution with parameter $1/\tilde{\sigma}$.

In practice, a difficulty arises when an adequate threshold has to be selected. If this value is chosen too small, a biased sample results. In this case observations would be selected which are too small, such that the GPD approximation would be violated. In contrast, if u is chosen too large, the sample size might become too small to yield reliable estimates for the unknown distribution parameters. Thus, a trade-off between a biased estimate and a large estimation error is required. An adequate threshold value can be determined graphically by means of a mean residual life (MRL) plot. This kind of plot is based on the expected value of the GPD: $E(Y) = \frac{\sigma}{1-\xi}$. For a given range of thresholds u_0 the conditional expected values

$$\begin{aligned} E(X - u_0 | X > u_0) &= \frac{\sigma_{u_0}}{1 - \xi} \\ &= \frac{\sigma_u}{1 - \xi} \\ &= \frac{\sigma_{u_0} + \xi u}{1 - \xi} \end{aligned} \quad (7.8)$$

are plotted against u . This equation is linear with respect to the threshold u :

$$\left\{ \left(u, \frac{1}{n_u} \sum_{i=1}^{n_u} (x_i - u) \right) : u < x_{\max} \right\}. \quad (7.9)$$

Hence, a suitable value for u is given when this line starts to become linear. Due to the central limit theorem (CLT), confidence bands can be calculated according to the normal distribution, owing to the fact that arithmetic means are depicted in an MRL plot.

Most often, the parameters of the GPD are estimated by applying the ML principle, but other methods, such as probability-weighted moments, are also encountered in empirical applications. The advantage of the former method is the possibility of plotting the profile log-likelihood for the shape parameter ξ and computing confidence bands based on this profile. In contrast to the bands derived from the normality assumption for the estimator, these are, in general, asymmetric. Hence, inference with respect to the sign of the shape parameter becomes more precise.

The VaR and ES risk measures can be inferred directly from the GPD as follows:

$$\text{VaR}_\alpha = q_\alpha(F) = u + \frac{\tilde{\sigma}}{\xi} \left(\left(\frac{1-\alpha}{\bar{F}(u)} \right)^{-\xi} - 1 \right), \quad (7.10)$$

$$ES_\alpha = \frac{1}{1-\alpha} \int_\alpha^1 q_x(F) dx = \frac{VaR_\alpha}{1-\xi} + \frac{\tilde{\sigma} - \xi u}{1-\xi}, \quad (7.11)$$

where $\bar{F}(u)$ denotes the number of non-exceedances relative to the sample size. Analogously to the confidence bands for the distribution parameters, intervals for these risk measures can also be derived.

7.3 Synopsis of R packages

7.3.1 The package evd

Although the packages are listed in alphabetical order in this section, another reason for beginning with the package **evd** (see Stephenson 2002) is that it has the longest history on CRAN. It was first published in February 2002. The package's name is an acronym for "Extreme Value Distributions." It is contained in the CRAN "Distributions," "Environmetrics," and "Finance" Task Views. Within the package, S3 methods and classes are implemented. The computationally burdensome algorithms are implemented as C routines.

The functions, methods, and data sets provided in this package cover basically all aspects of EVT. To cite from the package's description:

Extends simulation, distribution, quantile and density functions to univariate and multivariate parametric extreme value distributions, and provides fitting functions which calculate maximum likelihood estimates for univariate and bivariate maxima models, and for univariate and bivariate threshold models.

In particular, the following univariate extreme value distributions are implemented:

- extreme value distributions [`dqpr`] `extreme()`
- Fréchet distribution [`dqpr`] `frechet()`
- generalized extreme value distribution [`dqpr`] `gev()`
- generalized Pareto distribution [`dqpr`] `gpd()`
- Gumbel distribution [`dqpr`] `gumbel()`
- distributions of order statistics [`dqpr`] `order()`
- reversed Weibull distribution [`dqpr`] `rweibull()`.

For all these distributions the density, quantile, and distribution functions are implemented. Random number generation according to these distributions is also available. The `dqqr` naming convention referred to in Section 6.4.1 is followed: a prefix `d` is added for the values of the density function, `q` for the quantile function, `p` for

the distribution function, and `r` for generating random numbers from one of the above mentioned distributions. The ML method for estimating the unknown parameters of these distributions is used. The relevant functions are `fgev()`, `fpot()`, `forder()`, and `fextreme()`. Two models can be specified if the POT method is utilized, namely the generalized Pareto distribution or a Poisson point process. To aid the user's choice of an appropriate threshold value, the mean residual life plot (`mrlplot()`), the threshold choice plot (`tcpplot()`), and dependence measure plots (`chiplot()`) are implemented as preliminary diagnostic tools. Fitted model objects can be graphically inspected by the available `plot()` methods. In addition, `S3` methods for retrieving and plotting the profile log-likelihood are available. Confidence intervals are calculated for either fitted model objects or log-likelihood profiles with the `S3` method `confint()`. Finally, it can be checked whether nested models differ significantly from each other by means of a likelihood ratio test, which is implemented as an `anova()` method.

Apart from these univariate distributions, the package offers bivariate (`bvevd()`) and multivariate (`mvevd()`) extreme value distributions as well as non-parametric estimation methods (`abvnonpar()`, `amvnonpar()`, `qcbvnonpar()`). For bivariate extreme value distributions, parametric estimation is accomplished with the functions `fbvevd()` and `fbvpot()`. The parametric spectral density function of bivariate extreme value models can be computed and plotted with the function `hbvevd()`.

A distinctive feature of the package is the possibility of simulating extreme value stochastic processes. These can be specified as either a Markov chain (`evmc()`) or a maximum autoregressive moving average (`arma()`, `mar()`, `mma()`) process. Clusters of extreme values can be inspected with the function `clusters()`, and the extreme value index is implemented as function `exi()`.

The package is shipped with nine data sets that have been used extensively in the academic literature. A user guide in the form of a pdf file (“guide22.pdf”) within the `doc` subdirectory of the installed package is made available. Furthermore, the package does contain a vignette in which the figures, tables, and analysis contained in Chapter 9 of Beirlant et al. (2004) are replicated.

7.3.2 The package `evdbayes`

The package **evdbayes** provides functions and methods for the Bayesian analysis of extreme value models (see Stephenson and Ribatet 2014). Markov chain Monte Carlo (MCMC) techniques are utilized. Four different kinds of EVT models are implemented: the generalized extreme value distribution, the generalized Pareto distribution, order statistics, and the Poisson point process model. The likelihood functions, prior distributions, and Gibbs samplers (mixing and non-mixing) are interfaced from C routines. The latter routines are used for generating the Markov chains. The package is contained in the CRAN “Bayesian,” “Distributions,” “Environmetrics,” and “Finance” Task Views.

The main functions of the package can be split into four categories: generating prior distributions, generating Markov chains, local maximization of the prior/

posterior distribution, and diagnostic tools. First, the functions for specifying the prior distributions are named `prior.prob()`, `prior.quant()`, `prior.norm()`, and `prior.loglognorm()`. The user can choose the normal, beta, or gamma distribution as prior for the unknown model parameters. The functions return a list object of informal S3 class `evprior`. Next, a Markov chain must be generated with the function `posterior()`, with the previously generated `evprior` object as one of its arguments. The package's authors suggest the function `mcmc()` contained in the package `coda` (see Plummer et al. 2006) for inspecting the validity and well-behavedness of the generated Markov chain. The prior and posterior distributions are optimized with the function `mposterior()`. This function is a wrapper for the general-purpose optimizer `optim()` which is shipped with the base installation of R. Finally, two functions for graphical inspection of an EVD model are included: `rl.pst()` and `rl.pred()`. These functions depict the return levels of GEV quantiles and predictive distributions, respectively.

The `rainfall` data set is contained in the package, as well as user documentation. The latter is stored in the installation tree of the package in its `doc` subdirectory and named “`guide.pdf`” (see Stephenson and Ribatet 2006). Within this guide the theoretical underpinnings of Bayesian analysis in the context of EVT are outlined and the reader is guided through several empirical applications in which the usage of the functions provided is exhibited and discussed.

7.3.3 The package `evir`

The package `evir` has been ported from the S-PLUS package **EVIS** written by McNeil (1999) (see Pfaff and McNeil 2012). Its name is an acronym for “Extreme Values in R.” Because it was orphaned in 2008, the author of this book became the package's maintainer. In contrast to the two previously introduced packages, `evir` does not interface with any C routines, but is rather a self-contained collection of R functions that encompasses the most commonly encountered techniques in EVT. Within the package informal S3 classes and methods are employed. The package is contained in the CRAN “Distributions,” “Environmetrics,” and “Finance” Task Views.

The functions included can be loosely grouped into the following categories:

- exploratory data analysis (EDA)
- block maxima method
- peaks-over-thresholds (POT)
- bivariate POT
- POT with Poisson point processes
- generalized extreme value and generalized Pareto distributions.

Within the first group of functions, the user can investigate the data set of interest graphically by plotting the empirical distribution function (`emplot()`), the Hill

estimate of the tail index of heavy-tailed data (`hill()`), or the mean excesses over increasing thresholds (`meplot()`). Finally, QQ plots against the exponential distribution or the GPD can be created with the function `qplot()`. In addition to these graphical means of EDA the user can estimate an extremal index with the function `exindex()` and display the records of a data set against the expected behavior of an iid data set with the function `records()`.

The functions belonging to the second category, the block maxima method, are `gev()` and `gumbel()` for the generalized extreme value and the Gumbel distributions, respectively. The latter function is designed to help the user assess whether a Fréchet or Weibull distribution is more appropriate. Both functions return an object with class attribute `gev`. For these kinds of objects a plot and return level method are implemented.

The cornerstone function for applying the POT techniques is `gpd()`. A GPD can be fitted by either the ML or the PWM method. The function returns an object with class attribute `gpd`. A plot method exists for these fitted models. The user can choose from four different kind of plots: the excess distribution, the tail of the underlying distribution, a scatter-plot of the residuals, or a QQ plot of the residuals. The tail plot can be produced directly with the function `tailplot()`, bypassing the interactive plot menu. Furthermore, quantile estimates and confidence intervals for high quantiles above the threshold value can be superimposed on this plot with the function `gpd.q()`. In addition, an estimate of the expected shortfall can be included in the tail plot by utilizing the function `gpd.sfall()`. The responsiveness of the estimate for the shape parameter of the GPD with respect to the threshold values can be depicted with the function `shape()`. The relation of the estimate for a specified high quantile and the threshold or the number of extremes can be analyzed graphically with the function `quant()`. Both of these graphical displays can be used as a means of sensitivity analysis for the fitted GPD and thus help assess whether an appropriate threshold value or number of exceedances has been set. Finally, the function `riskmeasures()` returns a point estimate of a quantile for a given probability. Hence, this function can be swiftly employed to extract the value at risk of a fitted GPD object.

Similarly to the package **evd**, bivariate GPD can be fitted with **evir**. The function is termed `gpdbiv()` and the user can specify either the threshold values or the number of extreme observations. In addition, the user can choose to conduct a global or a two-stage optimization. With the latter optimization the marginal distributions are fitted first and, based thereon, the dependence parameter is determined in a second step. The function returns an object with class attribute `gpdbiv`. A `plot()` method exists for this kind of object. Similarly to the implemented `plot()` method for objects of class `gpd`, one can choose from five different plot types. Plots for the threshold exceedances, a contour plot of the fitted bivariate distribution function, a contour plot of the fitted joint survival function, and plots of the marginal distribution are all implemented. Finally, the `interpret()` method has been defined for calculating joint and conditional probabilities.

Turning to the fifth category, the user can estimate a Poisson point process with the function `pot()`. This function returns an object of informal class `potd` for which

a `plot()` method is defined. Eight different kinds of plot are available within this method. First, the exceedances of the point process can be plotted. Whether the gaps of this process are well-behaved can be inspected by either a scatter-plot of the gaps or a QQ plot, as well as the autocorrelation function thereof. In addition, the residuals of the fitted model object can be graphically analyzed in the form of a scatter-plot, a QQ plot, and their autocorrelation function. Finally, the user can employ the same `plot()` methods that are available for fitted GPD models.

Within the final category, functions for the GEV distribution and the GPD are subsumed. The usual `dpqr` naming convention for the various functions and for random number generation is followed. Hence, the relevant functions for the GEV are named `dgev()`, `qgev()`, `pgev()`, and `rgev()`, and for the GPD the corresponding functions are termed `dgpd()`, `qgpd()`, `pgpd()`, and `rgpd()`.

Seven data sets from the fields of hydrology (`nidd.annual` and `nidd.thresh`), insurance (`danish`), and finance (`bmw`, `siemens`, `sp.raw`, and `spto87`) are included in the package. Furthermore, `demo` files are provided to help the user understand how to apply the functions contained in the package.

7.3.4 The packages `extRemes` and `in2extRemes`

Since the first edition of this book, the package `extRemes` has received a major redesign (see Gilleland and Katz 2011). Within versions 1.65 and earlier a `Tcl/Tk` graphical user interface (GUI) for the package `ismev` (see Heffernan and Stephenson 2016) has been implemented. These versions were mainly intended to illustrate the application of EVT for pedagogical purposes. A GUI for this kind of analysis is now made available in a separate package `in2extRemes` (see Gilleland and Katz 2011), but now the underlying functions are taken from `extRemes` version 2.0-0 and above.

The package utilizes `S3` classes and methods and is almost purely written in `R`.¹ A total of 15 data sets—all originating from the fields of meteorology and hydrology—are included in the package to illustrate extreme value analysis.

The cornerstone function for fitting an extreme value distribution to data is `fevd()`. The implemented distribution models are the generalized extreme value, the Gumbel, the generalized Pareto, and the exponential distributions. The fitting of a point process model is available, too. The fit can be accomplished to either block maxima data or the threshold excesses. As estimation methods the user can chose between ML principles and/or Bayesian or L-moment methods. The function returns a `list` object with class attribute `fevd`. Functions for printing, plotting, and/or producing a summary are made available as methods `print()`, `plot()`, and `summary()`, respectively. The parameter estimates of an `fevd` object are returned by calling the `distill()` method. Confidence intervals for the parameters can be retrieved with `ci()`. Furthermore, probabilities and random draws from fitted EVD objects are returned by the functions `pextRemes()` and

¹ The function `abba()` interfaces with a C routine for fitting spatial extreme value models by means of the MCMC method.

`rextRemes()`, respectively. For the generalized extreme value and the generalized Pareto distributions, the values for the density, the quantiles, and the probabilities are returned by the functions `devd()`, `qevd()`, and `pevd()`, respectively. Random draws from these distributions are generated by invoking `revd()`.

More information can be found on the package's website, <http://www.assessment.ucar.edu/toolkit/>. In particular, a detailed tutorial on this package is available.

7.3.5 The package **fExtremes**

The package **fExtremes** is part of the Rmetrics suite of packages (see Würtz 2013). In contrast to the packages presented above, formal S4 classes and methods are utilized and, like **evir**, **fExtremes** is written purely in R. The package is considered a core package in the CRAN “Finance” Task View and is listed in the “Distributions” Task View. It has dependencies on other packages contained in the Rmetrics suite. Furthermore, **fExtremes** is endowed with the unit testing framework RUnit (see Burger et al. 2015). Quite a few functions have been directly ported and/or included from the packages **evd**, **evir**, **extRemes**, and **ismev**. The latter package is presented in the next subsection.

The functions and methods contained in **fExtremes** can be grouped loosely into five groups: data preprocessing, EDA, the GEV distribution, the GPD, and the calculation of risk measures. The `bmw` and `danish` data sets from **evir** are also contained in **fExtremes** as objects `bmwRet` and `danishClaims`, respectively.

The functions `blockMaxima()`, `findThreshold()`, `pointProcess()`, and `deCluster()` are provided for data preprocessing. The first function can be used for extracting the block maximum values that could then be used for fitting a GEV distribution. The next two functions can be used when POT techniques are employed. Within the function `findThreshold()` the number of exceedances must be specified and the function returns the values that exceed the threshold in accordance with this number. The function `pointProcess()` also returns exceedances, but here the user specifies an upper quantile. Finally, the function `deCluster()` can be used when a Poisson point process is to be fitted and the extreme observations occur closely with respect to time (e.g., volatility clustering).

Like the package **evir**, functions and methods for exploratory data analysis are also contained in this package. These can be split into functions with which the data can be analyzed for the occurrence of extremal events as well as tools for visual inspection. The functions `blockTheta()`, `clusterTheta()`, `runTheta()`, and `ferrosegersTheta()` belong to the former subgroup. The last of these, for calculating the extremal index, was proposed by Ferro and Segers (2003). All of these functions return an object of S4 class `fTHETA` for which a `show()` method is defined. The functions `exindexesPlot()` and `exindexPlot()` for graphically displaying extremal indexes are also available. It is further possible to simulate a series for given value of the extremal index with the function `thetaSim()`.

Similar to the package **evir**, functions for plotting the empirical distribution function (`emdPlot()`), producing a QQ plot against the GPD or exponential

distribution (`qqparetoPlot()`), displaying the mean excesses (`mePlot()` and `mxpPlot()`), as well as the development of records (`recordsPlot()` and `ssrecordsPlot()`), are available in **fExtremes**. In addition, the user can graph the mean residual lives (`mrlPlot()`), the ratio of maximum over sum (`msratioPlot()`), and the fits of the mean excesses according to either the normal (`normMeanExcessFit()`) or the generalized hyperbolic distribution (`ghMeanExcessFit()`) and its derived distributions, namely the normal inverse Gaussian (`nigMeanExcessFit()`) and the hyperbolic distribution (`hypMeanExcessFit()`). The ACF of the exceedances as well as their distance can be plotted with the function `xacfPlot()`. Finally, Kolmogorov's strong law of large numbers and Khinchine's law of the iterated logarithm can be verified with the functions `s1lnPlot()` and `l1lPlot()`, respectively.

The third group of functions encompass the block maxima technique. The density, quantile, and distribution functions of the GEV and the random number generator are named as in the package **evir**, namely, `dgev()`, `qgev()`, `pgev()`, and `rgev()`, respectively. In addition, the true moments of the GEV distribution can be calculated with the function `gevMoments()` and either the density or random numbers can be displayed for combinations of the distribution parameters with the function `gevSlider()`. As the function's name implies, the user can specify the values for the size and the distribution parameters by a `Tcl/Tk` interface and the resultant GEV or random numbers are displayed immediately. The parameters of the GEV and the Gumbel distribution can be fitted to block maxima data by the functions `gevFit()` and `gumbelFit()`. Both functions return an object of `S4` class `fGEVFIT`. Here, the unknown parameters can be estimated by either the ML principle or the PWM method. There are `plot()`, `print()`, and `summary()` methods for such objects. A return level plot can be generated with the function `gevrlevelPlot()`. Furthermore, data sets that are distributed according to either the GEV or Gumbel distribution can be generated with the functions `gevSim()` and `gumbelSim()`, respectively. These functions are wrappers for `rgev()`, but the user can specify the seed of the random number generator as a functional argument. Finally, the shape parameter can be determined according to maximum domain of attraction (MDA) estimators. The relevant functions are `hillPlot()` and `shaparmPlot()` for the Hill and shape parameter plots, respectively.

The same basic naming convention is adhered to when the POT technique is applied to the GPD. The `dpqr` naming convention is followed, hence `dgp` denotes the density, `pgp` the distribution, and `qgp` the quantile function of the GPD. Random numbers can be generated with the function `rgp`. A wrapper for this function for simulating data according to the GPD is provided as `gpdSim()`. A function for retrieving the true moments (`gpdMoments()`) and the dynamic plotting facility (`gpdSlider()`) are also available. The parameters of the GPD can be estimated with the function `gpdFit()`. This can be accomplished by using either the ML or the PWM method. The resulting object is of formal class `fGPDFIT` for which `print()`, `plot()`, and `summary()` methods exist.

With respect to the calculation and visual inspection of tail risks the functions `tailRisk()`, `gpdRiskMeasures()`, `tailplot()`, and `gpdTailPlot()`

are available for fitted GPD models. In addition, the empirical VaR and ES can be computed with the functions `VaR()` and `CVaR()`.

7.3.6 The package `ismev`

The package **ismev** is the second oldest contributed package on CRAN (see Heffernan and Stephenson 2016). It is an **R** port of functions originally written in **S** by Janet E. Heffernan. The package is contained in the CRAN “Environmetrics” and “Finance” Task Views. It includes routines for replicating the computations and results in Coles (2001). It is shipped with 12 data sets originating in the fields of finance and environmetrics. In addition, demonstration files are available to help the user understand how to use the package.

The functions can be grouped loosely into five categories: GEV models, GPD models, Poisson point processes, order statistics models, and utility functions to guide the user in choosing threshold values for the POT method.

The functions for fitting block maxima models (i.e., the GEV and Gumbel distributions) are `gev.fit()` and `gum.fit()`, respectively. The unknown distribution parameters are estimated by applying the ML principle. Diagnostic testing of the appropriateness of the model can be checked graphically with the functions `gev.diag()` for the GEV model and `gum.diag()` for the Gumbel distribution. The profile log-likelihood function for the GEV distribution can be plotted with the functions `gev.prof()` and `gevprofxi()`. The former function produces a return level plot for m years per block. The latter function returns the profile with respect to the shape parameter only. Confidence bands can be superimposed on both kinds of graph.

Two kinds of model for the POT techniques are available: the GPD distribution and a point process model. The unknown model parameters are estimated with the functions `gpd.fit()` and `pp.fit()` by applying the ML principle. Similarly to the block maxima method, the functions for graphically inspecting the model’s adequacy are named `gpd.diag()` for the GPD and `pp.diag()` for the point process model. The profile log-likelihood can be plotted analogously to the GEV with the functions `gpd.prof()` and `gpd.profxi()`.

Three functions are intended to guide the user by choosing an appropriate threshold value when the POT technique is applied. First, an MRL plot can be generated with the function `mrl.plot()`. The two other functions fit the GPD or the point process model over a user-specified range of threshold values and depict the path of the shape parameter for the GPD and the location, scale, and shape parameters for the point process model.

Finally, an order statistics model can be fitted with the function `rlarg.fit()` and the outcome can be graphically investigated with the function `rlarg.diag()`.

7.3.7 The package `QRM`

The **QRM** package was introduced in Section 6.4.4. Its EVT facilities include functions for block maxima, POT, point processes, and the calculation of risk measures. In

addition, methods for graphically depicting the fitted model objects or aspects thereof are included.

The GEV and Gumbel distributions are implemented, and the naming convention for the density, distribution, quantile function, and the generation of random numbers is followed, giving, for example, `dGPD()` and `dGumbel()` for the respective densities. The fitting of a GEV model is accomplished with the function `fit.GEV()`. Here, the ML principle is applied and the optimization is conducted with the general purpose optimizer `optim()`. The function is specified with an ellipsis argument within `optim()`. This gives the user a choice of optimizer and provides further control of the numeric optimization.

With respect to the POT techniques, the values of the exceedances for a given number thereof can be retrieved with the function `findthreshold()`. It is further possible to plot the sample mean excesses for varying thresholds with the function `MEplot()` and/or to produce a Hill plot (`hillPlot()`). The latter function is the same as the one in the package `evir`. To estimate a GPD the routine `fit.GPD()` can be used. Here, the user can choose between the functions `optim()` and `n1minb()` for numerically determining the unknown coefficients. The fitted model can be investigated with the functions `plotTail()` and `plotFittedGPDvsEmpiricalExcesses()`. The trajectory of the shape parameter can be displayed for a range of threshold values with the function `xiplot()`.

The risk measures VaR and ES for models fitted to the GPD can be calculated with the function `RiskMeasures()` or `showRM()`. The latter function returns a tail plot with the value of the estimated risk measure and a confidence band. Incidentally, estimates for the lower and upper bounds of the VaR for a given set of marginal distributions can be computed with the function `VaRbound()` (see Embrechts et al. 2013).

The extremal point process data can be retrieved via the function `extremalPP()` and visually inspected as either a marked point process (`plot.MPP()`) or a point process (`plot.PP()`). Data objects of this kind can be fitted to a GPD with the function `fit.POT()`. Furthermore, it is possible to estimate self-exciting point processes with the routines `fit.sePP()` and `fit.seMPP()`.

7.3.8 The packages `Renext` and `RenextGUI`

The package `Renext` is the latest addition to the R packages on CRAN dealing explicitly with EVT (see Deville 2015a). It was first added in 2010. Within the package, functions for the “*méthode de renouvellement*”—a technique similar to the POT—are implemented. The major difference of this technique compared to the POT is the assumption that exceedance times follow a homogeneous Poisson process and are independent of the sizes thereof. Given the validity of these assumptions, the exceedances can be modelled not only with the GPD but also with other distributions, such as the exponential, Weibull, gamma, log-normal, mixtures of the exponential, shifted left-truncated Weibull, and the shifted Pareto. As stated in the user guide “`RenextGuide.pdf`” (included in the package’s subdirectory `doc`), this method is quite popular among French hydrologists. The package is written purely in R and

a NAMESPACE is implemented. Similar to **extRemes**, a GUI is contained in a companion package **RenextGUI** (see Deville 2015b); use of the GUI is explained in a user guide (see “RenextGUIDe.pdf” in the package’s `doc` subdirectory).

The functions contained in the package can be grouped mainly into three categories: EDA and estimation; inference and simulation of this kind of model; and utility functions.

Within the first category, plot functions for investigating whether the exceedances can be approximated by the exponential or Weibull distribution are named `expplot()` and `weibplot()`, respectively. The number of exceedances within specified time plots can be depicted as a bar-plot with the function `barplotRenou()`.

The function for fitting renewal models is termed `Renou()` and that for generating random numbers from this kind of model is named `rRenou()`. The unknown parameters of the specified distribution are estimated according to the ML principle using the function `optim()`. By default, the data will be fitted to the exponential distribution, though the package also contains the relevant functions for fitting the above-listed distributions. A return level plot is generated in addition to the estimation results. This plot can also be produced separately with the function `RLplot()`.

The functions for reading Rendata objects in a specific XML format (`readXML()`), the plotting thereof (`plot.Rendata()`), the computation of parameter values from theoretical moments (`mom2par()`), and goodness-of-fit tests for the repartition of dates and the exponential distribution (functions `gof.date()` and `gofExp.test()`, respectively) can be grouped into the final category.

7.4 Empirical applications of EVT

7.4.1 Section outline

Extreme value theory will now be applied to financial data, showing how the methods and techniques presented above can be applied to the modelling of market risks. In particular, extreme value models using the block maxima, POT, and point process techniques are fitted to daily stock returns.

The empirical applications begin with an exploratory data analysis of the Siemens stock returns. This series will be used subsequently for applying the block maxima method to this data set. In the following examples the *r*-block maxima approach is applied to losses of BMW stock and the POT to losses of Boeing stock. In the last example the sensitivity of the declustering technique is investigated for losses of the NYSE index. These examples will show how the different implementations in the packages **evir**, **ismev**, and **fExtremes** can be utilized.

7.4.2 Block maxima model for Siemens

In this subsection the block maxima method is applied to the daily losses of Siemens stock. This data set was introduced in Section 3.1. The fitting of the GEV to this data set, as well as inference and diagnostic tests, are shown in Listing 7.1.

R code 7.1 Block maxima for Siemens losses.

```

library(evir)                                     1
data(siemens)                                    2
## Losses                                         3
SieLoss <- -100.0 * siemens                   4
## package evir:
SieGEV <- gev(SieLoss, block = "semester")     5
SieGEV                                         6
plot(SieGEV$data, type = "h", col = "blue", xlab = "", 7
      ylab = "Block Maxima",                      8
      main = "Maximum Biannual Losses of Siemens") 9
## package ismev:
library(ismev)                                    10
SieGEV2 <- gev.fit(SieGEV$data)                 11
SieGEV2                                         12
gev.diag(SieGEV2)                                13
par(mfrow = c(2, 1))                            14
gev.prof(SieGEV2, m = 20, xlow = 5, xup = 16, conf = 0.95) 15
gev.profxi(SieGEV2, xlow = 0.0, xup = 0.7, conf = 0.95) 16
mLoss <- max(SieGEV$data)                      17
mYears <- 1 / (1 - pgev(mLoss, mu = SieGEV2$mle[1], 18
                         sigma = SieGEV2$mle[2], 19
                         xi = SieGEV2$mle[3])) / 2 20
## package fExtremes:
library(fExtremes)                             21
SieGEV3 <- gevFit(SieGEV$data, type = "pwm") 22
SieGEV3                                         23

```

First, the daily returns are converted to positive loss figures expressed as percentages. Next, this series is fitted against the GEV distribution by employing the function `gev()` of the package `evir`. The function's argument `block` has been set to "semester". This will extract the biannual maxima of the series if the series contains a `time` attribute with time stamps of class `POSIXct` or of a class that can be coerced to it. The data points extracted are shown in Figure 7.2. The finding of an increased volatility during the second half of the sample period is mirrored in this graph. Hence, the assumption of identically distributed block maxima might be violated. However, for the time being we will neglect this potential violation and address it again later, when a trend as a covariate is included in the model.

The ML estimation results are provided in Table 7.1. All coefficients are significantly different from zero. The estimate of the shape parameter, $\hat{\xi}$, implies the existence of heavy tails and non-finite losses—that is, the GEV is of the Fréchet type.

Alternatively, the block maxima data could be fitted by applying the ML principle to the GEV with the function `gev.fit()` of the package `ismev`. The parameter estimates are equal to those returned by `gev()`. Diagnostic plots can be swiftly produced with the function `gev.diag()`, as shown in Figure 7.3. As indicated by the

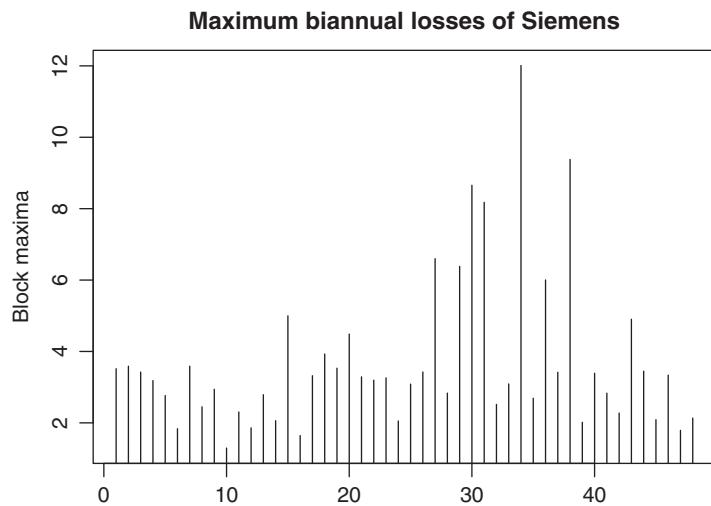


Figure 7.2 Block maxima for Siemens losses.

Table 7.1 Fitted GEV to block maxima of Siemens.

GEV	ξ	σ	μ
Estimate	0.287	1.047	2.700
Standard error	0.117	0.141	0.170

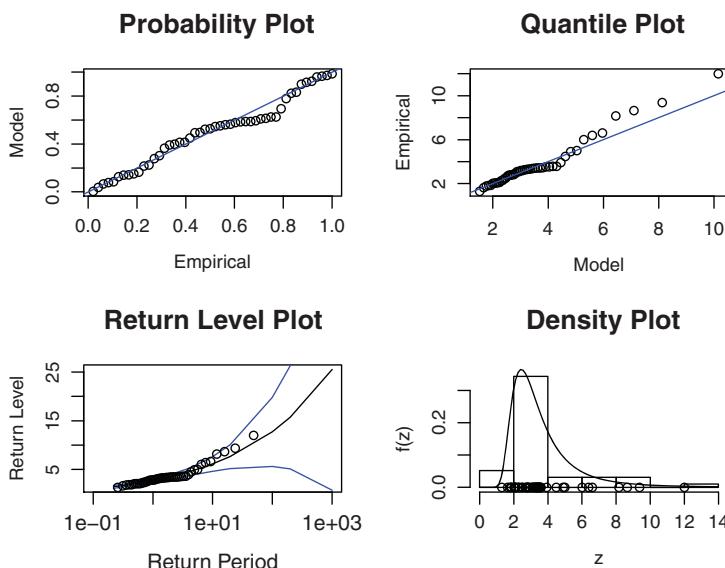


Figure 7.3 Diagnostic plots for fitted GEV model.

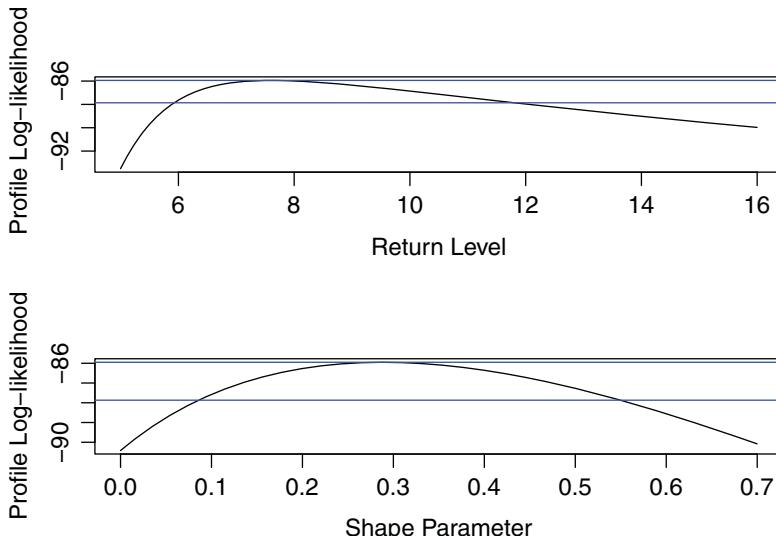


Figure 7.4 Profile log-likelihood plots for fitted GEV model.

probability and quantile plots, data points in the far right tail are not captured well by this model specification. This can be attributed to the relatively small losses witnessed during the beginning of the sample period. This artifact shows up in the return level plot. For data points in the far right tail the estimated return levels systematically fall short compared to the empirical levels, though the latter stayed within the 95% confidence bands.

In addition to these diagnostic plots, further inference from the model can be made using the profile log-likelihoods. Figure 7.4 shows these for a 10-year return level (upper panel) and for the shape parameter (lower panel). A daily loss as high as 7.6% would be observed once every 10 years. This point estimate falls within a 95% confidence level ranging from 6% to 11.75%. Hence, the maximum observed loss of 12.01% would not have been covered as a “once in every 10 years” event, but rather this loss would occur only once every 42 years or so. The relevant computation for this time span is included in Listing 7.1 (see lines 20–22). In the lower panel of Figure 7.4 the profile log-likelihood for the shape parameter is shown with a 95% confidence band (the horizontal light gray lines) superimposed. As can clearly be seen, the confidence band is asymmetric and to the right for the point estimate of $\hat{\xi} = 0.287$. A value of almost 0.6 would be covered by this confidence band.

In the next lines of the R code listing, the parameters of the GEV are determined by the PWM methods. The estimates are fairly close to the ML estimates. The shape, scale, and location parameters take the values $\hat{\xi} = 0.319$, $\hat{\sigma} = 2.675$, and $\hat{\mu} = 1.001$, respectively.

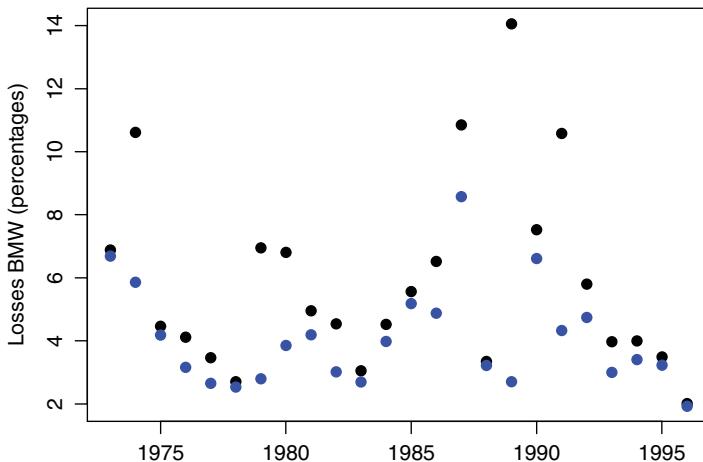


Figure 7.5 Two largest annual losses of BMW.

7.4.3 r -block maxima for BMW

We will now fit an r -block maxima model to the daily losses of BMW. Here, the two largest annual losses are extracted from the data set. The return series is contained in the package **evir** and the functions for fitting and diagnosing models of this type are provided in the package **ismev**, namely `rlarg.fit()` and `rlarg.diag()`, respectively. The time span and structure of the data object `bmw` is the same as for `siemens`.

In Listing 7.2 the above-mentioned packages are loaded first. Next, the percentage losses for BMW stock are computed as positive figures (i.e., object `BmwLoss`). The object `bmw` has a `time` attribute that is of class `POSIXct`. In order to derive the two largest annual losses of the series and prepare a `matrix` object suitable for fitting, a series of unique years is recovered from the `time` attribute by employing the `format()` method for objects of this kind. The resulting object is added as attribute `years` to the object `BmwLoss`. The unique years are stored as object `Yearu` and an index `idx` is created equal to the length of this item. Next, the order r is set to 2, and this is then used to retrieve the two largest losses per year. Here, the losses per year are sorted in decreasing order and the first r values are extracted. The resulting object `BmwOrder` is of class `matrix` and has a column dimension of 2 for the two largest observed losses and a row dimension of 24 which coincides with the annual time span from 1973 to 1996. The losses retrieved are plotted in Figure 7.5.

As is evident from the plot, the second largest observed loss is not in all years of similar magnitude to the largest annual loss. In particular, the differences are quite marked in the years 1989 and 1991, at 11.357 and 6.249 percentage points, respectively.

Having prepared the data in a format suitable for fitting, the GEV is fitted to the r -block maxima data. The estimation results are provided in Table 7.2. All estimates

R code 7.2 r -block maxima for BMW losses.

```

## Loading of packages
library(evir)
library(ismev)
## Order statistics
data(bmw)
BmwLoss <- -1.0 * bmw * 100
Years <- format(attr(BmwLoss, "time"), "%Y")
attr(BmwLoss, "years") <- Years
Yearu <- unique(Years)
idx <- 1:length(Yearu)
r <- 2
BmwOrder <- t(sapply(idx, function(x)
  head(sort(BmwLoss[ attr(BmwLoss, "years") ==
    Yearu[x]], decreasing =
  TRUE), r)))
rownames(BmwOrder) <- Yearu
colnames(BmwOrder) <- paste("r", 1:r, sep = "")
## Plot of order data
plot(Yearu, BmwOrder[, 1], col = "black",
  ylim = range(BmwOrder), ylab = "Losses BMW(percentages)",
  xlab = "", pch = 21, bg = "black")
points(Yearu, BmwOrder[, 2], col = "blue", pch = 23,
  bg = "blue")
## Fit and diagnostics
BmwOrderFit <- rlarg.fit(BmwOrder)
rlarg.diag(BmwOrderFit)

```

Table 7.2 Fitted r -block maxima of BMW.

GEV	μ	σ	ξ
Estimate	4.480	1.752	0.303
Standard error	0.341	0.289	0.159

are significantly different from zero, but for ξ only at the 10% level. The sign of the parameter value indicates a Fréchet-type distribution, which implies the potential occurrence of unbounded losses.

The goodness of fit can be inspected visually with the function `rlarg.diag()`, which returns two graphics. First, plots for the largest losses are produced similar to the function `gev.diag()`, and then PP and QQ plots for $r = 1, 2$ are returned. These diagnostic tools are shown in Figures 7.6 and 7.7, respectively.

Overall, the PP plot indicates a decent fit. However, the QQ plot reveals that losses in the range from roughly 8% to 10% are not covered so well. This is also reflected in the return level plot. Here, the values for a one-year return period are at the boundary

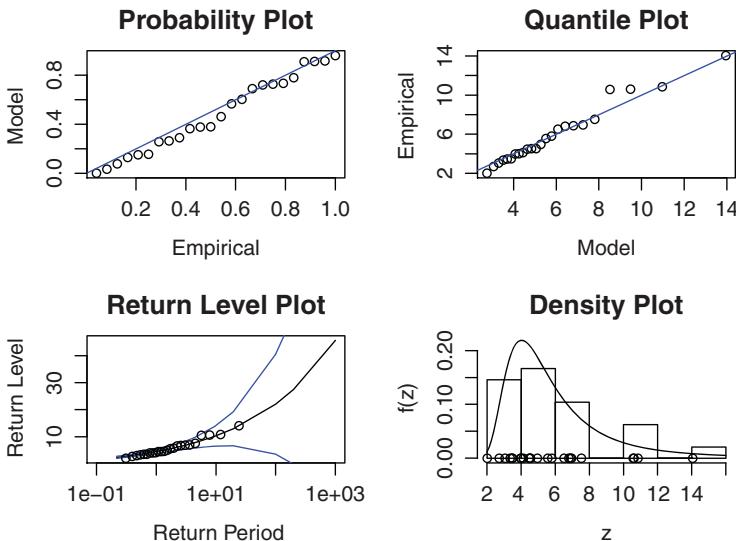


Figure 7.6 Diagnostic plots for r -block maxima: largest losses.

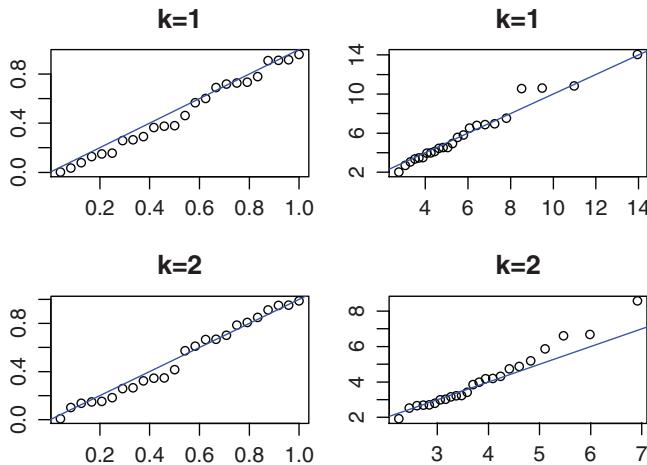


Figure 7.7 Diagnostic plots for r -block maxima: PP and QQ plots.

of the upper confidence band. The upward sloping return level curve reiterates that the GEV is of Fréchet type. The relatively large estimated standard error for the shape parameter is mirrored by quite sharply widening confidence bands for higher return periods, alerting the researcher to the vagaries of extrapolating too far into the tails of the distribution.

Figure 7.7 shows the PP and QQ plots for each of the two orders. As should be expected from the descriptive analysis of this data set, the results for $k = 2$ look less favorable in terms of the QQ plot (lower right-hand panel).

R code 7.3 POT GPD for Boeing losses.

```

## Loading of packages
library(fBasics)
library(fExtremes)
## Data handling
data(DowJones30)
DJ <- timeSeries(DowJones30[, -1],
                  charvec = as.character(DowJones30[, 1]))
BALoss <- -1.0 * returns(DJ[, "BA"], percentage = TRUE,
                         trim = TRUE)
## MRL-plot
mrlPlot(BALoss, umin = -10, umax = 10)
## GPD
BAFit <- gpdFit(BALoss, u = 3)
## Diagnostic plots
par(mfrow = c(2, 2))
plot(BAFit, which = 1)
plot(BAFit, which = 2)
plot(BAFit, which = 3)
plot(BAFit, which = 4)
## Risk measures
gpdRiskMeasures(BAFit, prob = c(0.95, 0.99, 0.995))

```

7.4.4 POT method for Boeing

In this subsection the GPD distribution is fitted to the daily losses of Boeing stock by utilizing the POT method. The closing prices for this stock are contained in the object `DowJones30` which is part of the `fBasics` package (see Würtz et al. 2014). The sample runs from 31 December 1990 to 2 January 2001 and comprises 2529 observations.

In Listing 7.3 the necessary packages are loaded first. The package `fExtremes` will be used to conduct the analysis. Next, the data set is loaded into the workspace and is converted to a `timeSeries` object. The daily losses of Boeing are computed and expressed as percentage figures. The resulting object has been termed `BALoss`.

To apply the POT method, a suitable threshold value must be determined. This choice can be guided by the MRL plot which is shown in Figure 7.8. In this graph 95% confidence bands around the mean excesses have been superimposed and are colored light gray. Unfortunately, and this is often encountered empirically, a definite choice for the threshold value can hardly be deduced from this kind of plot. However, it seems reasonable to assume a daily loss as high as 3% as a threshold value, given that a linear relationship exists between the plotted thresholds and the mean excesses above this value. In principle, a threshold slightly higher or lower than 3% could be chosen, but then there is a trade-off between greater uncertainty for the estimates and bias. For the given threshold a total of 102 exceedances result. This data set corresponds roughly to the upper 96% quantiles of the empirical distribution function.

Having fixed the threshold value, the GPD can be fitted to the exceedances. This is accomplished with the function `gpdFit()`. The estimates for the scale and shape parameters as well as their estimated standard deviations are shown in Table 7.3. The shape parameter is greater than zero and significantly different from one, indicating heavy tails.

In addition to the estimate, the appropriateness of the fitted model can be investigated graphically by means of the `plot()` method for objects of class `fGPDFIT`. Figure 7.9 shows the diagnostic plots obtained. The upper panels show the fitted excess distribution and a tail plot. Both indicate a good fit of the GPD to the exceedances. The lower panels display the residuals with a fitted ordinary least-squares (OLS) line on the left and a QQ plot on the right. Neither plot gives cause for concern as to a size effect; that is to say, the OLS line stays fairly flat and in the QQ plot the points plotted do not deviate much from the diagonal.

Finally, point estimates for the VaR and ES risk measures can be swiftly computed with the function `GPDRiskMeasures()`. In Listing 7.3 these are computed for the 95%, 99%, and 99.5% levels. The results are shown in Table 7.4. These measures would qualify as unconditional risk assessments for the next business day.

In the example above it was implicitly assumed that the exceedances are iid data points. However, this assumption is barely tenable for financial market returns. In particular, given the validity of the stylized facts, returns large in absolute value are clustered with respect to time and this empirical characteristic also holds *cum grano salis* for losses that exceed a high value chosen as a threshold. The focus is now shifted to the time domain of the exceedances, called marks, rather than the actual values. The issue raised here is exemplified in Listing 7.4 by employing the daily returns of the New York Stock Exchange Index.

First, the necessary packages `fBasics` and `fExtremes` are loaded into the workspace. Next, the daily continuous losses of the stock index are computed and expressed as percentages. The data for exceedances above the 95th percentile are recovered from the losses with the function `pointProcess()`. This function returns an object with the time stamps and marks, that is, the losses above the threshold. For an iid point process the time gaps between consecutive exceedances should be Poisson distributed. This can easily be checked by means of an appropriate QQ plot. Another graphical means to detect departure from the iid assumption is to portray the ACF and/or PACF of the gaps. If the exceedances occurred randomly during the sample period, the gaps should not be autocorrelated. Such graphs are shown in Figure 7.10.

All four graphs indicate that the exceedances cannot be assumed to be iid. The upper left panel shows the point process. It is fairly evident from this graph that the exceedances are clustered. This is mirrored by the QQ plot in the upper right panel. Here the time gaps between consecutive marks are plotted against a Poisson distribution. The scatter points deviate considerably from the fitted line. The non-randomness of the occurrences is also reflected by the ACF and PACF plot in the lower panel. The ACF tapers off only slowly and the PACF indicates a significant second-order autocorrelation.

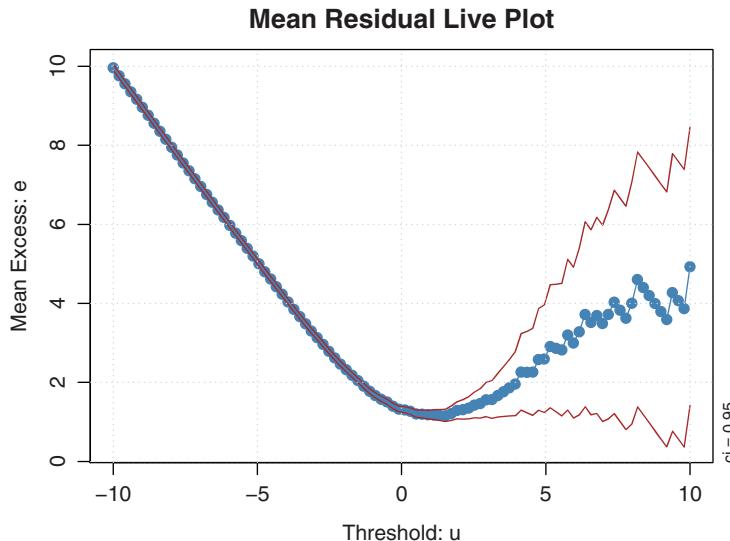


Figure 7.8 MRL plot for Boeing losses.

Table 7.3 Fitted GPD of Boeing.

GPD	ζ	β
Estimate	0.331	1.047
Standard error	0.128	0.166

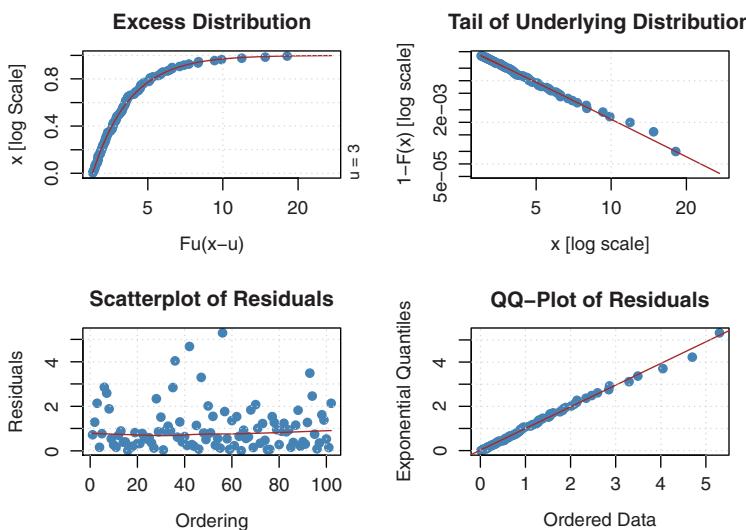


Figure 7.9 Diagnostic plots of fitted GPD model.

Table 7.4 Risk measures for Boeing.

Confidence level	VaR	ES
95.0%	2.783	4.240
99.0%	4.855	7.336
99.5%	6.149	9.268

R code 7.4 Declustering of NYSE exceedances.

```

1 library(fExtremes)
2 library(fBasics)
3 data(nyse)
4 NYSELevel <- timeSeries(nyse[, 2],
5                           charvec = as.character(nyse[, 1]))
6 NYSELoss <- na.omit(-1.0 * diff(log(NYELevel)) * 100)
7 colnames(NYSELoss) <- "NYSELoss"
8 ## Point process data
9 NYSEPP <- pointProcess(x = NYSELoss,
10                         u = quantile(NYSELoss, 0.95))
11 ## Declustering
12 DC05 <- deCluster(x = NYSEPP, run = 5, doplot = FALSE)
13 DC10 <- deCluster(x = NYSEPP, run = 10, doplot = FALSE)
14 DC20 <- deCluster(x = NYSEPP, run = 20, doplot = FALSE)
15 DC40 <- deCluster(x = NYSEPP, run = 40, doplot = FALSE)
16 DC60 <- deCluster(x = NYSEPP, run = 60, doplot = FALSE)
17 DC120 <- deCluster(x = NYSEPP, run = 120, doplot = FALSE)
18 ## Fit of declustered data
19 DC05Fit <- gpdFit(DC05, u = min(DC05))
20 DC10Fit <- gpdFit(DC10, u = min(DC10))
21 DC20Fit <- gpdFit(DC20, u = min(DC20))
22 DC40Fit <- gpdFit(DC40, u = min(DC40))
23 DC60Fit <- gpdFit(DC60, u = min(DC60))
24 DC120Fit <- gpdFit(DC120, u = min(DC40))

```

As a means of data preprocessing, the exceedances can be declustered. In other words, only the maximum is recovered within a cluster of exceedances as the representative extreme loss. Hence, the declustered exceedances are at least the assumed width of the cluster apart from each other with respect to time. The aim of this data preprocessing technique is to ensure the validity of the GPD assumptions. Note that the results are sensitive to the choice of number of runs. To highlight this issue, declustered series have been retrieved for various periodicities in Listing 7.4. That is, the point process data have been declustered for weekly, biweekly, monthly, bi-monthly, and quarterly as well as half-yearly runs. Next, the GPD is fitted to these series and the results are provided in Table 7.5.

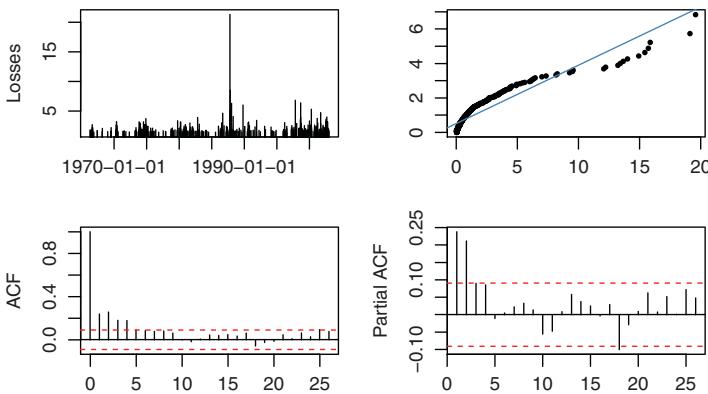


Figure 7.10 Plots for clustering of NYSE exceedances.

Table 7.5 Results for declustered GPD models.

Decluster	$\hat{\xi}$	$\hat{\beta}$	Exceedances	NLLH
Weekly	0.252 [0.066]	0.536 [0.046]	303	189.9
Bi-weekly	0.324 [0.084]	0.518 [0.055]	222	147.5
Monthly	0.33 [0.102]	0.59 [0.076]	149	118.76
Bi-monthly	0.376 [0.146]	0.752 [0.133]	86	92.67
Quarterly	0.421 [0.181]	0.824 [0.178]	61	73.59
Semi-annually	0.581 [0.342]	0.928 [0.352]	25	37.66

In this table the estimates for ξ and β are reported. The estimated standard errors are reported in square brackets. In the two columns to the right the numbers of declustered exceedances and the values of the negative log-likelihood are provided. The estimates for the shape parameter increase with run frequency (as one moves down the table), given that each run frequency is a valid choice. However, these point estimates become less reliable due to the decreased number of observations available for minimizing the negative log-likelihood, though the values for the latter do pick up for longer run periods. With respect to the derivation of VaR and/or ES these results imply quite different assessments of the riskiness inherent in the NYSE index. In particular, the point processes for the monthly, bimonthly, quarterly, and half-yearly declustered series do not indicate that their time gaps depart to a large extent from the exponential distribution, but the inference drawn from the fitted models does differ markedly.

References

- Beirlant J., Goegebeur Y., Segers J., and Teugels J. 2004 *Statistics of Extremes: Theory and Applications*. John Wiley & Sons, Chichester, England.
- Burger M., Jünemann K., and König T. 2015 *RUnit: R Unit Test Framework*. R package version 0.4.28.
- Coles S. 2001 *An Introduction to Statistical Modeling of Extreme Values*. Springer Verlag, London, New York.
- Deville Y. 2015a *Renext: Renewal Method for Extreme Values Extrapolation*. R package version 3.0-0.
- Deville Y. 2015b *RenextGUI: GUI for Renext*. R package version 1.3-0.
- Embrechts P., Klüppelberg C., and Mikosch T. 1997 *Modelling Extremal Events for Insurance and Finance* vol. 33 of *Stochastic Modelling and Applied Probability* 1st edn. Springer-Verlag, Berlin.
- Embrechts P., Puccetti G., and Rüschendorf L. 2013 Model uncertainty and VaR aggregation. *Journal of Banking and Finance* **38**(8), 2750–2764.
- Ferro C. and Segers J. 2003 Inference for clusters of extreme values. *Journal of the Royal Statistical Society, Series B* **65**(2), 545–556.
- Gilleland E. and Katz R. 2011 New software to analyze how extremes change over time. *Eos* **92**(2), 13–14.
- Heffernan E. and Stephenson A. 2016 *ismev: An Introduction to Statistical Modeling of Extreme Values*. R package version 1.41.
- Leadbetter M., Lindgren G., and Rootzén, H. 1983 *Extremes and Related Properties of Random Sequences and Series*. Springer-Verlag, New York.
- McNeil A. 1999 Extreme value theory for risk managers *Internal Modelling and CAD II* RISK Books pp. 93–113.
- McNeil A., Frey R., and Embrechts P. 2005 *Quantitative Risk Management: Concepts, Techniques and Tools*. Princeton University Press, Princeton, NJ.
- Pfaff B. and McNeil A. 2012 *evir: Extreme Values in R*. R package version 1.7-3.
- Plummer M., Best N., Cowles K., and Vines K. 2006 CODA: Convergence diagnosis and output analysis for MCMC. *R News* **6**(1), 7–11.
- Stephenson A. 2002 evd: Extreme value distributions. *R News* **2**(2), 0.
- Stephenson A. and Ribatet M. 2006 *A User's Guide to the evdbayes Package (Version 1.1)*.
- Stephenson A. and Ribatet M. 2014 *evdbayes: Bayesian Analysis in Extreme Value Theory*. R package version 1.1-1.
- Würtz D. 2013 *fExtremes: Rmetrics – Extreme Financial Market Data*. R package version 3010.81.
- Würtz D., Setz T., and Chalabi Y. 2014 *fBasics: Rmetrics – Markets and Basic Statistics*. R package version 3011.87.

8

Modelling volatility

8.1 Preliminaries

The previous two chapters introduced quantitative methods for risk modelling in the case of non-normally distributed returns, that is, extreme value theory and the generalized hyperbolic and generalized lambda distribution classes. The first method addresses the tail modelling of a return process, whereas the second focuses on adequately capturing the entire distribution. With respect to the value-at-risk and expected shortfall risk measures it was assumed that the financial market returns are iid. Hence, these risk measures are unconditional in the sense that these measures do not depend on prior information. As already shown in Chapter 3, volatility clustering is one of the stylized facts of financial market returns. Given this stylized fact, the assumption of iid returns is clearly violated. Therefore, this chapter introduces a model class that takes volatility clustering explicitly into account. As will be shown, conditional risk measures can be deduced from these models. Here the phenomenon of volatility clustering directly feeds into the derived risk measures for future periods in time.

8.2 The class of ARCH models

The class of autocorrelated conditional heteroscedastic (ARCH) models was introduced in the seminal paper by Engle (1982). This type of model has since been modified and extended in several ways. The articles by Engle and Bollerslev (1986), Bollerslev et al. (1992), and Bera and Higgins (1993) provide an overview of the

model extensions during the decade or so after the original paper. Today, ARCH models are not only well established in the academic literature but also widely applied in the domain of risk modelling. In this section the term “ARCH” will be used both for the specific ARCH model and for its extensions and modifications.

The starting point for ARCH models is an expectations equation which only deviates from the classical linear regression with respect to the assumption of independent and identically normally distributed errors:

$$y_t = \mathbf{x}'_t \boldsymbol{\beta} + \epsilon_t, \quad (8.1)$$

$$\epsilon_t | \Psi_{t-1} \sim \mathcal{N}(0, h_t). \quad (8.2)$$

The error process ϵ_t is assumed to be normally distributed, but the variance is not constant, being allowed to change over time. This variability of the variance is expressed as dependence with respect to the available information at time $t-1$, denoted by Ψ_{t-1} . An alternative notation is commonly encountered in the literature:

$$\epsilon_t = \eta_t \sqrt{h_t}, \quad (8.3)$$

$$\eta_t \sim \mathfrak{D}_v(0, 1), \quad (8.4)$$

where η_t denotes a random variable with distribution \mathfrak{D} with expected value zero and unit variance. Additional parameters of this distribution are subsumed in the parameter vector v . This notation is more general in the sense that now a normally distributed error process is no longer required, but distributions with non-zero excess kurtosis and/or skewness can be interspersed. In the literature one often encounters the Student's t distribution, the skewed Student's t distribution, or the generalized error distribution.

The second building block of ARCH models is the variance equation. In an ARCH model of order q the conditional variance is explained by the history of the squared errors up to time lag q :

$$h_t = \alpha_0 + \alpha_1 \epsilon_{t-1}^2 + \cdots + \alpha_q \epsilon_{t-q}^2, \quad (8.5)$$

where $\alpha_0 > 0$ and $\alpha_i \geq 0, i = 1, \dots, q$. These parameter restrictions guarantee a positive conditional variance. The inclusion of the available information up to time $t-1$ is evident from

$$\epsilon_{t-i} = y_{t-i} - \mathbf{x}'_{t-i} \boldsymbol{\beta}, i = 1, \dots, q. \quad (8.6)$$

It can already be deduced why this model class can capture the stylized fact of volatility clustering: the conditional variance is explained by the errors in previous periods. If these errors are large in absolute value, a large value for the conditional variance results, and vice versa. By way of illustration, Figure 8.1 shows the plots of a white noise process and simulated ARCH(1) and ARCH(4) processes.

A clear pattern of volatility clustering is evident for the ARCH(1) and ARCH(4) processes, and it is more pronounced for the latter. One might be tempted to conclude that the two ARCH processes are more volatile than the white noise. However, all

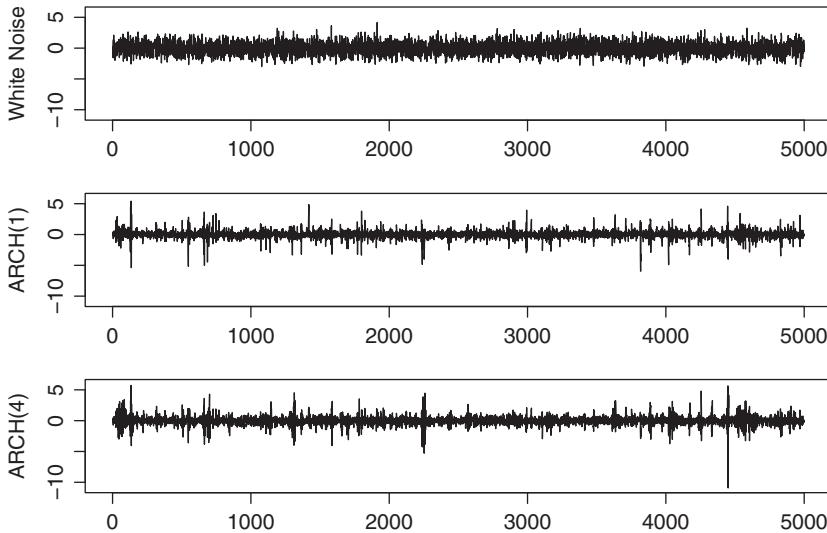


Figure 8.1 Plots of simulated white noise, ARCH(1), and ARCH(4) processes.

processes have been simulated to have unit unconditional variances. For an ARCH(1) process the unconditional variance is given by

$$\sigma_\epsilon^2 = \mathbb{E}(\epsilon_t^2) = \frac{\alpha_0}{1 - \alpha(1)}, \quad (8.7)$$

where $\alpha(1)$ represents the sum of the coefficients for the lagged squared errors. The processes in Figure 8.1 have been generated according to $h_t = 0.1 + 0.9\epsilon_t^2$ and $h_t = 0.1 + 0.36\epsilon_{t-1}^2 + 0.27\epsilon_{t-2}^2 + 0.18\epsilon_{t-3}^2 + 0.09\epsilon_{t-4}^2$ for the ARCH(1) and ARCH(4) models, respectively. Hence, for both models the unconditional variance equals unity. Note that the fourth moment (i.e., the kurtosis) for ARCH(1) models,

$$\frac{\mathbb{E}(\epsilon_t^4)}{\sigma_\epsilon^4} = 3 \left(\frac{1 - \alpha_1^2}{1 - 3\alpha_1^2} \right), \quad (8.8)$$

is greater than 3 and thus has more probability mass in the tails than the normal distribution. This is further evidence that with ARCH models the stylized facts of financial market returns can be captured well.

Having discussed the baseline ARCH model, the focus is now shifted to its modification and extensions. Bollerslev (1986) introduced the GARCH(p, q) model into the literature. This differs from the ARCH model in the inclusion of lagged endogenous variables in the variance equation—that is, now the conditional variance depends not only on past squared errors but also on lagged conditional variances:

$$h_t = \alpha_0 + \alpha_1 \epsilon_{t-1}^2 + \cdots + \alpha_q \epsilon_{t-q}^2 + \beta_1 h_{t-1} + \cdots + \beta_p h_{t-p}, \quad (8.9)$$

with the restrictions $\alpha_0 > 0$, $\alpha_i \geq 0$ for $i = 1, \dots, q$, and $\beta_j \geq 0$ for $j = 1, \dots, p$ such that the conditional variance process is strictly positive. The advantage of this approach is that a GARCH model is the same as an ARCH model with an infinite number of lags, if the roots of the lag polynomial $1 - \beta(z)$ lie outside the unit circle, hence a more parsimonious specification is possible when GARCH-type specifications are used. The unconditional variance for a GARCH(1, 1) model is given by $\sigma_e^2 = \mathbb{E}(\epsilon_t^2) = \frac{\alpha_0}{1 - \alpha_1 - \beta_1}$. From this, the condition for a stable variance process can be directly deduced, namely, $\alpha_1 + \beta_1 < 1$.

So far it has been assumed that the sign of the shock does not have an impact on the conditional variance. This is because the past errors enter as squares into the variance equation. Nelson (1991) extended the class of ARCH to make it possible to take this effect into account. He proposed the class of *exponential* GARCH (EGARCH) models to capture such asymmetries. The modelling of asymmetric effects can be justified from an economic point of view with leverage effects, in particular when equity returns are investigated. For these, a negative relationship between volatility and past returns is postulated (see Black 1976). The variance equation now takes the form

$$\log(h_t) = \alpha_0 + \sum_{i=1}^q \alpha_i g(\eta_{t-i}) + \sum_{j=1}^p \beta_j \log(h_{t-j}), \quad (8.10)$$

where the previously introduced alternative specification of ARCH models is employed and the function $g(\eta_t)$ is defined as:

$$g(\eta_t) = \theta \eta_t + \gamma [|\eta_t| - \mathbb{E}(|\eta_t|)]. \quad (8.11)$$

This approach has some peculiarities worth mentioning. First, a multiplicative relationship for explaining the conditional variances is assumed. This becomes evident from the logarithmic form for the variance equation. Hence, there is no need for non-negativity constraints on the parameter space, $\alpha_i, i = 1, \dots, q$ and $\beta_j, j = 1, \dots, p$, because $h_t = \exp(\cdot)$ will always be positive. Second, the impact of the error variables is piecewise linear and takes for a positive shock a value of $\alpha_i(\theta + \gamma)$ and for a negative one $\alpha_i(\theta - \gamma)$. Here, the sign-dependent impact of past errors with respect to the contemporaneous conditional variance shows up. The first term, $g(\eta_t)$, in the equation depicts the correlation between the error process η_t and the future conditional volatility. The ARCH effects themselves are captured by the coefficient γ . The greater the deviation of the variable η_t from its expected value, the higher the value of the function $g(\eta_t)$ and hence of the log-value for the conditional variance.

The last of the many extensions of the ARCH model to be discussed is *asymmetric power* ARCH (APARCH) proposed by Ding et al. (1993). The reason for this choice is that the APARCH model encompasses other ARCH specifications, as will be shown below. Its variance equation is defined as

$$\epsilon_t = \eta_t h_t, \quad (8.12)$$

$$\eta_t \sim \mathcal{D}_v(0, 1), \quad (8.13)$$

$$h_t^\delta = \alpha_0 + \sum_{i=1}^q \alpha_i (|\epsilon_{t-i}| - \gamma_i \epsilon_{t-i})^\delta + \sum_{j=1}^p \beta_j h_{t-j}^\delta, \quad (8.14)$$

with parameter restrictions $\alpha_0 > 0$, $\delta \geq 0$, $\alpha_i \geq 0$, $i = 1, \dots, q$, $-1 < \gamma_i < 1$, $i = 1, \dots, q$, and $\beta_j \geq 0$, $i = 1, \dots, p$. One peculiarity of this approach for modelling the conditional variance is the exponent δ . For $\delta = 2$ the conditional variance results, similarly to the models discussed above. But the parameter is defined for non-negative values in general and hence the *long memory* characteristic often encountered for absolute and/or squared daily return series can be taken explicitly into account. The long memory characteristic can be described simply by the fact that for absolute and/or squared returns the autocorrelations taper off only slowly and hence dependencies exist between observations that are further apart. Potential asymmetries with respect to positive/negative shocks are reflected in the coefficients γ_i , $i = 1, \dots, q$. The APARCH model includes the following special cases:

- ARCH model, if $\delta = 2$, $\gamma_i = 0$, and $\beta_j = 0$;
- GARCH model, if $\delta = 2$ and $\gamma_i = 0$;
- TS-GARCH (see Schwert 1990; Taylor 1986), if $\delta = 1$ and $\gamma_i = 0$;
- GJR-GARCH (see Glosten et al. 1993), if $\delta = 2$;
- T-ARCH (see Zakoian 1994), if $\delta = 1$;
- N-ARCH (see Higgins and Bera 1992), if $\gamma_i = 0$ and $\beta_j = 0$;
- Log-ARCH (see Geweke 1986; Pentula 1986), if $\delta \rightarrow 0$.

The reader is referred to the citations above for further details on these special cases.

Estimates for the unknown model parameters are often obtained by applying the maximum likelihood or quasi-maximum likelihood principle. Here, numerical optimization techniques are applied. Less commonly encountered are Bayesian estimation techniques for the unknown parameter in applied research. Whence a model fit is obtained, forecasts for the conditional variances can be computed recursively and the desired risk measures can be deduced together with the quantiles of the assumed distribution for the error process from these.

8.3 Synopsis of R packages

8.3.1 The package bayesGARCH

The package **bayesGARCH** implements the Bayesian estimation of GARCH(1, 1) models with Student's t innovations (see Ardia 2008, 2009, 2015; Ardia and Hoogerheide 2010; Nakatsuma 2000). The package is contained in the CRAN “Bayesian,” “Finance,” and “TimeSeries” Task Views. It has dependencies on the packages **mvtnorm** and **coda**. The latter is used to generate MCMC objects which

can be utilized to form the joint posterior sampler. Furthermore, **bayesGARCH** interfaces with routines written in C for recursively computing conditional variances. The package is shipped with a NAMESPACE file in which the two cornerstone functions `bayesGARCH()` and `formSmp1()` are exported. It includes a demo file in which the proper usage of the functions is exemplified, as well as a data set of Deutschmark/Sterling (DEM/GBP) continuous returns. A model description, the priors, and the implemented MCMC scheme are provided in a vignette.

Within the call to `bayesGARCH()`, the priors for the coefficients and the associated covariance matrices can be specified. In addition, the hyper-parameters for either the translated exponential distribution or the degrees of freedom parameter v of the t distribution can be provided. If the values for these are set to reasonably high values, a model with normal innovations is obtained at the limit. Finally, the estimation can be influenced by providing a list object with named elements for the `control` argument of the function. Here, the number of Markov chains ("n.chain"), the length of the chains ("l.chain"), starting values ("start.val"), prior conditions ("addPriorConditions"), and the list elements that determine the shape of the estimation report (frequency thereof ("refresh")) and number of digits to be printed ("digits")) can be supplied. The prior conditions have to be provided in the form of a function that returns the logical values TRUE/FALSE.

8.3.2 The package `ccgarch`

This package is one of three in which multivariate GARCH models can be dealt with. In particular, the conditional correlation approach to multivariate GARCH (CC-GARCH) is implemented (see Nakatani 2014). The package is contained in the CRAN “Finance” Task View. It is shipped with a NAMESPACE file. The package itself is a collection of functions and employs neither S3 nor S4 methods or classes. Ordinarily the functions return `list` objects and their elements can then be used as functional arguments in other calls. The computationally burdensome estimation routines are interfaced from C code.

The unknown parameters of a CC-GARCH model are estimated by utilizing the function `dcc.estimation()`. Within this function the two-step estimation procedure is carried out. Each of these estimators can also be accessed and employed on a standalone basis through the functions `dcc.estimation1()` and `dcc.estimation2()`. These functions return `list` objects and the elements thereof can then be used in subsequent analysis. For instance, robust standard errors for the model’s parameters can be returned with the function `dcc.results()`, by providing the necessary arguments explicitly. Similarly, the values of the log-likelihoods can be computed with the functions `loglik.dcc()`, `loglik.dcc1()`, and `loglik.dcc2()` for the CC-GARCH model, the first-stage optimization, and the second-stage optimization, respectively. The stationarity condition of an empirical CC-GARCH model can be checked by utilizing the function `stationarity()`. For diagnostic testing, the causality tests proposed by Hafner and Herwartz (2006) (`hh.test()`) and by Nakatani and Teräsvirta (2009) and Nakatani (2010) (`nt.test()`) are implemented. The validity of the

normality assumption can be assessed with the function `jb.test()`. Routines are included in **ccgarch** for simulating data according to pre-specified CC-GARCH parameterizations, namely `dcc.sim()` and `eccc.sim()`. The innovations can follow either the normal or Student's t distribution. The package lacks a routine for obtaining forecasts from CC-GARCH type models, however.

8.3.3 The package **fGarch**

The package **fGarch** is part of the Rmetrics suite of packages (see Würtz and Chalabi 2013). It is contained in the CRAN "Finance" and "TimeSeries" Task Views and is considered a core package in the former. This package is the broadest implementation of univariate ARCH models and the extensions thereof. It interfaces with FORTRAN routines for the more computationally burdensome calculations. Within the package, S4 methods and classes are utilized. As a technicality, a unit testing framework based on the package **RUnit** is implemented (see Burger et al. 2015).

The cornerstone function for fitting ARCH-type models is `garchFit()`. Within this function the mean and variance models to be used are specified in the form of a `formula`. Currently, pure GARCH/APARCH models can be supplied in which the mean equation is a simple intercept, or more general specifications such as ARMA/GARCH or ARMA/APARCH models can be estimated. The innovations can be distributed according to the normal, Student's t , the skewed Student's t , the GEV, or the skewed GEV. The maximization of the log-likelihood is conducted by means of numerical optimizers. Here, the user can choose between `nlminb`, `lbfgsb`, `nlminb+nm`, and `lbfgsb+nm`; in the latter two cases the Nelder–Meade optimizer is employed in a first stage. Null restrictions can be superimposed for the more complex ARCH-type models by employing include directives as logical flags for these coefficients. The function returns an object of formal class `fGARCH`. For objects of this kind, `coef()`, `fitted()`, `formula()`, `initialize()`, `plot()`, `predict()`, `residuals()`, `show()`, `summary()`, and `update()` methods have been defined. Data for univariate ARCH-type models can be generated with the function `garchSim()`. Here, a particular model can be specified with the function `garchSpec()` and the object returned can then be used in calls to this function.

8.3.4 The package **GEVStableGarch**

The package **GEVStableGarch** has recently been added to CRAN (see do Rego Sousa et al. 2015). It is listed in the task views "Finance" and "Time Series." The package is written purely in R and employs neither the S3 nor the S4 class/method scheme.

Similar to **fGarch**, ARMA/GARCH and ARMA/APARCH models can be fitted to data. This is accomplished by the function `GSGarch.Fit()`. Even though the package's name implies that only the generalized extreme value distribution is used for the error term ("gev"), one can specify via the argument `cond.dist` that the error process adheres to either the normal ("norm"), the Student's t ("std"), the skewed Student's t ("sstd"), or the stable ("stable") distribution instead. Hence,

from a modelling point of view pretty much the same can be accomplished as with the function `garchFit()` of **fGarch**. For estimation of the unknown model parameters the user can choose between `nlminb()` or a sequential quadratic optimization routine, as implemented in **RSolnp** (see Ghalanos and Theussl 2015; Ye 1987). The user is guided in selecting an appropriate lag order for the ARMA/GARCH or ARMA/APARCH models by calling the function `GSGarch.FitAIC()`. Here, one provides upper bounds for the lag orders and the model with the minimum AIC is returned. Finally, for a given set of estimates, the progression of the data-generation process (DGP) can be simulated with the function `GSGarch.Sim()`.

8.3.5 The package **gogarch**

The package **gogarch** (see Pfaff 2012) implements the generalized orthogonal GARCH (GOGARCH) model, a multiple GARCH model proposed by Boswijk and van der Weide (2006); van der Weide (2002) and Boswijk and van der Weide (2009). The package is contained in the CRAN “Finance” and “TimeSeries” Task Views. It utilizes formal S4 classes and methods and is written purely in R.

The cornerstone function in the package is `gogarch()`, which initializes the model and its specification and carries out the estimation according to the methods chosen. The unknown parameters of a GOGARCH model can be estimated by maximum likelihood, nonlinear least squares, the method of moments, or by applying an independent component analysis via the `fastICA()` function contained in the package of the same name. When the unknown model parameters have been estimated the user can then proceed by employing the methods shown in Table 8.1. The availability of the methods is partly dependent on the estimation technique chosen. However, regardless of this, the user can not only retrieve the fitted values, residuals, coefficients, covariances, and/or correlations, but can also obtain a summary of the fitted model. Furthermore, forecasts can be swiftly produced by the `predict()` method. Although there are quite a few methods available in the package, the coding effort is reduced to a minimum by defining the methods for the parent class GOGARCH such that the child classes defined inherit the methods by making use of `callNextMethod()`.

8.3.6 The package **lgarch**

The focus of the package **lgarch** is on the estimation and simulation of univariate and multivariate log-GARCH models. The package has recently been contributed to CRAN is contained in the task views “Finance” and “Time Series.” Within the package the S3 class/method engine is used. Log-GARCH models can be represented in the form of a (V)ARMA-X model (see Francq and Sucarrat 2013; Sucarrat et al. 2013). This representation is chosen for estimating the unknown model parameters in the case of the quasi-ML method. The computations of the necessary recursions that are implied by this approach are interfaced from routines written in C++, which yields faster execution compared to a pure R implementation. As an

Table 8.1 Overview of package *gogarch*.

alternative to QML, estimates for the model parameters can also be determined by applying nonlinear least-squares.

Univariate log-GARCH models are fitted by means of the function `lgarch()`. This function returns a `list` object with class attribute `lgarch`. Methods for recovering/extracting the estimated coefficients (`coef()`), the fitted values (`fitted()`), the residuals (`residuals()`), the residual sum of squares (`rss()`), the value of the log-likelihood (`logLik()`), a summary (`summary()`), and the variance-covariance matrix of the coefficients (`vcov()`) are made available. Simulated series can be generated by calling `lgarchSim()`.

The function `mlgarch` can be employed for the estimation of multivariate log-GARCH models whereby constant conditional correlations between the error processes are superimposed. Simulations for this type of multivariate DGP and a given set of parameters can be computed with `mlgarchSim()`.

8.3.7 The packages `rugarch` and `rmgarch`

A pretty comprehensive suite of GARCH-type models for univariate series is made available in the package **`rugarch`** (see Ghalanos 2015b), which is contained in the “Finance” and “Time Series” Task Views. S4 classes and methods are employed and the package is shipped with a NAMESPACE file. The computationally burdensome routines are interfaced from C++ functions. This is accomplished with the packages **Rcpp** (Eddelbüttel and François 2011) and **RcppArmadillo** (see Eddelbüttel and Sanderson 2014), on which this package depends. Because of the numerical complexity and the demanding task of fitting the covered models to data, wrapper functions that support multi-core capabilities are available if models are fitted to more than one endogenous variable. The package is shipped with a vignette in which the major capabilities and applications are elucidated by examples. Four data sets are included in **`rugarch`**: a return series of the Dow Jones Index (`dji30ret`), a return series of the S&P 500 index (`sp500ret`), the SPDR S&P 500 open–close daily returns and the realized kernel volatility (`spyreal`) as used by Hansen et al. (2012), and a spot exchange rate series for DEM/GBP (`dmbp`), all daily.

A typical work flow would start by specifying the kind of GARCH model with the function `ugarchspec()`. This is similar in design and purpose to the function `garchSpec()` in **fGarch** introduced earlier. It takes five arguments. The kind of variance model is determined by a list object for the `variance.model` argument, the mean equation is set with the `mean.model` argument, and the distribution of the error process with the `distribution.model` argument. Starting values for the parameters according to these first three arguments, as well as whether any of them should be kept fixed, can be controlled with the `start.pars` and `fixed.pars` arguments. The function returns an object of formal class `uGARCHspec`.

Objects of this class can then be used for fitting data to the chosen specification, which is achieved by calling `ugarchfit()`. Apart from a `uGARCHspec`, which is passed as argument `spec` to the function, the data set is passed to the body of the function as argument `data`. This can be either a numeric `vector`, a `matrix` or `data.frame` object, or one of the specific time series class objects: `zoo`, `xts`,

`timeSeries`, or `irts`. The numerical solution of the model can be determined by one of the optimizers `nlminb()`, `solnp()`, or `gosolnp()`, which is set by the argument `solver`. Control arguments pertinent to these solvers can be passed down by providing a `list` object for the argument `solver.control`. In addition, the conformance of stationarity constraints, the calculation of standard errors in the case of fixed parameters, and/or whether the data is to be scaled prior to optimization can be set with the argument `fit.control`. A feature of the function is the argument `out.sample`. Here the user can curtail the size of the sample used for fitting, leaving the remaining n data points available for pseudo *ex ante* forecasts. The function `ugarchfit()` returns an object of S4 class `uGARCHfit`. This class has the slots `fit` for the fitted model and `model` for its specification. A total of 22 methods are defined for these objects, including data extractions, displaying and analyzing the fitted model, diagnostic testing, and plotting facilities.

Forecasts can be generated with the function `ugarchforecast()`. Here, either a fitted or a specified model has to be provided. If the latter is used, a valid set of fixed parameters must have been set and a data set must be passed to the function as argument `data`. The number of forecast periods and/or whether the forecasts should be derived from a rolling window are determined by the arguments `n.ahead` and `n.roll`, respectively. If exogenous regressors are specified in either the mean or the variance equation, values for these must be supplied as a `list` object for `external.forecast`. The forecast function returns an object of S4 class `uGARCHforecast`, for which data extraction, plotting, and performance measure methods are available. Confidence bands for the forecasts can be generated by means of bootstrapping, which is implemented as function `ugarchboot()`.

In addition to using a fitted model for forecasting purposes, the function `ugarchsim()` enables the user to simulate data from it. Data extraction, show, and plot methods are implemented for the returned object of class `uGARCHsim`.

The function `ugarchroll()` can be used to back-test the VaR of a return/loss series. The user can swiftly produce VaR forecasts of a GARCH model and analyze the results of the object returned, which is of class `uGARCHroll`, with the methods provided for data extraction of the VaR numbers, plotting, reporting, testing of the forecast performance, and/or summarizing the back-test results. The back-test can be conducted either for a recursively extending or a moving sample window.

The user can filter data for a given model specification by utilizing the function `ugarchfilter()`. The function returns an S4 object `uGARCHfilter` for which the same set of methods is available as in the case of `uGARCHfit` objects. Hence, filtering could in principle be utilized to assess the robustness of results, given differing parameter values, and to check whether an alternative specification yields reasonable results.

Lastly, it should be pointed out that the package contains a routine for benchmark comparison of results (`ugarchbench()`) and supports parallel computation through the wrapper functions `multispec()`, `multifit()`, `multiforecast()`, and `multifilter()`, as mentioned earlier in this subsection.

A companion package to **rugarch** by the same author is **rmgarch** Ghalanos (2015a). This package is also listed in the CRAN Task Views on “Finance” and

“Time Series,” and is endowed with a vignette. In this package multivariate GARCH model classes and concepts are implemented, namely, the dynamic conditional correlation (DCC) GARCH models as proposed by Engle (2002), the GOGARCH models, and copula-GARCH models. The latter concept will be presented in Chapter 9 when the copula concept is discussed. For each of these multivariate GARCH models a set of S4 classes and methods are defined. The naming convention is `DCCfoo`, `goGARCHfoo`, and `cGARCHfoo` for the three models, where `foo` serves as a placeholder for the object’s purpose—`spec` for the specification, `fit` for the fitted model, `filter` for the filter solution of a specified model, `roll` for a rolling estimation of the model, `sim` for simulating trajectories from a given model specification, and `forecast` for containing the forecasts of a model. Hence, a total of 16 classes have been defined. In addition to these class definitions, virtual classes from which the model-specific ones inherit are defined as `mGARCHfoo`. Objects for each of the classes can be created by calling the constructor functions of the same name as the S4 class, but spelled in lower-case letters. For instance, a DCC model is specified by calling the function `dccsim()`. The resulting object can then be used to estimate the unknown coefficients by invoking `dccfit()`.

8.3.8 The package `tseries`

The package `tseries` was the first contributed package on CRAN in which time series models and related statistical tests are primarily implemented (see Trapletti and Hornik 2016). Its history dates back to the late 1990s. It is contained in the “Econometrics,” “Finance,” “TimeSeries,” and “Environmetrics” Task Views, and it is a core package in the former three views. Within the package S3 classes and methods are employed, and it is shipped with a NAMESPACE file.

With respect to ARCH models, the function `garch()` is provided. This function allows the user to fit GARCH models to data. Errors are assumed to be normally distributed. The order of the GARCH model is set by the argument `order`. This two-element vector determines the number of lags to be included in the model, the first element determining the consecutive lags for the lagged conditional variances and the second element determining the number of lags for the squared residuals. It is not possible to omit the intermediate lags for either the GARCH or the ARCH part. ML estimates for the unknown model parameters are determined by employing a quasi-Newton optimizer. This optimizer is interfaced from C/FORTRAN routines. Optimizer-specific arguments can be provided via a `list` object, `garch.control`. The ellipsis argument of `garch()` is passed down to the function `qr()`, which is employed in the computation of the asymptotic covariance matrix. The function returns a `list` object with class attribute `garch`.

For objects of this kind, `print()`, `coef()`, `vcov()`, `residuals()`, `fitted()`, `logLik()`, `plot()`, and `predict()` methods are defined. The estimation result is returned with the first method. The values of the estimated coefficients can be extracted by `coef()` and the associated variance-covariance matrix by `vcov()`. The value of the log-likelihood can be retrieved with the method `logLik()`. This method is useful for setting up likelihood ratio tests for statistically

discriminating between nested models. The \pm conditional standard deviations are returned by the method `fitted()`. The appropriateness of a given GARCH order can be graphically investigated with the `plot()` method. Here, time series plots of the series itself and the residuals can be returned as well as a histogram, a QQ plot, and an ACF representation thereof. Finally, predictions for the \pm conditional standard deviations can be generated with the `predict()` method.

8.4 Empirical application of volatility models

In this section a back-test of the expected shortfall at a 99% confidence level for New York Stock Exchange (NYSE) daily returns is conducted. The R code is shown in Listing 8.1. This data set is a supplement to the monograph of Stock and Watson (2007) and is contained in the **AER** package (see Kleiber and Zeileis 2008).

R code 8.1 Expected shortfall derived from GARCH(1, 1) models

```

library (AER)                                     1
library (fGarch)                                2
data (NYSESW)                                   3
NYSELOSS <- timeSeries (-1.0 * diff (log (NYSESW)) * 100,      4
                        char . vec = time (NYSESW))      5
## Function for ES of t-GARCH                   6
ESgarch <- function (y, p = 0.99){             7
  gfit <- garchFit (formula = ~ garch (1, 1), data = y,      8
                     cond . dist = "std", trace = FALSE)
  sigma <- predict (gfit, n . ahead = 1)[3]          9
  df <- coef (gfit) ["shape"]                      10
  ES <- sigma * (dt (qt (p, df), df) / (1 - p)) *      11
         ((df + (qt (p, df)) ^ 2) / (df - 1))        12
  return (ES)                                     13
}
## Date vectors for backtest                   14
from <- time (NYSELOSS)[-c ((nrow (NYSELOSS) - 999) :      15
                           nrow (NYSELOSS))]          16
to <- time (NYSELOSS)[ -c (1:1000)]           17
NYSEES <- fapply (NYSELOSS, from = from, to = to, FUN = ESgarch) 18
NYSEESL1 <- lag (NYSEES, k = 1)                19
res <- na . omit (cbind (NYSELOSS, NYSEESL1))      20
colnames (res) <- c ("NYSELOSS", "ES99")          21
plot (res[, 2], col = "red", ylim = range (res),      22
      main = "NYSE: t-GARCH(1,1) ES 99%",           23
      ylab = "percentages", xlab = "")            24
points (res[, 1], type = "p", cex = 0.2, pch = 19, col = "blue") 25
legend ("topleft", legend = c ("Loss", "ES"),          26
        col = c ("blue", "red"), lty = c (NA, 1), pch = c (19, NA)) 27

```

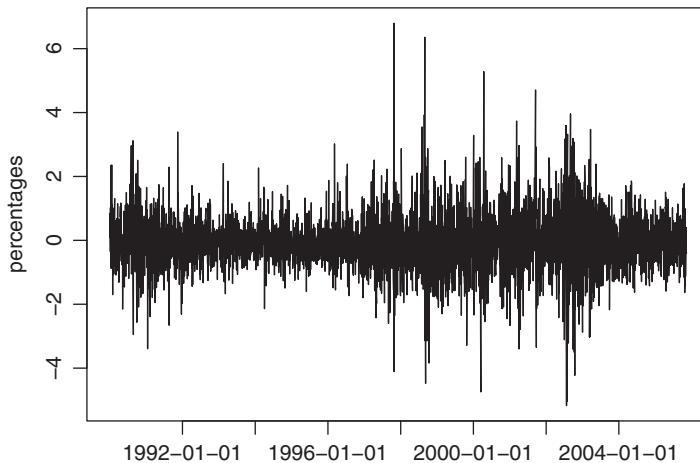


Figure 8.2 *Time series plot for daily losses of NYSE.*

The sample runs from 2 January 1990 to 11 November 2005 and contains 4003 observations. The necessary packages and the data set are loaded into the workspace with the first three commands in the R code listing. Next, the daily compound losses are computed as percentages and expressed as positive numbers. The stylized facts of this series are pretty evident from Figure 8.2.

In order to conduct the back-test, the function `ESgarch()` is defined in lines 6–14. Within this function a GARCH(1, 1) model with a Student’s t -distributed innovation process is estimated first. The one-step-ahead forecast of the conditional standard deviation is computed next, and the fitted value of the degrees-of-freedom

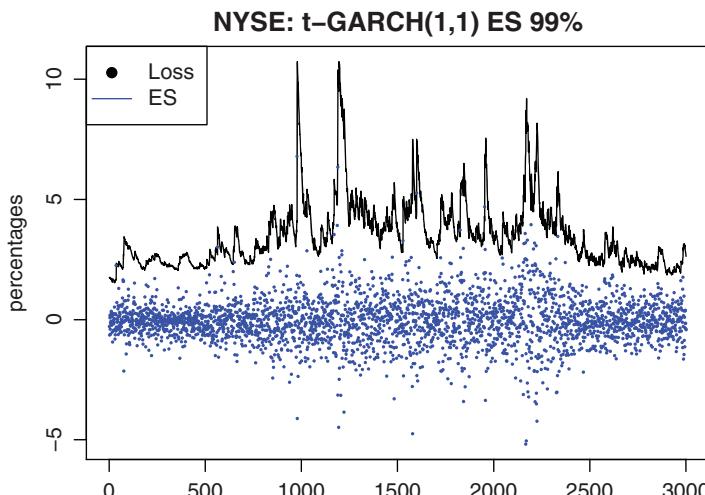


Figure 8.3 *Comparison of daily losses of NYSE and ES.*

parameter is assigned to the object `df`. The expected shortfall is then computed for the default confidence level of $p = 0.99$. Incidentally, the mean equation of this GARCH model consists of a constant only and is omitted in the calculation of the ES.

Given that its estimate represents the mean of the series which is empirically close to zero, it can safely be discarded from the computation of the ES. The back-test itself is then conducted by utilizing a sliding window with 1000 observations. The function `fapply()` comes in handy for conducting back-tests of this kind. Two date vectors are created in which the start and end dates of this moving window through time are stored. In the next line the function `fapply()` will call `ESgarch()` with the subsamples of the losses according to the date values contained in the objects `from` and `to`. It should be noted that the ES numbers are now associated with the date values of `from`. But these conditional risk measures pertain to the next trading day and hence the series must be lagged by one period (object `NYSEESL1`) for comparison with the actual losses. The size of the back-test therefore consists of 3001 risk measure–loss pairs. A graphical comparison of the actual losses and the conditional ES for a 99% confidence level is produced in the final lines. The outcome is shown in Figure 8.3.

Not only are the spikes of the daily losses captured pretty well by this conditional risk measure, but also the level of the risk measure decreases rapidly during the more tranquil periods. During this back-test simulation only five violations occurred. This is a little too conservative, given that roughly 15 exceedances can be expected.

References

- Ardia D. 2008 *Financial Risk Management with Bayesian Estimation of GARCH Models: Theory and Applications* vol. 612 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin, Germany.
- Ardia D. 2009 Bayesian estimation of a Markov-switching threshold asymmetric GARCH model with Student-t innovations. *Econometrics Journal* **12**(1), 105–126.
- Ardia D. 2015 *bayesGARCH: Bayesian Estimation of the GARCH(1,1) Model with Student-t Innovations in R*. Version 2.0.2.
- Ardia D. and Hoogerheide L. 2010 Bayesian estimation of the GARCH(1,1) model with Student-t innovations. *The R Journal* **2**(2), 41–47. URL <http://journal.r-project.org/>.
- Bera A. and Higgins H. 1993 ARCH models: Properties, estimation and testing. *Journal of Economic Surveys* **7**(4), 305–362.
- Black F. 1976 Studies of stock price volatility changes *Proceedings of the 1976 Meeting of Business and Economics Statistics Section*, pp. 177–181 American Statistical Association.
- Bollerslev T. 1986 Generalised autoregressive conditional heteroscedasticity. *Journal of Econometrics* **31**, 307–327.
- Bollerslev T., Chou R., and Kramer K. 1992 Arch modeling in finance. *Journal of Econometrics* **52**, 5–59.
- Boswijk H. P. and van der Weide R. 2006 Wake me up before you GO-GARCH. Discussion Paper TI 2006-079/4, University of Amsterdam and Tinbergen Institute, Amsterdam.
- Boswijk H. P. and van der Weide R. 2009 Method of moments estimation of GO-GARCH models. Working paper, University of Amsterdam and Tinbergen Institute and World Bank, Amsterdam.

- Burger M., Jünemann K., and König T. 2015 *RUnit: R Unit Test Framework*. R package version 0.4.31.
- Ding Z., Granger C., and Engle R. 1993 A long memory property of stock market returns and a new model. *Journal of Empirical Finance* **1**, 83–106.
- do Rego Sousa T., Otiniano C., and Lopes S. 2015 *GEVStableGarch*. R package version 1.1.
- Eddelbüttel D. and François R. 2011 Rcpp: Seamless R and C++ integration. *Journal of Statistical Software* **40**(8), 1–18.
- Eddelbüttel D. and Sanderson C. 2014 RcppArmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics and Data Analysis* **71**, 1054–1063.
- Engle R. 1982 Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation. *Econometrica* **50**(4), 987–1007.
- Engle R. 2002 Dynamic conditional correlation. *Journal of Business & Economic Statistics* **20**(3), 339–350.
- Engle R. and Bollerslev T. 1986 Modelling the persistence of conditional variances. *Econometric Reviews* **5**, 1–50.
- Francq C. and Sucarrat G. 2013 An exponential chi-squared QMLE for log-GARCH models via the ARMA representation. Working Paper 51783, University of Munich, Munich.
- Geweke J. 1986 Modelling the persistence of conditional variances: A comment. *Econometric Reviews* **5**, 57–61.
- Ghalanos A. 2015a *rmgarch: Multivariate GARCH models*. R package version 1.3-0.
- Ghalanos A. 2015b *rugarch: Univariate GARCH models*. R package version 1.3-6.
- Ghalanos A. and Theussl S. 2015 *Rsolnp: General Non-linear Optimization Using Augmented Lagrange Multiplier Method*. R package version 1.16.
- Glosten L., Jagannathan R., and Runkle D. 1993 On the relation between expected value and the volatility of the nominal excess return on stocks. *Journal of Finance* **48**, 1779–1801.
- Hafner C. and Herwartz H. 2006 A Lagrange multiplier test for causality in variance. *Economics Letters* **93**(1), 137–141.
- Hansen P., Huang R., and Shek H. 2012 Realized GARCH: a joint model for returns and realized measures of volatility. *Journal of Applied Econometrics* **27**(6), 877–906.
- Higgins M. and Bera A. 1992 A class of nonlinear ARCH models. *International Economic Review* **33**, 137–158.
- Kleiber C. and Zeileis A. 2008 *Applied Econometrics with R*. Springer-Verlag, New York.
- Nakatani T. 2010 *Four Essays on Building Conditional Correlation GARCH Models* PhD thesis Stockholm School of Economics.
- Nakatani T. 2014 *ccgarch: An R Package for Modelling Multivariate GARCH Models with Conditional Correlations*. R package version 0.2.3.
- Nakatani T. and Teräsvirta T. 2009 Testing for volatility interactions in the constant conditional correlation GARCH model. *Econometrics Journal* **12**(1), 147–163.
- Nakatsuma T. 2000 Bayesian analysis of ARMA-GARCH models: A markov chain sampling approach. *Journal of Econometrics* **95**(1), 57–69.
- Nelson D. 1991 Conditional heteroscedasticity in asset returns: A new approach. *Econometrica* **59**(2), 347–370.
- Pentula S. 1986 Modelling the persistence of conditional variances: A comment. *Econometric Reviews* **5**, 71–74.

- Pfaff B. 2012 *gogarch: Generalized Orthogonal GARCH (GO-GARCH) models*. R package version 0.7-2.
- Schwert G. 1990 Stock volatility and the crash of '87. *Review of Financial Studies* **3**(1), 77–102.
- Stock J. and Watson M. 2007 *Introduction to Econometrics*. Addison Wesley, Boston.
- Sucarrat G., Grönneberg S., and Escribano A. 2013 Estimation and inference in univariate and multivariate log-GARCH models when the conditional density is unknown. Working Paper 49344, University of Munich, Munich.
- Taylor S. 1986 *Modeling Financial Time Series*. John Wiley, New York.
- Trapletti A. and Hornik K. 2016 *tseries: Time Series Analysis and Computational Finance*. R package version 0.10-35.
- van der Weide R. 2002 GO-GARCH: A multivariate generalized orthogonal GARCH model. *Journal of Applied Econometrics* **17**(5), 549–564.
- Würtz D. and Chalabi Y. 2013 *fGarch: Rmetrics—Autoregressive Conditional Heteroskedastic Modelling*. R package version 3010.82.
- Ye Y. 1987 *Interior Algorithms for Linear, Quadratic, and Linearly Constrained Non-Linear Programming* PhD thesis Department of ESS, Stanford University.
- Zakoian J. 1994 Threshold heteroscedasticity models. *Journal of Economic Dynamics and Control* **15**, 931–955.

9

Modelling dependence

9.1 Overview

In the previous chapters, alternatives to the Gaussian model have been put forward for assessing market price risks of single financial instruments. The potential losses arising from holding a position in an asset were interpreted as a random variable, and unconditional as well as conditional risk measures were derived.

In this chapter the topic of financial risk modelling in the context of multiple financial instruments is addressed. Section 9.2 introduces the correlation coefficient between two assets and investigates its appropriateness as a measure of dependence between two assets. Section 9.3 discusses alternative measures of dependence, namely the use of rank correlations and the concept of the copula. Section 9.4 provides a synopsis of the R packages that specifically include copula modelling. Finally, Section 9.5 shows how copula models can be fruitfully combined with the techniques outlined in Chapters 6–8. In particular, a copula-GARCH model is proposed for measuring the market risk of a portfolio.

9.2 Correlation, dependence, and distributions

The computation and usage of Pearson’s correlation coefficient is quite ubiquitous in the quantitative analysis of financial markets. However, applied quantitative researchers are often unaware of the pitfalls involved in careless application and usage of correlations as a measure of risk. It is therefore appropriate to investigate this dependence concept in some detail and point out the shortcomings of this measure.

Pearson's correlation coefficient between two random variables X_1 and X_2 with finite variances is defined as

$$\rho(X_1, X_2) = \frac{\text{COV}(X_1, X_2)}{\sqrt{\text{VAR}(X_1)}\sqrt{\text{VAR}(X_2)}}, \quad (9.1)$$

where $\text{COV}(X_1, X_2) = E((X_1 - E(X_1))(X_2 - E(X_2)))$ denotes the covariance between the random variables and $\text{VAR}(X_1)$ and $\text{VAR}(X_2)$ are their variances, respectively. As is evident from (9.1), the correlation coefficient is a scalar measure. In the case of a perfect linear dependence the value of this measure is $\rho(X_1, X_2) = |1|$, where the perfect linear dependence is given by $X_2 = \beta + \alpha X_1$, with $\alpha \in \mathbb{R} \setminus 0, \beta \in \mathbb{R}$, otherwise the correlation coefficient can take values in the interval $-1 < \rho(X_1, X_2) < 1$. Furthermore, the linear correlation coefficient is invariant with respect to linear affine transformations of the random variables X_1 and X_2 : $\rho(\alpha_1 X_1 + \beta_1, \alpha_2 X_2 + \beta_2) = \text{signum}(\alpha_1 \alpha_2) \rho\{X_1, X_2\}$ for $\alpha_1, \alpha_2 \in \mathbb{R} \setminus 0, \beta_1, \beta_2 \in \mathbb{R}$.

Hence, Pearson's correlation coefficient depicts the strength of a linear relationship between two random variables. The logical reasoning that $\rho(X_1, X_2) = 0$ implies independence between X_1 and X_2 is only valid in the case of multivariate elliptically distributed random variables. Put differently, lack of correlation is only a sufficient condition for independence and only in the case of elliptical distributions can the two concepts be used interchangeably. Elliptic distributions are characterized by the fact that the points of the density function which yield the same values represent an ellipse; that is, horizontal cuts through the probability mass are elliptically shaped.

The implication for multivariate risk modelling is that only in the case of jointly elliptically distributed risk factors can the dependence between these be captured adequately by the linear correlation coefficient. Given the stylized facts of financial market returns this assumption is barely met. It should be pointed out at this point that with respect to risk modelling one is usually more concerned with the dependence structure in the tail of a multivariate loss distribution than with an assessment of the overall dependence, but a correlation measure just depicts the latter, in principle.

Furthermore, the linear correlation coefficient is not invariant with respect to nonlinear (e.g., log-transformed) random variables, or if nonlinear (e.g., quadratic or cubic) dependence between variables exists. This fact will be illustrated with the following two examples.

First, a standardized normal random variable $X_1 \sim N(0, 1)$ with quadratic dependence $X_2 = X_1^2$ is investigated. Between these two variables a direct relationship is evident; for a given realization x_1 of X_1 the value for x_2 can be concluded. However, if one were to calculate the linear correlation coefficient between these two variables, the result would be $\text{COV}(X_1, X_2) = E(X_1 \cdot (X_1^2 - 1)) = E(X_1^3) - E(X_1) = 0$, because the skewness of normally distributed random variables is zero.

The purpose of the second example is to highlight the fact that the correlation coefficient depends on the marginal distributions of the random variables in question and the possibility that the correlation coefficient cannot take all values in the interval $-1 \leq \rho \leq 1$ if one views the multivariate distribution of these random variables.

Given two log-normal random variables $X_1 \sim \log N(0, 1)$ and $X_2 \sim \log N(0, \sigma^2)$ with $\sigma > 0$, the feasible range of values for the correlation coefficient has a lower bound of $\rho_{\min} = \rho(e^Z, e^{-\sigma Z})$ and an upper bound of $\rho_{\max} = \rho(e^Z, e^{\sigma Z})$, where $Z \sim N(0, 1)$. The sign of the exponent depends on whether the random variables move in the same direction or opposite directions. In the first case a positive dependence (*co-monotonicity*) results, and in the latter a negative one (*counter-monotonicity*). The lower bound of the correlation coefficient is therefore given by

$$\rho_{\min} = \frac{e^{-\sigma} - 1}{\sqrt{(e - 1)(e^{\sigma^2} - 1)}} \quad (9.2)$$

and the upper bound is determined by

$$\rho_{\max} = \frac{e^{\sigma} - 1}{\sqrt{(e - 1)(e^{\sigma^2} - 1)}}. \quad (9.3)$$

As is evident from (9.2) and (9.3), these bounds depend only on the variance of the marginal distribution for X_2 . Here, the lower bound is greater than -1 and the upper bound coincides with the case of perfect dependence only when $\sigma = 1$, and is less than 1 in all other instances. Incidentally, with ever increasing variance the correlation coefficient approaches zero.

With respect to risk modelling the following conclusions can be drawn. For instance, a correlation coefficient as high as 0.2 would only indicate a weak relationship between two risk factors. Indeed, the opposite can be true and therefore a tentative diversification gain in the context of a portfolio is void when capital is allocated to the respective assets. By converse reasoning, the information of a correlation coefficient as high as 0.7 between two log-normally distributed risk factors with expected values of zero and variances of 1 and 4 , respectively, is void. If these two random variables are jointly distributed, such a high value for the correlation coefficient cannot be achieved.

In summary, the dependence between financial instruments can only be depicted correctly with the linear correlation coefficient if these are jointly elliptically distributed. It was also shown that the value of the correlation coefficient depends on the marginal distributions and that not all values in the range $[-1, 1]$ are attainable. In particular, a perfect positive dependence between two random variables is not interchangeable with a correlation coefficient of one, but the value can be much lower and at the limit a value of zero is obtained. Conversely, a correlation coefficient of zero does not lead to a conclusion of independence. This reasoning is only valid in the case of jointly elliptical distributions. Last, but not least, it should be pointed out that the linear correlation coefficient is only defined for pairs of random variables with finite variance. This assumption may be violated for risk factors with great probability masses located in the tails. As a consequence, the application of alternative concepts for measuring the dependence between risk factors becomes necessary. In addition to the rank correlation coefficients of Kendall and Spearman, the copula concept is a promising route to take. The methods will be introduced in the next section.

9.3 Copulae

9.3.1 Motivation

The copula approach was introduced by Sklar (1959). Detailed textbook expositions can be found in Schweizer and Sklar (1983), Nelsen (2006), Joe (1997), and McNeil et al. (2005). However, only since the mid-1990s have copulae been used as a tool for modelling dependencies between assets in empirical finance. The word “copula” derives from the Latin verb *copulare* and means to “bond” or “tie.” What exactly is bonded by a copula will be the subject of the rest of this subsection.

The marginal distributions of jointly distributed random variables as well as their dependence are contained in their joint distribution function,

$$F(x_1, \dots, x_d) = \mathbb{P}[X_1 \leq x_1, \dots, X_d \leq x_d]. \quad (9.4)$$

It was shown in the previous section that in the case of normal random variables the value of the correlation coefficient is dependent upon the marginal distributions. This measure may take values less than 1 even if there is perfect dependence between the two random variables. Hence, it is necessary to separate the marginal distributions from the dependence structure between the random variables. This separation can be achieved by means of a copula.

For this purpose an element-wise transformation of the random vector $X = (X_1, \dots, X_d)$ is required, such that the resulting variables follow a uniform distribution $U(0, 1)$. Given the assumption that all distribution functions F_1, \dots, F_n of the random vector X are continuous, this projection is given by the probability integral: $\mathbb{R}^d \mapsto \mathbb{R}^d, (x_1, \dots, x_n)^t \mapsto (F_1(x_1), \dots, F_d(x_d))^t$. The joint density function \mathcal{C} of the transformed random variables $(F_1(x_1), \dots, F_d(x_d))^t$ is the copula of the vector $X = (X_1, \dots, X_d)$:

$$F(x_1, \dots, x_d) = \mathbb{P}[F_1(X_1) \leq F_1(x_1), \dots, F_d(X_d) \leq F_d(x_d)] \quad (9.5)$$

$$= \mathcal{C}(F_d(x_d), \dots, F_d(x_d)). \quad (9.6)$$

Hence, a copula is the distribution function in \mathbb{R}^d space of a d -element random vector with standard uniform marginal distributions $U(0, 1)$. Alternatively, a copula can be interpreted as a function that maps from the d -dimensional space $[0, 1]^d$ into the unit interval: $\mathcal{C} : [0, 1]^d \mapsto [0, 1]$. In this interpretation further conditions must be met by the function \mathcal{C} in order to qualify as a copula. The dependence structure of the random vector X is embodied in the copula. If the marginal distributions are continuous, then the copula is uniquely defined. It is therefore possible to employ different distributions for the random variables (i.e., risk factors) as marginals and capture the dependences between those with a copula. In principle, all of the previously introduced distributions would classify as potential candidates for modelling the risk factors in a portfolio’s loss function.

9.3.2 Correlations and dependence revisited

It was shown in Section 9.2 that the linear correlation coefficient does not capture dependencies well in the case of non-elliptically distributed random variables. Furthermore, while this measure depicts the overall dependence, in the assessment of the riskiness of a portfolio what matters most is the dependence in the tail of the joint distribution. Hence, in this subsection two further concepts for capturing the dependence between risk factors are introduced, namely, concordance and tail dependence. The first concept entails the rank correlation coefficients of Kendall and Spearman.

The concept of concordance will be illustrated by the following example. Suppose one has two realizations (x_i, y_i) and (x_j, y_j) of a two-dimensional random vector (X, Y) . If one plots these two points in a Cartesian coordinate system and joins them by a line, one says that a positive dependence exists if the line slopes upward (concordance) and a negative one prevails if the line slopes downward (discordance). Equivalently, if one calculates the product of the pairwise differences $(x_i - x_j)(y_i - y_j)$ then concordance obtains if the result is positive and discordance if the result is negative. It is now assumed that the continuous and independent random vectors (X_1, Y_1) and (X_2, Y_2) each have joint density functions H_1 and H_2 with copulae \mathcal{C}_1 and \mathcal{C}_2 . Let Q denote the difference in probability for the cases of concordance and discordance of (X_1, Y_1) and (X_2, Y_2) :

$$Q = \mathbb{P}((X_1 - X_2)(Y_1 - Y_2) > 0) - \mathbb{P}((X_1 - X_2)(Y_1 - Y_2) < 0). \quad (9.7)$$

The probability expression of (9.7) can be cast in terms of copulae. A detailed derivation can be found in Genest and Favre (2007).

$$Q = Q(\mathcal{C}_1, \mathcal{C}_2) = 4 \int_0^1 \int_0^1 \mathcal{C}_2(u, v) \, d\mathcal{C}_1(u, v) - 1. \quad (9.8)$$

If the two random vectors share the same dependence, then (9.8) can be simplified to:

$$Q = Q(\mathcal{C}_1, \mathcal{C}_2) = 4 \int_0^1 \int_0^1 \mathcal{C}(u, v) \, d\mathcal{C}(u, v) - 1. \quad (9.9)$$

Kendall's rank correlation coefficient, referred to as "tau" and denoted by ρ_τ , is defined as

$$\rho_\tau(X_1, X_2) := \mathbb{E}[\text{sign}((X_1 - X_2)(Y_1 - Y_2))], \quad (9.10)$$

where \mathbb{E} denotes the expectation operator. The definition of Kendall's tau in (9.10) is identical to the probability expression of (9.7). Therefore, Kendall's tau can be interpreted as a probability integral of the copulae (see (9.8) and (9.9)) and depends only on these. Like the linear correlation coefficient, Kendall's tau is invariant with respect to monotone transformations of the random variables and measures an overall dependence between them. In contrast to the linear correlation coefficient, rank correlations are less sensitive to outliers or extreme observations. Furthermore, it can be shown that for elliptical copulae $\rho_\tau(X_1, X_2) = \frac{2}{\pi} \arcsin \rho$, where ρ denotes Pearson's correlation coefficient.

We now turn to Spearman's rank correlation coefficient, ρ_s . This coefficient is defined for the bivariate case as $\rho_s = \text{COR}(F_1(X_1), F_2(X_2))$. The dependence on a copula can be expressed analogously to that for Kendall's tau shown above:

$$\rho_s(X_1, X_2) = 12 \int_0^1 \int_0^1 \mathcal{C}(u, v) \, du \, dv - 3. \quad (9.11)$$

The following relation holds between Spearman's and Pearson's correlations: $\rho_s(X_1, X_2) = \frac{6}{\pi} \arcsin \frac{\rho}{2}$. The difference between the two concepts might appear to be negligible, but Spearman's rank correlation coefficient possesses the characteristic that $E(\rho_s) = \pm 1$ if there is a functional relationship between the two random variables (in this case the copula would coincide with one of the two Fréchet–Hoeffding bounds). In contrast, $E(\rho_p) = \pm 1$ holds only for a linear relationship between the two random variables. In addition, ρ_s is a distribution parameter and as such always defined. Pearson's ρ_p , however, cannot be theoretically derived from all distributions.

An alternative to the concordance concept and rank correlations for modelling dependencies is the concept of tail dependencies, which is of great importance in the risk assessment of a portfolio. The focus of the dependence measure is solely on the tails of the joint distribution. It is therefore possible to determine the likelihood that for a given loss in one financial instrument an equivalent or even greater loss would occur in a second holding. Formally, the upper and lower tail dependencies for two random variables (X, Y) with marginal distributions F_X and F_Y are defined as

$$\lambda_u = \lim_{q \nearrow 1} \mathbb{P}(Y > F_Y^{-1}(q) | X > F_X^{-1}(q)), \quad (9.12a)$$

$$\lambda_l = \lim_{q \searrow 0} \mathbb{P}(Y \leq F_Y^{-1}(q) | X \leq F_X^{-1}(q)). \quad (9.12b)$$

In other words, tail dependence is a conditional probability expression, namely, the likelihood of observing a large (small) value for Y , given a large (small) value for X . If $\lambda_u > 0$ then there is upper tail dependence between the two random variables, and if $\lambda_l > 0$ then there is lower tail dependence. For the cases of $\lambda_u = 0$ or $\lambda_l = 0$, it is said that (X, Y) are asymptotically independent at the upper or lower end of the distribution. According to Bayes' formula, the tail dependencies for continuous distribution functions are given as:

$$\begin{aligned} \lambda_l &= \lim_{q \searrow 0} \frac{\mathbb{P}(Y \leq F_Y^{-1}(q) | X \leq F_X^{-1}(q))}{\mathbb{P}(X \leq F_X^{-1}(q))} \\ &= \lim_{q \searrow 0} \frac{\mathcal{C}(q, q)}{q} \end{aligned} \quad (9.13)$$

for the lower tail dependence coefficient, and analogously

$$\lambda_u = 2 + \lim_{q \nearrow 0} \frac{\mathcal{C}(1 - q, 1 - q) - 1}{q} \quad (9.14)$$

for the upper tail dependence coefficient.

9.3.3 Classification of copulae

The previous two subsections gave a brief outline of the concept of the copula and how it relates to rank correlations. In this subsection copulae are introduced more formally and their classification into sub-categories is described. It will be shown how tail dependence as a measure of risk can be deduced from them. It is beyond the scope of this book to provide a detailed exposition of copulae; the aim is rather to better grasp the underlying concept and apply copulae as a tool for modelling dependencies. Thorough textbook expositions can be found in Joe (1997) and Nelsen (2006).

In general, a copula is expressed in the form of a multivariate density function

$$\mathcal{C}(\mathbf{u}) = \mathbb{P}(U_1 \leq u_1, U_2 \leq u_2, \dots, U_d \leq u_d) = \int_0^{u_1} \cdots \int_0^{u_d} c(\mathbf{u}) \, d\mathbf{u}, \quad (9.15)$$

where \mathbf{u} is a d -dimensional random vector.

The implicit form of a copula for a continuous distribution function F with continuous marginal distributions F_1, \dots, F_d can therefore be written as

$$\mathcal{C}(\mathbf{u}) = \frac{f(F_1^{-1}(u_1), \dots, F_d^{-1}(u_d))}{f_1(F_1^{-1}(u_1)) \cdots f_d(F_d^{-1}(u_d))}. \quad (9.16)$$

Equivalently, (9.16) can be written as

$$c(F_1(x_1), \dots, F_d(x_d)) = \frac{f(x_1, \dots, x_d)}{f_1(x_1) \cdots f_d(x_d)}. \quad (9.17)$$

As can be seen from this equation, a d -dimensional density function can be represented as a multiplicative concatenation of the copula with the marginal distributions.

It was shown above that certain conditions must be met by a copula with respect to its domain. These Fréchet–Hoeffding bounds were also introduced in the context of the linear correlation coefficient for jointly normally distributed random variables. These bounds will now be derived for arbitrary copulae. Two uniformly distributed random variables U_1 and U_2 are assumed. Between these two variables three extreme cases of dependence can be distinguished: concordance, independence, and discordance. First, the case of perfect positive dependence will be investigated (e.g., $U_1 = U_2$). The copula for this case could then be written as

$$\mathcal{C}(u_1, u_2) = \mathbb{P}(U_1 \leq u_1, U_2 \leq u_2) = \min(u_1, u_2). \quad (9.18)$$

The copula would also be valid if a monotone transformation is applied to the random variables. As an intermediate step between this case and that of discordant random variables, independence will be investigated next. The copula is then the product of the two random variables $\mathcal{C}(u_1, u_2) = u_1 \cdot u_2$. Symbolically one can imagine this as flipping a coin twice. The joint density function of independent random variables is the product of the respective marginal distributions, which equals the independence copula:

$$\begin{aligned}\mathcal{C}(\mathbf{u}) &= \prod_{i=1}^d u_i = \mathbb{P}(U_1 \leq u_1) \cdots \mathbb{P}(U_d \leq u_d) \\ &= \mathbb{P}(U_1 \leq u_1, \dots, U_d \leq u_d) = f(\mathbf{u}).\end{aligned}\quad (9.19)$$

The case of discordance occurs if $U_2 = 1 - U_1$ is fulfilled. The copula is then given by

$$\begin{aligned}\mathcal{C}(u_1, u_2) &= \mathbb{P}(U_1 \leq u_1, 1 - U_1 \leq u_2) \\ &= \mathbb{P}(U_1 \leq u_1, 1 - u_2 \leq U_1) = u_1 + u_2 - 1,\end{aligned}\quad (9.20)$$

and it is zero for all other instances. The domain of a copula for the d -dimensional case $\mathcal{C}(\mathbf{u}) = \mathcal{C}(u_1, \dots, u_d)$ is then bounded by

$$\max\left\{\sum_{i=1}^d u_i + 1 - d, 0\right\} \leq \mathcal{C}(\mathbf{u}) \leq \min\{u_1, \dots, u_d\} \quad (9.21)$$

and the lower bound is only attainable for the bivariate case.

Hitherto, the discussion of the copula concept has been quite abstract, in terms of its main characteristics and bounds. We will now classify copulae into two broad categories, namely *Archimedean* copulae on the one hand and distribution-based copulae on the other. Within the latter group of copulae the dependence between the random variables will be captured implicitly by a distribution parameter. For instance, the bivariate case of a Gauss copula is defined as

$$\begin{aligned}\mathcal{C}_\rho^{GA}(u_1, u_2) &= \Phi_\Sigma(\Phi^{-1}(u_1), \Phi^{-1}(u_2)) \\ &= \int_{-\infty}^{\Phi^{-1}(u_1)} \int_{-\infty}^{\Phi^{-1}(u_2)} \frac{1}{2\pi\sqrt{1-\rho^2}} \\ &\quad \exp\left(-\frac{x_1^2 - 2\rho x_1 x_2 + x_2^2}{2(1-\rho^2)}\right) dx_1 dx_2.\end{aligned}\quad (9.22)$$

The distribution function in (9.22) depends on the variance-covariance matrix Σ . Because monotone transformations do not alter the dependence structure, the variance-covariance matrix can be replaced by the correlation matrix. Hence, the Gauss copula includes the extreme cases of *co-monotonicity* ($\rho = 1$), the independence copula ($\rho = 0$), and *counter-monotonicity* ($\rho = -1$). A characteristic of the Gauss copula is that the tail dependence is zero. Therefore it has limited application to risk modelling.

As a second example, we will now focus on the Student's t copula, which is defined as

$$\mathcal{C}_{\rho, v}^t(u_1, u_2) = t_{\Sigma, v}(t_v^{-1}(u_1), t_v^{-1}(u_2))$$

$$= \int_{-\infty}^{t_v^{-1}(u_1)} \int_{-\infty}^{t_v^{-1}(u_1)} \frac{1}{2\pi\sqrt{1-\rho^2}} \left[1 + \frac{x_1^2 - 2\rho x_1 x_2 + x_2^2}{v(1-\rho^2)} \right]^{-(v+2)/2} dx_1 dx_2, \quad (9.23)$$

where v denotes the degrees of freedom as an additional distribution parameter. In contrast to the Gauss copula, the coefficient of tail dependence for the Student's t copula can be computed as

$$\lambda_u = \lambda_l = 2t_{v+1}(-\sqrt{v+1}\sqrt{(1-\rho)/(1+\rho)}). \quad (9.24)$$

The advantages of distribution-based copulae are twofold: first, and most importantly, their simplicity; and second, that simulations based upon these copula models can be carried out easily. Among the disadvantages is the fact that many parameters have to be estimated. Furthermore, in the case of elliptical distributions their symmetry is a problem, given the stylized fact that the empirical distributions of loss functions are ordinarily skewed. Finally, a closed form for distribution-based copulae cannot be derived. Archimedean copulae do not share the latter disadvantage. An Archimedean copula is defined as

$$\mathcal{C}(u_1, u_2) = \psi^{-1}(\psi(u_1) + \psi(u_2)), \quad (9.25)$$

where ψ is the copula-generating function. Two prominent examples of this category are the Clayton and Gumbel copulae. The copula-generating function for the Clayton copula is given by $\phi(t) = (t^{-\delta} - 1)/\delta$, with $\delta \in (0, \infty)$. For $\delta \rightarrow \infty$ perfect dependence results, and for $\delta \rightarrow 0$ independence. If the generating function is inserted into (9.25), the Clayton copula is then given by

$$\mathcal{C}_\delta^{CL} = \psi^{-1}(\psi(u_1) + \psi(u_2)) = (u_1^{-\delta} + u_2^{-\delta} - 1)^{1/\delta}. \quad (9.26)$$

The Clayton copula possesses lower tail dependence. The relevant coefficient is calculated according to $\lambda_l = 2^{-1/\delta}$.

The copula-generating function for the Gumbel copula is given as $\psi(t) = (-\ln t)^\theta$, with the parameter restriction $\theta \geq 1$. Inserting this generating function into (9.25) leads to the Gumbel copula,

$$\mathcal{C}_\theta^{GU} = \psi^{-1}(\psi(u_1) + \psi(u_2)) = \exp(-[(-\ln u_1)^\theta + (-\ln u_2)^\theta]^{1/\theta}). \quad (9.27)$$

The Gumbel copula possesses upper tail dependence. Perfect dependence exists for $\theta \rightarrow \infty$ and independence for $\theta \rightarrow 1$. The coefficient of tail dependence can be determined according to $\lambda_u = 2 - 2^{1/\theta}$.

The main advantages of Archimedean copulae are twofold: these copulae do have a closed form and manifold dependence structures can be modelled with them. A drawback is their quite complex structure for higher dimensions. Ordinarily this is circumvented through nested/hierarchical modelling designs.

In principle, the unknown parameters of a copula can be estimated in two ways. The first procedure is fully parametric. Because for multivariate distribution models derived from a copula of higher dimensions this procedure can be quite burdensome, Joe and Xu (1996) and Shih and Louis (1995) proposed a two-step estimation. Here, the unknown parameters for the assumed models of the marginal distributions are estimated first. Based upon these fitted models, the pseudo-uniform variables are retrieved from the inverse distribution functions. These can then be used for maximizing the likelihood (copula). This approach is, for obvious reasons, often termed *inference functions for margins*. The second procedure is based upon a semi-parametric approach. In contrast to the first procedure, no models are assumed for the marginals, but rather the empirical distribution functions are employed to retrieve the pseudo-uniform variables. These are then used for maximizing the pseudo-likelihood. The parameters of the copula are determined by means of numerical optimization techniques. For the case of a Student's t copula a simplification results if Kendall's tau is calculated first and then the pseudo-likelihood has to be maximized only with respect to the degrees of freedom parameter v . For the bivariate Archimedean copulae of Clayton and Gumbel type, the following simplifications are given, where $\hat{\rho}_\tau$ denotes the empirical Kendall rank correlation (see, for instance, Genest and Favre 2007):

$$\hat{\delta} = \frac{2\hat{\rho}_\tau}{1 - \hat{\rho}_\tau}, \quad \hat{\theta} = \frac{1}{1 - \hat{\rho}_\tau}. \quad (9.28)$$

It was argued in Section 9.2 that the linear correlation coefficient is not well suited to capturing dependencies between loss factors, namely that quite distinct shapes of dependence can yield the same value for this coefficient. We conclude this section by highlighting this fact. Figure 9.1 shows four scatter diagrams. The sample size for each of the data pairs has been set to 2000 observations and the samples have been generated such that the correlation coefficient for each of the four sets equals 0.7. From these scatter diagrams, it is quite evident that the dependence structures vary and that the information content of the correlation coefficient with respect to the tail dependence is quite limited. In the case of the Gumbel copula a concentration of positive pairs and in the case of the Clayton copula a clustering of negative pairs is evident. The Student's t copula is characterized by a wider dispersion of pairs with opposite signs.

9.4 Synopsis of R packages

9.4.1 The package BLCOP

The package **BLCOP** (see Gochez et al. 2015) implements the Black–Litterman approach (see Black and Litterman 1990) to portfolio optimization and the framework of copula opinion pooling (see Meucci 2006a,b, 2010). It is presented in this chapter because of the latter feature, but an application is deferred to Part III of the book (Chapter 13). The package is written purely in R, and S4 classes and methods are

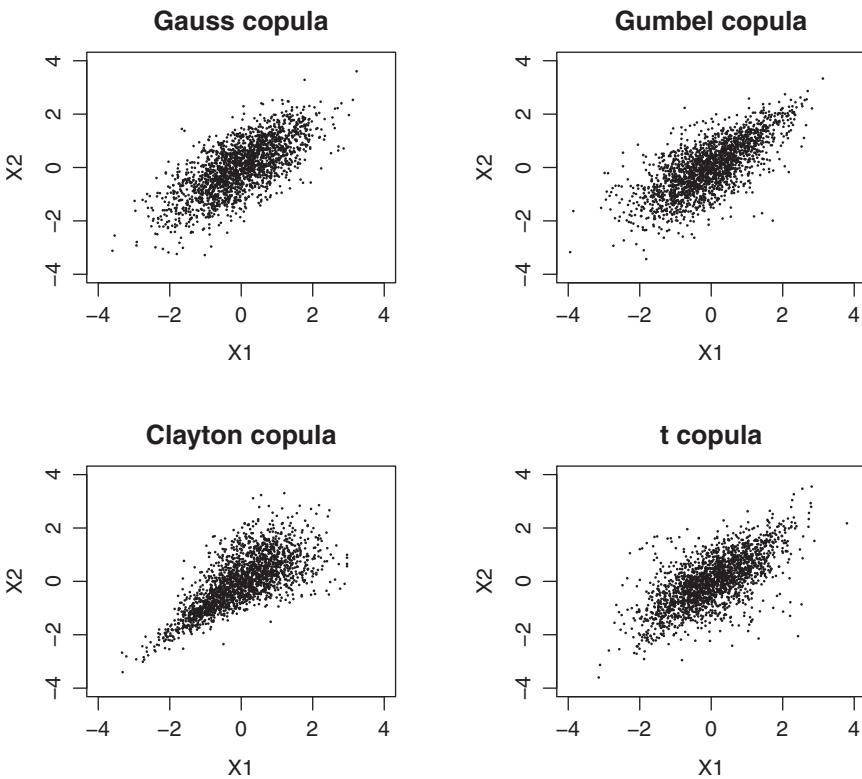


Figure 9.1 *Different types of copulae.*

employed. The package depends on the packages **methods**, **MASS** (see Venables and Ripley 2002), and **quadprog**—the latter is a non-standard package and will be installed automatically if not found in the search path. **BLCOP** is shipped with a **NAMESPACE** file and there is a vignette to guide the user. In addition, a unit testing framework is implemented such that the proper working of the package’s functions can be controlled. For this, the package **RUnit** (see Burger et al. 2015) needs to be installed. **BLCOP** is shipped with three data sets: **monthlyReturns**, containing the monthly returns of six US stocks, and **sp500Returns** and **US13wTB**, containing the monthly returns of the S&P 500 Index and the 13-week Treasury bill interest rates.

With regard to the copula opinion pooling framework the class **COPViews** is defined. The various opinions about certain assets are stored in objects of this class. This kind of object can be generated with a function of the same name. A **deleteViews()** and a **show()** method are available. The former facilitates the removal of views from such an object, whereas the latter method produces a user-friendly print of the views. Views (i.e., opinions about certain assets) can be added to an existing **COPViews** object with the function **addCOPViews()**. As an aside, a graphical user interface—by means of invoking R’s built-in data editor—for accessing and manipulating views is made available by the function **createCOPViews()**.

The posterior distribution is generated by means of Monte Carlo simulations through the function `COPPosterior()`. Here, a multivariate distribution object for the overall market distribution has to be provided as well as an object of class `COPViews`. The function returns an object of class `COPResult` for which methods are defined that display the prior and marginal prior distributions of each asset (`densityPlot()`), print information about the posterior results (`show()`), and generate optimal prior and posterior portfolios (`optimalPortfolios.fPort()`). The last function utilizes routines that are contained in the package **fPortfolio** (see Würtz et al. 2014).

In addition to these cornerstone classes and methods that deal with the copula opinion pooling framework, the package also offers utility functions for retrieving and manipulating slots from objects of the two classes presented.

9.4.2 The package `copula`

In the package **copula** the copula concept is implemented in a broad and self-contained manner (see Hofert and Mächler 2011; Hofert et al. 2015; Kojadinovic and Yan 2010; Yan 2007). The package is considered to be a core package of the CRAN “Distributions” Task View and is also listed in the “Finance” and “Multivariate” Task Views. S4 classes and methods not only for elliptical and Archimedean but also for extreme value copulae are defined, enabling the user to estimate copulae and conduct statistical tests with respect to the appropriateness of a chosen copula. Indeed, applications of a goodness-of-fit tests are available in the form of R DEMO files, besides other R code examples. The computationally burdensome routines are interfaced from code written in the C language. Finally, the package contains four data sets originating in the fields of insurance, finance, and chemistry. Incidentally, the finance data set is used in McNeil et al. (2005, Chapter 5).

The virtual class `copula` is defined to deal with copulas. This class is extended by classes for elliptical, Archimedean, and extreme value copulae—`ellipCopula`, `archmCopula`, and `evCopula`, respectively. Particular classes for copulae belonging to and inheriting from one of these three kinds are defined, too. On the one hand the peculiarities of each specific copula can be taken into account, and on the other hand methods need only be defined for the parent class in most cases. Copula objects can be created swiftly by employing creator functions of the same name as the kind of copula. The specific copula is determined by the common argument `family` within these functions. The proper creation of objects belonging to these classes is controlled by validity functions. Specific copula objects can also be generated by calling the function `fooCopula()`, where `foo` is a placeholder for the name of the copula; for example, a normal copula can be created by a call to `normalCopula()` and a Gumbel copula by utilizing `gumbelCopula()`.

Methods for returning values of the density and distribution functions for objects of class `copula` are available as well as the generation of random numbers. The R naming convention first introduced in Section 6.4 is used whereby the name of the copula is prefixed with `[d]` for the density, `[p]` for the distribution, and `[r]` for random number generation. Contour and perspective plots of a `copula` object can be drawn

with the defined methods `contour()` and `persp()`, respectively. Furthermore, methods calculating the measures of association due to Kendall (`tau()`), Spearman (`rho()`), and the tail index (`tailIndex()`) are available. It is possible to calibrate the parameters of a copula for a given rank correlation with the functions `iTau()` and `iRho()` for the Kendall and Spearman coefficients, respectively. The generating functions and their inverses for Archimedean copulae can be accessed through the functions `genFun()` and `genInv()`. In addition, the first and second derivatives of the generating function for this group of copulae are termed `genFunDer1()` and `genFunDer2()`. Similarly, the generating function for extreme value copulae is `Afun()`, and a data frame of the first and second derivatives thereof can be returned by the function `AfunDer()`.

Estimation of the unknown copula parameters is implemented in the function `fitCopula()`. The user can choose between maximum likelihood, maximum pseudo-likelihood, and the inversion of Kendall's tau or Spearman's ρ as estimation methods. The function returns an object of class `fitCopula`, for which `show()` and `summary()` methods are defined. The maximization of the likelihood is carried out with the function `optim()`. The user can supply the optimizer to be used as well as a list of control parameters in accordance with this function. It was briefly mentioned in Section 9.3.3 that it can be quite burdensome to estimate all of the unknown parameters for higher dimensions and hence that a two-stage estimation should be conducted (i.e., inference functions for margins). Yan (2007, Section 4) provides an example showing how this can be achieved with the function `fitCopula()`.

The second cornerstone class defined in `copula` is `mvdc`, which contains the relevant items for multivariate distributions constructed from copulae. Objects belonging to this class can be swiftly generated with the function `mvdc()`. The arguments of the function refer to the copula and marginal distributions to be used, and a list object containing the parameters for each of the marginal distributions. Similarly to the class `copula`, methods for the density and distribution function as well as the generation of random numbers are available. The fitting of a multivariate distribution model defined by a copula is implemented in the function `fitMvdc()`, which returns an object of class `fitMvdc`. The unknown parameters are estimated by applying the ML principle via `optim()`, and the log-likelihood is defined through the function `loglikMvdc()`. Similarly to objects of class `fitCopula`, `show()` and `summary()` methods are defined for the results of `fitMvdc()`.

Furthermore, goodness-of-fit tests and tests for independence and extreme value dependence are implemented. The appropriateness of a certain copula can be assessed with the functions `gofCopula()` and `gofEVCopula()`. Whether the assumption of independence for a given multivariate sample holds can be tested with the functions `indepTest()` and/or `multIndepTest()`. The outcome of the former function can be inspected graphically using a *dependogram*. This kind of figure is created by a function of the same name. Similarly, the functions `serialIndepTest()` and `multSerialIndepTest()` can be utilized to test serial independence of the data set. These tests are described in Deheuvels (1979), Genest and Rémillard (2004), Genest et al. (2006), and Genest et al. (2007). Three different tests of extreme value

dependence are also implemented. A bivariate test based on Pickand's dependence function is available through the function `evTestA()`, and one based on a Kendall process is available as `evTestK()`. A large-sample test of multivariate extreme value dependence can be carried out by calling `evTestC()`. Further information about these statistical tests is provided in the help pages of the functions.

Compared to the first edition of this book, the package **nacopula** has been merged completely into **copula**. The sources of **nacopula** can still be accessed from the CRAN archive. As such, the approach of nesting Archimedean copulae (AC) is made available within **copula** using S4 classes and methods. The Archimedean copulae are interfaced from routines written in the C language. The usage of the available routines for specifying and estimating nested AC are detailed in a vignette (see Hofert and Mächler 2011). Furthermore, a standard testing framework for proper working of the methods is available.

The two cornerstone classes for (nested) AC are `acopula` and `nacopula` for single arbitrary and nested Archimedean copulae, respectively. In addition, the class `outer_nacopula` inherits from the latter class, but the two differ only with respect to their validation function. The copula families available are Ali–Mikhail–Haq, Clayton, Frank, Gumbel, and Joe. Objects of these copulae can be nested with the function `onacopula()`. Random variates of a nested copula object can be generated with the function `rnacopula()`, and for its children the function `rnch1d()` is used. Nested AC objects can be evaluated, that is, the probability is computed for a given d -dimensional vector of observations, with the function `pnacopula()`. Probabilities that refer to intervals are returned with the function `prob()`. Both functions refer to points/limits of the hypercube.

9.4.3 The package **fCopulae**

Similar to the package **copula**, the package **fCopulae** offers a unified treatment of copulae (see Würtz and Setz 2014). The package is part of the Rmetrics suite of packages and is considered a core package in the CRAN “Distributions” and “Finance” Task Views. It employs S4 as well as S3 classes and methods. A utility function for multidimensional numerical integration interfaces to a FORTRAN routine. The package is shipped with a NAMESPACE file, and a unit testing framework is implemented through the package **RUnit** (see Burger et al. 2015). Furthermore, **fCopulae** offers graphical user interfaces written in Tcl/Tk for displaying the impact of alternative copula parameter/settings. A drawback of the package is that only two-dimensional copulae are implemented, but the functions that address the skewed normal (`[dpr] mvsnrm()`) and skewed Student's t (`[dpr] mvst()`) multivariate distributions are not confined to this case. Incidentally, data can be fitted to these multivariate distributions by the function `mvfit()`.

Similarly to the package **copula**, the copulae in the package **fCopulae** are classified into three categories: Archimedean, elliptical, and extreme value. In addition, functions for investigating bivariate empirical copulae are made available, too. With regard to the Archimedean copulae a total of 22 variants are available. The density and distribution functions thereof and the generation of random numbers

can be accessed with the functions `[dpr]archmCopula()`. As stated in the previous paragraph, interactive displays of the density, the distribution function, and random variates can be generated with the functions `[dpr]archmSlider()`, respectively. Because of its importance, the density, the distribution function, and the generation of random variates of the Gumbel copula can be accessed by calling `[dpr]gumbelCopula()`. As measures of association, Kendall's tau, Spearman's ρ , and the tail coefficient can be computed with the functions `archmTau()`, `archmRho()`, and `archmTailCoeff()`, respectively. The latter can be plotted with the function `archmTailPlot()`. The fitting and simulation of AC can be conducted with the functions `archmCopulaFit()` and `archmCopulaSim()`. In contrast to the package **copula**, the optimization routine employed for estimating the unknown copula parameters is `nlminb()`. The ellipsis argument of `archmCopulaFit()` is passed down to this optimizer.

In contrast to the package **copula**, the Cauchy copula is available in addition to the normal and Student's t copulae, although all of these distribution-based copulae are confined to the two-dimensional case. The nomenclature for the density and distribution function and the generation of random variates follows the **R** naming convention in the same way as for the Archimedean copulae, namely, `[dpr]ellipticalCopula()`. Interactive displays thereof can be created through a **Tcl/Tk** graphical user interface which is started by calling `[dpr]ellipticalSlider()`. The type of elliptical copula is selected by the type argument pertinent to these functions. Similarly, Kendall's tau, Spearman's ρ , and the tail coefficient are computed by the functions `ellipticalTau()`, `ellipticalRho()`, and `ellipticalTailCoeff()`. For the latter measure of association a plot can be displayed via `ellipticalTailPlot()`. Needless to say, the tail coefficient is zero in the case of a normal copula. Bivariate elliptical copulae can be fitted and/or simulated with the functions `ellipticalCopulaFit()` and `ellipticalCopulaSim()`, respectively. Within the former function `nlminb()` is utilized as the optimizer and the ellipsis argument of the fitting function is passed down to it.

For extreme value copulae the same naming convention and set of functions are used as in the cases of Archimedean and elliptical copulae. The acronym used for this class of copulae is **ev**; for example, the functions `[dpr]evCopula()` refer to the density and distribution function and the generation of random variates. The package **fCopulae** implements not only the copulae available in **copula** but also the BB5 copula.

9.4.4 The package **gumbel**

The package **gumbel** is dedicated solely to the Gumbel copula (see Dutang 2015). It is contained in the CRAN "Distribution" Task View. The package is written purely in **R** and is accompanied by a vignette (in French only). Neither **S3** nor **S4** classes and methods are employed, but the functions are exported via the **NAMESPACE** mechanism. In addition to the density and distribution function and the generation of random variates, the generating function of the Gumbel copula and its

inverse are available. For the former group of functions the **R** naming convention is followed (i.e., `[dpr]gumbel()`). The generating function and inverse generating function are named `phigumbel()` and `invphigumbel()`, respectively. A strength of the package is the inclusion of functions for estimating the unknown parameters of the Gumbel copula and the margins through the methods of maximum likelihood (`gumbel.EML()`), canonical maximum likelihood (`gumbel.CML()`), inference functions for margins (`gumbel.IFM()`), and a moment-based method (`gumbel.MBE()`). The functions `optim()` and `optimize()` are utilized for numerically maximizing the objective functions. Unfortunately, only the estimates are returned as a vector object, hence statistical inference is not possible. An additional drawback of the implementation is that the marginal distributions must be either exponential or gamma for all estimation methods except canonical maximum likelihood.

9.4.5 The package QRM

The package **QRM** was introduced in Section 6.4.4 and encountered again in Section 7.3.7. For our present purposes, it contains functions for the Frank, Gumbel, Clayton, normal, and Student's t copulae. Here, the densities and the generation of random variates are implemented. The fitting of Archimedean copulae is conducted with the function `fit.AC()`. Numerical maximization of the likelihood is accomplished by `nlminb()`. The function returns a list object with the maximized value of the log-likelihood, the estimated coefficients, the standard errors thereof, and a logical flag indicating whether or not the algorithm converged. The parameters of the elliptical normal and Student's t copulae are estimated by means of the functions `fit.gausscopula()` and `fit.tcopula()`, respectively. It is advantageous to employ the function `fit.tcopula()` for higher-dimensional data by setting the argument `method` to either "Kendall" or "Spearman" when a t copula is used. In these cases a two-step estimation is carried out whereby first the rank correlations are determined, and then the optimization is done only with respect to the degrees-of-freedom parameter v .

9.5 Empirical applications of copulae

9.5.1 GARCH-copula model

Motivation

The stylized facts of daily returns were set out in Part I of this book. In particular, it was stated that the assumptions of an iid and normal return process are generally not met in practice. The stylized facts regarding "fat tails" and "volatility clustering" can be viewed as the flip side of these violated assumptions. As should be evident from this chapter the appropriate determination of the dependence between the financial instruments in a portfolio is of pivotal importance. It has been shown that the variance-covariance or correlation matrix is only appropriate for elliptically

distributed random variables. This assumption is as detrimental to the observed stylized facts for multivariate financial return series as the iid assumption is for the univariate case. Hence, the utilization of the copula concept has been advocated in this chapter.

That said, the question of the kind of model to employ in simultaneously addressing the univariate and multivariate stylized facts for market returns in the portfolio context arises. We are now in a position to put together the pieces from the previous chapters of this part of the book, that is, to combine GARCH and copula models. Incidentally, the modelling of the marginals could in principle also be achieved by applying extreme value theory or a generalized hyperbolic or generalized lambda distribution, but the advantages of employing volatility models instead are striking if one contrasts the GARCH–copula model with a “plain vanilla” variance-covariance approach and the assumption of normally distributed returns:

1. GARCH models possess a higher kurtosis than the normal distribution, hence the higher probability of witnessing extreme returns can be captured with these volatility models.
2. Risk measures based on the variance-covariance approach are unconditional in nature and therefore imply an iid process. It is therefore impossible to model volatility clusters. The GARCH models specifically capture this empirical artifact and allow the computation of conditional risk measures.
3. Quite often a risk assessment for the next 10 days is desired. This is a requirement of the Basel II Accord (see Basel Committee on Banking Supervision 2006, Part 2, D.4). In practice, the standard deviation derived from daily returns is used and scaled up by a factor of 10. The so-called “square-root-of-time” scaling is explicitly mentioned in the document just cited, but in a more recent draft it has only explanatory status (see Basel Committee on Banking Supervision 2008). However, this kind of scaling can result in a severely underestimated volatility if it is not applied to a homoscedastic process (see, for instance, Danielsson and Zigrand 2005). In contrast, outright 10-day forecasts of the volatility can be derived from GARCH models.¹
4. If the variance-covariance approach with the assumption of normally distributed portfolio returns/losses is employed, then one implicitly also assumes that all marginal distributions are of the same type. One cannot exclude this possibility *per se*, but it is unlikely to be met in empirical applications. The GARCH–copula approach explicitly enables the use of different distributions/models for the margins, thus providing great flexibility.

¹ For the sake of completeness it is worth mentioning that the Basel Committee on Banking Supervision now advocates the utilization of the expected shortfall as a risk measure with a confidence level of 97.5% (see Basel Committee on Banking Supervision, 2013, Section 1.4).

5. To reiterate, only in the case of elliptical distributions do the correlations correctly depict the dependence between the random variables. But in practice the assumption of jointly elliptically distributed random variables seldom holds, hence the dependencies are depicted wrongly. The copula approach circumvents this problem by distinguishing between the dependence structure on the one hand and the modelling of the marginal distributions on the other.

Description

The GARCH-copula approach was probably first introduced by Rockinger and Jondeau (2001). This combination of models has since been used not only for the modelling of portfolio risk, but also in the domain of financial derivatives. For instance, Patton (2006), Jondeau and Rockinger (2006), Micocci and Masala (2005), Embrechts and Dias (2003), Hsu et al. (2008), and Chiou and Tsay (2008) have utilized this approach in their empirical research. The GARCH-copula model rests on the concept of “inference functions for margins.” Here, the dependence between financial risk factors is deduced from the marginal distributions, which are the specified GARCH models.

GARCH models were introduced in Chapter 8. We now focus on the expectation and variance equations for ARMA-GARCH models for processes of the form

$$X_{k,t} = \mu_{k,t} + \epsilon_{k,t}, \quad (9.29)$$

$$\mu_{k,t} = \mu_k + \sum_{i=1}^{p_{1,k}} \phi_i (X_{k,t-i} - \mu_k) + \sum_{j=1}^{q_{1,k}} \theta_j \epsilon_{k,t-j}, \quad (9.30)$$

$$\epsilon_{k,t} = \sigma_{k,t} Z_{k,t}, \quad (9.31)$$

$$Z_{k,t} \sim \mathfrak{D}_{k,v}(0, 1), \quad (9.32)$$

$$\sigma_{k,t}^2 = \alpha_0 + \sum_{i=1}^{p_{2,k}} \alpha_i \epsilon_{k,t-i}^2 + \sum_{j=1}^{q_{2,k}} \beta_j \sigma_{k,t-j}^2, \quad (9.33)$$

where $(Z_{k,t})_{k \in \mathbb{N}, t \in \mathbb{Z}}$ is a $\mathfrak{D}_{k,v}$ -distributed random variable with expected value 0 and variance 1, and X_k denotes the return/loss of the k th financial instrument contained in the portfolio, $k = 1, \dots, K$. Assumptions about the distributions thereof are needed. Potential candidates are the standard normal, the Student’s t , and its skewed version. Estimates for the unknown model parameters, $\zeta_k = (\mu_k, \phi_k, \theta_k, \alpha_k, \beta_k, v_k)'$ for $k = 1, \dots, K$, could then be computed by applying the ML principle. It should again be stressed that the specifications of the GARCH models chosen for each financial return/loss series can differ. This is not confined to the order of the GARCH model, but applies equally to the distribution assumed for the innovations $Z_{k,t}$. Estimates for the standardized residuals of these GARCH models can be computed as $\hat{z}_{k,t} = (x_{k,t} - \hat{\mu}_{k,t})/\hat{\sigma}_{k,t}$.

The joint distribution of the K processes can be expressed with respect to a copula \mathcal{C} as follows: $F(x_1, \dots, x_K; \xi) = \mathcal{C}(F_1(x_1; \zeta_1), \dots, F_K(x_K; \zeta_K)).$ (9.34)

Here, ξ denotes the parameter vector of the copula \mathcal{C} and $F_1(\cdot), \dots, F_K(\cdot)$ are the marginal distributions. If the k -dimensional density function of the copula, c , exists, then the joint density can be expressed as the product of the marginal distributions $f_1(\cdot), \dots, f_K(\cdot)$ with density

$$f(x_1, \dots, x_K; \xi) = c(F_1(x_1; \zeta_1), \dots, F_K(x_K; \zeta_K); \xi) \prod_{i=1}^K f_i(x_i; \zeta_i), \quad (9.35)$$

$$c(u_1, \dots, u_K) = \frac{\delta^K c(u_1, \dots, u_K)}{\delta u_1, \dots, \delta u_K}. \quad (9.36)$$

In principle it would be possible to estimate the unknown parameters by setting up the pseudo-log-likelihood and maximizing it. However, this approach can be computationally burdensome and time-consuming. Instead, a two-step estimation is carried out in practice. First, the parameters of the GARCH models are estimated and the standardized residuals, $\hat{z}_1, \dots, \hat{z}_K$, are determined. These are then transformed into pseudo-uniform variables by employing either the assumed distribution functions or their empirical distributions. The former option is called parametric IFM (inference functions for margins) and the latter semi-parametric IFM, for obvious reasons. The log-likelihood to be maximized is then given by

$$L(\xi; \hat{z}_1, \dots, \hat{z}_k) = \sum_{i=m}^T \log c(F_1(\hat{z}_{1,i}; \xi), \dots, F_1(\hat{z}_{K,i}; \xi)), \quad (9.37)$$

where $m = \max(p_{i,k}, q_{i,k})$ for $i = 1, 2$ and $k = 1, \dots, K$. The estimator for maximizing this likelihood has the properties of consistency and asymptotic normality, if the data can be assumed to be independent and identically normally distributed (see Genest et al. 1995). This assumption can be accepted if the standardized residuals are employed.

Therefore, the GARCH–copula approach for determining portfolio risks consists of the following steps:

1. Specify and estimate the GARCH models for each loss factor.
2. Determine the standardized residuals.
3. Calculate the pseudo-uniform variables from the standardized residuals:
 - either parametrically from the assumed distributions for the GARCH error processes,
 - or semi-parametrically from the empirical distribution functions.
4. Estimate the copula model.
5. Use the dependence structure determined by the estimated copula for generating N data sets of random variates for the pseudo-uniformly distributed variables.
6. Compute the quantiles for these Monte Carlo draws.

7. Use these quantiles in conjunction with the weight vector to calculate the N portfolio return scenarios.
8. Finally, use this series in the calculation of the desired risk measure for a given confidence level.

Application

The GARCH–copula framework is now applied to a fictional passive portfolio consisting of European stocks, and the 95% daily expected shortfall is computed. The `EuStockMarkets` data set is used, which first appears in Chapter 3. It is assumed that the portfolio consists of 40% DAX, with the remaining 60% distributed equally between the FTSE, SMI, and CAC.²

The R code is shown in Listing 9.1. First, the packages **QRM** and **fGarch** are loaded into the workspace. Then the data is retrieved and the losses are expressed

R code 9.1 GARCH–copula model: expected shortfall

```

library (QRM)                                     1
library (fGarch)                                    2
## Losses                                         3
data (EuStockMarkets)                           4
loss <- as . data . frame (na . omit ( - 1.0 * diff (log (EuStockMarkets)) 5
                           * 100.0))           6
## GARCH                                         7
gfit <- lapply (loss , garchFit , formula = ~ garch (1 , 1) ,          8
                cond . dist = "std" , trace = FALSE)                      9
gprog <- unlist (lapply (gfit , function (x)           10
                        predict (x , n . ahead = 1) [3]))                  11
gshape <- unlist (lapply (gfit , function (x) x @ fit $ coef [5]))    12
gresid <- as . matrix (data . frame (lapply (gfit ,           13
                                         function (x) x @ residuals / sqrt (x @ h . t ())))) 14
## Copula                                         15
U <- sapply (1:4 , function (y) pt (gresid [ , y] , df = gshape [y])) 16
cop <- fit . tcopula (Udata = U , method = "Kendall")                  17
rcop <- rcopula . t (100000 , df = cop $ nu , Sigma = cop $ P)        18
qcop <- sapply (1:4 , function (x) qstd (rcop [ , x] , nu = gshape [x])) 19
ht . mat <- matrix (gprog , nrow = 100000 , ncol = ncol (loss) ,          20
                    byrow = TRUE)                                         21
pf <- qcop * ht . mat                                22
## ES 95 percent                                    23
weights <- c (0.4 , 0.2 , 0.2 , 0.2)                24
pfall <- (qcop * ht . mat) %*% weights            25
pfall . es95 <- median (tail (sort (pfall) , 5000)) 26
pfall . es95                                         27

```

² Incidentally, the GARCH–copula approach is also exhibited in the DEMO file `garch_copula` contained in the package **copula**.

as continuous percentages. In lines 7–9, the GARCH models for the marginals are estimated and the standardized residuals are computed. For the sake of simplicity GARCH(1, 1) models with t -distributed innovation processes are specified for the four markets. The task of estimating can be swiftly carried out with the `lapply()` function. The object `gfit` is a list with elements of class `fGARCH`. The estimation results are provided in Table 9.1.

As can be seen from this table, for each model all the coefficients are significantly different from zero and the stability requirement for GARCH(1, 1) models is not violated, that is, $\alpha_1 + \beta_1 < 1$. Furthermore, the magnitude of the estimates for the degrees of freedom parameter v indicate a pronounced deviation from normality, that is, the heavy tail characteristic is reflected by these estimates.

Figure 9.2 shows the QQ plots of the standardized residuals. Apart from a few outliers, it can be concluded that the assumption of a t -distributed error process does hold in general.

Table 9.1 Fitted GARCH(1, 1) models.

GARCH	Estimate	Std. error	t value	$P(> t)$
DAX				
μ	−0.076	0.019	−4.046	0.000
ω	0.022	0.009	2.509	0.012
α_1	0.079	0.016	4.886	0.000
β_1	0.904	0.020	44.950	0.000
v	6.038	0.814	7.418	0.000
SMI				
μ	−0.114	0.018	−6.439	0.000
ω	0.058	0.019	3.062	0.002
α_1	0.114	0.024	4.742	0.000
β_1	0.822	0.039	21.079	0.000
v	5.697	0.727	7.839	0.000
CAC				
μ	−0.052	0.023	−2.234	0.025
ω	0.042	0.025	1.672	0.094
α_1	0.044	0.016	2.822	0.005
β_1	0.922	0.033	27.894	0.000
v	7.986	1.364	5.856	0.000
FTSE				
μ	−0.051	0.016	−3.128	0.002
ω	0.006	0.003	1.754	0.079
α_1	0.036	0.009	3.790	0.000
β_1	0.956	0.013	75.026	0.000
v	9.526	1.787	5.331	0.000

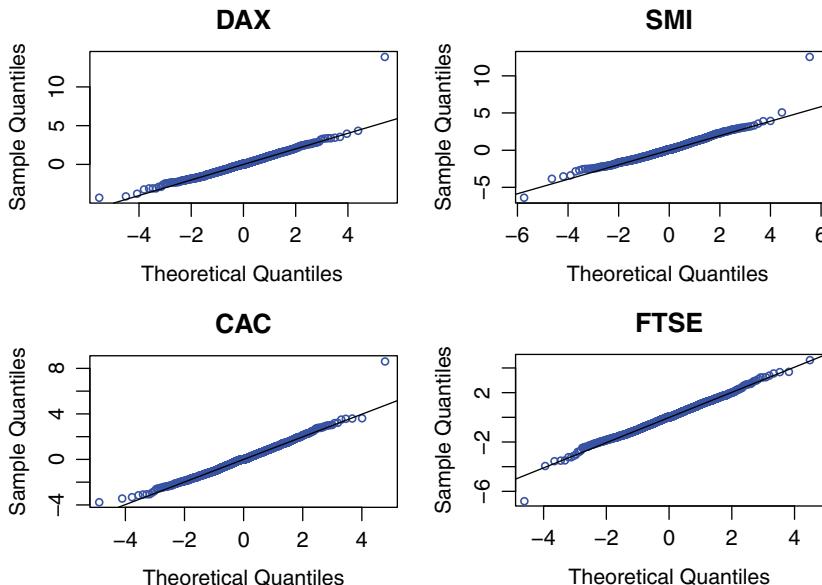


Figure 9.2 *QQ plots for GARCH(1, 1) models.*

Whether the chosen GARCH(1, 1) specification is appropriate for each of the loss factors can be graphically inspected by the ACF of the squared residuals. These are portrayed in Figure 9.3. In neither of the ACF plots does a significant spike at the 95% confidence level occur, hence it can be concluded that the GARCH models assumed for the marginal distributions appropriately capture the characteristics of the loss factors.

In lines 10–14 of the R code the conditional volatilities are computed, and the estimates for the degrees-of-freedom parameters and the standardized residuals are extracted. All of these tasks can be swiftly accomplished by utilizing `lapply()`. The one-step-ahead forecasts of the conditional variance will be used in the computation of the portfolio loss variates, and both of these are needed for the calculation of the pseudo-uniform variables. These are obtained with the expression in line 13. Next, a Student's *t* copula is estimated based on Kendall's rank correlations. The result is assigned to the object `cop`, which is then used in the generation of 100 000 random sets. The spectrum of simulated losses for each financial instrument is determined in lines 20–22. The volatility forecasts are repeated for each simulation of random sets and the Hadamard product of these two matrices is computed. The simulated portfolio losses are then determined as the outcome of the matrix–weight vector product (i.e., object `pfall`). This item is then sorted by size and the expected shortfall at the 95% confidence level is expressed as the median of the 5000 largest losses, which takes a value of 2.585. Incidentally, it has been implicitly assumed that the one-day mean returns/losses are zero.

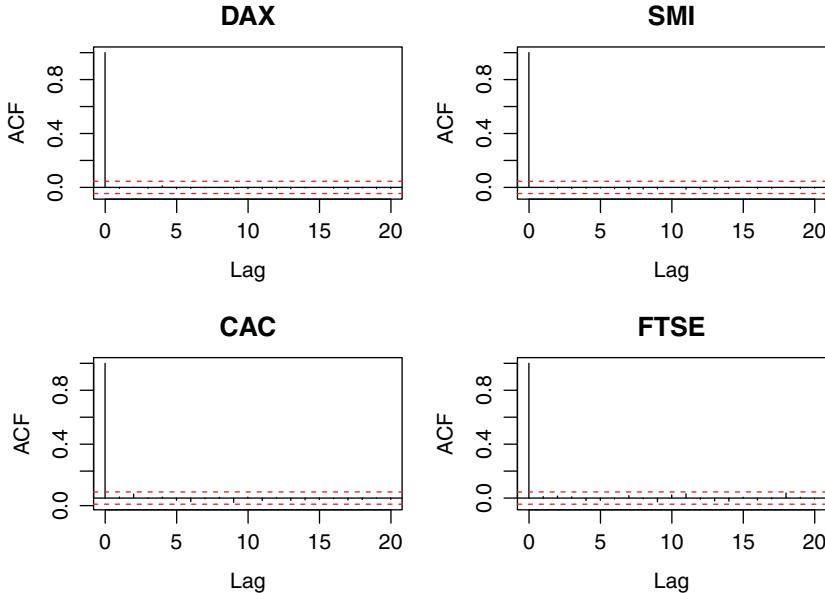


Figure 9.3 Autocorrelations of squared standardized residuals.

9.5.2 Mixed copula approaches

The Clayton and Gumbel copulae were introduced in Section 9.3.3. With these Archimedean copulae either lower or upper tail dependence can be modelled, but not both. In principle, this can be rectified by using the Joe–Clayton copula (see Joe 1997),

$$\mathcal{C}_{\alpha}^{JC} = 1 - \{1 - [(1 - \bar{u}_1^k)^{-\gamma} + (1 - \bar{u}_2^k)^{-\gamma} - 1]^{-1/\gamma}\}^{1/k}, \quad (9.38)$$

with $\bar{u}_i = 1 - u_i$, for $i = 1, 2$, and $\alpha = (k, \gamma)'$. This copula has lower tail dependence equal to $\lambda_l = 2^{-1/\gamma}$ and upper tail dependence equal to $\lambda_u = 2 - 2^{1/k}$. For $k = 1$, the Joe–Clayton copula reduces to the Clayton copula, and for $\gamma \rightarrow 0$, it approaches the Gumbel copula. The Joe–Clayton copula is characterized by an increasing upper tail dependence for increasing parameter values k . However, this copula does not account for extreme events. Because a linear combination of two Archimedean copulae is an Archimedean copula (see Nelsen 2006), one can create a mixture of the Clayton and Gumbel copula in the form

$$\mathcal{C}_{\xi}^{CG} = \pi \mathcal{C}_{\delta}^C(u_1, u_2) + (1 - \pi) \mathcal{C}_{\theta}^G(u_1, u_2), \quad (9.39)$$

where $\pi \in (0, 1)$ is the weighting coefficient between these two copulae. The tail dependencies for this copula are given by $\lambda_l = \pi 2^{-1/\delta}$ for the lower and $\lambda_u = (1 - \pi)(2 - 2^{1/\theta})$ for the upper coefficient, respectively.

R code 9.2 Mixing of copulae: Clayton and Gumbel.

```

library(copula)                                     1
library(FRAPO)                                    2
library(QRM)                                      3
## Retrieving data and edf                      4
data(DJ.df)                                       5
Data <- DJ.df[, c("GM", "UTX")]                 6
R <- returnseries(Data, method = "discrete", trim = TRUE) 7
U <- apply(R, 2, edf)                           8
detach(package:QRM)                            9
## Initialising copula objects                  10
copC <- claytonCopula(2)                         11
copG <- gumbelCopula(2)                         12
## Objective function                           13
LLCG <- function(params, x, copC, copG){          14
  slot(copC, "parameters") <- params[1]           15
  slot(copG, "parameters") <- params[2]           16
  pi <- params[3]                                17
  ldens <- log(pi * dCopula(x, copC) + (1 - pi) * dCopula(x, copG)) 18
  if(any(is.infinite(ldens))){                   19
    ldens[which(is.infinite(ldens))] <- 0        20
  }
  sum(ldens)                                     21
}
## Parameter bounds & initialisation          22
lower <- c(copC@param.lowbnd, copG@param.lowbnd, 0.0) 23
upper <- c(copC@param.upbnd, copG@param.upbnd, 1.0) 24
par1 <- copula::fitCopula(copC, U, "itau")@estimate 25
par2 <- copula::fitCopula(copG, U, "itau")@estimate 26
par3 <- 0.5                                       27
## Optimisation                                28
opt <- optim(c(par1, par2, par3), LLCG, x = U, copC = copC, 29
             copG = copG, lower = lower, upper = upper, 30
             method = "L-BFGS-B", 31
             control = list(fnscale = -1, trace = 2), 32
             hessian = TRUE) 33
## Variance-Covariance                         34
varcov <- round(solve(-opt$hessian), 4)           35

```

In Listing 9.2 the relevant code lines for estimating a mixed copula model are exhibited. First, the necessary packages are loaded into the workspace. Here, the functions and methods contained in the package **copula** will be used for creating the log-likelihood consisting of the weighted densities of the Clayton and Gumbel copulae. The function **returnseries()** contained in the package **FRAPO** will be used for calculating the discrete returns expressed as percentages, and the data set will be taken from the package **QRM**, as will the function **edf()** for determining

Table 9.2 Fitted mix of Clayton and Gumbel.

Clayton–Gumbel	$\hat{\delta}$	$\hat{\theta}$	$\hat{\pi}$
Estimated values	0.7377	1.1147	0.3540
Standard errors	0.2166	0.0265	0.0927

the pseudo-uniform variables of the stock returns for GM and UTX. In the next block of statements, two copula objects are initialized. The chosen parameters for δ and θ are arbitrarily set to 2 and will be updated during the numerical minimization of the negative log-likelihood. Next comes the definition of the objective function `LLCG()`. Its arguments are `params` for the three-element parameter vector, `x` for the matrix of pseudo-uniform variables, and `copC` and `copG` for the copula objects defined earlier for the Clayton and Gumbel copulas, respectively. In the subsequent chunk of code the lower and upper bounds of the parameter space are set and initial values for δ and θ are determined according to the rank correlations. The minimization of the negative log-likelihood is carried out by means of `optim()`. Because the value of the log-likelihood is returned by `LLCG()`, the control argument `fnscale` must be set to -1 . In addition, for better tracing of the numerical optimization, the control argument `trace` has been set to 2.

The parameter estimates and standard errors are shown in Table 9.2.

The estimated value of π implies a roughly one-third to two-thirds weighting between the Clayton and Gumbel copulae and is significantly different from zero. From the estimates for δ and θ , the tail dependence coefficients are 0.138 for the lower and 0.945 for the upper.

References

- Basel Committee on Banking Supervision 2006 *International Convergence of Capital Measurement and Capital Standards, A Revised Framework* Bank for International Settlements Basel, Switzerland.
- Basel Committee on Banking Supervision 2008 *Proposed Revisions to the Basel II Market Risk Framework* Bank for International Settlements Basel, Switzerland.
- Basel Committee on Banking Supervision 2013 *Consultative Document—Fundamental Review of the Trading Book: A Revised Market Risk Framework* Bank for International Settlements Basel, Switzerland.
- Black F. and Litterman R. 1990 *Asset allocation: Combining investor views with market equilibrium*. Technical report, Goldman Sachs Fixed Income Research.
- Burger M., Jünemann K., and König T. 2015 *RUnit: R Unit Test Framework*. R package version 0.4.31.
- Chiou S. and Tsay R. 2008 A copula-based approach to option pricing and risk assessment. *Journal of Data Science* **6**, 273–301.

- Danielsson J. and Zigrand J. 2005 *On Time-Scaling of Risk and the Square-Root-of-Time Rule* efa 2004 Maastricht Meetings paper no. 5399 edn London School of Economics London, UK.
- Deheuvels P. 1979 La fonction de dépendance empirique et ses propriétés: un test non paramétrique d'indépendance. *Académie Royale de Belgique, Bulletin de la Classe des Sciences*, 5e **65**, 274–292.
- Dutang C. 2015 *gumbel: Package for Gumbel copula*. R package version 1.10-1.
- Embrechts P. and Dias A. 2003 *Dynamic copula models for multivariate high-frequency data in finance* Discussion Paper, Department of Mathematics, ETH Zurich.
- Genest C. and Favre A.-C. 2007 Everything you always wanted to know about copula modeling but were afraid to ask. *Journal of Hydrologic Engineering* pp. 347–368.
- Genest C. and Rémillard B. 2004 Tests of independence and randomness based on the empirical copula process. *Test* **13**, 335–369.
- Genest C., Ghoudi K., and Rivest L.-P. 1995 A semiparametric estimation procedure of dependence parameters in multivariate families of distributions. *Biometrika* **82**(3), 543–552.
- Genest C., Quesey J.-F., and Rémillard B. 2006 Local efficiency of a Cramér–von Mises test of independence. *Journal of Multivariate Analysis* **97**, 274–294.
- Genest C., Quesey J.-F., and Rémillard B. 2007 Asymptotic local efficiency of Cramér–von Mises tests for multivariate independence. *The Annals of Statistics* **35**, 166–191.
- Gochez F., Chandler-Mant R., Jin S., and Xie J. 2015 *BLCOP: Black–Litterman and Copula Opinion Pooling Frameworks*. R package version 0.3.1.
- Hofert M. and Mächler M. 2011 Nested archimedean copulas meet R: The nacopula package. *Journal of Statistical Software* **39**(9), 1–20.
- Hofert M., Kojadinovic I., Mächler M., and Yan J. 2015 *copula: Multivariate Dependence with Copulas*. R package version 0.999-14.
- Hsu C., Wang Y., and Tseng C. 2008 Dynamic hedging with futures: A copula-based GARCH model. *Journal of Futures Markets* **28**, 1095–1116.
- Joe H. 1997 *Multivariate Models and Dependence Concepts* number 73 in *Monographs on Statistics and Applied Probability*. Chapman & Hall, London.
- Joe H. and Xu J. 1996 *The estimation method of inference functions for margins for multivariate models*. Technical Report 166, University of British Columbia, Department of Statistics.
- Jondeau E. and Rockinger M. 2006 The copula–GARCH model of conditional dependencies: An international stock market application. *Journal of International Money and Finance* **25**, 827–853.
- Kojadinovic I. and Yan J. 2010 Modeling multivariate distributions with continuous margins using the copula R package. *Journal of Statistical Software* **34**(9), 1–20.
- McNeil A., Frey R., and Embrechts P. 2005 *Quantitative Risk Management: Concepts, Techniques and Tools*. Princeton University Press, Princeton, NJ.
- Meucci A. 2006a Beyond Black–Litterman in practice: A five-step recipe to input views on non-normal markets. *Risk* **19**(9), 114–119.
- Meucci A. 2006b Beyond Black–Litterman: Views on non-normal markets. *Risk* **19**(2), 96–102.
- Meucci A. 2010 The Black–Litterman approach: Original model and extensions In *The Encyclopedia of Quantitative Finance* (ed. Cont R.) vol. I John Wiley & Sons Chichester, UK p. 196.

- Micocci M. and Masala G. 2005 *Backtesting value-at-risk estimation with non-Gaussian marginals Working Paper*, 8th International Congress on Insurance: Mathematics & Economics, Rome.
- Nelsen R. 2006 *An Introduction to Copulas* number 139 in *Lecture Notes in Statistics* 2nd edn. Springer Verlag, New York.
- Patton A. 2006 Modelling asymmetric exchange rate dependence. *International Economic Review* **47**(2), 527–556.
- Rockinger M. and Jondeau E. 2001 *Conditional dependency of financial series: An application of copulas*. Notes d' Études et de Recherche NER 82, Banque de France, Paris, France.
- Schweizer B. and Sklar A. 1983 *Probabilistic Metric Spaces*. North-Holland/Elsevier, New York.
- Shih J. and Louis T. 1995 Inferences on the association parameter in copula models for bivariate survival data. *Biometrics* **51**, 1384–1399.
- Sklar A. 1959 Fonctions de répartition à dimensions e leurs marges. *Publications de l'Institut de l'Université de Paris* **8**, 229–231.
- Venables W. N. and Ripley B. D. 2002 *Modern Applied Statistics with S* 4th edn. Springer, New York.
- Würtz D. and Setz T. 2014 *fCopulae: Rmetrics—Bivariate Dependence Structures with Copulae*. R package version 3011.81.
- Würtz D., Setz T. and Chalabi Y. 2014 *fPortfolio: Rmetrics—Portfolio Selection and Optimization*. R package version 3011.81/r5908.
- Yan J. 2007 Enjoy the joy of copulas: With a package copula. *Journal of Statistical Software* **21**(4), 1–21.

Part III

PORTFOLIO OPTIMIZATION APPROACHES

10

Robust portfolio optimization

10.1 Overview

Markowitz portfolios and their application in **R** were introduced in Chapter 5. The problems that one might encounter by directly optimizing portfolios of this kind were addressed. In particular, it was stated that the use of sample estimators for the expected returns and the covariance matrix can result in sub-optimal portfolio results due to estimation error. Furthermore, extreme portfolio weights and/or erratic swings in the asset mix are commonly observed in *ex post* simulations. In general, this empirical fact is undesirable because of transaction costs. From a statistical point of view, these artifacts can mainly be attributed to the sensitivity of the ordinary sample estimators with respect to outliers. These outlying data points influence the dispersion estimates to a lesser extent than the means, *ceteris paribus*. Hence, but not only for this reason, minimum-variance portfolios are advocated compared to mean-variance portfolios (see the references in Chapter 5). It would therefore be desirable to have estimators available which lessen the impact of outliers and thus produce estimates that are representative of the bulk of sample data, and/or optimization techniques that incorporate estimations errors directly. The former can be achieved by utilizing robust statistics and the latter by employing robust optimization techniques.

In the next two sections these two means are presented from a theoretical point of view. Then the sources in **R** are discussed. The chapter concludes with empirical applications in the form of a Monte Carlo simulation and back-test comparisons, where these robust portfolio optimizations are compared to portfolio solutions based on ordinary sample estimators.

10.2 Robust statistics

10.2.1 Motivation

It has already been pointed out in Chapter 3 that the normality assumption quite often does not hold for financial market return data (see also Pfaff 2010, Chapter 1). The violation of this assumption is justified on empirical grounds by the stylized facts for single and multivariate returns. But it was also shown in Chapter 6 that the normality assumption is violated to a lesser extent the lower the data frequency is (see also Pfaff 2010, Chapter 2).

It is fair to say that in empirical work the arithmetic mean and the sample covariance are widely employed in estimating the corresponding theoretical location and dispersion moments of the population. These estimators can be derived from the ML principle and are asymptotically efficient and consistent. Furthermore, they are asymptotically normally distributed. However, when the distributional assumption is not met, the estimators lose their desirable properties. In fact, the arithmetic mean—as an estimator for the location of a population—is sensitive to extreme observations, such that the estimate does not reflect the bulk of the data well. On a similar note, the dependence between two random variables can be highly distorted by a single outlying data pair. In light of this, it would be desirable to have recourse to methods and techniques that are relatively immune to such outliers and/or to violations of the underlying model assumptions, but could still deliver reasonable results for the majority of observations in a sample. The field of robust statistics deals with problems of this kind and offers solutions in the form of robust estimators and inference based upon these. Textbook expositions can be found, for instance, in Huber (1981), Hampel et al. (1986), Rousseeuw and Leroy (1987), Staudte and Sheather (1990), and Maronna et al. (2006).

Formerly, the outlier problem sketched above would be resolved by means of trimming (removing of outliers) or winsorizing (equalizing extreme observations to a fixed quantile value). Indeed, both methods can be considered as means of robustification. Unfortunately, the outcome of these procedures is greatly dependent on the subjective choice of the researcher. The median could perhaps be employed as an estimator for the location and the mean absolute deviation for the scale. Both produce more robust estimates than the ordinary sample estimators of the arithmetic mean and standard deviation. The advantage of these robust estimators is that they avoid specifying in advance which data points are viewed as outliers. Hence, a subjective element in the data analysis is removed. Before the most commonly encountered robust statistics are introduced in the next subsection, it is worth mentioning that so far the term “outlier” or “extreme observation” has not been precisely defined. The reason for this is simple: there is no clear-cut way to assess whether a data point is an outlier or not. The question is always a relative one and crucially depends on the underlying model/distribution assumption. For example, given the standard normal distribution and a sample observation of 5, one could surely classify this data point as an outlier. But would this classification still hold when a Student’s t distribution with four degrees of freedom or a Cauchy distribution is assumed?

10.2.2 Selected robust estimators

The following presentation of selected robust estimators and their properties is based on Maronna et al. (2006) and Todorov and Filzmoser (2009). The most commonly utilized measure for assessing the robustness of an estimator is the *breakdown point* (BP). This measure is defined as the relative share of outliers in a sample such that the estimator does not take an arbitrary large value. By definition, the BP can take values between 0 and 0.5. The arithmetic mean has a BP of 0, because if a single observation is replaced by one value, the location estimate can be made arbitrarily high. In contrast, the median has a BP of 0.5. The upper bound of the BP is explained by the fact that if more than half of the observations are outliers, the sample is falsified to a degree such that no inference with respect to the population can be drawn from it. A further criterion for assessing the appropriateness of a robust estimator is the *relative efficiency* (RE). Here, the asymptotic variance of a robust estimator is expressed relative to the variance of an optimal estimator which has been derived under strict adherence to the model/distribution assumption. As such, it can be interpreted as a percentage figure, indicating by how much the sample size has to be increased such that the variances of the two estimators are equalized.

Class of M- and MM estimators

As early as 1964 the class of M-estimators was introduced by Huber. The class name should indicate the resemblance of this estimator to the ML principle (see Huber 1964, 1981). It is convenient to restrict the following presentation of the estimator to univariate samples. Recall that, according to the ML principle, the unknown parameters, θ , are determined such that they have most likely produced a given iid sample, $\{x_1, \dots, x_n\}$, drawn from the distribution $F(x, \theta)$ with density function $f(\cdot)$. Due to the iid assumption, the joint distribution is equal to the product of the marginals, which is maximized:

$$\hat{\theta} = \arg \max \left(\prod_{i=1}^n f(x_i, \theta) \right). \quad (10.1)$$

Because the logarithm is a strictly monotone transformation, the optimization is generally carried out by minimizing the negative log-likelihood function:

$$\hat{\theta} = \arg \min \left(- \sum_{i=1}^n \log(f(x_i, \theta)) \right). \quad (10.2)$$

Analogously, the M-estimators are defined as the minimum of a sum of functions $\rho(x, \theta)$:

$$\hat{\theta} = \arg \min \left(\sum_{i=1}^n \rho(x_i, \theta) \right). \quad (10.3)$$

This broad class of estimators includes the ML principle ($\rho(\cdot) = -\log f(x, \theta)$) and the method of least squares (LS). In the latter case the function $\rho(\cdot)$ is the quadratic error.

The function $\rho(\cdot)$ must meet the requirements of symmetry, positive definiteness, and a global minimum at zero. Of course, the function should provide decent estimates when the model/distribution assumptions are met and not be negatively affected by instances of violation. The difference between the robust forms of the M-estimators and those of the ML and LS principles is in the specification of $\rho(\cdot)$. For the former estimators extreme data points obtain a smaller weight and are thus less influential with respect to the parameter estimates. Two kinds of specification for $\rho(\cdot)$ are usually employed in empirical work. First, there is the class of Huber functions, defined as

$$\rho_k(x) = \begin{cases} x^2 & \text{if } |x| \leq k, \\ 2k|x| - k^2 & \text{if } |x| > k. \end{cases}$$

This function is quadratic in a central region around k and linear outside it. For $k \rightarrow \infty$ and $k \rightarrow 0$ the M-estimators are identical to the arithmetic mean and the median, respectively.

Second, the *bi-square* function proposed by Tukey is also encountered in empirical work. It is given by

$$\rho_k(x) = \begin{cases} 1 - [1 - (x/k)^2]^3 & \text{if } |x| \leq k, \\ 1 & \text{if } |x| > k. \end{cases}$$

This function is bounded for large absolute values of x . The bi-square function is preferred to the Huber functions for symmetric distributions characterized by excess kurtosis, because the influence of outliers can be completely suppressed.

Instead of directly optimizing (10.3), it is often easier to find the solution by equating the gradient $\psi(x) = \frac{\delta \rho(x, \theta)}{\delta \theta}$ to zero.

The class of MM estimators was originally introduced by Yohai (Yohai 1987; Yohai et al. 1991) in the context of robust regression analysis. Lopuhaä (1991, 1992) adapted these estimators for multivariate data analysis. In a first step, the dispersion of the data is estimated by an M-estimator. Based upon this, a second M-type estimation for the location parameters is carried out that is in close accord with the variance-covariance matrix of the first step.

Estimators based on robust scaling

A general class of robust estimators is now presented that can be derived from robust scaling, namely the *minimum volume ellipsoid* (MVE) estimator, the *minimum covariance determinant* (MCD) estimator, and the S-estimator. A p -dimensional random variable $\mathbf{x} = (x_1, \dots, x_p)' \in \mathbb{R}^p$ is considered which is jointly normally distributed, $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$. The location vector is given by $\boldsymbol{\mu} = \mathbf{E}(\mathbf{x}) = (\mathbf{E}(x_1), \dots, \mathbf{E}(x_p))'$ and the covariance matrix by $\Sigma = \text{VAR}(\mathbf{x}) = \mathbf{E}((\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})')$. The distance measure is the squared Mahalanobis distance between the sample and the location and dispersion parameters: $d(\mathbf{x}, \boldsymbol{\mu}, \Sigma) = (\mathbf{x} - \boldsymbol{\mu})' \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})$. The trivial solution of zero distance can be achieved if the smallest eigenvalue of an estimated covariance matrix $\hat{\Sigma}$ tends to zero. In order to exclude this trivial solution, $|\hat{\Sigma}| = 1$ is required,

where $|\cdot|$ denotes the determinant of a matrix. The three above-mentioned robust estimators for $\hat{\mu}$ and $\hat{\Sigma}$ are based on the optimization

$$\arg \min \hat{\sigma}(\mathbf{d}(X, \hat{\mu}, \hat{\Sigma})), \quad (10.4)$$

where X is the full sample and \mathbf{d} is the vector of distances $d(x_i, \hat{\mu}, \hat{\Sigma})$ for $i = p + 1, \dots, N$ and $\hat{\sigma}$ is a robust metric.

The MVE and MCD estimators were introduced by Rousseeuw (Rousseeuw 1985; Rousseeuw and Leroy 1987). In the case of the MVE estimator the smallest data cloud is determined such that at least half of the sample observations is included. This is achieved by utilizing the median of $\hat{\sigma}$. The convergence rate for this estimator is only $N^{-1/3}$. In the case of the MCD estimator the robust subsample with size $h > N/2$ is chosen such that the determinant of the variance-covariance matrix is minimal. The location vector is determined as the p arithmetic means of the selected data points, and the estimator for the variance-covariance matrix is adjusted by correction factors, such that the resultant estimate $\hat{\Sigma}$ follows the normal distribution. The BP is maximal for $h = \text{int}[(N + p + 1)/2]$, but any value in the interval $[(N + p + 1)/2, N]$ can be chosen. Davies (1987) introduced the class of S-estimators. The letter "S" stands for scale. With respect to the above optimization, an M-estimator for the scale N according to $1/N \sum_{i=1}^N \rho(d_i) = \delta$ is inserted, where $\rho(\cdot)$ must be a bounded function (e.g., the bi-square function) and $\delta \in (0, 1)$.

The Stahel–Donoho estimator

The Stahel–Donoho estimator (SDE) is due to Stahel (1981) and Donoho (1982). According to this procedure, the degree of outlying is determined in terms of a projection of the data matrix X with a vector $\mathbf{a} \in \mathbb{R}^p$. This vector is defined for given observations \mathbf{x} , robust location $\hat{\mu}$, and scale estimator with respect to the complete data set X as

$$t(\mathbf{x}, \mathbf{a}) = \frac{\mathbf{x}'\mathbf{a} - \hat{\mu}(\mathbf{X}\mathbf{a})}{\hat{\sigma}(\mathbf{X}\mathbf{a})}. \quad (10.5)$$

The degree of outlines is given as the maximum of (10.5) for all vectors \mathbf{a} where the norms of these vectors equal 1. This solution is then used as the weighting scheme for X such that a non-increasing function $t(\mathbf{x}_i)$ is applied.

The OGK estimator

One problem with robust estimators that possess the property of affine invariance is the resulting non-convex optimization. The estimator proposed by Maronna and Zamar (2002) avoids this problem by estimating the covariances between two random variables as

$$s_{jk} = \frac{1}{4} \left(\sigma \left[\frac{Y_j}{\sigma(Y_j)} + \frac{Y_k}{\sigma(Y_k)} \right]^2 - \sigma \left[\frac{Y_j}{\sigma(Y_j)} - \frac{Y_k}{\sigma(Y_k)} \right]^2 \right). \quad (10.6)$$

This form of dispersion estimation was introduced by Gnanadesikan and Kettenring (1972). If the covariances are determined according to (10.6) then the property of affine invariance is traded off against a simpler computation. However, if one applied this dispersion estimator directly to the data pairs of X , the resulting variance-covariance matrix would cease to be positive definite. For this reason, Maronna and Zamar (2002) proposed an orthogonalization of X and hence the estimator is termed “orthogonalized Gnanadesikan–Kettenring” (OGK). If a robust estimator for σ is used for the pairwise covariances s_{jk} , with $j = 1, \dots, p$ and $k = 1, \dots, p$, then the resulting variance-covariance matrix is also robust.

10.3 Robust optimization

10.3.1 Motivation

In this section robust approaches and techniques for optimizing portfolios will be presented. The previous section dealt with the utilization of robust rather than classical estimators for the unknown location and dispersion parameters. In both cases the resulting estimates—whether obtained by maximum likelihood or by robust estimation techniques—have been taken as fixed and would be used directly as mathematical program inputs. However, in both instances these parameter values are subject to uncertainty. In contrast, we will now present and investigate optimization approaches whereby the parameter uncertainty is directly taken into account in the formulation of the mathematical program. The term “robust” is now defined differently than in the previous section. There it was used for point estimators that are less prone to outliers when compared to the behavior of the classical ML estimators. Now it is defined as an optimization technique that will produce a solution which is not negatively impacted by an alternative parameter specification—for example, if the return expectations are turning out to be less favorable. Incidentally, robust optimization techniques differ from stochastic optimization in the sense that the latter are based on a specific distribution assumption for the parameters. This is not generally required for robust optimization. In a nutshell, the aim of robust optimization is the derivation of an optimal solution for sets of possible parameter constellations. Extended presentations of the approaches to tackling such problems are given in Ben-Tal and Nemirovski (1998), Cornuejols and Tütüncü (2007), Fabozzi et al. (2007), Meucci (2005, Chapter 9), Scherer (2010, Chapter 5), and Tütüncü and König (2004).

10.3.2 Uncertainty sets and problem formulation

The concept of robust optimization will now be elucidated for mean-variance portfolios, although the approach is also applicable to other kinds of optimization. The classical portfolio optimization is given by¹

$$P_\lambda = \arg \min_{\omega \in \Omega} (1 - \lambda) \sqrt{\omega' \Sigma \omega} - \lambda \omega' \mu, \quad (10.7)$$

¹ Most of this subsection is based upon (Schötte 2007, Chapter 4).

where ω denotes the $n \times 1$ portfolio weight vector and $\Omega \subset \{\omega \in \mathbb{R}^N | \omega' \mathbf{1} = 1\}$ is the set of all allowable solutions. The (expected) returns of the N assets are contained in the vector μ , with variance-covariance matrix $\Sigma \in \mathbb{R}^{N \times N}$ (assumed to be positive definite). The parameter λ is allowed to take all values in the interval $[0, 1]$ and determines the weighting between the portfolio return and its risk. Of course, the above problem formulation can be amended by further constraints, such as a budget constraint ($\omega' \mathbf{1} = 1$), a non-negativity constraint ($\omega \geq 0$), and/or bounds on the weights for single assets or groups of assets ($A\omega \leq b$). The mathematical program in the above formulation includes the special cases of a **minimum-variance** and a **maximum-return portfolio** if the values for λ are chosen as $\lambda = 0$ or $\lambda = 1$, respectively. All values of λ between these bounds yield portfolio solutions that lie on the feasible efficient frontier.

So far, point estimates for the unknown parameters (μ, Σ) have been utilized. These can be derived from classical or robust estimators, as shown in the previous section. Incidentally, it will be shown by means of simulation in Section 10.5 that the latter class of estimators turn out to be beneficial compared to the ML estimators. However, regardless of whether the point estimates are obtained from classical or robust estimators, these **point estimates are subject to uncertainty and in either case small deviations from the true but unknown parameter values can result in quite distinct portfolio solutions**. Viewed from this angle, **it would be desirable to include the parameter uncertainties directly in the formulation of the portfolio optimization**. Because parameter uncertainty exerts a greater impact on the portfolio composition in the case of expected returns compared to their dispersion, **we will limit the following exposition to the inclusion of uncertain return parameters only and treat the parameters pertinent to portfolio risk as fixed**. So far, the term “uncertainty” has been used rather vaguely. For the purpose of applying robust optimization techniques this gap needs to be filled with a concrete definition. This is achieved by **defining an uncertainty set $\mathcal{U}_{\hat{\mu}}$ for all allowable parameter values**. In principle, three different kinds of sets are possible:

$$\mathcal{U}_{\hat{\mu}} = \{\mu \in \mathbb{R}^N | \hat{\mu}_1, \dots, \hat{\mu}_i, \dots, \hat{\mu}_M, i = 1, \dots, M\}, \quad (10.8a)$$

$$\mathcal{U}_{\hat{\mu}} = \{\mu \in \mathbb{R}^N | |\hat{\mu}_i - \mu_i| \leq \delta_i, i = 1, \dots, N\}, \quad (10.8b)$$

$$\mathcal{U}_{\hat{\mu}} = \{\mu \in \mathbb{R}^N | (\mu - \hat{\mu})' \hat{\Sigma}^{-1} (\mu - \hat{\mu}) \leq \frac{\delta^2}{T}\}. \quad (10.8c)$$

For the uncertainty set in (10.8a), M scenarios for the expected return vector μ must be specified. These can comprise subjective expectations and/or location estimates derived from alternative estimators. In (10.8b) the uncertainty set is defined as intervals around the true, but unknown, return vector. These bounds can be subjectively set, but can also be derived from a distributional assumption. For instance, **if it is assumed that each of the asset returns is normally distributed, the central fluctuation interval with $(1 - \alpha)$ confidence level is given as $\delta_i = \Phi^{-1}(\alpha/2) \cdot \hat{\sigma}_i / \sqrt{T}$** , where Φ^{-1} is the quantile function, $\hat{\sigma}_i$ the standard deviation of the returns for the i th asset, and T denotes the sample size. The stylized fact of excess kurtosis of empirical return processes can be taken into account, by utilizing a Student’s t distribution, for

example. In this case, an ML estimation is conducted first, to obtain an estimate of the degrees-of-freedom parameter v . Then the quantile at the desired confidence level is determined by using this empirical estimate. It is worth mentioning that in the case of an uncertainty set as in (10.8b) no dependencies between the uncertainty margins for the different assets are taken into account, but rather these are treated as being independent of each other. For the uncertainty set in (10.8c) an elliptical uncertainty set is defined. In contrast to the uncertainty set in (10.8b), it is now assumed that the uncertainties originate from a multivariate elliptical distribution; the covariances between assets' returns are now explicitly included in the uncertainty set. Last, but not least, it should be pointed out that the first and second uncertainty sets can be combined. The scenarios (e.g., ML- and robust-based estimators for the unknown returns) would then be given by $M := \{\hat{\mu}^{ML}, \hat{\mu}^{MCD}, \hat{\mu}^M, \hat{\mu}^{MM}, \hat{\mu}^{MVE}, \hat{\mu}^S, \hat{\mu}^{SD}, \hat{\mu}^{OGK}\}$ and an elliptical uncertainty set can be generated as

$$\mathcal{U}_{\text{est}} = \{\boldsymbol{\mu} \in \mathbb{R}^N \mid (\boldsymbol{\mu} - \bar{\boldsymbol{\mu}})' \bar{\Sigma}^{-1} (\boldsymbol{\mu} - \bar{\boldsymbol{\mu}}) \leq \bar{\delta}^2\}, \quad (10.9)$$

with

$$\bar{\boldsymbol{\mu}} = \frac{1}{|M|} \sum_{m \in M} \boldsymbol{m}, \quad (10.10a)$$

$$\bar{\Sigma} = \text{diag}(\bar{\sigma}_{11}, \dots, \bar{\sigma}_{NN}) \text{ where } \bar{\sigma}_{ii} = \frac{1}{|M| - 1} \sum_{m \in M} (m_i - \bar{\mu}_i)^2, \quad (10.10b)$$

$$\bar{\delta} = \arg \max_{m \in M} (m - \bar{\boldsymbol{\mu}})' \bar{\Sigma}^{-1} (m - \bar{\boldsymbol{\mu}}). \quad (10.10c)$$

Having introduced alternative specifications for the uncertainty sets, it will next be shown how these uncertainties can be expressed in a mathematical program. In general, and independent of the chosen form of the uncertainty set, a worst-case approach is employed for solving optimization tasks robustly. This approach is also known as a min–max approach. This tackles the question of what the optimal weight vector is given the least favorable parameter constellation, that is, the smallest portfolio return for a given risk level. Loosely speaking: expect the worst and you will at least not be disappointed.

In the case of an uncertainty set as in (10.8a), in a first step the weight vectors for the M scenarios would be determined and then the one that yields the lowest portfolio return would be selected. It should be pointed out that for this kind of uncertainty set the determination of the optimal solution can become computationally expensive and depends on the number of scenarios M . It should also be stressed that all scenarios are treated as equally likely, but that the solution is determined solely by the worst scenario. It can be deduced from this that the approach is very conservative and that the optimal outcome can be highly influenced by single outliers for the expected returns of the assets. Hence, a sound specification of the M scenarios is of the utmost importance.

Next is the description of how the problem for an uncertainty set of a symmetrical interval around the location parameters $\boldsymbol{\mu}$ as shown in (10.8b) can be implemented.

The true but unknown subsequent return of the i th asset is contained in the interval $\mu_i \in [\hat{\mu}_i - \delta_i, \hat{\mu}_i + \delta_i]$ for a given confidence level. The least favorable return for the i th asset is therefore given as $\mu_i = \hat{\mu}_i - \delta_i$ for a long position and as $\mu_i = \hat{\mu}_i + \delta_i$ for a short position. These N confidence intervals form a polyhedron and can be expressed as a system of linear inequality constraints. However, it is unknown beforehand whether an asset enters into the portfolio with a positive or negative weight. To solve this problem, two slack variables, ω_+ and ω_- , are included in the objective function:

$$\begin{aligned} PR_\lambda &= \arg \max_{\omega, \omega_+, \omega_-} \omega' \hat{\mu} - \delta' (\omega_+ - \omega_-) - (1 - \lambda) \sqrt{\omega' \Sigma \omega}, \\ \omega &= \omega_+ - \omega_-, \\ \omega_+ &\geq 0, \\ \omega_- &\geq 0. \end{aligned} \tag{10.11}$$

Of course, the above problem formulation can be amended by a budget constraint $\omega' \mathbf{1} = 1$ and/or other constraints. During optimization for positive weights $\omega_i > 0$ the returns are equal to $(\hat{\mu}_i - \delta_i)$, and for negative weights the returns are set as $(\hat{\mu}_i + \delta_i)$. In the first case, the i th element of ω_+ is positive and that of ω_- is set equal to zero according to the inequality constraint, and vice versa. Assets with a higher return uncertainty will obtain a lower portfolio weight than those for which the expected return is more certain. The uncertainty of the expected returns can also be limited to a subset $\mathcal{U} \subset N$ of the assets. In this case, $\delta_i \notin \mathcal{U}$ are set to zero. If long-only constraints are included as side constraints, then the formulation of the mathematical program reduces to that exhibited in (10.7), where the return vector is set as $\hat{\mu} - \delta$ with the additional non-negativity constraint $\omega \geq 0$.

Finally, the treatment of an elliptical uncertainty set for the returns as in (10.8c) is elucidated. Analogously to the previous approaches, this problem is also solved in the form of a worst-case approach:

$$PR_\lambda = \arg \min_{\omega \in \Omega} \arg \max_{\mu \in \mathcal{U}} (1 - \lambda) \sqrt{\omega' \Sigma \omega} - \lambda(\omega' \mu). \tag{10.12}$$

If the returns are multivariate normally distributed, then $(\mu - \hat{\mu})' \Sigma^{-1} (\mu - \hat{\mu})$ is distributed as χ^2 with N degrees of freedom. The scalar δ^2 is then the corresponding quantile value for a given confidence level $(1 - \alpha)$. The stochastic return vector μ therefore lies in the ellipse defined by the level of confidence. The maximal distance between this uncertainty ellipsoid and the empirical location vector is now determined, such that the returns correspond to the least favorable outcome. It is evident from (10.12) that the uncertainty is confined to the returns only and that the variance-covariance matrix is taken as given. Therefore, in a first step the maximal distance can be determined by utilizing a Lagrange approach, where $\hat{P} = \frac{1}{T} \hat{\Sigma}$ is the variance-covariance matrix of the empirical returns and γ denotes the Lagrangian multiplier:

$$\mathcal{L}(\mu, \gamma) = \omega' \hat{\mu} - \omega' \mu - \left[\frac{\gamma}{2} (\hat{\mu} - \mu)' \hat{P}^{-1} (\hat{\mu} - \mu) - \delta^2 \right]. \tag{10.13}$$

The optimal solution is then found by taking the partial derivatives of (10.13) with respect to μ and γ and setting these to zero. This yields a system of two equations, which is solved for μ :

$$\mu = \hat{\mu} - \left(\frac{\delta}{\sqrt{\omega' \hat{P} \omega}} P \omega \right). \quad (10.14)$$

After left-multiplication by ω' one obtains, for the portfolio returns,

$$\begin{aligned} \omega' \mu &= \omega' \hat{\mu} - \delta \sqrt{\omega' \hat{P} \omega} \\ &= \omega' \hat{\mu} - \delta \|\hat{P}^{\frac{1}{2}} \omega\|. \end{aligned} \quad (10.15)$$

It is evident from this equation that the portfolio return in the case of a robust optimization with elliptical uncertainty is smaller than the classical solution by the term $\delta \sqrt{\omega' P \omega}$. The square root of the quantile value can be interpreted as a risk aversion parameter with respect to the uncertainty of the estimates. Substituting the inner solution from equation (10.15) into the robust optimization specification as in (10.12), one obtains

$$\begin{aligned} PR_\lambda &= \arg \min_{\omega \in \Omega} \arg \max_{\mu \in \mathcal{U}} (1 - \lambda) \sqrt{\omega' \Sigma \omega} - \lambda (\omega' \mu) \\ &= \arg \min_{\omega \in \Omega} (1 - \lambda) \sqrt{\omega' \Sigma \omega} - \lambda (\omega' \mu) + \lambda \frac{\delta}{\sqrt{T}} \sqrt{\omega' \hat{\Sigma} \omega} \\ &= \arg \min_{\omega \in \Omega} \left(1 - \lambda + \lambda \frac{\delta}{\sqrt{T}} \right) \sqrt{\omega' \Sigma \omega} - \lambda \omega' \hat{\mu}. \end{aligned} \quad (10.16)$$

Equation (10.16) has the following implications:

- The efficient frontier of a portfolio when optimized robustly under elliptical uncertainty is, except for a shortening factor, the same as the efficient frontier of a classical mean-variance portfolio.
- The optimal weight vector for a minimum-variance portfolio is the same for both kinds of optimization. This must be true by definition, because the uncertainty has been limited to returns only.

With respect to the first point, the risk–return trade-off parameter in (10.16) is now expressed as θ . The equivalent trade-off parameter λ in the problem formulation of (10.7) is then given by

$$\lambda := \frac{\theta}{1 + \theta \frac{\delta}{\sqrt{T}}}. \quad (10.17)$$

The defined interval for $\theta \in [0, 1]$ is now bounded above for the equivalent classical mean-variance portfolios: $\lambda \in [0, 1/(1 + \delta/\sqrt{T})]$.

But the solution is also conservative in the case of an elliptical uncertainty set, in the sense that the optimal portfolio weights correspond to a situation where for all assets the least favorable returns are realized. The mathematical problem formulated as in (10.16) can be expressed in the form of a second-order cone program (SOCP). SOCPs are solved by means of interior-point methods. More precisely, a mathematical program is expressed as an SOCP if it can be written as

$$\begin{aligned} & \arg \min_{\mathbf{x}} \mathbf{f}' \mathbf{x} \\ & \text{subject to } \|A_j \mathbf{x} + \mathbf{b}_j\| \leq \mathbf{c}' \mathbf{x} + d_j \text{ for } j = 1, \dots, J, \end{aligned} \quad (10.18)$$

where J denotes the number of cone constraints. The SOCP formulation includes linear programs (the A_j would be null matrices and the \mathbf{b}_j would be null vectors), quadratic programs, and problems with hyperbolic constraints, as well as problems that can contain sums and maxima of vector norms (see Lobo et al. 1998). As already implied by the name of this formulation, a quadratic cone optimization requires a corresponding cone—also known as a Lorenz or ice-cream cone—with the property that the first element is at least as great as the Euclidean norm of its remaining elements. Analytically, the cone is defined as:

$$\mathcal{C} = \{\mathbf{x} = (x_1, \dots, x_N) \in \mathbb{R}^N : x_1 \geq \|\mathbf{x}_2, \dots, \mathbf{x}_N\|\}. \quad (10.19)$$

The SOCP in (10.18) can also be expressed in its dual form:

$$\begin{aligned} & \arg \max_{\mathbf{z}, \mathbf{w}} - \sum_{j=1}^J (\mathbf{b}'_j \mathbf{z}_j + d_j w_j) \\ & \text{subject to } \sum_{j=1}^J (A'_j \mathbf{z}_j + c_j w_j) = \mathbf{f}, \\ & \|\mathbf{z}_j\| \leq w_j \text{ for } j = 1, \dots, J. \end{aligned} \quad (10.20)$$

Here, $\mathbf{z} \in \mathbb{R}^{n_j-1}$ and $\mathbf{w} \in \mathbb{R}^J$ denote the optimization variables. Like its primal form, this is a convex program, since the objective function to be maximized is concave and the constraints are convex. Further details about these kinds of optimization can be found, for instance, in Nesterov and Nemirovsky (1994) or Boyd and Vandenberghe (2004).

Now, with regard to the task of bringing the problem formulation in (10.12) into the SOCP form, one defines as slack variable for the unknown worst-case portfolio return $t = \hat{\mu}' \boldsymbol{\omega}$. The vector \mathbf{x} in (10.18) and (10.19) is then $\mathbf{x} = (t, \boldsymbol{\omega})'$ and the problem can be written as

$$\begin{aligned} & \arg \min_{t, \boldsymbol{\omega}} t \\ & \text{subject to } t \leq \hat{\mu}' \boldsymbol{\omega} - \delta_\alpha \|P^{\frac{1}{2}} \boldsymbol{\omega}\|, \\ & \sigma_{\max} \geq \|\Sigma^{\frac{1}{2}} \boldsymbol{\omega}\|. \end{aligned} \quad (10.21)$$

The first inequality constraint is the cone constraint and the second is a quadratic constraint with respect to the portfolio risk. Of course, the problem specification in (10.21) can be amended by other linear constraints, such as budget, non-negativity, and/or bound and group constraints.

10.4 Synopsis of R packages

In this section, only packages that are partly or wholly dedicated to robust estimation methods in the context of multivariate data analysis are presented. Thus, R packages that cover robust regression, robust time series analysis, and other robust topics are not included in this synopsis. The reader is referred to the CRAN “Robust” Task View for an overview and brief description of these packages.

10.4.1 The package covRobust

The package **covRobust** was written by Wang et al. (2013). It is contained in the CRAN “Robust” and “Multivariate” Task Views. The package consists of the function `cov.nnve()` only, which is an implementation of the covariance and location estimator introduced by Wang and Raftery (2002). Outliers or groups thereof are detected by a nearest neighbor (NN) procedure and hence the function’s suffix is an acronym for “nearest neighbor variance estimator.” The function is specified with the two main arguments `datamat` for the matrix of sample observations and `k` for choosing the number of neighbors to be considered, with a default value of 12. The remaining arguments can be used to control the optimization procedure. The function returns a `list` object with elements for the robust covariance estimate (`cov`), the robust mean vector (`mu`), and the posterior probabilities (`postprob`), that is, the odds that data points will have to be considered as outliers. The list element `classification` contains the (0, 1) discretized posterior probabilities. Finally, the initial results of the NN cleaning are contained in the list element `innc`, which itself is a list with the four above-named elements.

10.4.2 The package fPortfolio

The package **fPortfolio** is a powerful R package for conducting many different kinds of portfolio optimization task. It is part of the Rmetrics bundle of packages. It is considered a core package in the CRAN “Finance” Task View. The capabilities of the package are concisely described in Würtz et al. (2009, 2014). In this subsection, we focus on how robust estimators can be employed for portfolio optimizations; the package is discussed further in Chapter 12. The pivotal slot for an object of class `fPFOLIOSPEC` is `model`. This slot is itself a `list` object and contains the named element `estimator`. The value of this list element denotes the function for estimating mean and covariance of the assets’ returns and its default value is `"covEstimator"`, which refers to the classical product-moment estimator. The kind of estimator for a given portfolio specification can be queried with the function

`getEstimator()` and altered by the function `setEstimator()`. The latter function expects the name of a function that returns estimates for the locations and dispersion. It returns a `list` object with elements `mu` and `sigma` for the location and scatter, respectively. In **fPortfolio** the functions for robustifying portfolios are `kendallEstimator()`, `spearmanEstimator()`, `mcdEstimator()`, `mveEstimator()`, `covMcdEstimator()`, and `covOGKEstimator()`. The first two functions employ Kendall and Spearman's rank correlations as measures of association between the assets' returns. The difference between the functions `mcdEstimator()` and `covMcdEstimator()` is that the former employs the function contained in the package **MASS** and the latter uses the function contained in the package **robustbase**. Similarly, the MVE estimator, `mveEstimator()`, is imported from the package **MASS**, and the OGK estimator, `covOGKEstimator()`, from the package **robustbase**. However, it should be stressed again that the user can provide any estimator, and in Würtz et al. (2009, Chapter 20) examples are provided in which this approach is elucidated.²

10.4.3 The package MASS

MASS is the package accompanying the book *Modern Applied Statistics with S* (Venables and Ripley 2002). Because of its importance and wide-ranging implementation of methods encountered in statistics, it is part of the base **R** distribution. The package **MASS** is contained in the CRAN "Distributions," "Econometrics," "Environmetrics," "Multivariate," "Pharmacoetics," "Psychometrics," "Robust," and "SocialSciences" Task Views. Within the package, S3-type classes and methods are employed and the package is shipped with a NAMESPACE file. In the following, only the functions pertinent to robust statistics are presented.

The Huber M-estimators are implemented as functions `huber()` and `hubers()`. The former function estimates the location of a data vector. The median absolute deviation (MAD) is used as a scale estimate and the default value for winsorizing the sample is set to $k = 1.5$. The latter function can be employed when either the location or the scale is specified or if neither is supplied by the user. Both functions return a `list` object with elements for the location and scale estimates.

For multivariate data, robust estimates for the location vector and scatter can be computed with the function `cov.rob()`. Two methods are available, `minimum volume ellipsoid` and `minimum covariance determinant`, the former being the default. A classical covariance estimate according to the product moments is calculated if `method = "classical"` has been set as an argument. The functions `cov.mve()` and `cov.mcd()` are wrappers for the former two methods. The function returns a `list` object containing named elements for the location ("center") and dispersion estimates ("cov"). If correlations are also to be

² For completeness' sake, it should be mentioned that shrinkage and bagged estimators for the dispersion matrix are also included in the package **fPortfolio**, namely `shrinkEstimator()` and `baggedEstimator()`, respectively. Furthermore, the reader is directed to the package **tawny** (see Lee and Rowe 2014) for facilitating portfolio optimizations based on shrinkage estimators.

computed (the default is `cor = FALSE`), these will be returned as named list element "`cor`". The remaining list items contain information about the outcome of the numerical optimization.

Finally, the dispersion matrix of a multivariate Student's t distribution can be estimated with the function `cov.trob()`. The t distribution provides some robustness with respect to outliers due to a higher probability mass in the tails compared to the normal distribution. The function allows the inclusion of a weighting vector (`wt`) for the sample observations. The default is to weight the observations equally. The degrees of freedom have to be specified by the argument `nu`; the default value is 5. The argument `cor` is a logical switch (the default is `FALSE`) which serves the same purpose as in `cov.rob()`. The optimization process can be controlled by the arguments `maxit` and `tol`, which limit the maximum number of iterations and the convergence criteria, respectively; the default values are 25 and 0.01. The function returns a `list` object and the elements "`cov`", "`center`", and "`cor`" denote the covariance, the location, and correlation estimates, respectively. Akin to the function `cov.rob()`, the remaining list elements contain information about the numerical optimization.

10.4.4 The package `robustbase`

The package `robustbase` includes the base and essential functions for robust estimation (see Rousseeuw et al. 2015; Todorov and Filzmoser 2009). The package serves as a building block for other packages in the domain of robust statistics. This CRAN package is contained in the “Multivariate,” “Robust,” and “SocialSciences” Task Views. It employs `S4` methods and classes, but also imports `S3` methods. The burdensome computations for some robust methods, in particular the linear model, are interfaced from `C` routines.

With respect to robust statistics the functions `covMcd()` and `covOGK()` can be used to estimate the dispersion. The former is an implementation of the minimum covariance determinant, and the latter of the orthogonalized Gnanadesikan–Kettenring estimator. Both functions return a `list` object from which the location ("`center`") and dispersion ("`cov`") estimates can be extracted.

The generalized class of Huber M-estimators is implemented as function `huberM()`. This returns a `list` object with robust location estimate ("`mu`") and scale estimate ("`s`"), where the MAD is used as default, and the number of iterations is returned as list element "`it`".

10.4.5 The package `robust`

The package `robust` is an `R` port of the Insightful (now TIBCO) package of the same name (see Wang et al. 2014). It is aimed at casual users of robust methods and statistics. It is part of the CRAN “Robust” and “SocialSciences” Task Views and considered a core package—similar to `robustbase`—of the former. The package has dependencies on the packages `fit.models`, `MASS`, `lattice`, `robustbase`, `rrcov`, and `stats`. The estimation of linear and generalized linear models is interfaced from routines written in `C`.

For robust multivariate statistics the function `covRob()` is available. The data for which the robust statistics are required can be specified either as a matrix or `data.frame` object. The user can choose whether covariances or correlations are returned by setting the logical argument `corr` accordingly. The default is to compute a variance-covariance matrix. Further, the argument `distance` is used as a switch to determine whether or not squared Mahalanobis distances are computed (the default value is `TRUE`). Missing data values can be handled by specifying a function for the argument `na.action`; the default action is `na.fail`, but `na.omit` can be utilized instead. The robust estimation method to be employed is determined by the argument `estim`. A character string can be provided to choose between the following multivariate estimators: "`mcd`" for the fast MCD algorithm; "`weighted`" for the re-weighted MCD estimator; "`donostah`" for the Stahel–Donoho estimator; "`M`" for the class of M-estimators; "`pairwiseQC`" for the orthogonalized quadrant correlation estimator; and "`pairwiseGK`" for the orthogonalized Gnanadesikan–Kettenring estimator. The default value of the argument is "`auto`", in which case the function automatically selects one of either the Stahel–Donoho, MCD, or pairwise quadrant correlation estimators by trading off computation time against reasonable estimates for location and scatter. The rule is to use the Stahel–Donoho estimator for data sets that contain less than 1000 observations and less than 10 variables, or less than 5000 observations and less than five variables. If the count of variables is greater than 10 and less than 20 with a maximum of 5000 data points, the MCD estimator is used. In all higher-dimensional cases the orthogonalized pairwise correlation method is used. The MCD estimator is imported from the package `robustbase` and the rest from `rrcov` (see Section 10.4.6). The function `covRob()` accepts an object for controlling the robust estimation. This functional argument can be created with the function `covRob.control()`, which returns a named list of control parameters. These can also be provided via the ellipsis argument when `estim` is not set to its default value "`auto`".

The function returns a `list` object of informal S3 class `covRob`. The list element "`call`" shows how the user invoked the function, that is, the kind of arguments used. Hence, the generic `update()` method can be used for this object. The dispersion estimate (covariances or correlations) is returned as element "`cov`" and the location vector as "`center`". If Mahalanobis distances have been computed, they will appear in the list element "`dist`". If the algorithm has been initialized with robust estimates, these values are returned as "`raw.cov`", "`raw.center`", and "`raw.dist`", for dispersions, locations, and distances, respectively. The name of the chosen estimator is returned as list element "`estim`" and the control parameters as list object "`control`". For objects of this informal class `print()`, `plot()`, and `summary()` methods are provided, and the latter also has its own `print()` method. For the `plot()` method, the user can interactively choose between a display of the eigenvalues, the square root of the Mahalanobis distances, or an ellipses plot. The eigenvalues of the dispersion estimates are contained in objects of informal S3 class `summary.covRob`.

Finally, the function `ccov()` is an implementation of the classical estimators and returns a `list` object of informal S3 class `cov`. The main intention of the package's

developer is to provide a function that returns an object similar in structure to objects of S3 class `covRob` and hence facilitate an easy comparison of estimates. For these objects `plot()`, `print()`, and `summary()` methods are defined.

10.4.6 The package `rrcov`

The package `rrcov` is **dedicated solely to robust multivariate statistics**. As such it depends on the package **robustbase**, but offers a self-contained S4 classes and methods approach for implementing robust estimators (see Todorov and Filzmoser 2009). Further dependencies and/or import directives of the package are **methods**, **lattice**, **cluster**, **pcaPP** (see Filzmoser et al. 2014), and **mvtnorm** (see Genz and Bretz 2009). It is contained in the CRAN “Multivariate” and “Robust” Task Views and is considered to be a core package in the latter. Its structure with regard to the classes defined, methods provided, and robust statistics covered is detailed in a vignette. The package is shipped with a `NAMESPACE` file in which all relevant functions, methods, and classes of the package are exported and S3 methods are imported as generic functions. The computationally intensive calculations are interfaced from C routines.

In addition to robust multivariate statistics, robust methods for **principal component** and **linear discriminant analysis** are also implemented, but the focus here will be on the robust estimators only. A hierarchical class structure is employed, with a virtual class `CovRobust` for which the usual `show()`, `summary()`, and `plot()` methods are defined. An S4 class for each of the multivariate robust estimators is defined; these are derived classes of this virtual class. Similarly, a virtual class `CovControl` is defined from which the classes that control a specific robust method are derived. Basically, **all robust methods for multivariate statistics are implemented**: **M**-, **MM**, and **S**-estimators, as well as **MVE**, **MCD**, and **OGK** methods and the **Stahel–Donoho estimator**. The naming convention for the classes is that a corresponding constructor function of the same name exists. Thus, for instance, the class pertinent to the minimum-covariance determinant method is `CovMcd`, the object’s constructor function is `CovMcd()`, and the construction of these objects (i.e., parameter settings for estimation) is controlled by `CovControlMcd()`, which returns an object of class `CovControlMcd`. Therefore, the classes of the above-mentioned robust methods are `CovMest`, `CovMMest`, `CovSest`, `CovMve`, `CovMcd`, `CovOgk`, and `CovSde` respectively, and the associated constructor functions have the same names. In addition, a **wrapper function `CovRobust()` for these robust methods is available**. In addition to an argument `x` for the data object, it is defined with a `control` argument that requires an object created by one of the control routines for a particular estimation method, for example, the output of `CovControlMve()`. The slots of the formal S4 class objects can be extracted by calling one of the functions `getfoo()`, where `foo` is the name of the slot.

Similar to the package `robust`, the classic estimators for location and scatter are **implemented as function `CovClassic()`**, which returns an S4 object `CovClassic`. The user can choose whether sample or population moments are estimated by employing the logical argument `unbiased`; the default is `TRUE`.

The user can choose between five different kinds of plot for either classical or robust estimates of location and scatter: an index plot of the robust and Mahalanobis distances, a distance–distance plot (for robust methods only), a χ^2 QQ plot of the robust and Mahalanobis distances, an ellipses plot, and a scree plot of the eigenvalues. The `plot()` method takes an argument which the user can employ to control the kind of plot produced; the default is `which = "all"`, whereby all plots are returned successively if executed in interactive mode.

10.4.7 Packages for solving SOCPs

The package `cccp`

The package `cccp` has been added recently to CRAN and is listed in the Task View on “Optimization” (see Pfaff 2015). The package’s name is an acronym for “cone constrained convex programs” and as such is suitable for solving convex objective functions with second-order cone constraints. This is accomplished by means of interior-point methods. The implemented algorithms are partially ported from CVXOPT, a Python module for convex optimization (see <http://cvxopt.org> for more information). Within the package S4 classes and methods are employed. All solver-related computations are interfaced from C++ modules, for which the packages `Rcpp` (see Eddelbüttel 2013; Eddelbüttel and François 2011) and `RcppArmadillo` (see Eddelbüttel and Sanderson 2014) are utilized. `cccp` is endowed with a unit testing framework as provided by `RUnit` (see Burger et al. 2015). A detailed account of the package will be provided in the next chapter and this exposition is just confined to second-order cone constrained programs.

The cornerstone function for solving convex programs is `cccp()`. Here, the user can specify the objective function by means of the arguments `P`, `q`, or `f0`. The cone constraints are provided as a list object and its elements must have been created by one of the functions `nncoc()` (for linear inequality constraints), `psdc()` (for positive semi-definite matrix inequality constraints), and/or `socc()`. The latter is used for specifying second-order cone constraints and has arguments `F`, `g`, `d`, and `f` for providing the second-order cone constraint in the form $\|Fx - g\|_2 \leq d^T x + f$. The function `cccp()` does return a reference class object `Rcpp_CPS` for which a `show()` method is defined. Getter functions for the fields contained in `Rcpp_CPS` objects are made available, too.

The package `Rsocp`

The package `Rsocp` is part of the Rmetrics suite of packages (see Chalabi and Würtz 2014). At the time of writing, the package was hosted on R-Forge only. Within the package, wrapper functions for the software `socp` of Lobo et al. (1997) are implemented. The algorithm for solving SOCPs has been developed in C. These C routines are interfaced from R. In addition to these high-level language functions, the authors provide MatLab scripts. The original software is still available at http://stanford.edu/~boyd/old_software/SOCP.html.

The package contains two functions: `socp()` and `socpControl()`. The former function serves as the wrapper around the underlying C code, and to control the behavior of the algorithm a `list` object created by the latter function can be specified. The SOCP has to be formulated at least in its primal form (see (10.18)) and provides arguments for specifying the parameters of its dual form. If the latter arguments are not provided, then an internal solution is computed before the parameters of the SOCP are passed down to the C routine. This is accomplished by the internal functions `.socp.phase1()` and `.socp.phase2()`. The first of these functions returns an internal solution for x and the second determines corresponding values for z . Incidentally, the package contains a further hidden function `.SqrtMatrix()` for computing the square root of a matrix. The user's guide (Lobo et al. 1997) and a draft version of Lobo et al. (1998) are contained in the package's `doc` directory.

10.5 Empirical applications

10.5.1 Portfolio simulation: robust versus classical statistics

In this first empirical application a simulation comparing classical and robust estimators will be conducted. Returns for five fictional assets will be randomly generated according to one of the following data-generating processes (DGPs):

- Gauss copula with normally distributed margins
- Gauss copula with t -distributed margins
- Student's t copula with t -distributed margins.

The first DGP corresponds to the case where the classical estimators are the best linear unbiased estimators for the Gauss copula with normal margins; the other two DGPs reflect the stylized facts of financial market returns, namely excess kurtosis and tail dependence. For the t distribution five degrees of freedom have been chosen, and the sample sizes for all DGPs are 60, 120, and 240 observations. Assuming a monthly frequency, this corresponds to a 5-, 10-, and 20-year time span for the data in each of the portfolio optimizations. For each DGP a set of 1000 samples has been generated. An equally correlated dependence structure between the five assets has been assumed with a value of $\rho = 0.5$. The generation of these random samples is shown in Listing 10.1.

First, the required packages are loaded into the workspace. The random samples are generated by functions contained in the package `copula` (see Section 9.4.2). Next, copula objects for the Gauss and Student's t copula are created and named `ncop` and `t.cop`, respectively. The DGPs are created by utilizing the function `mvdc()`. This requires an object of class `copula` and a `list` object that contains the parameter information about the marginal distributions assumed. The objects are labeled `NcopMargN`, `NcopMargT`, and `TcopMargT` for the three multivariate distribution models.

R code 10.1 Portfolio simulation: data generation.

```

## Loading of packages
library(cccp)
library(copula)
library(rrcov)
## Creating copula objects
ncop <- normalCopula(param = 0.5, dim = 5)
tcop <- tCopula(param = 0.5, dim = 5, df = 5, df.fixed = TRUE)
## Creating DGPs
NcopMargN <- mvdc(ncop, margins = "norm",
                    paramMargins = list(list(mean = 0, sd = 1)),
                    marginsIdentical = TRUE)
NcopMargT <- mvdc(ncop, margins = "t",
                    paramMargins = list(df = 5),
                    marginsIdentical = TRUE)
TcopMargT <- mvdc(tcop, margins = "t",
                    paramMargins = list(df = 5),
                    marginsIdentical = TRUE)
## Initialising list objects for DGP
Lobj <- list()
length(Lobj) <- 1000
## Setting a seed
set.seed(12345)
## Generating random samples
rNcopMargN <- lapply(Lobj, function(x) rMvdc(240, NcopMargN))
rNcopMargT <- lapply(Lobj, function(x) rMvdc(240, NcopMargT))
rTcopMargT <- lapply(Lobj, function(x) rMvdc(240, TcopMargT))

```

These objects are then employed to draw the random samples. In order to do so, first a `list` object, `Lobj`, is created and the size of the simulation is assigned as its length. Second, a seed is set for replication purposes. In principle, the 1000 random samples for each of the DGPs could be assigned as list elements with a `for` loop, but it is more in the nature of R to use `lapply()` instead. In the last three lines the list object `Lobj` is used for this purpose and the results are stored in the objects `rNcopMargN`, `rNcopMargT`, and `rTcopMargT`. These `list` objects consist of 1000 samples for each of the DGPs with 240 rows and five columns for the fictional asset returns. Sample data for the shorter sample spans can then be swiftly extracted from the list elements.

As the next step, a `function` is created that returns the dispersion estimates for the classical and robust methods. This function can then be applied for the simulated data sets for each of the DGPs and then used for optimizing the minimum-variance portfolios. The comparative simulation will encompass the classical estimators and the M-, MM, S-, MCD, MVE, SD, and OGK robust estimators. The function specification is shown in Listing 10.2, where the estimators of the `rrcov` package are used.

The `function` is specified with three arguments. The first, `x`, is used for the random data set, the second for determining what kind of estimator is employed, and the

R code 10.2 Portfolio simulation: function for estimating moments.

```

## Function for Moment Estimation
1
Moments <- function(x, method = c("CovClassic", "CovMcd",
2
  "CovMest", "CovMMest", "CovMve", "CovOgk",
3
  "CovSde", "CovSest"), ...){
4
  method <- match.arg(method)
5
  ans <- do.call(method, list(x = x, ...))
6
  return(getCov(ans))
7
}
8

```

R code 10.3 Portfolio simulation: estimates for data processes.

```

## Dimensions of Simulation
1
DGP <- c("rNcopMargN", "rNcopMargT", "rTcopMargT")
2
EST <- c("CovClassic", "CovMcd", "CovMest", "CovMMest",
3
  "CovMve", "CovOgk", "CovSde", "CovSest")
4
SAMPLE <- c(60, 120, 240)
5
## Creating list objects for combinations of
6
## DGP and sample sizes
7
## initialising vector for data objects
8
datnames <- NULL
9
for(i in DGP){
10
  for(j in SAMPLE){
11
    objname <- paste(i, j, sep = "")
12
    datnames <- c(datnames, objname)
13
    cat(paste("Creating list object", objname, "\n"))
14
    assign(objname, lapply(eval(as.name(i)),
15
      function(x) x[1:j, ]))
16
  }
17
}
18
## Creating list objects with estimates of
19
## location and dispersion for combinations of
20
## DGP, sample sizes and estimators
21
## initialising vector for list objects
22
objnames <- NULL
23
for(i in datnames){
24
  for(j in EST){
25
    objname <- paste(j, i, sep = "")
26
    objnames <- c(objnames, objname)
27
    cat(paste("Creating list object", objname, "\n"))
28
    assign(objname, lapply(eval(as.name(i)), Moments,
29
      method = j))
30
  }
31
}
32

```

third is the ellipsis argument that is passed down to `do.call()` so that the user has command of the arguments of the estimating function. In the first line of the function body, partial matching for the name of the estimating function is included. In the second line, the function determined by the argument `method` is applied to the data set `x`. Finally, the estimate is extracted from the object `ans` by means of the access function `getCov()` and its result is returned.

This function can now be used to estimate the second moment for each of the list elements and for each of the DGPs and sample sizes. Listing 10.3 shows how this task can be swiftly accomplished. First, the dimension of the simulation study is defined: there are three models for the multivariate distributions, `DGP`. These list objects were created in Listing 10.1. Each list object contains 1000 elements in the form of (240×5) matrices. Next, the function names of the estimators to be used are collected in the character vector `EST`. Finally, the sample sizes are set in the numeric vector `SAMPLE`. In the ensuing code section the samples of different sizes are created. In the first double `for` loop, the object names are created and displayed by the `cat()` function so that the user can better track the progress of the loop. In addition, the names of these list objects are saved in the vector `datnames` for use in the second `for` loop construction. In the last line, `lapply()` is used to extract for each list element and for each `DGP` the number of rows according to `SAMPLE`. Thus, after the execution of the loop has finished, nine new list objects have been created. With these data sets at hand, one can now proceed and estimate the moments for each single list element. This is done in the second double `for` loop. First, the names of the list objects that will store the estimates for the mean and covariance are created, and these names are displayed to enable progress monitoring when the `for` loop is executed. Similar to the first loop, the names of these objects are saved in the character vector `objnames`. In the final line, `lapply()` is used to apply the function `Moments()` to each list element for each of the DGPs and sample sizes. After the successful completion of this loop, a total of $3 \times 3 \times 8 = 72$ list objects have been created and each contains a dispersion estimate. The portfolio optimizations are then conducted with respect to these 72 objects, which implies a total of 72 000 optimizations to be done.

Having created all the necessary data objects, one can proceed with the portfolio optimizations. In Listing 10.4 the function `PortMinVar()` has been defined. The optimization is carried out by employing `cccp()` from the package `cccp`. The minimum-variance optimization takes place under the constraints of being fully invested (objects `a1` and `b1`) and with only long positions allowed (objects `nno1`). The function returns the weight vector. In the ensuing `for` loop the optimizations are carried out and the 1000 portfolio risk figures are stored for each of the `DGP`, estimator, and sample size combinations. Finally, the median and interquartile range (IQR) are computed for the portfolio risks.

The simulation results are summarized in Table 10.1. The medians and interquartile ranges are reported in the columns for each of the three DGPs. The results are grouped by sample size. A couple of conclusions can be drawn from this table. Even in the case of the `Gauss` copula with normally distributed margins the classical covariance estimator does not yield a portfolio structure of lowest risk on average, and the dispersion of the portfolio risks is also not the smallest. This result holds for all

R code 10.4 Portfolio simulation: minimum-variance optimizations.

```

## Function for minimum-variance portfolio
## Constraints: Fully invested, long-only
PortMinVar <- function(x){
  k <- ncol(x)
  nn01 <- nnoc(G = -diag(k), h = rep(0, k))
  A1 <- matrix(rep(1, k), nrow = 1)
  b1 <- 1.0
  opt <- cccp(P = x, q = rep(0, k), A = A1, b = b1,
               cList = list(nn01),
               optctrl = ctrl(trace = FALSE))
  getx(opt)
}
## Conduct optimisation
portnames <- NULL
idx <- 1:1000
for(i in objnames){
  objname <- paste("Port", i, sep = "")
  portnames <- c(portnames, objname)
  obj <- eval(as.name(i))
  weights <- lapply(obj, PortMinVar)
  assign(objname, sapply(idx, function(x)
    sqrt(crossprod(weights[[x]], obj[[x]]) %*%
      weights[[x]])))
}
## Calculate median and IQR of portfolio risks
mednames <- NULL
iqrnames <- NULL
for(i in portnames){
  objname1 <- paste("Med", i, sep = "")
  objname2 <- paste("IQR", i, sep = "")
  mednames <- c(mednames, objname1)
  iqrnames <- c(iqrnames, objname2)
  assign(objname1, median(eval(as.name(i))))
  assign(objname2, IQR(eval(as.name(i))))
}

```

sample sizes, but the differences are becoming negligible for $T = 240$ with respect to the median risks. Lower portfolio risks can be achieved with the robust OGK and Stahel–Donoho estimators for this DGP. For $T = 60$ the M-, MM, and S-estimators excel likewise.

For the second DGP, a multivariate distribution with excess kurtosis but no tail dependence, two observations can be made. First, the risk levels increase for all types of estimator and this increase is most pronounced for the classical estimator. In contrast, the increase in the median risks for the robust estimators is rather negligible, and

Table 10.1 Portfolio simulation: summary of portfolio risks

Estimator	Normal / Normal		Normal / Student		Student / Student	
	Median	IQR	Median	IQR	Median	IQR
<i>T = 60</i>						
Classic	0.75	0.098	0.91	0.142	0.92	0.158
MCD	0.76	0.147	0.79	0.178	0.82	0.172
M	0.73	0.131	0.76	0.161	0.79	0.159
MM	0.74	0.100	0.82	0.121	0.82	0.129
MVE	0.74	0.154	0.77	0.173	0.80	0.165
OGK	0.66	0.117	0.67	0.129	0.67	0.129
SDE	0.70	0.141	0.72	0.149	0.75	0.164
S	0.73	0.110	0.80	0.127	0.82	0.144
<i>T = 120</i>						
Classic	0.76	0.067	0.95	0.106	0.96	0.121
MCD	0.77	0.091	0.82	0.102	0.83	0.111
M	0.75	0.081	0.79	0.105	0.81	0.107
MM	0.76	0.067	0.85	0.084	0.84	0.090
MVE	0.76	0.096	0.81	0.106	0.83	0.114
OGK	0.69	0.080	0.71	0.089	0.69	0.090
SDE	0.74	0.094	0.77	0.098	0.80	0.109
S	0.75	0.075	0.82	0.086	0.84	0.096
<i>T = 240</i>						
Classic	0.77	0.052	0.97	0.084	0.97	0.097
MCD	0.77	0.058	0.83	0.074	0.85	0.077
M	0.77	0.059	0.82	0.077	0.82	0.077
MM	0.76	0.051	0.86	0.067	0.86	0.066
MVE	0.77	0.064	0.84	0.079	0.85	0.083
OGK	0.70	0.056	0.72	0.068	0.71	0.065
SDE	0.76	0.061	0.80	0.074	0.81	0.075
S	0.76	0.052	0.84	0.068	0.86	0.068

all of these estimators outperform the classic estimator, that is, producing portfolio allocations of lower risk. The OGK estimator fares best.

This picture remains pretty much unchanged for the third DGP, where now tail dependencies exist between the included assets. The differences between the second and third DGPs in the median levels are rather small for all estimators and sample sizes. One can conclude that tail dependence does not have a material impact on the overall portfolio risk for the given simulation design.

The two main take-home messages are, first, that even in the case of the multivariate normal distribution, it can be advantageous to employ a robust estimator, in particular OGK, instead of the classical estimator; and second, all robust estimators are quite

immune with respect to different DGPs, but violations of the model assumptions exert a great impact on portfolio riskiness when a classical estimator is utilized.

When the interquartile ranges of the portfolio risks for each of the estimators and sample sizes are compared, two features emerge. First, in the case of the Gaussian DGP, the portfolio risks are more concentrated compared to the robust estimators for all sample sizes. This result confirms the best linear unbiased estimator property of the classical estimator. Second, by slightly violating the assumption of normality the implied portfolio risks of the classical estimator are dispersed more widely compared to most of the robust estimators. For the latter, the interquartile range remains pretty much the same, regardless of DGP.

10.5.2 Portfolio back test: robust versus classical statistics

In this subsection a back-test of a minimum-variance portfolio optimization with long-only and full investment for major stock market indexes is conducted. The classical and the robust estimators are utilized for moment estimation. The six stock indexes covered are the S&P 500, Nikkei 225, FTSE 100, CAC 40, DAX, and Hang Seng. The sample of month's-end index levels starts on 31 July 1991 and ends on 30 June 2011, thus comprising 240 observations.

In Listing 10.5 the necessary packages are first loaded into memory. The data set is contained in the package **FRAPO** accompanying this book. In principle, the portfolio back-test could be conducted with the facilities offered in **fPortfolio**, but for ease of comprehension a customized back-test will be described in which the functions and methods of the packages loaded in lines 3–6 will be employed.

With the packages in memory, the data set is loaded and converted to a **zoo** object. Percentage discrete returns are calculated, and by way of preliminary data analysis descriptive statistics are calculated and box-plots drawn for each index.

R code 10.5 Portfolio back-test: descriptive statistics of returns.

```

## Loading of packages
library(FRAPO)                                     1
library(PerformanceAnalytics)                      2
library(quadprog)                                    3
library(rrcov)                                     4
library(zoo)                                       5
## Loading data, calculate returns
data(StockIndex)                                    6
pzoo <- zoo(StockIndex, order.by = rownames(StockIndex))  7
rzoo <- (pzoo / lag(pzoo, k = -1) - 1) * 100        8
## boxplot and descriptive statistics
boxplot(coredata(rzoo))                           9
rstats <- rbind(apply(rzoo, 2, summary),          10
                skewness(rzoo),                      11
                kurtosis(rzoo))                     12
                )

```

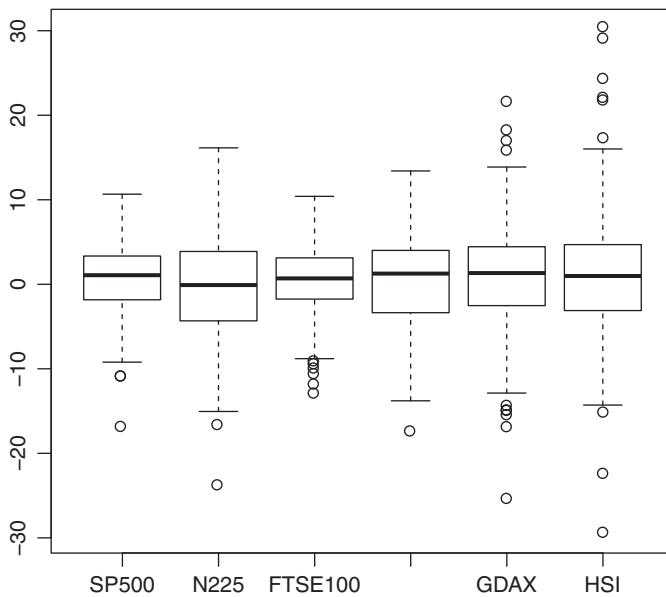


Figure 10.1 Box-plots of stock index returns.

The box-plot is shown in Figure 10.1, from which the stylized facts of financial market returns are clearly evident. The returns are skewed to the left and the span of the returns indicates excess kurtosis. The most volatile returns series is the Hang Seng index. All returns series have outliers in the left tail, but only the German DAX and the Hang Seng have them in the right tail. The key descriptive statistics are summarized in Table 10.2.

The code for conducting the back-test is shown in Listing 10.6. First, a character vector, `EST`, with the names of the estimators to be used, is created. Next, a function, `PfBack()`, is defined for determining the weights of the minimum-variance portfolio for a given estimator. The previously defined functions `Moments()`

Table 10.2 Portfolio back-test: descriptive statistics of returns

Statistics	SP500	N225	FTSE100	CAC40	GDAX	HSI
Minimum	-16.940	-23.830	-13.020	-17.490	-25.420	-29.410
First quartile	-1.854	-4.319	-1.762	-3.351	-2.533	-3.122
Median	1.067	-0.082	0.690	1.266	1.320	0.959
Mean	0.604	-0.190	0.436	0.499	0.829	1.009
Third quartile	3.323	3.886	3.115	4.017	4.431	4.679
Maximum	10.640	16.150	10.400	13.410	21.380	30.160
Skewness	-0.558	-0.249	-0.481	-0.296	-0.457	0.242
Excess kurtosis	0.985	0.484	0.420	0.215	1.763	2.310

R code 10.6 Portfolio back test: rolling window optimization.

```

## Conduct back test
EST <- c("CovClassic", "CovMcD", "CovMest", "CovMMest",
        "CovMve", "CovOgk", "CovSde", "CovSest")
## Function for back test
PfBack <- function(x, method = c("CovClassic", "CovMcD",
        "CovMest", "CovMMest", "CovMve", "CovOgk",
        "CovSde", "CovSest"), ...) {
  cov <- Moments(x, method = method)
  return(PortMinVar(cov))
}
## Conducting back test
PfWeights <- lapply(EST, function(x)
  rollapply(rzoo, width = 120, FUN = PfBack,
            method = x, by.column = FALSE,
            align = "right"))

periods <- as.Date(index(PfWeights[[1]]))
## Calculate portfolio returns / relative performance
PfReturns <- lapply(PfWeights, function(x)
  rowSums(lag(x, k = -1) * rzoo))
PfReturns <- zoo(matrix(unlist(PfReturns),
  ncol = length(PfReturns)), periods)
colnames(PfReturns) <- EST
PortOut <- (PfReturns[, -1] - PfReturns[, 1])
## plot relative performance
plot(PortOut, type = "h",
      xlab = "",
      ylab = EST[-1],
      main = "Relative Performance",
      ylim = range(PortOut))
## statistics on relative performance
PortRelStats <- rbind(apply(PortOut, 2, summary),
                      skewness(PortOut)
)

```

and `PortMinVar()` are used. The function `PfBack()` is then used by `rollapply()` to calculate the portfolio weights by using a rolling window of 120 observations. The resulting `zoo` object is named `PfWeights`. The function `rollapply()` is called within an `lapply()` enclosure and the character vector `EST` has been used as a `list` object. Hence, the object `PfWeights` is itself a list of `zoo` objects with the weights for each of the estimation methods as in `EST`. Next, its index is stored as the object `periods`. For each of the underlying estimators, the portfolio returns can be computed swiftly by employing the `lag()` function such that the weights—lagged by one period—are multiplied by the respective returns and the row sums of these products are the portfolio returns. This calculation

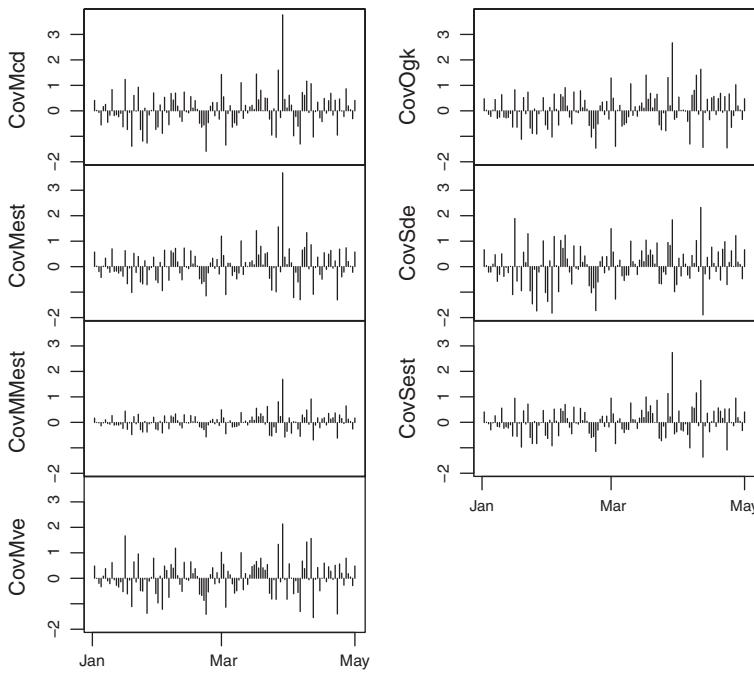


Figure 10.2 Relative performance of robust portfolios.

again involves the `lapply()` function. In the next two lines this `list` object is transformed to a `zoo` object and the excess returns of the portfolios based on robust estimators compared to the classical covariance estimator are computed. In the last two blocks of code the excess returns are displayed and summary statistics computed.

The excess returns of the back-test based upon the robust statistics compared to the classical estimators are displayed in Figure 10.2 and the relevant summary statistics in Table 10.3. For the given measures, the outcome of the robust optimizations is more favorable than that of the classical estimator. The average excess return is positive

Table 10.3 Portfolio back-test: descriptive statistics of excess returns

Estimator	Min	1st quartile	Median	Mean	3rd quartile	Max	Skew
MCD	-1.592	-0.350	-0.003	0.034	0.427	3.768	1.054
M	-1.312	-0.314	-0.003	0.039	0.438	3.682	1.279
MM	-0.685	-0.166	0.004	0.024	0.195	1.696	1.115
MVE	-1.545	-0.372	-0.020	0.021	0.454	2.132	0.143
OGK	-1.468	-0.337	0.017	0.037	0.484	2.670	0.363
SDE	-1.898	-0.376	0.061	0.035	0.515	2.319	-0.018
S	-1.360	-0.294	0.000	0.044	0.423	2.749	0.810

for all robust portfolios, and they are skewed to the right. Certainly the skewness is affected by the large positive excess return in 2009, but even if one disregards this data point a positive skew results. This can be concluded by the difference between the third quartile and the absolute value of the first quartile, which is greater than zero for all robust estimators. Compared to the other robust estimators, the performance of the MM estimator is the least favorable, giving the lowest mean and maximum excess return. Among the robust estimators with a positive median excess return, the MCD and MM estimators produced portfolio allocations that yielded excess returns of highest skewness, followed by the OGK and Stahel–Donoho estimators.

10.5.3 Portfolio back-test: robust optimization

In this subsection robust portfolio optimizations with elliptical uncertainty for the expected returns are applied to the data set of the previous section. The elliptical uncertainty will be based on the ML point estimates. The efficient frontiers for a classical mean-variance optimization as given in (10.7) and the robust counterpart optimization as given in (10.16) will then be compared. Please recall that these two problem formulations only differ with respect to the factor in the first term. The SOCP will be solved by utilizing the function `cccp()` contained in the package `cccp` (see Pfaff 2015). That is, the portfolio optimization problems in (10.7) and (10.16) are cast so as to maximize the expected portfolio return for a given portfolio standard deviation risk. Before doing so, the function `PMV()` for carrying out this task is defined; it returns the portfolio weights and is exhibited in the Listing 10.7.

Here, the SOCP is expressed in its primal form as in (10.18). The arguments of the function are `SRoot` for the square root of the variance-covariance matrix, `mu` for the

R code 10.7 Robust portfolio optimization with elliptical uncertainty.

```

## Defining function for points on efficient frontier
PMV <- function(SRoot, mu, SigTerm,
                  optctrl = ctrl(trace = FALSE)){
  N <- nrow(SRoot)
  ## Portfolio risk constraint
  soc1 <- socc(F = SRoot, g = rep(0, N), d = rep(0, N),
                f = SigTerm)
  ## non-negativity constraint
  nn01 <- nnoc(G = -diag(N), h = rep(0, N))
  ## Budget constraint
  A1 <- matrix(rep(1, N), nrow = 1)
  b1 <- 1.0
  ## optimization
  ans <- cccp(q = -mu, A = A1, b = b1,
              cList = list(nn01, soc1), optctrl = optctrl)
  getx(ans)
}

```

point estimates of the expected returns, `SigTerm` for the first term as in (10.7) and (10.16), and `optctrl` for providing user-specified control options with respect to the function `cccp()`. As a default, this argument is initialized such that the solver's trace is not printed. In the function's body, first the number of assets is determined by the row count of `SRoot`. In the following lines, the portfolio constraints are created with respect to a fully invested, long-only portfolio. The portfolio standard deviation risk constraint is defined by calling the function `socc`. This constraint consists of the square root of the variance covariance matrix as the left-hand side argument and the constant term `SigTerm` as the upper bound. This second-order cone constraint is assigned to the object `soc1`. The non-negativity constraints are created by calling the function `nnoc` for creating constraints with respect to the non-negative orthant cone. Because all cone constraints are formulated as lower or equal constraints with respect to a cone, the negative diagonal matrix has to be provided as the left-hand side argument `G`. This constraint is assigned to the object `nn01`. Finally, the fully invested requirement is cast as an equality constraint (objects `A1` and `b1`). The optimization is then swiftly conducted by calling the function `cccp()`, and the linear objective to be minimized is set to minus the negative returns. The weighting parameter λ as in (10.7) and (10.16) does not have to be provided, given that it is a scalar and does not influence the optimization outcome, but only the value of the objective function. The remaining arguments passed to `cccp()` are the budget constraint (arguments `A` and `b`), the two previously created cone constraints cast in a `list` object as argument `cList`, and the control settings for the optimizer (argument `optctrl`).

With this wrapper function in Listings 10.7 one can now trace the efficient frontiers of the mean-variance and the robustly optimized portfolios. The relevant R code for doing so is exhibited in Listing 10.8.

First, the package `cccp` is loaded into the workspace. In lines 2–8 the parameters used in the optimizations are defined. These consist of the count assets and observations, the expected return vector set as the column means of `rzoo`, the variance-covariance matrix of the returns and the square-root thereof, and the 90% quantile for the returns uncertainties. In lines 9–12 a sequence of ten risk aversion parameters is determined for computing the points on the efficient frontier. These are constructed such that the range encompasses the portfolio standard deviation risks greater than 110% of the least risky and 90% of the riskiest asset in terms of the standard deviation risk.

Commencing in line 14, two `matrix` objects for storing the optimization results of the classical mean-variance portfolio approach (object `MVans`) and its robust counterpart (object `RCans`) are defined, respectively.

The computation of the ten points on the efficient frontier for each optimization scheme is conducted in a `for` loop starting on line 18. In the loop's body, the implied allocation by the classical mean-variance approach for a given risk aversion parameter `ra[i]` is determined and assigned to the object `wmv`. Next, the associated portfolio standard deviation risk, the portfolio return to be expected and the allocation itself is stored in the *i*th row of the `matrix` object `MVans`. Similarly, the same is accomplished for the robust counterpart setting in the remaining part of the loop's

R code 10.8 Efficient frontiers for mean-variance and robust counterpart optimization with elliptical uncertainty of μ .

```

library(cccp)                                     1
## Setting of parameters                         2
Nassets <- ncol(rzoo)                           3
Nobs <- nrow(rzoo)                             4
mu <- colMeans(rzoo)                           5
S <- cov(rzoo)                                 6
SR <- sqrtm(S)                                7
delta <- sqrt(qchisq(0.9, Nassets))            8
## Determining feasible risk aversion           9
SigMax <- max(colSds(rzoo))                   10
SigMin <- min(colSds(rzoo))                   11
ra <- seq(SigMin * 1.1, SigMax * 0.9, length.out = 10) / SigMax 12
## Initializing objects for MV and robust counterpart results 13
RCans <- MVans <- matrix(NA,                      14
                           nrow = 10,                  15
                           ncol = Nassets + 2)        16
## Computing points on efficient frontier and allocations 17
for(i in 1:10){                                18
  ## minimum-variance                           19
  wmv <- PMV(SRoot = SR, mu = mu, SigTerm = SigMin / ra[i]) 20
  MVans[i, ] <- c(sqrt(t(wmv) %*% S %*% wmv),           21
                  crossprod(mu, wmv),                         22
                  wmv)                                         23
  ## robust counterpart                         24
  theta <- ra[i] + (1 - ra[i]) * delta / sqrt(Nobs)        25
  wrc <- PMV(SRoot = SR, mu = mu, SigTerm = SigMin / theta) 26
  RCans[i, ] <- c(sqrt(t(wrc) %*% S %*% wrc),           27
                  crossprod(mu, wrc),                         28
                  wrc)                                         29
}

```

body. As an outcome of this `for` loop, two `matrix` objects have been filled with respect to the classical and robust counterpart optimizations. As is evident from the invocation of the `cccp()` function, the two optimizations only differ from each other with respect to the provided argument `sigTerm`.

Now, recall (10.17) in which the relation between the robust counterpart and the classical mean-variance risk-weighting parameter λ was stated. For a given value of the risk-weighting scalar $\theta \in [0, 1]$ in the robust counterpart optimization, an equivalent value for the classical mean-variance setting can be computed. This is the subject in Listing 10.9.

R code 10.9 Determining equivalent mean-variance allocation for a given robust counterpart risk weighting.

```

1  ## Equivalent weighting parameter for theta
2  theta2lambda <- function(theta, Nassets, Nobs, level){
3    delta <- sqrt(qchisq(level, Nassets))
4    lambda <- theta / (1 + theta * (delta / sqrt(Nobs)))
5    lambda
6  }
7  ## robust allocation and equivalent
8  ## mean-variance allocation
9  theta <- 0.7
10 wrc <- PMV(SRoot = SR, mu = mu, SigTerm = SigMin / theta)
11 ## RC point on efficient frontier
12 rceq <- c(sqrt(t(wrc) %*% S %*% wrc), crossprod(mu, wrc))
13 ## Equivalent risk weighting
14 rweq <- theta2lambda(theta, Nassets = Nassets, Nobs = Nobs,
15                      level = 0.9)
16 ## Equivalent MV point on efficient frontier
17 wmv <- PMV(SRoot = SR, mu = mu, SigTerm = SigMin / rweq)
18 mveq <- c(sqrt(t(wmv) %*% S %*% wmv), crossprod(mu, wmv))

```

First, the function `theta2lambda()` is defined for computing the equivalent risk weighting for the mean-variance portfolio. The function's closure has arguments `theta` for a given risk weighting of the robust counterpart portfolio, `Nassets` for the count of assets, `Nobs` for the sample size, and `level` for the desired confidence level. The arguments `Nassets` and `level` are employed in the computation of the quantile according to a χ^2 distribution. Starting in line 10, the robust allocation for a value of `theta = 0.7` is determined (object `wrc`) and the associated point on the efficient frontier is assigned to the object `rceq` in line 12. Next, the equivalent risk weighting is computed by calling `theta2lambda()`, and its returned value is assigned to the object `rweq`. This scalar is then utilized in the call to `PMV()` to determine the equivalent classical mean-variance allocation (object `wmv`). Finally, the associated point on the efficient frontier is computed and stored as `mveq`.

The R code for producing a graphical display of the efficient frontier with equivalence points is exhibited in Listing 10.10.

First, the limits of the x - and y -axes are set. Next, the efficient frontier for the computed points according to the classical mean-variance allocations are plotted as a line. Superimposed on this graphic are the efficient points with respect to the mean-variance and robust counterpart allocations. Finally, the equivalent mean-variance and robust counterpart points are drawn in as computed in Listing 10.9. The outcome is displayed in Figure 10.3.

R code 10.10 Graphical display of efficient frontier for mean-variance and robust counterpart portfolios.

```

## Efficient Frontier
## determining ranges
1
2
3
xlims <- c(SigMin * 0.9 ,
            max(cbind(MVans[, 1], RCans[, 1])))
4
5
6
ylims <- c(min(cbind(MVans[, 2], RCans[, 2])),
            max(cbind(MVans[, 2], RCans[, 2])))
7
## plotting efficient frontier for MV
8
plot(MVans[, 1], MVans[, 2], type = "l",
      9
      xlim = xlims,
      10
      ylim = ylims,
      11
      xlab = expression(sigma),
      12
      ylab = expression(mu))
13
## Superimposing points
14
for(i in 1:nrow(MVans)){
      15
      points(x = MVans[i, 1], y = MVans[i, 2], col = "blue",
      16
              pch = 19)
      17
      points(x = RCans[i, 1], y = RCans[i, 2], col = "red",
      18
              pch = 19)
}
19
## Superimposing equivalence points
20
points(x = rceq[1], y = rceq[2], col = "darkgreen",
      21
              bg = "darkgreen", pch = 23)
22
points(x = mveq[1], y = mveq[2], col = "green", bg = "green",
      23
              pch = 23)
24
## Legend
25
legend("topleft", legend = c("Efficient Frontier", "MV points",
      26
              "RC points",
      27
              expression(paste("RC allocation with ",
      28
                  theta == 0.7)),
      29
              "Equivalent MV-Portfolio"),
      30
              lty = c(1, NA, NA, NA, NA), pch = c(NA, 19, 19, 23, 23),
      31
              pt.bg = c(NA, NA, NA, "darkgreen", "orange"),
      32
              col = c("black", "blue", "red", "darkgreen", "orange")))
      33

```

It can be seen that the robust optimization solutions selected are part of the mean-variance efficient frontier, though the solutions of the former fall short of the latter. This becomes apparent when one compares the equivalent solutions of the mean-variance and robust optimizations. The solution indicated as a point to the far north-east is the corresponding mean-variance solution to the adjacent robust counterpart, which lies to the south-west of it.

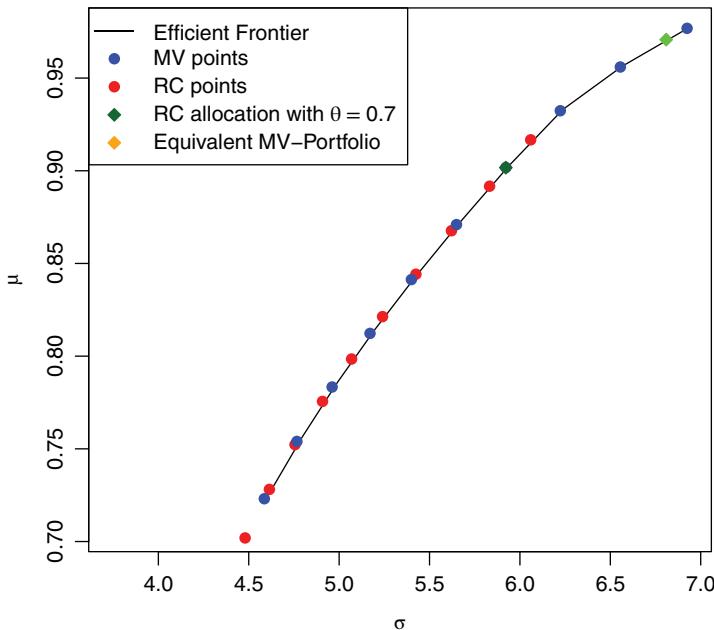


Figure 10.3 Efficient frontier of mean-variance and robust portfolios.

References

- Ben-Tal A. and Nemirovski A. 1998 Robust convex optimization. *Mathematics of Operations Research* **23**(4), 769–805.
- Boyd S. and Vandenberghe L. 2004 *Convex Optimization*. Cambridge University Press, Cambridge.
- Burger M., Jünemann K., and König T. 2015 *RUnit: R Unit Test Framework*. R package version 0.4.28.
- Chalabi Y. and Würtz D. 2014 *Rsocp: An R extension library to use SOCP from R*. R package version 271.1.
- Cornuejols G. and Tütüncü R. 2007 *Optimization Methods in Finance*. Cambridge University Press, Cambridge.
- Davies P. 1987 Asymptotic behavior of S-estimators of multivariate location parameters and dispersion matrices. *The Annals of Statistics* **15**, 1269–1292.
- Donoho D. 1982 Breakdown properties of multivariate location estimators. Technical report, Harvard University, Boston.

- Eddelbüttel D. 2013 *Seamless R and C++ Integration with Rcpp*. Springer, New York.
- Eddelbüttel D. and François R. 2011 Rcpp: Seamless R and C++ integration. *Journal of Statistical Software* **40**(8), 1–18.
- Eddelbüttel D. and Sanderson C. 2014 RcppArmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics and Data Analysis* **71**, 1054–1063.
- Fabozzi F., Focardi S., Kolm P., and Pachamanova D. 2007 *Robust Portfolio Optimization and Management*. John Wiley & Sons, New Jersey.
- Filzmoser P., Fritz H., and Kalcher K. 2014 *pcaPP: Robust PCA by Projection Pursuit*. R package version 1.9-60.
- Genz A. and Bretz F. 2009 *Computation of Multivariate Normal and t Probabilities*. Lecture Notes in Statistics. Springer-Verlag, Heidelberg.
- Gnanadesikan R. and Kettenring J. 1972 Robust estimates, residuals and outlier detection with multiresponse data. *Biometrics* **28**, 81–124.
- Hampel F. R., Rochetti E. M., Rousseeuw P. J., and Stahel W. A. 1986 *Robust Statistics, The Approach Based on Influence Functions*. John Wiley & Sons, New York.
- Huber P. 1964 Robust estimation of a location parameter. *Annals of Mathematical Statistics* **35**, 73–101.
- Huber P. J. 1981 *Robust Statistics*. John Wiley & Sons, New York.
- Lee B. and Rowe Y. 2014 *tawny: Provides various portfolio optimization strategies including random matrix theory and shrinkage estimators*. R package version 2.1.2.
- Lobo M., Vandenberghe L., and Boyd S. 1997 *SOCP: Software for Second-order Cone Programming, User's Guide*. Stanford University Stanford. Beta Version.
- Lobo M., Vandenberghe L., Boyd S., and Lebret H. 1998 Applications of second-order cone programming. *Linear Algebra and its Applications* **284**, 193–228.
- Lopuhaä H. 1991 Multivariate τ -estimators for location and scatter. *Canadian Journal of Statistics* **19**, 307–321.
- Lopuhaä H. 1992 Highly efficient estimators of multivariate location with high breakdown point. *The Annals of Statistics* **20**, 398–413.
- Maronna R. and Zamar R. 2002 Robust estimates of location and dispersion of high-dimensional datasets. *Technometrics* **44**(4), 307–317.
- Maronna R., Martin D., and Yohai V. 2006 *Robust Statistics: Theory and Methods*. John Wiley & Sons, New York.
- Meucci A. 2005 *Risk and Asset Allocation*. Springer-Verlag, New York.
- Nesterov Y. and Nemirovsky A. 1994 *Interior-Point Polynomial Methods in Convex Programming* vol. **13** of *Studies in Applied Mathematics*. SIAM, Philadelphia.
- Pfaff B. 2010 *Modelling Financial Risks: Fat Tails, Volatility Clustering and Copulae*. Frankfurt Allgemeine Buch, Frankfurt am Main.
- Pfaff B. 2015 *cccp: Cone Constrained Convex Problems*. R package version 0.2-4.
- Rousseeuw P. 1985 Multivariate estimation with high breakdown point In *Mathematical Statistics and Applications* (ed. Grossmann W., Pflug G., Vincze I., and Wertz W.) vol. B Reidel Publishing Dordrecht pp. 283–297.
- Rousseeuw P. and Leroy A. 1987 *Robust Regression and Outlier Detection*. John Wiley & Sons, New York.
- Rousseeuw P., Croux C., Todorov V., Ruckstuhl A., Salibian-Barrera M., Verbeke T., Koller M., and Mächler M. 2015 *robustbase: Basic Robust Statistics*. R package version 0.92-5.

- Scherer B. 2010 *Portfolio Construction and Risk Budgeting* 4th edn. Risk Books, London.
- Schötte K. 2007 *Robust Optimization with Application in Asset Management* Dissertation Technische Universität München München.
- Stahel W. 1981 *Robuste Schätzungen: Infinitesimale Optimalität und Schätzungen von Kovarianzmatrizen* PhD thesis Swiss Federal Institute of Technology (ETH) Zürich.
- Staudte R. G. and Sheather S. J. 1990 *Robust Estimation and Testing*. John Wiley & Sons, New York.
- Todorov V. and Filzmoser P. 2009 An object-oriented framework for robust multivariate analysis. *Journal of Statistical Software* **32**(3), 1–47.
- Tüntüci R. and König M. 2004 Robust asset allocation. *Annals of Operation Reserach* **132**, 132–157.
- Venables W. N. and Ripley B. D. 2002 *Modern Applied Statistics with S* 4th edn. Springer, New York.
- Wang J., Zamar R., Marazzi A., Yohai V., Salibian-Barrera M., Maronna R., Zivot E., Rocke D., Martin D., Maechler M., and Konis K. 2014 *robust: Robust Library*. R package version 0.4-16.
- Wang N. and Raftery A. 2002 Nearest neighbor variance estimation (NNVE): Robust covariance estimation via nearest neighbor cleaning (with discussion). *Journal of the American Statistical Association* **97**, 994–1019.
- Wang N., Raftery A., and Fraley C. 2013 *covRobust: Robust Covariance Estimation via Nearest Neighbor Cleaning*. R package version 1.1-0.
- Würtz D., Setz T., and Chalabi Y. 2014 *fPortfolio: Rmetrics—Portfolio Selection and Optimization*. R package version 3011.81.
- Würtz D., Setz T., Chalabi Y., Chen W., and Ellis A. 2009 *Portfolio Optimization with R/Rmetrics Update 2015* Rmetrics eBooks. Rmetrics Association and Finance Online Publishing, Zürich.
- Yohai V. 1987 High breakdown-point and high efficiency estimates for regression. *The Annals of Statistics* **15**, 642–656.
- Yohai V., Stahel W., and Zamar R. 1991 A procedure for robust estimation and inference in linear regression In *Directions in Robust Statistics and Diagnostics (Part II)* (ed. Stahel W. and Weisberg S.) vol. **34** of *The IMA Volumes in Mathematics and its Applications* Springer-Verlag New York pp. 365–374.

Diversification reconsidered

11.1 Introduction

It almost goes without saying that one purpose of wealth allocation is the diversification of risks. That being so, the utility of a risk-averse investor is increased when the wealth is allocated to an asset mix instead of holding a single risky asset. Concepts and methods for the assessment of market risks have been presented in Part II of this book. The risk measures derived from the statistical models presented have been applied on an *ex post* basis—that is, for a given portfolio allocation and its implied loss function, the riskiness of the investment can be assessed for a given confidence level. In this chapter and the next, approaches to portfolio optimization are presented that directly address the issue of asset allocation such that the purpose of risk diversification is directly taken into account.

So far, the term “diversification” has been used rather loosely and the approaches presented have not questioned the view that the variance-covariance matrix of the returns is an appropriate statistic for measuring diversification, except in the chapter on modelling dependencies (i.e., copulae). Diversification entails at least two dimensions. The first dimension addresses the question of the underlying common characteristic with respect to which the assets are diverse. The second addresses the question of how to measure the degree of diversification with respect to this characteristic. For instance, a risk-averse long-only investor would not be better off if he held a portfolio with strong co-movements in the sign of the returns compared to an asset mix where the returns of the constituent assets do not share this characteristic, *ceteris paribus*. The variance-covariance matrix of the returns is then often employed as the measure to assess the riskiness of the assets and the dependencies between them. But

here diversification is defined simply with respect to the overall dependencies in the returns, measured correctly with the variance-covariance matrix. In general, this is no safeguard against risk per se.

In Section 11.2 the focus is first on how to measure the degree of diversification and how to determine a portfolio allocation such that it is “most diversified.” Then, in Section 11.3, “diversification” is characterized by the contribution of the portfolio’s constituent assets to the overall portfolio risk. Here, diversification is favorable if the risk contributions are not concentrated on only a few assets, and hence a portfolio allocation is regarded as well diversified if each of its constituents contributes an equal amount to the overall portfolio risk. In Section 11.4, in which diversification is reconsidered, optimal tail-(in)dependent portfolios are constructed. Now “diversification” is used in the sense of extreme market events, such that the focus is shifted toward the portfolio’s behavior in overall market downturns. This can then be measured by the (lower) tail dependence coefficient (TDC; see Chapter 9). In Section 11.5, the R packages in which the portfolio optimization approaches either are directly implemented or can be employed for optimization are presented, and the chapter concludes with examples in which the solutions of the most diversified, the equal-risk contributed, and the minimum tail-dependent portfolio approaches are contrasted with the allocation of the global-minimum variance portfolio. All optimization approaches have a common goal, namely to minimize the investor’s risk through diversification, but they differ in the two dimensions of “diversity with respect to which characteristic?” and “how to measure it?”

11.2 Most-diversified portfolio

In a series of papers, Choueifaty and Coignard (2008) and Choueifaty et al. (2011) addressed the theoretical and empirical properties of portfolios when diversification is used as a criterion. In order to do so, they first established a measure by which the degree of diversification for a long-only portfolio can be assessed. Let Σ denote the variance-covariance matrix of the returns for N assets and σ the vector of asset volatilities measured by their respective standard deviations. The diversification ratio (DR) is then defined for a given ($N \times 1$) weight vector ω in the allowed set of portfolio solutions Ω as

$$DR_{\omega \in \Omega} = \frac{\omega' \sigma}{\sqrt{\omega' \Sigma \omega}}. \quad (11.1)$$

The numerator is the weighted average volatility of the single assets, and is given by the scalar product of the weight vector and the standard deviations of the assets’ returns. The denominator is the portfolio standard deviation. By this definition, the higher the DR, the more the portfolio is diversified. This ratio has a lower bound of one, which will only be achieved in the case of a single-asset portfolio. Portfolio solutions that are characterized by either a highly concentrated allocation or highly correlated asset returns would qualify as being poorly diversified. The authors confirm this statement by considering the following decomposition of the

diversification ratio (proof provided in Appendix A of Choueifaty et al. 2011):

$$DR_{\omega \in \Omega} = \frac{1}{\sqrt{\rho + CR} - \rho CR}, \quad (11.2)$$

where ρ and CR denote the volatility-weighted average correlation and the volatility-weighted concentration ratio, respectively. The former is defined as

$$\rho_{\omega \in \Omega} = \frac{\sum_{i \neq j}^N (\omega_i \sigma_i \omega_j \sigma_j) \rho_{ij}}{\sum_{i \neq j}^N (\omega_i \sigma_i \omega_j \sigma_j)}, \quad (11.3)$$

and the latter as the normalized Herfindahl–Hirschmann index (see Hirschman 1964), which is bounded in the interval $[1/N, 1]$:

$$CR_{\omega \in \Omega} = \frac{\sum_{i=1}^N (\omega_i \sigma_i)^2}{(\sum_{i=1}^N \omega_i \sigma_i)^2}. \quad (11.4)$$

The partial derivatives of (11.2) are

$$\frac{\partial DR}{\partial \rho} = -\frac{1}{2} \frac{\frac{1-CR}{\sqrt{\rho+CR}-\rho CR}}{\rho + CR - \rho CR}, \quad (11.5a)$$

$$\frac{\partial DR}{\partial CR} = -\frac{1}{2} \frac{\frac{1-\rho}{\sqrt{\rho+CR}-\rho CR}}{\rho + CR - \rho CR}, \quad (11.5b)$$

from which the correctness of the relationship stated above can be directly inferred.

Because of the relationship between the number of assets and the CR with respect to the lower bound of the CR, a comparison of different portfolio solutions is only valid if these have been derived from the same set of investment opportunities. Furthermore, the DR depends only on the volatility-weighted average correlations in the case of a naive (i.e., equally weighted) allotment between assets. If one assumes that the assets' expected excess returns are proportional to their respective volatilities, then the diversification ratio is also proportional to the Sharpe ratio of a portfolio. Hence, under this assumption the solutions of a maximum DR portfolio and that of the tangency portfolio will coincide. Of course, excess returns do not enter into (11.1) explicitly, but by cross-comparing the allocation of a portfolio which maximizes the DR with the solution of the tangency portfolio, one can infer for which assets a higher/lower return expectation has implicitly been factored in.

Choueifaty et al. (2011) then state the **condition for a most diversified portfolio (MDP)**:

$$P_{MDP} = \arg \max_{\omega \in \Omega} DR. \quad (11.6)$$

By introducing a set of synthetic assets that share the same volatility, the diversification ratio is maximized by minimizing $\omega' C \omega$, where C denotes the correlation matrix of the initial assets' returns. Hence, the objective function coincides with

Global Minimum Variance (GMV)

that for a GMV portfolio, but instead of using the variance-covariance matrix, the correlation matrix is employed. The final weights are then retrieved by rescaling the intermediate weight vector (based on the correlation matrix) with the standard deviations of the assets' returns. Put differently, the optimal weight vector is determined in two stages. First, an allocation is determined that yields a solution for a least correlated asset mix. This solution is then inversely adjusted by the asset volatilities. This procedure yields a different outcome than the GMV, because in the latter approach the asset volatilities enter directly into the quadratic form of the objective function to be minimized. Hence, the impact of the assets' volatilities is smaller for an MDP than for a GMV portfolio. This can be directly deduced by intuitive reasoning from the decomposition of the DR as in (11.2)–(11.4). The asset volatilities only enter the computation of the CR as a relative share. Finally, the **MDP possesses the following two core properties** (see Choueifaty et al. 2011, pp. 7ff):

1. “Any stock not held by the MDP is more correlated to the MDP than any of the stocks that belong to it. Furthermore, all stocks belonging to the MDP have the same correlation to it.”
2. “The long-only MDP is the long-only portfolio such that the correlation between any other long-only portfolio and itself is greater than or equal to the ratio of their DRs.”

The second core property can be stated equivalently by noting that the more a long-only portfolio is diversified, the higher will be the correlation of the portfolio's returns with those of the MDP solution.

11.3 Risk contribution constrained portfolios

In this section the term “diversification” is applied to the portfolio risk itself. The **portfolio risk can in principle be captured by either a volatility-based measure (i.e., the portfolio's standard deviation) or a downside-based measure (i.e., the conditional value at risk (CVaR) or expected shortfall)**. Hence, two approaches will be presented here. Heuristically these approaches are motivated by the empirical observation that the risk contributions are a good predictor for actual portfolio losses, and hence, by diversifying directly on these contributions, portfolio losses can potentially be limited compared to an allocation that witnesses a high risk concentration on one or a few portfolio constituents (see Qian 2005).

In the first approach, an asset allocation is sought such that the contributions of the portfolio's constituents contribute the same share to the overall portfolio volatility. In this case, the **equal-weight allocation is now applied to the assets' risk contributions, leading to the term “equal-risk contribution” (ERC)**. **Diversification is therefore defined and achieved by a weight vector that is characterized by a least-concentrated portfolio allocation with respect to the risk contributions of its constituents**. It was introduced in the literature by Qian (2005, 2006, 2011) and the properties of this portfolio optimization approach were analyzed in Maillard

et al. (2009, 2010). Zhu et al. (2010) showed how this approach can be adapted when the risk contributions are constrained or budgeted. Spinu (2013) introduced risk-parity portfolio optimizations as a convex program formulation with predefined risk contributions. Hereby, the ERC portfolio is entailed as a special case. Last but not least, the monograph by Roncalli (2013) is an encompassing treatment of risk-parity and risk-budgeting portfolio optimization techniques.

In the second approach, the risk contributions of the portfolio constituents are measured by the portfolio's downside risk, such as the CVaR. The term "diversification" is now related to the notion that the downside risk contributions of the assets contained in a portfolio are either bounded by an upper threshold or evenly spread out between the constituents. This approach was introduced by Boudt et al. (2010, 2011) and Ardia et al. (2011b), based on results in Boudt et al. (2007, 2008) and Peterson and Boudt (2008).

In the following paragraphs, ERC portfolios are more formally presented and then the focus is shifted toward the second kind of portfolio optimization which limits the contributions to a downside risk measure. Recall from Section 4.3 the [definition of the risk contribution of an asset](#), which is restated in more general terms below:

$$C_i M_{\omega \in \Omega} = \omega_i \frac{\partial M_{\omega \in \Omega}}{\partial \omega_i}, \quad (11.7)$$

where $M_{\omega \in \Omega}$ denotes a linear homogeneous risk measure and ω_i is the weight of the i th asset. As such, the risk measure $M_{\omega \in \Omega}$ can be the portfolio's standard deviation, the value at risk, or the expected shortfall. All of these measures have in common the above characteristic and hence, by Euler's homogeneity theorem, the total portfolio risk is equal to the sum of the risk contributions as defined in (11.7). These contributions can also be expressed as percentage figures by dividing the risk contributions by the value of $M_{\omega \in \Omega}$:

$$\%C_i M_{\omega \in \Omega} = \frac{C_i M_{\omega \in \Omega}}{M_{\omega \in \Omega}} \times 100. \quad (11.8)$$

In the empirical application, the marginal contributions will be provided in this notation.

If one inserts the [formula for the portfolio standard deviation](#) $\sigma(\omega) = \sqrt{\omega' \Sigma \omega}$ for $M_{\omega \in \Omega}$, where ω is the $(N \times 1)$ weight vector and Σ denotes the variance-covariance matrix of asset returns with off-diagonal elements σ_{ij} and with σ_i^2 the i th element on its main diagonal (i.e., the variance of the returns for the i th asset), then partial derivatives in the above equation are given by

$$\frac{\partial \sigma(\omega)}{\partial \omega_i} = \frac{\omega_i \sigma_i^2 + \sum_{j \neq i}^N \omega_j \sigma_{ij}}{\sigma(\omega)}. \quad (11.9)$$

These $i = 1, \dots, N$ partial derivatives are proportional to the i th row of $(\Sigma \omega)$; and hence the problem for an ERC portfolio with a long-only and a budget constraint can be

stated as

$$\begin{aligned} P_{\text{ERC}} : \quad & \omega_i(\Sigma\omega)_i = \omega_j(\Sigma\omega)_j \forall i, j, \\ & 0 \leq \omega_i \leq 1 \text{ for } i = 1, \dots, N, \\ & \omega' \mathbf{i} = 1, \end{aligned} \quad (11.10)$$

where \mathbf{i} is an $(N \times 1)$ vector of 1s. A solution to this problem can be found numerically by minimizing the standard deviation of the risk contributions. The optimal ERC solution is valid if the value of the objective function is equal to zero, which is only the case when all risk contributions are equal. A closed-form solution can only be derived under the assumption that all asset pairs share the same correlation coefficient. Under this assumption, optimal weights are determined by the ratio of the inverse volatility of the i th asset and the average of the inverse asset volatilities. Assets with a more volatile return stream are penalized by a lower weight in an ERC portfolio (see Maillard et al. 2010). It was further shown by these authors that with respect to the portfolio's standard deviation, the ERC solution takes an intermediate position between the solution of a GMV and an equal-weighted portfolio. They also showed that under the assumption of constant correlations, and if the Sharpe ratios of the assets are all identical, then the ERC solution coincides with that of the tangency portfolio.

As mentioned in Section 11.1, Spinu (2013) cast the problem of risk-parity optimizations as a convex program. The objective is then given as

$$F(x) = \frac{1}{2} \omega' \Sigma \omega - \sum_{i=1}^N b_i \log(\omega_i), \quad (11.11)$$

where b_i is the marginal risk contribution of the i th asset. If $b_i = 1/N$ for $i = 1, \dots, N$ in (11.11) then an ERC portfolio optimization results. Thus, non-negativity constraints for the weights must be included in the convex program definition.

If one inserts the portfolio's CVaR instead of its standard deviation as a measure of risk in (11.7), one obtains for the marginal CVaR contribution of the i th asset to the portfolio's CVaR,

$$C_i \text{CVaR}_{\omega \in \Omega, \alpha} = \omega_i \frac{\partial \text{CVaR}_{\omega \in \Omega, \alpha}}{\partial \omega_i}. \quad (11.12)$$

Obviously, by employing the CVaR or any other quantile-based risk measure, all portfolio optimizations will be dependent on the prespecified nuisance parameter α , which is the confidence level pertinent to the downside risk. The derivation of the partial derivatives in (11.12) depends on the distribution model assumed, but can be provided fairly easily in the case of elliptical distributions. As such the CVaR either based on the normal assumption or modified by the Cornish–Fisher extension is assumed in practice (see Section 4.3). If one assumes that the returns follow a multivariate normal distribution, then the CVaR of a portfolio allocation is given by

$$\text{CVaR}_{\omega \in \Omega, \alpha} = -\omega' \mu + \sqrt{\omega' \Sigma \omega} \frac{\phi(z_\alpha)}{\alpha}, \quad (11.13)$$

where μ denotes the $(N \times 1)$ vector of expected returns, ϕ the standard normal density function, and z_α the α -quantile of the standard normal distribution. The marginal contribution of the i th asset is then given by

$$C_i \text{CVaR}_{\omega \in \Omega, \alpha} = \omega_i \left[\mu_i + \frac{(\Sigma \omega)_i}{\sqrt{\omega' \Sigma \omega}} \frac{\phi(z_\alpha)}{\alpha} \right]. \quad (11.14)$$

It was shown in Scaillet (2002) that the CVaR contributions are equal to the loss contributions for portfolio losses that exceed the portfolio's VaR at the confidence level α . Incidentally, if one further assumes that the assets' returns behave like a random walk and hence the best forecast for the expected returns is $\mu = \mathbf{0}$, then the portfolio allocation obtained by equating all marginal CVaR contributions will be identical to an ERC allocation, regardless of the specified confidence level. This is because the first summand drops out of the objective function and $\phi(z_\alpha)/\alpha$ is a constant which does not affect the portfolio composition. However, (11.14) could be fruitfully employed in portfolio optimizations when either placing an upper bound on the marginal CVaR contributions or demanding a well-diversified allocation with respect to the constituents' downside risk contributions. The latter objective can be achieved by minimizing the maximum marginal CVaR contribution:

$$C_{\omega \in \Omega, \alpha} = \max_i C_i \text{CVaR}_{\omega \in \Omega, \alpha}, \quad (11.15)$$

as proposed in Boudt et al. (2011). The authors termed this approach “minimum CVaR concentration” (MCC), as it turns out that by using this objective in practice one ordinarily obtains a more balanced solution—not equal—with respect to the marginal contributions, and the CVaR level itself is reasonably low, compared to a solution where the risk contributions are budgeted (BCC). Instead of minimizing the maximum marginal contribution to CVaR, one could directly utilize a measure of divergence instead.

11.4 Optimal tail-dependent portfolios

The concept presented in this section can be viewed as a synthesis of the approaches presented in the previous two sections. In an MDP portfolio an asset allocation is determined which yields the greatest diversification, by definition. However, this approach is based on the symmetric correlations between the assets. As already pointed out in Chapter 9, the Pearson correlation coefficient only measures the dependence between two random variables correctly if these are jointly normally (elliptically) distributed. Furthermore, covariance/correlation takes deviations below and above the mean into account, but a risk-averse (long-only) investor seeks positive returns (the upside or right tail of the distribution) and only considers negative returns (the downside) as risk. This point was made by Markowitz (1952) when he noted the appropriateness of the lower partial second moment as a dispersion measure. In that sense, the lower tail dependence coefficient measures the strength of the relationship between asset returns when both are extremely negative at the same time, and is

therefore a downside risk measure, as is VaR or CVaR. Dependent on the underlying copula assumption (e.g., the EVT copula and/or the empirical copula), the value of the TDC also depends on the count of ranked observations used in its calculation, which is akin to the confidence level of VaR/CVaR. For Archimedean copulae the TDC only depends on the copula parameter. In this section the **TDC for a distributional copula, such as the Student's t copula, is discarded because it is a dependence measure for the lower and the upper tail and, hence, does not fit well in the task of risk management** (see Section 9.3). In the following, a brief account of the non-parametric estimation of the TDC is given, followed by a description of how optimal portfolio solutions can be derived from it. A general account and synopsis of tail dependence is provided, for instance, in Coles et al. (1999) and Heffernan (2000).

A synopsis of the non-parametric TDC estimators is provided in Dobrić and Schmid (2005), Frahm et al. (2005), and Schmidt and Stadtmüller (2006). Let (X, Y) denote the percentage losses of two investments. **The lower tail dependence coefficient, λ_L , between these two random variables is invariant with respect to strictly increasing transformations applied to (X, Y) and does not depend on the marginal distributions of the assets' returns. It is solely a function of the assumed copula.** The joint distribution function of X and Y is given by

$$F_{X,Y}(x, y) = \mathbb{P}(X \leq x, Y \leq y) \text{ for } (x, y) \in \mathbb{R}^2. \quad (11.16)$$

This bivariate distribution can be stated equivalently in terms of the copula, \mathcal{C} , as

$$F_{X,Y}(x, y) = \mathcal{C}(F_X(x), F_Y(y)), \quad (11.17)$$

where $F_X(x)$ and $F_Y(y)$ are the marginal distributions of X and Y , respectively. The copula is the joint distribution of the marginal distribution functions: $\mathcal{C} = \mathbb{P}(U \leq u, V \leq v)$, with $U = F_X(x)$ and $V = F_Y(y)$, and therefore maps from $[0, 1]^2$ into $[0, 1]$. If the limit exists, then **the lower tail dependence coefficient is defined as**

$$\lambda_L = \lim_{u \rightarrow 0} \frac{\mathcal{C}(u, u)}{u}. \quad (11.18)$$

This limit **can be interpreted as a conditional probability**, and as such the lower tail dependence coefficient is bounded in the interval $[0, 1]$. The bounds are realized for an independence copula ($\lambda_L = 0$) and a co-monotonic copula ($\lambda_L = 1$), respectively.

The non-parametric estimators for λ_L are derived from the empirical copula. For a given sample of N observation pairs $(X_1, Y_1), \dots, (X_N, Y_N)$ with corresponding order statistics $X_{(1)} \leq X_{(2)} \leq \dots \leq X_{(N)}$ and $Y_{(1)} \leq Y_{(2)} \leq \dots \leq Y_{(N)}$, the **empirical copula is defined as**

$$\mathcal{C}_N \left(\frac{i}{N}, \frac{j}{N} \right) = \frac{1}{N} \sum_{l=1}^N I(X_l \leq X_{(i)} \wedge Y_l \leq Y_{(j)}) \quad (11.19)$$

with $i, j = 1, \dots, N$; I is the indicator function, which takes a value of 1 if the condition stated in the parentheses is true. By definition, \mathcal{C}_N takes a value of zero for $i = j = 0$.

In the literature cited above, three consistent and asymptotically unbiased estimators for λ_L are provided, which all depend on a threshold parameter k (i.e.,

the number of order statistics considered). The first estimator, $\lambda_L^{(1)}(N, k)$, is an approximation of the derivative of λ_L with respect to u by the slope of a secant in the neighborhood of it (note that u is written as k/N):

$$\lambda_L^{(1)}(N, k) = \left[\frac{k}{N} \right]^{-1} \cdot \mathcal{C}_N \left(\frac{k}{N}, \frac{k}{N} \right). \quad (11.20)$$

The second estimator is based on the slope coefficient of a simple affine linear regression between the values of the copula as regressand and the tail probabilities i/N , $i = 1, \dots, k$, as the regressor. It is defined as

$$\lambda_L^{(2)}(N, k) = \left[\sum_{i=1}^k \left(\frac{i}{N} \right)^2 \right]^{-1} \cdot \sum_{i=1}^k \left[\frac{i}{N} \cdot \mathcal{C}_N \left(\frac{i}{N}, \frac{i}{N} \right) \right]. \quad (11.21)$$

A third means of estimating λ_L non-parametrically is derived from a mixture of the co-monotonic and independence copulae. Here the lower tail dependence coefficient is the weighting parameter between these two copulae. The estimator is then defined as

$$\lambda_L^{(3)}(N, k) = \frac{\sum_{i=1}^k \left(\mathcal{C}_N \left(\frac{i}{N}, \frac{i}{N} \right) - \left(\frac{i}{N} \right)^2 \right) \left(\left(\frac{i}{N} \right) - \left(\frac{i}{N} \right)^2 \right)}{\sum_{i=1}^k \left(\frac{i}{N} - \left(\frac{i}{N} \right)^2 \right)^2}. \quad (11.22)$$

Of crucial importance for estimating the lower tail dependence coefficient is an appropriate selection of k . This parameter is akin to the threshold value for the peaks-over-threshold method in the field of EVT and hence the trade-off between bias and variance is applicable here, too. Choosing k too small will result in an imprecise estimate, and too high in a biased estimate. The conditions stated above of consistency and unbiasedness are met if $k \sim \sqrt{N}$, as shown in Dobrić and Schmid (2005).

The information on the size of lower tail dependence between assets can be utilized in several ways. For instance, a straightforward application would be to replace Pearson's correlation matrix—employed in determining the allocation for a most diversified portfolio—with the lower tail dependence coefficients, whereby the main diagonal elements are set equal to one. After rescaling the weight vectors by the assets' volatilities, one would obtain a “most diversified/minimum tail-dependent” portfolio. Alternatively, if one ranked the TDCs between the constituent assets of a benchmark and the benchmark itself, one could employ this information to select the financial instruments that are the least dependent with extreme losses in the benchmark, but independent with respect to the upside. It should be clear that this asset selection will yield a different outcome than following a low- β strategy. Even though the present account has focused on the lower tail dependence, the approach sketched above for selecting constituent assets could also be based on the difference between the upper and lower tail dependence. These two TDCs could be retrieved from the Gumbel and Clayton copula, respectively. This procedure is akin to selecting assets

Value Theory

according to their risk–reward ratio. However, it should be stressed that using tail dependence as a criterion in portfolio optimization should always be accompanied by an additional measure of risk for the assets. A low value of tail dependence is non-informative with respect to the asset risk in terms of either its volatility or its downside risk.

11.5 Synopsis of R packages

11.5.1 The package cccp

The package **cccp** has been briefly introduced in Section 10.4.7. With respect to risk-parity portfolio optimizations it contains the function `rp()`. This function is an **implementation of the convex program formulation** as proposed by Spinu (2013)—see (11.11) for the objective. This R function interfaces with the C++ routine `rpp()` which is a member of the package’s **Rcpp** module. The gradient and Hessian of the objective are given as $\nabla F(\omega) = \Sigma\omega - \mathbf{b}\omega^{-1}$ and $\nabla^2 F(\omega) = \Sigma + \text{diag}(\mathbf{b}\omega^{-2})$, respectively, which are implemented as separate C++ routines. The C++ function `rpp()` returns an object of C++ class **CPS** for which an R-side reference class **Rcpp_CPS** is defined. The function `rp()` has four arguments: `x0` for the feasible starting values, `P` for the dispersion matrix, `mrc` for the marginal risk contributions, and `optctrl` for providing parameters pertinent to the interior-point solver. The latter argument requires an object returned by `ctrl()`, that is, an object of reference class **Rcpp_CTRL**. Methods for showing the solution of the risk-parity portfolio optimization (`show()`); for displaying the state, status, and count of iterations (`getstate()`, `getstatus()`, and `getniter()`, respectively); and retrieving the primal and dual variables (`getx()`, `gets()`, `getz()`, and `gety()`) are made available.

11.5.2 The packages **DEoptim**, **DEoptimR**, and **RcppDE**

It should be evident from the portfolio optimizations presented above, in particular the equal-risk contributions to the ES of an asset allocation, that the task of finding a best global solution can become quite complicated from a numerical point of view. In light of this, and also due to the dependence of the package **PortfolioAnalytics** (see Section 11.5.4) on **DEoptim**, a separate presentation of R packages that implement differential evolution is in order. Of course, the solvers implemented are not solely dedicated to finding a portfolio allocation which gives equal marginal contributions to the ES for a given confidence level, but can also be used in other optimizations.

The differential evolution (DE) algorithm was introduced by Storn and Ulrich (1997), and its application is elucidated in Price et al. (2006). It is classified as a genetic algorithm and is hence a derivative-free global optimizer. In this respect, **DE** is an evolutionary algorithm where an initial population of candidate solutions is subjected to alteration and selection operations to yield a global minimum of the objective function. Because the DE algorithm is derivative-free, the only requirements are that the objective function and the parameters are real-valued. **Neither differentiability nor**

continuity of the objective function is necessary. However, because the initial population of candidate solutions is random, one should set a seed for the random number generator in order to replicate results. A thorough account of genetic algorithms can be found in Mitchell (1998).

Historically, the first package on CRAN to implement the DE algorithm was **DEoptim** by Ardia et al. (2011a,b, 2015) and Mullen et al. (2011). The package is contained in the “Optimization” Task View. In earlier releases, the DE algorithm was implemented solely in R (written by David Ardia), but in order to improve the computation speed the DE routines were ported to the C language, similar to the MS Visual C++ implementation in Price et al. (2006). The package utilizes S3 classes and methods and is shipped with two vignettes. Incidentally, in one of the vignettes Ardia et al. (2011b) give an example showing a large-scale portfolio optimization using the function `DEoptim()` which fails if gradient-based optimizations are used.

The function `DEoptim()` is specified with four arguments in its closure and the ellipsis argument. The objective function to be minimized is provided as argument `fn`. The function should return a real scalar value. Lower and upper bounds for the parameters can be set by means of the arguments `lower` and `upper`, respectively. Incidentally, constraints other than box constraints must be specified as penalty terms in the objective function. Constraints can be directly included in the objective function as additive terms. Each constraint is multiplied by a positive constant factor. A violation of a constraint would, however, increase the value of the objective function to be minimized. The algorithm can be tailored by the `control` argument, which expects a `list` object returned by the function `DEoptim.control()`.

The function `DEoptim()` returns a `list` object of S3 class `DEoptim` for which a `summary()` and a `plot()` method are available. The `list` object consists of two named elements, `optim` and `member`, which are themselves `list` objects. The former contains the optimal solution (`bestmem`), the value of the objective function evaluated at the optimum (`bestval`), the count of function evaluations (`nfeval`), and the number of procedure iterations (`iter`). The latter `list` element contains the specified lower and upper bounds (`lower` and `upper`), the best value of the objective function at each iteration (`bestvalit`), and its associated parameter values (`bestmemit`), as well as the population generated at the last iteration (`pop`) and a `list` of the intermediate populations (`storepop`). As indicated above, the user can fine-tune the algorithm by means of the function `DEoptim.control()`, which returns a `list` with class attribute of the same name. Here, the optimization process can be terminated by specifying a value of the objective function (argument `VTR`). Its default value is set to `-Inf`; if changed to a higher value, the process is terminated when either the maximum number of iterations is reached (which can be changed by the argument `itermax` with a default value of 200) or the real value of the objective function is less than `VTR`. Further, the user can control the termination of the optimization process by specifying a value for the relative convergence tolerance (argument `reltol`). Depending on the platform on which R is run, this amounts roughly to `1-e8`. Next, the user can choose between six strategies for propagating the populations from one iteration to the next by means of the argument `strategy`. These strategies are detailed in the package manual and vignette and are based on

the specification as in Price et al. (2006). The number of population members is determined from the argument `NP`. If its default value of `NA` remains unchanged, then 10 sets of parameter vectors are created randomly. On the other hand, the user can provide a matrix object for the argument `initialpop` in which the initial population of parameter vectors is specified. The row dimension of this matrix refers to the parameters of the objective function and the column dimension to the population.

The parameters' progress through the iterations can be controlled by the arguments `storepopfrom` and `storepopfreq`. The former sets the generation from which the following intermediate population will be stored with a default setting such that no intermediate population is stored. The latter argument determines the frequency with which the populations are stored. Its default value is 1, whereby every intermediate population is stored. The step size from one iteration to the next is determined by the value assigned to `F`. The remaining arguments in the function's closure are pertinent to the kind of parameter heritage and are partly dependent on the strategy chosen. The reader is referred to the package manual for more details on these controls. Finally, the progress of the optimization can be monitored by means of the argument `trace`. If set to `TRUE` (the default) then the intermediate results of every iteration are printed, and if specified as an integer value `i` then the outcome of only every `i`th iteration is output.

The package **DEoptimR** was added in 2014 to CRAN (see Conceicao and Mächler 2015) and is contained in the Task View “Optimization.” In contrast to **DEoptim**, the package is purely written in R and does contain only one function, namely `JDEoptim()`. As its name implies, within this function the jDE algorithm proposed by Brest et al. (2006) has been implemented. As stated in the previous paragraph, convergence of DE algorithms do depend on the (randomly) chosen seed values and the mutation strategy. A feature of the jDE algorithm is the implementation of a self-adaptive scheme to perform the setting of control parameters and thereby lessens the burden in cases when the default values for the control parameters do not result in convergence. A second feature of the `JDEoptim()` function is the fairly easy provision of nonlinear equality/inequality constraints. The function's argument `constr` expects a function with first argument for the parameters and must return a vector, whereby each element is the outcome of the constraint evaluated at the current parameter values. The constraints must be supplied as $h_i(x) = 0$ with $i = 1, \dots, K$ for the equality constraints and $g_j(x) \leq 0$ with $j = 1, \dots, L$ for the inequality constraints. Therefore, the returned vector has $K + L$ elements. If the argument `constr` is left `NULL` (the default), then a bounded (box-constrained) optimization is carried out, whereby the allowable lower and upper bounds for the parameters are set by the arguments `lower` and `upper`, respectively.

The package **RcppDE** is closely related to the package **DEoptim** (see Eddelbüttel 2016). It is also contained in the CRAN “Optimization” Task View. The major difference between the two packages is the interface to the DE algorithm. As the package's name implies, the routines are interfaced from code written in C++. As shown in the package's vignette by cross-comparison of this implementation with the C-based one in **DEoptim**, the former results in faster execution of various optimization tasks. In contrast to the implementation in the package **DEoptim**, the function `DEoptim()`

now has an additional argument `env` to define the environment in which the objective function is evaluated. If not specified, then the function assigned to `fn` will be evaluated in a new environment. The methods defined for objects with class attribute `DEoptim` are also available, as well as control of the algorithm's behavior using `DEoptim.control()`.

11.5.3 The package FRAPO

This subsection presents the functions and methods of the package `FRAPO` that relate to the topics introduced in this chapter. The functions `dr()`, `cr()`, and `rhow()` can be used to measure the degree of diversification for a given weight vector and variance-covariance matrix. These functions return the diversification ratio, concentration ratio, and volatility-weighted average correlation as introduced in Section 11.2. All three functions have the same closure, consisting of arguments for the weight vector (`weight`) and the variance-covariance matrix (`Sigma`). The solution of the most diversified portfolio can be computed with the function `PMD()`. The array of returns is provided as argument `Returns`, and the logical argument `percentage` determines whether the weights are returned as percentages (the default) or decimal values. The ellipsis argument is passed down to the call of `cov()`, allowing the user to control how the variance-covariance matrix is computed. The resulting object is then converted to a correlation matrix by means of the function `cov2cor()`. The function `PMD()` returns an object of formal S4 class `PortSol` for which `show()`, `Weights()`, `Solution()`, and `update()` methods are available.

The marginal contributions to risk for a given asset allocation and dispersion matrix are returned by the function `mrc()`. In addition to the arguments `weights` and `Sigma` for the weight vector and the variance-covariance matrix, the user can specify whether the contributions are returned as percentages that sum to 100% (the default) or as decimal numbers. The solution of an equal risk contribution portfolio can be determined with the function `PERC()`. The function's closure takes four arguments: `Sigma` for the variance-covariance matrix, `par` for the initial weight vector to be used, `percentage` as a logical switch governing whether the weights should be returned as percentages (the default) or decimal figures, and the argument `optctrl` which is passed down to the `rp()` function of the package `cccp` (see Pfaff 2015) in order to carry out the optimization. By default, `par = NULL` and an equal weights vector is used in the call to `rp()`.

Finally, for optimal tail-dependent portfolio construction the functions `tdc()` and `PMTD()` are available. The former function returns the bivariate TDCs; the user can choose between a non-parametric estimation according to the empirical tail copula (the default) or the stable tail function by means of the `method` argument. The array of returns has to be provided for the argument `x` with the only requirement that it can be coerced to a `matrix` object. Whether lower or upper tail dependence coefficients are returned is determined by the logical argument `lower`, with default value `TRUE`. The threshold value `k` is specified as `NULL`, and if not otherwise stated the square root of the count of observations is used. The ellipsis argument is passed

down to the `rank()` function, hence the user can determine how the order statistics are computed with respect to ties in the data. The function returns a `matrix` object with the bivariate TDCs and 1s on the diagonal, as by definition a series has a dependence of unity with itself. For a long-only investor, the minimum tail-dependent portfolio solution is returned by calling `PMTD()`. The function body is akin to that of the function `PGMV()` by which the solution of a global minimum-variance portfolio is returned. But instead of employing the variance-covariance matrix as a measure of dispersion, the TDC matrix returned from the call to `tdc()` is utilized. In addition to the arguments pertinent to this function, the array of returns has to be assigned to the argument `Returns`. The function returns an object of formal S4 class `PortSol` for which the above-mentioned methods are available.

11.5.4 The package `PortfolioAnalytics`

The package **PortfolioAnalytics** (see Peterson and Carl 2015) takes a conceptually different approach to portfolio optimization compared to the packages discussed above. It allows the user to obtain a numerical portfolio solution for complex constraints and/or objective functions. With regard to the risk-measure-related approaches to portfolio optimization presented in this chapter, one could directly use a function that returns the risk measure in question as the objective function, for instance. The numerical optimization is then carried out either with the differential evolution optimizer contained in the package **DEoptim** (see Ardia et al. 2015) or by means of randomly generated portfolios obeying the specified constraints. The former algorithm was introduced by Storn and Ulrich (1997). The general approach is that the objective function is amended by the constraint functions, which are multiplied by penalty factors. The package is hosted on CRAN and is shipped with five vignettes. One of these is solely dedicated to showing how the included functions in the package can be used to optimize a portfolio with CVaR budgets.

The four cornerstone functions are `portfolio.spec()` for specifying a portfolio structure, `add.objective()` for adding/altering the objective of the returned informal S3 class object returned by `portfolio.spec()`, `add.constraint()` for adding constraints to the `portfolio/portfolio.spec` class object, and `optimize.portfolio()` for carrying out the numerical optimization of the portfolio problem thus stated. The function `optimize.portfolio()` returns an object of informal S3 class `optimize.portfolio.DEoptim`, `optimize.portfolio.random`, or `optimize.portfolio`. For such objects `plot()` and `extractStats()` methods are provided. The latter method returns statistics for the optimal portfolio solution. The values of the objective function for a given set of weights can be queried by the function `constrained_objective()`. Two utility functions are worth highlighting: `optimize.portfolio.rebalancing()` for carrying out portfolio rebalancing at a specified frequency and `optimize.portfolio.parallel()` for carrying out multiple calls to `optimize.portfolio()` on a multiprocessor computer. The default value is set to use four nodes. The remaining functions of

the package are primarily intended for plotting and querying/extracting information from optimal portfolio solutions.

11.6 Empirical applications

11.6.1 Comparison of approaches

The first R code example, Listing 11.1, shows how the portfolio solutions for the sector indexes of the Swiss Performance Index (SPI) according to a global minimum variance, equal-risk contribution, most-diversified, and a minimum-tail-dependence allocation can be determined. All optimizations have in common that no return forecasts are required, but that the shares to be invested are solely dependent on different risk measures. The GMV allocation will serve as the benchmark in the portfolio comparisons. Aside from the allocations themselves, the marginal risk contributions and the overall portfolio characteristics will be evaluated. The covered sectors are: Basic Materials (BASI), Industrials (INDU), Consumer Goods (CONG), Healthcare (HLTH), Consumer Services (CONS), Telecommunications (TELE), Utilities (UTIL), Financial (FINA), and Technology (TECH).

In the first three code lines of Listing 11.1 the necessary packages are loaded. All optimizations will be carried out with the functions provided in **FRAPO**. The utilized data set is contained in the package **fPortfolio** and the base package **lattice** will be employed in the graphical analysis of the results. In the next lines, the level data of the sector indexes are extracted from **SPISECTOR**, NA values are interpolated, and discrete daily returns are computed as well as the variance-covariance matrix thereof. The sample starts on 30 December 1999 and ends on 17 October 2008; a total of 2216 observations per sector. The allocations according to the four portfolio optimizations are then extracted by means of the **Weights()** method and assigned to the objects **GMVw**, **MDPw**, **MTDw**, and **ERCw** for the global-minimum variance, most-diversified, minimum tail-dependent, and equal-risk contributed solutions, respectively. The weights are then collected in the matrix object **W**. This object is then used in the call to the **apply()** function to determine the marginal risk contributions for each of the four optimal weight vectors. The results are then shown graphically from line 20 onwards.

First, the weights are displayed in the form of dot charts. Usually, asset allocations are portrayed as pie charts. However, this kind of visualization is problematic, because the relation of surfaces and their sizes cannot be well digested by visual inspection. In the north-west panel of Figure 11.1 the sector allocations according to the GMV portfolio are shown and relative to this set of weights the solutions of the MDP, ERC, and MTD portfolios are depicted in the remaining three panels.

In Figures 11.2 and 11.3, the marginal risk contributions (MRCs) for each of the portfolios are shown as dot plots. Here, the MRCs are ordered by sector and per portfolio optimization and by portfolio and per sector. In order to do so, the sectors and the kind of portfolio have to be coerced to factors and then this kind of lattice plot

R code 11.1 Comparison of portfolio solution for Swiss equity sectors.

```

library(FRAPO)                                     1
library(fPortfolio)                                2
library(lattice)                                   3
## Loading data and calculating returns          4
data(SPISECTOR)                                   5
Idx <- interpNA(SPISECTOR[, -1], method = "before") 6
R <- returnseries(Idx, method = "discrete", trim = TRUE) 7
V <- cov(R)                                       8
## Portfolio Optimizations                      9
GMVw <- Weights(PGMV(R))                         10
MDPw <- Weights(PMD(R))                          11
MTDw <- Weights(PMTD(R))                         12
ERCw <- Weights(PERC(V))                          13
## Combining solutions                         14
W <- cbind(GMVw, MDPw, MTDw, ERCw)               15
MRC <- apply(W, 2, mrc, Sigma = V)                 16
rownames(MRC) <- colnames(Idx)                     17
colnames(MRC) <- c("GMV", "MDP", "MTD", "ERC") 18
## Plot of allocations                         19
oldpar <- par(no.readonly = TRUE)                  20
par(mfrow = c(2, 2))                             21
dotchart(GMVw, xlim = c(0, 40), main = "GMV Allocation", 22
         pch = 19)                                23
dotchart(MDPw - GMVw, xlim = c(-20, 20), main = "MDP vs. GMV", 24
         pch = 19)                                25
abline(v = 0, col = "grey")                         26
dotchart(MTDw - GMVw, xlim = c(-20, 20), main = "MTD vs. GMV", 27
         pch = 19)                                28
abline(v = 0, col = "grey")                         29
dotchart(ERCw - GMVw, xlim = c(-20, 20), main = "ERC vs. GMV", 30
         pch = 19)                                31
abline(v = 0, col = "grey")                         32
par(oldpar)                                       33
## lattice plots of MRC                      34
Sector <- factor(rep(rownames(MRC), 4),           35
                  levels = sort(rownames(MRC))) 36
Port <- factor(rep(colnames(MRC), each = 9),        37
               levels = colnames(MRC))           38
MRCdf <- data.frame(MRC = c(MRC), Port, Sector) 39
dotplot(Sector ~ MRC | Port, groups = Port, data = MRCdf, 40
        xlab = "Percentages",                      41
        main = "MR Contributions by Sector per Portfolio", 42
        col = "black", pch = 19)                    43
dotplot(Port ~ MRC | Sector, groups = Sector, data = MRCdf, 44
        xlab = "Percentages",                      45
        main = "MR Contributions by Portfolio per Sector", 46
        col = "black", pch = 19)                    47

```

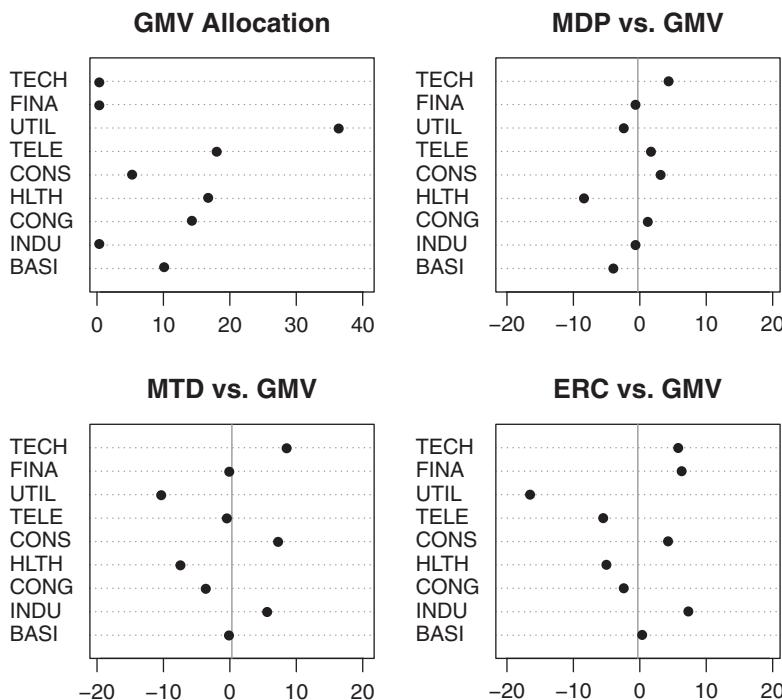


Figure 11.1 Comparison of sector allocations.

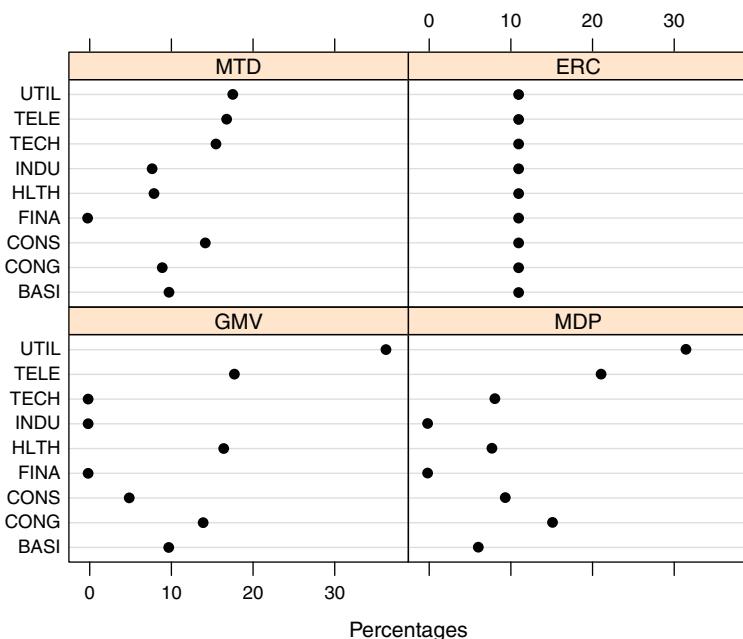


Figure 11.2 Marginal risk contributions by sector per portfolio.

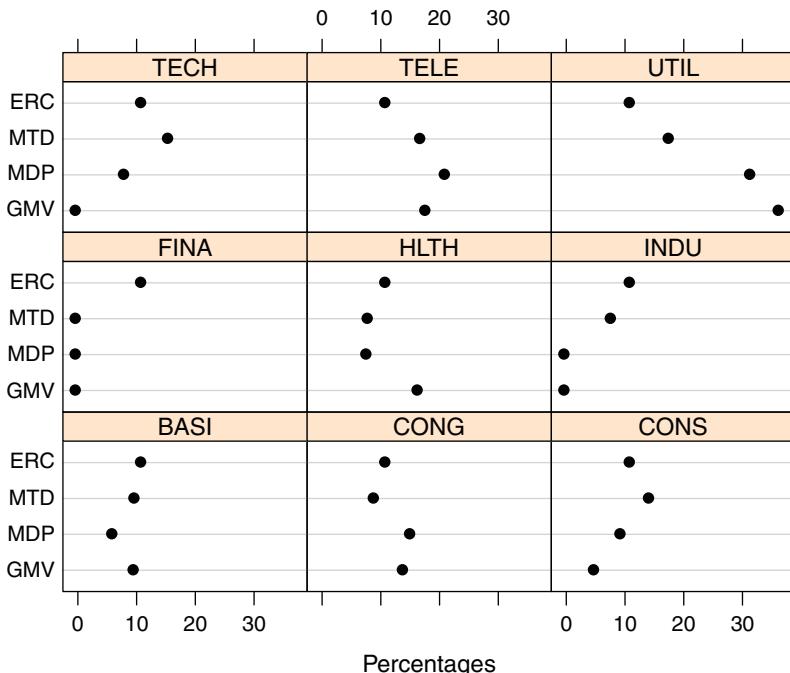


Figure 11.3 Marginal risk contributions by portfolio per sector.

can be produced by the formula arguments in the calls to the `dotplot()` function which is part of the package **lattice** (see Sarkar 2008).

As is evident from Figure 11.1, the GMV solution is characterized by a dominant allocation in the Utility sector (roughly 40%) followed by the Telecommunications, Healthcare, and Consumer Goods sectors. The Technology, Financial, and Industrials sectors are basically excluded from the portfolio. More than 80% of the wealth is allocated to only four out of nine sectors. This portfolio concentration is quite typical for GMV approaches and governed by the low volatility of the dominating sectors. The relative allocation of the most-diversified portfolio is depicted in the north-east panel. The biggest difference is a distinct underweight of the Healthcare sector. The remaining sector weights do not differ materially from the GMV solution and hence the highest weight is attributed to the Utility sector, too. Quite a different picture emerges for the MTD and ERC allocations. In both cases, the Utility sector is underweight and the Technological, Consumer Services, and Industrials sectors are overweight. In addition, the Financial sector is also overweight in the ERC portfolio. All in all, the latter two portfolio approaches result in a more balanced wealth allocation. These characteristics are mirrored by the marginal contributions to risk in Figures 11.2 and 11.3. Of course, for the ERC portfolio these are all equal. In absolute terms the greatest share of risk contribution is taken by the Utility sector in the GMV and MDP cases, whereas the risk contributions of the MTP portfolio are more evenly dispersed.

R code 11.2 Key measures of portfolio solutions for Swiss equity sectors.

```

library(PerformanceAnalytics)                                1
## Portfolio Risk Measures and Characteristics          2
Rdec <- R / 100                                         3
Pret <- apply(W, 2, function(x) Rdec %*% x / 100)       4
SD <- apply(Pret, 2, sd) * 100                           5
ES95 <- apply(Pret, 2, function(x)                      6
            abs(ES(R = x, method = "modified") * 100))     7
DR <- apply(W, 2, dr, Sigma = V)                         8
CR <- apply(W, 2, cr, Sigma = V)                         9
## Summarising results                                  10
Res <- rbind(SD, ES95, DR, CR)                         11

```

Summary statistics for the four portfolio solutions are computed within the R code in Listing 11.2. Here, the functions contained the package **PerformanceAnalytics** are used to compute the standard deviations and the expected shortfalls at the 95% level. The diversification and the concentration ratios can be calculated with the functions **dr()** and **cr()** described in Section 11.5.3, respectively. All computations are carried out by utilizing the **apply()** function with respect to the matrix object **W**, which contains the weight vectors in its columns. The results are provided in Table 11.1.

Judged by these selected key measures, the **GMV** solution does deliver an optimal weight vector that yields the lowest portfolio standard deviation, *ex post*, it is more risky in terms of its expected shortfall, and has the highest concentration ratio. On the other extreme, the **ERC** allocation implies the most risky allocation in terms of portfolio standard deviation and its expected shortfall. Furthermore, it embodies the smallest portfolio concentration, but its diversification ratio fairs the worst. Intermediate positions are taken by the **MDP** and **MTD** solutions; the former is more akin to the **GMV** allocation and the latter resembles more the wealth allotment according to the **ERC** approach.

11.6.2 Optimal tail-dependent portfolio against benchmark

This subsection shows how the lower tail dependence coefficient between a broader market aggregate and its constituents can be utilized such that the selected stocks that enter into a long-only portfolio are solely least concordant only to negative market

Table 11.1 Key measures of portfolio solutions for Swiss equity sectors.

Measures	GMV	MDP	MTD	ERC
Standard deviation	0.813	0.841	0.903	0.949
ES (modified, 95%)	2.239	2.189	2.313	2.411
Diversification ratio	1.573	1.593	1.549	1.491
Concentration ratio	0.218	0.194	0.146	0.117

returns. This approach contrasts with a low- β strategy in the sense that now the aim is to shelter investors from concurrently incurring index losses and the stocks held in the portfolio. The intention of a low- β strategy is basically the same, by selecting stocks that co-move less proportionally than the market in absolute terms. However, this kind of selection might deter stocks that are characterized by a high value of the upper tail dependence coefficient with the market and hence such a portfolio allocation would miss on the upside. Incidentally, it should be stressed that neither of the two strategies takes the riskiness of the portfolio members explicitly into account.

In the R code in Listings 11.3 and 11.4, the two strategies are applied to the constituents of the S&P 500 index. In the first two lines the necessary packages **FRAPO** and **copula** for conducting the comparison are loaded into the workspace. In contrast to the application in the previous subsection, the lower tail dependence will be derived from a Clayton copula and this task can be swiftly accomplished with the facilities offered in the latter package, as will be shown in due course. Next, the data set **INDTRACK6** is brought into memory; this is contained in the package **FRAPO**. This object is a data frame with 291 weekly observations of the S&P 500 index and 457 members of it. The sample starts in March 1991 and ends in September 1997. The data set was first used in Canakgoz and Beasley (2008) (see also Beasley 1990). Stocks with missing values during the sample period have been discarded. Originally, the data was downloaded from DATASTREAM and has been made anonymous. The first column refers to the index data itself.

The index and stock returns are computed in lines 6 and 8 of Listing 11.3. Here, only the first 260 data points are used and the remaining ones are saved for a *pseudo ex ante* evaluation of the portfolios' wealth progressions. The β values of the stocks are assigned to the object **Beta** in the next line by using the **apply()** function on the column dimension of **RA** and the estimator for the slope coefficient. The lower tail dependence coefficients derived from the Clayton copula are calculated in lines 11 to 14. These measures are derived from Kendall's τ rank correlations by first returning estimates of the copula parameter θ from which the lower tail dependence coefficients can be deduced. In the next block of R statements, the stocks with $|\beta|$ and tail dependence coefficients below their respective median value are determined and the weights are calculated by an inverse logarithmic scale. By using this scheme, roughly 80% of the stocks are common to both selections. The allocation employed is only as an example; any other kind of weight assignment can be chosen for the two sets of stocks, for example, an allocation in accordance with a global minimum variance approach. The progression of the out-of-sample benchmark equity and for the two strategies are computed in lines 25 to 43. In the R code in Listing 11.4 these progressions are depicted in the form of a time series plot (see Figure 11.4) and the relative-to-benchmark performances as a bar plot (see Figure 11.5).

Clearly, both strategies out-performed the benchmark, albeit the low- β strategy did worse during roughly the first half of the out-of-sample period. The lower-tail-dependence strategy exceeded the benchmark consistently throughout the *ex ante* period except for eight data points and kept a slight edge at the sample end compared to the low- β strategy. The low- β portfolio underperformed its benchmark

R code 11.3 S&P 500: tail-dependence versus low- β portfolio.

```

library(FRAPO)                                     1
library(copula)                                    2
## S&P 500                                         3
data(INDTRACK6)                                    4
## Market and Asset Returns                      5
RM <- returnseries(INDTRACK6[1:260, 1], method = "discrete", 6
                   trim = TRUE)                      7
RA <- returnseries(INDTRACK6[1:260, -1], method = "discrete", 8
                   trim = TRUE)                      9
Beta <- apply(RA, 2, function(x) cov(x, RM) / var(RM)) 10
Tau <- apply(RA, 2, function(x) cor(x, RM, method = "kendall")) 11
## Clayton Copula: Lower Tail Dependence        12
ThetaC <- copClayton@iTau(Tau)                  13
LambdaL <- copClayton@lambdaL(ThetaC)           14
## Selecting Stocks below median; inverse log-weighted and 15
## scaled                                         16
IdxBeta <- Beta < median(Beta)                  17
WBeta <- -1 * log(abs(Beta[IdxBeta]))           18
WBeta <- WBeta / sum(WBeta) * 100               19
## TD                                            20
IdxTD <- LambdaL < median(LambdaL)            21
WID <- -1 * log(LambdaL[IdxTD])                22
WID <- WID / sum(WID) * 100                     23
Intersection <- sum(names(WID) %in% names(WBeta)) / 24
                  length(WBeta) * 100               25
## Out-of-Sample Performance                   26
RMo <- returnseries(INDTRACK6[260:290, 1], method = "discrete", 27
                     percentage = FALSE) + 1        28
RAo <- returnseries(INDTRACK6[260:290, -1], method = "discrete", 29
                     percentage = FALSE) + 1        30
## Benchmark                                     31
RMo[1] <- 100                                    32
RMEquity <- cumprod(RMo)                        33
## Low Beta                                      34
LBEquity <- RAo[, IdxBeta]                      35
LBEquity[1, ] <- WBeta                          36
LBEquity <- rowSums(apply(LBEquity, 2, cumprod)) 37
## TD                                            38
TDEquity <- RAo[, IdxTD]                        39
TDEquity[1, ] <- WID                           40
TDEquity <- rowSums(apply(TDEquity, 2, cumprod)) 41
## Collecting results                         42
y <- cbind(RMEquity, LBEquity, TDEquity)        43

```

R code 11.4 Plotting of wealth progression and relative performance.

```

## Time series plots of equity curves
plot(RM Equity, type = "l", ylim = range(y),
      ylab = "Equity Index",
      xlab = "Out-of-Sample Periods")
lines(LB Equity, lty = 2, col = "blue")
lines(TD Equity, lty = 3, col = "red")
legend("topleft",
      legend = c("S&P 500", "Low Beta", "Lower Tail Dep."),
      lty = 1:3, col = c("black", "blue", "red"))
## Bar plot of relative performance
RelOut <- rbind((LB Equity / RM Equity - 1) * 100,
                  (TD Equity / RM Equity - 1) * 100)
RelOut <- RelOut[, -1]
barplot(RelOut, beside = TRUE, ylim = c(-5, 17),
        names.arg = 1:ncol(RelOut),
        legend.text = c("Low Beta", "Lower Tail Dep."),
        args.legend = list(x = "topleft"))
abline(h = 0)
box()

```

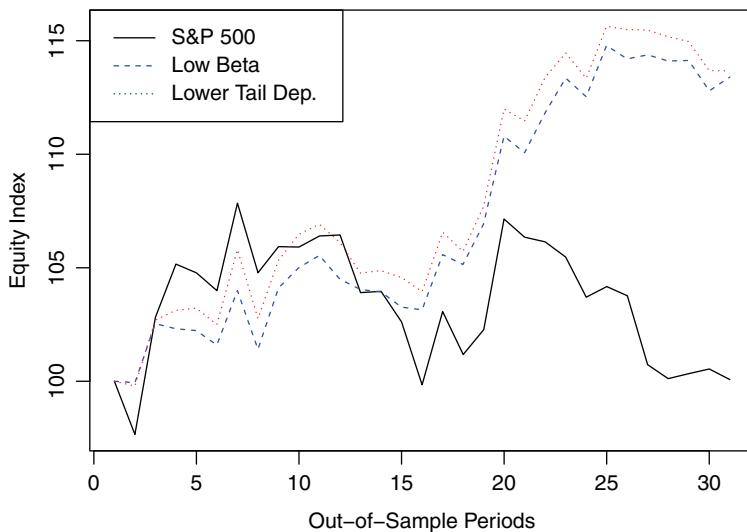


Figure 11.4 Progression of out-of-sample portfolio wealth.

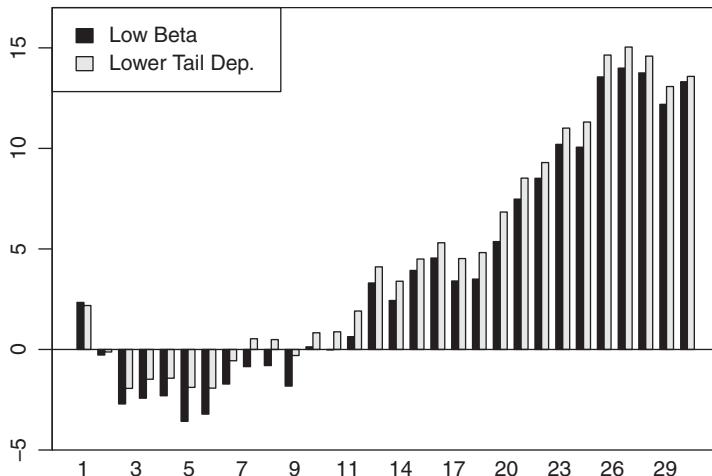


Figure 11.5 Relative out-performance of strategies versus S&P 500.

in 11 out of 30 instances and delivered only for the first out-of-sample observation a portfolio value greater than the tail-dependent strategy, albeit only marginally.

Akin to the key measures in the previous example for quantitatively assessing portfolio allocations, a similar set of figures is computed in the R code in Listing 11.5 for the low- β and minimum-tail-dependence portfolios. All key measures are computed by employing the facilities of the package *PerformanceAnalytics*. Because Listing 11.3 encompasses an out-of-sample part, the quantitative key figures are split into a block of “in-sample” and “out-of-sample” statistics. Within the former category the same set of statistics is computed as in Listing 11.2 and need no further elaboration. The latter are related to the benchmark returns, except for the annualized returns of the two portfolio approaches. The out-of-sample performance is evaluated by means of the information ratio, the upside capture ratio, and downside capture ratio. A high value is desirable for the upside capture ratio and a low value for the downside capture ratio. A reference for these statistics is provided in Bacon (2004, page 47). The computation of these statistics is assembled in the function `km()` and the outcome is assigned to the objects `LbKM` and `TdKM` for the low- β and lower-tail-dependent portfolios, respectively. The results are provided in Table 11.2.

With respect to the in-sample statistics the low- β strategy is slightly less risky than the lower-tail-dependent allocation. However, the latter possesses a greater diversification ratio and is less concentrated. The *ex ante* information ratio is greater for the tail-dependence strategy and also fares better in terms of the upside capture ratio, whereas the low- β strategy has an edge over the former allocation with respect to the downside capture ratio.

R code 11.5 Key measures of portfolio solutions for S&P 500.

```

library(PerformanceAnalytics) 1
## Key measures (ex post) 2
RAdec <- RA / 100 3
RALB <- RAdec[, names(WBeta)] 4
RATD <- RAdec[, names(WID)] 5
LbStd <- StdDev(rowSums(RALB * WBeta / 100)) * 100 6
TdStd <- StdDev(rowSums(RATD * WID / 100)) * 100 7
LbES95 <- abs(ES(R = rowSums(RALB * WBeta / 100), 8
                  method = "gaussian")) * 100 9
TdES95 <- abs(ES(R = rowSums(RATD * WID / 100), 10
                  method = "gaussian")) * 100 11
LbDr <- dr(WBeta, Sigma = cov(RALB)) 12
TdDr <- dr(WID, Sigma = cov(RATD)) 13
LbCr <- cr(WBeta, Sigma = cov(RALB)) 14
TdCr <- cr(WID, Sigma = cov(RATD)) 15
## Key measure (ex ante) 16
LbRetO <- returnseries(LBEquity, method = "discrete", 17
                        percent = FALSE, trim = TRUE) 18
TdRetO <- returnseries(TDEquity, method = "discrete", 19
                        percent = FALSE, trim = TRUE) 20
BmRetO <- RMo[-1] - 1 21
timeArtificial <- timeSequence(from = "1997-03-01", by = "7d", 22
                                  length.out = length(LbRetO)) 23
LbRetO <- timeSeries(LbRetO, as.Date(timeArtificial)) 24
TdRetO <- timeSeries(TdRetO, as.Date(timeArtificial)) 25
BmRetO <- timeSeries(BmRetO, as.Date(timeArtificial)) 26
km <- function(pf, bm, scale = 52){ 27
  ra <- Return.annualized(pf, scale = scale) * 100 28
  ap <- ActivePremium(pf, bm, scale = scale) 29
  te <- sd(pf - bm) * sqrt(scale) 30
  ir <- ap / te 31
  upr <- UpDownRatios(pf, bm, method = "Capture", side = "Up") 32
  dnr <- UpDownRatios(pf, bm, method = "Capture", side = "Down") 33
  res <- c(ra, ir, upr, dnr) 34
  names(res) <- c("Return", "IR", "UpRatio", "DownRatio") 35
  return(res) 36
}
LbKM <- km(LbRetO, BmRetO) 37
TdKM <- km(TdRetO, BmRetO) 38

```

11.6.3 Limiting contributions to expected shortfall

The last example in this chapter shows how a portfolio allocation can be determined with respect to limits on the downside risk contributions. These limitations can be defined in two ways: first, as a budget constraint such that the ES contribution is below

Table 11.2 Key measures of portfolio solutions for S&P 500.

Measures	Low- β	Lower tail dependence
In-sample		
Standard deviation	1.948	2.050
ES (95%)	3.813	3.954
Diversification ratio	2.350	2.576
Concentration ratio	0.008	0.007
Out-of-sample		
Return (annualized)	24.357	24.870
Information ratio	2.364	2.847
Upside capture ratio	0.671	0.784
Downside capture ratio	0.111	0.214

an upper bound, and second, such that the maximum contribution to the portfolio's downside risk of an asset is minimized. These two approaches are contrasted with the solutions of global minimum-variance and equal-risk contributed allocations. The latter two portfolio optimizations are directed to the portfolio's volatility, whereas the former two explicitly relate to the portfolio's downside risk. In all cases, it is assumed that only long positions in an asset are allowed and that the wealth is fully invested in the risky assets.

The R code by which this comparison is carried out is shown in Listing 11.6. The required packages **FRAPO** and **PortfolioAnalytics** are brought into memory first. Next, the data set `MultiAsset` is loaded. This object is a data frame with 85 month-end observations of stock and bond indexes and gold (represented by ETF SPDR Gold). In total, ten asset classes represented by their indexes are included. The sample starts at 30 November 2004 and ends on 30 November 2011. The data set has been obtained from Yahoo Finance and the unadjusted closing prices have been retrieved. If a month-end value was not reported, the value of the previous day has been used. More information about the data set can be found in its manual page. In the subsequent two lines, the discrete decimal returns for the assets are computed (R) and the count of assets is assigned to the object N.

In the next block of statements a portfolio structure is defined by which the marginal contribution to the ES is bounded by an upper limit of 20%. The portfolio structure consists of three elements. First, the object `C1` contains the non-negativity and budget restrictions. Next, the first objective is added to this constraint, namely to find an asset allocation characterized by a minimal CVaR at the 95% confidence level. These two elements are combined into the object `ObjCVaR` and could be employed as such for finding a weight vector that yields a minimum downside risk at the specified confidence level. The third element defines the budget constraint with respect to the assets' downside risk contribution. Here, it is set to be at most 20%.

R code 11.6 Comparison of restricted ES portfolios with GMV and ERC.

```

library (FRAPO)
library (PortfolioAnalytics)
## Loading data and computing returns
data (MultiAsset)
R <- returnseries (MultiAsset, percentage = FALSE, trim = TRUE)
N <- ncol (R)
## Defining constraints and objective for CVaR budget
C1 <- constraint (assets = colnames (R), min = rep (0, N),
                   max = rep (1, N), min_sum = 1, max_sum = 1)
ObjCVaR <- add . objective (constraints = C1, type = "risk",
                             name = "ES", arguments = list (p = 0.95),
                             enabled = TRUE)
ObjCVaRBudget <- add . objective (constraints = ObjCVaR,
                                    type = "risk_budget",
                                    name = "ES", max_prisk = 0.2,
                                    arguments = list (p = 0.95),
                                    enabled = TRUE)
SolCVaRBudget <- optimize . portfolio (R = R,
                                         constraints = ObjCVaRBudget,
                                         optimize_method = "DEoptim",
                                         itermax = 50,
                                         search_size = 20000,
                                         trace = TRUE)
WCVaRBudget <- SolCVaRBudget $ weights
CVaRBudget <- ES (R, weights = WCVaRBudget, p = 0.95,
                   portfolio_method = "component")
## Minimum CVaR concentration portfolio
ObjCVaRMinCon <- add . objective (constraints = ObjCVaR,
                                    type = "risk_budget",
                                    name = "ES",
                                    min_concentration = TRUE,
                                    arguments = list (p = 0.95),
                                    enabled = TRUE)
SolCVaRMinCon <- optimize . portfolio (R = R,
                                         constraints = ObjCVaRMinCon,
                                         optimize_method = "DEoptim",
                                         itermax = 50,
                                         search_size = 20000,
                                         trace = TRUE)
WCVaRMinCon <- SolCVaRMinCon $ weights
CVaRMinCon <- ES (R, weights = WCVaRMinCon, p = 0.95,
                   portfolio_method = "component")
## GMV Portfolio
WGMV <- Weights (PGMV (R, percentage = FALSE))
CVaRGMV <- ES (R, weights = WGMV, p = 0.95,
                 portfolio_method = "component")
## ERC Portfolio
WERC <- Weights (PERC (cov (R), percentage = FALSE))
CVaRERC <- ES (R, weights = WERC, p = 0.95,
                 portfolio_method = "component")

```

Incidentally, by utilizing an upper bound equal to $1/N$, one would obtain the same solution as in the case of an ERC portfolio. The reason for this is that the marginal contributions to CVaR scale linearly to the marginal contributions with respect to the variance-covariance matrix of returns. This portfolio setting is then used in the call to `optimize.portfolio()`. The optimizer has been selected as `DEoptim` and as optimizing parameters `itermax` has been set to 50 and the `search_size` to 20 000. It is recommended that the ratio between the search size and the maximum number of iterations should exceed the count of parameters by a factor of ten. The weight vector is extracted from the returned `list` object `SolCVaRBudget` and assigned to `CVaRBudget`. In lines 24 onwards, the weight result obtained is inspected with respect to the portfolio's ES and whether the solution found does comply with the downside risk budget. This information can be returned by the function `ES()` contained in the package **PerformanceAnalytics**, which is stated as a dependency of **PortfolioAnalytics**.

In the next block of R statements, a minimum concentrated portfolio with respect to the downside risk contributions of its constituents is defined. Here, the object `ObjCVaR` has been reutilized and the minimization of the maximum downside contribution is achieved by setting the argument `min_concentration` to TRUE for the portfolio type `risk_budget`. The object `ObjCVaRMinCon` holds the specification for an MCC portfolio. The solution to this portfolio setting can—similarly to the above optimization—be determined by calling `optimize.portfolio()`. The allocation is assigned to `WCVaRMinCon`, and the downside risk as well as the assets' contributions to it can be queried by employing the `ES()` function. In the last section of Listing 11.5 the solutions pertinent to an equal-risk contributed strategy and a global minimum-variance strategy are determined.

The four asset allocations and the associated marginal CVaR contributions are summarized in Table 11.3. The concentration of the GMV solution to be expected with respect to the least volatile asset, namely German Bunds, is striking. Almost 90% of wealth is allocated to this asset and the remaining share is roughly allotted in equal chunks to US and German stocks. The remaining assets do not enter into the GMV solution, or with a negligible amount only. The ERC solution is more balanced in terms of its weight distribution, but still almost two thirds would have been invested in sovereign bonds of Germany and the UK. The MCC portfolio takes a middle stance between the GMV and ERC solutions and the BCC is the most akin to an equal-weighted solution. This picture is qualitatively mirrored by the percentage contributions to each of the portfolios' downside risk. Worth highlighting is the negative contribution of German bonds in the ERC, BCC and MCC allocations. This is primarily because of the comparatively high allotment to stocks and hence the bond allocation serves as a risk diversifier for tail losses. In the case of the BCC optimization this was forced by limiting the maximum contribution to 20%, which resulted in a smaller weight for German bonds. This can be seen by comparing the BCC with the MCC allocations. The minimum CVaR allocation delivers an allocation whereby the risk contributions are the least concentrated, but the German REX still contributes almost 40% to risk, which is twice as high as allowed for the BCC setting.

Table 11.3 Weight and downside risk contributions of multi-asset portfolios.

Assets	Weights				Risk contributions			
	GMV	ERC	BCC	MCC	GMV	ERC	BCC	MCC
S&P 500	4.55	3.72	7.80	6.40	9.83	16.63	16.61	17.06
Russell 3000	0.00	3.59	6.20	2.20	0.00	16.80	13.74	6.15
DAX	4.69	3.47	7.60	6.80	12.19	14.34	15.93	16.76
FTSE 100	0.00	4.12	1.20	12.20	0.00	11.20	1.58	20.62
Nikkei 225	1.35	3.38	5.00	4.00	3.16	22.36	15.49	15.22
MSCI EM	0.00	2.14	5.80	3.20	0.00	14.22	17.69	12.53
US Treasury	0.00	16.42	2.40	0.20	0.00	5.40	0.76	0.05
German REX	88.72	42.44	3.20	21.60	74.75	-17.60	-0.32	-3.94
UK Gilts	0.40	15.93	60.80	38.80	0.50	5.00	18.52	8.90
Gold	0.29	4.78	0.00	4.60	-0.43	11.63	0.00	6.65

Table 11.4 Key measures of portfolio solutions for multi-asset portfolios.

Measures	GMV	ERC	BCC	MCC
Standard deviation	0.815	1.151	1.923	1.766
ES (95%)	1.437	2.923	5.366	4.613
Diversification ratio	1.853	2.035	1.546	1.661
Concentration ratio	0.417	0.110	0.183	0.133

Summary statistics on the portfolio level for the four approaches are provided in Table 11.4. With respect to the portfolios' return volatility, the GMV allocation yields the lowest outcome and the BCC solution implies the highest return dispersion, measured by the portfolio standard deviation. The latter is due to the binding CVaR budget constraint which forces a lower weight on the least volatile asset and hence the remaining wealth has to be allocated into riskier constituents. This fact is mirrored by the highest value of the portfolios' downside risk. Here, the MCC allocation yields a less risky wealth distribution, followed by the ERC and GMV solutions. The ERC outcome is with respect to the volatility and downside risk characteristics situated between the GMV and MCC/BCC portfolios. The diversification ratio is maximal for the ERC portfolio and the least diversified is the BCC; the orders are reversed for the concentration ratio. With respect to this characteristic the ERC and MCC solution both deliver an allocation of roughly the same concentration and the allotment according to the BCC approach fares slightly worse.

References

- Ardia D., Arango J., and Gomez N. 2011a Jump-diffusion calibration using Differential Evolution. *Wilmott Magazine* **55**, 76–79.
- Ardia D., Boudt K., Carl P., Mullen K., and Peterson B. 2011b Differential Evolution (DEoptim) for non-convex portfolio optimization. *The R Journal* **3**(1), 27–34.
- Ardia D., Mullen K., Peterson B., and Ulrich J. 2015 *DEoptim: Differential Evolution in R*. version 2.2-3.
- Bacon C. 2004 *Practical Portfolio Performance Measurement and Attribution*. John Wiley & Sons, Chichester, UK.
- Beasley J. 1990 Or-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society* **41**(11), 1069–1072.
- Boudt K., Carl P., and Peterson B. 2010 Portfolio optimization with CVaR budgets. Presentation at R/Finance Conference, Katholieke Universiteit Leuven and Lessius, Chicago, IL.
- Boudt K., Carl P., and Peterson B. 2011 Asset allocation with conditional value-at-risk budgets. Technical report, <http://ssrn.com/abstract=1885293>.
- Boudt K., Peterson B., and Croux C. 2007 Estimation and decomposition of downside risk for portfolios with non-normal returns. Working Paper KBI 0730, Katholieke Universiteit Leuven, Faculty of Economics and Applied Economics, Department of Decision Sciences and Information Management (KBI), Leuven.
- Boudt K., Peterson B., and Croux C. 2008 Estimation and decomposition of downside risk for portfolios with non-normal returns. *The Journal of Risk* **11**(2), 79–103.
- Brest J., Greiner S., Boskovic B., Mernik M., and Zumer V. 2006 Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation* **10**, 646–657.
- Canakgoz N. and Beasley J. 2008 Mixed-integer programming approaches for index tracking and enhanced indexation. *European Journal of Operational Research* **196**, 384–399.
- Choueifaty Y. and Coignard Y. 2008 Toward maximum diversification. *Journal of Portfolio Management* **34**(4), 40–51.
- Choueifaty Y., Froidure T., and Reynier J. 2011 Properties of the most diversified portfolio. Working paper, TOBAM, Paris.
- Coles S., Heffernan J., and Tawn J. 1999 Dependence measures for extreme value analysis. *Extremes* **2**(4), 339–365.
- Conceicao E. and Mächler M. 2015 *DEoptimR: Differential Evolution Optimization in pure R*. R package version 1.0-4.
- Dobrić J. and Schmid F. 2005 Nonparametric estimation of the lower tail dependence λ_l in bivariate copulas. *Journal of Applied Statistics* **32**(4), 387–407.
- Eddelbüttel D. 2016 *RcppDE: Global optimization by Differential Evolution in C++*. R package version 0.1.5.
- Frahm G., Junker M., and Schmidt R. 2005 Estimating the tail dependence coefficient: Properties and pitfalls. *Insurance: Mathematics and Economics* **37**(1), 80–100.

- Heffernan J. 2000 A directory of coefficients of tail dependence. *Extremes* **3**(3), 279–290.
- Hirschman A. 1964 The paternity of an index. *The American Economic Review* **54**(5), 761–.
- Maillard S., Roncalli T., and Teiletche J. 2009 On the properties of equally-weighted risk contributions portfolios. Working paper, SGAM Alternative Investments and Lombard Odier and University of Paris Dauphine.
- Maillard S., Roncalli T., and Teiletche J. 2010 The properties of equally weighted risk contribution portfolios. *The Journal of Portfolio Management* **36**(4), 60–70.
- Markowitz H. 1952 Portfolio selection. *The Journal of Finance* **7**(1), 77–91.
- Mitchell M. 1998 *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA.
- Mullen K., Ardia D., Gil D., Windover D., and Cline J. 2011 DEoptim: An R package for global optimization by Differential Evolution. *Journal of Statistical Software* **40**(6), 1–26.
- Peterson B. and Boudt K. 2008 Component VaR for a non-normal world. *Risk*. November.
- Peterson B. and Carl P. 2015 *PortfolioAnalytics: Portfolio Analysis, Including Numerical Methods for Optimization of Portfolios*. R package version 1.0.3636.
- Pfaff B. 2015 *cccp: Cone Constrained Convex Problems*. R package version 0.2-4.
- Price K., Storn R., and Lampinen J. 2006 Differential Evolution: A Practical Approach to Global Optimization *Natural Computing*. Springer-Verlag, New York.
- Qian E. 2005 Risk parity portfolios: Efficient portfolios through true diversification. White paper, PanAgora, Boston, MA.
- Qian E. 2006 On the financial interpretation of risk contribution: Risk budgets do add up. *Journal of Investment Management* **4**(4), 1–11.
- Qian E. 2011 Risk parity and diversification. *The Journal of Investing* **20**(1), 119–127.
- Roncalli T. 2013 *Introduction to Risk Parity and Budgeting*. Chapman and Hall/CRC, Boca Raton, FL.
- Sarkar D. 2008 *Lattice: Multivariate Data Visualization with R*. Springer, New York.
- Scaillet O. 2002 Nonparametric estimation and sensitivity analysis of expected shortfall. *Mathematical Finance* **14**(1), 74–86.
- Schmidt R. and Stadtmüller U. 2006 Nonparametric estimation of tail dependence. *The Scandinavian Journal of Statistics* **33**, 307–335.
- Spinu F. 2013 An algorithm for computing risk parity weights SSRN. OMERS Capital Markets.
- Storn R. and Ulrich J. 1997 Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization* **11**(4), 341–359.
- Zhu S., Li D., and Sun X. 2010 Portfolio selection with marginal risk control. *Journal of Computational Finance* **14**(1), 1–26.

12

Risk-optimal portfolios

12.1 Overview

In this chapter portfolio optimization methods are presented in which some sort of risk measure and its level are directly factored into the weightings of the financial instruments. These approaches can be distinguished from Markowitz portfolios in the sense that now a certain VaR or ES level is not the result of an efficient portfolio allocation, but an objective. Hence the user can ask questions such as: “How should the portfolio weights be chosen such that the resulting 99% VaR is 3.5%?” A thorough and concise comparison between mean-VaR and mean-variance portfolios is provided in Alexander and Baptista (1999, 2002, 2004, 2008), as well as in De Giorgi (2002). This approach has recently been investigated by Durand et al. (2011). In the light of the Basel Accord, these kinds of portfolio optimization have attracted interest from practitioners, because the capital adequacy requirements can be derived from these kinds of risk measures (see Alexander and Baptista 2002, for instance). Portfolio optimizations involving the expected shortfall have been analyzed by Rockafellar and Uryasev (2000, 2001) and Uryasev and Rockafellar (1999).

In a similar vein, portfolios can be optimized with respect to their draw-down. This type of optimization is applicable when a portfolio is benchmarked against a broader market index.

In the next two sections, the mean-VaR and mean-ES portfolio approaches are presented and contrasted with the classical mean-variance portfolios. In Section 12.4 portfolios for which some sort of draw-down measure is minimized or targeted are presented. Section 12.5 presents the relevant **R** packages, and the applications in Section 12.6 conclude the chapter.

12.2 Mean-VaR portfolios

In this section the mean-VaR portfolio approach is presented and contrasted with Markowitz portfolios as presented in Section 5.2. This comparison draws on the work of Alexander and Baptista (2002) and De Giorgi (2002). The VaR risk measure was defined in (4.4), and is repeated here for the reader's convenience:

$$\text{VaR}_\alpha = \inf\{l \in \mathbb{R} : P(L > l) \leq 1 - \alpha\} = \inf\{l \in \mathbb{R} : F_L(l) \geq \alpha\}, \quad (12.1)$$

where F_L is the distribution function of the losses, which is now assumed to be elliptical; that is, it is now assumed that the distribution of the returns is either multivariate normal or multivariate Student's t . In the synopsis of the risk measures the mean-VaR risk measure has been defined as the difference between the expected return and VaR. This double nomenclature is perhaps unfortunate, because we are using "mean-VaR" in two ways: first for the risk measure, and second in naming the axis in the efficient frontier plane. However, it should be evident from the context when mean-VaR refers to the risk measure and when it is used in the context of optimal mean-VaR portfolios along the mean-VaR efficient frontier.

Akin to the classical Markowitz portfolio, mean-VaR portfolio optimization can be approached from two perspectives: either determining the weights for a specified VaR at a given confidence level which maximizes the portfolio return, or minimizing the mean VaR for a given confidence level for a fixed portfolio return. In accordance with the exposition in Section 5.2, the latter route is chosen in the following. In a first step, it is assumed that there are N risky assets and the wealth is completely allocated among them. Let $z_\alpha = \Phi^{-1}(\alpha)$ denote the quantile of the loss distribution at the confidence level $\alpha \in (1/2, 1)$. Then the mean VaR is defined as

$$\text{VaR}_\alpha^m = z_\alpha \sigma_W - \bar{r}, \quad (12.2)$$

where \bar{r} is the portfolio return $\omega' \mu$. A portfolio $\bar{\omega} \in \Omega$ then belongs to the mean-VaR boundary at confidence level α if it solves the following problem:

$$\begin{aligned} P_{\text{VaR}^m} &= \arg \min_{\omega \in \Omega} \text{VaR}_\alpha^m = z_\alpha \sigma_W - \bar{r} \\ \omega' \mu &= \bar{r}, \\ \omega' i &= 1, \end{aligned} \quad (12.3)$$

where i is an $(N \times 1)$ vector of 1s. This is a quadratic objective problem with linear constraints, due to $\sigma_W = \omega' \Sigma \omega$. This optimization problem is quite similar to mean-variance portfolios as described in Section 5.2, and indeed it is shown in Alexander and Baptista (2002) that mean-VaR portfolios are a subset of the mean-variance boundary. Hence, the condition for a mean-variance efficient portfolio (see Merton 1972),

$$\frac{\sigma_w^2}{1/C} - \frac{(\bar{r} - A/C)^2}{D/C^2} = 1, \quad (12.4)$$

where the scalars A , B , C , and D are defined as

$$A = \mathbf{i}' \Sigma^{-1} \boldsymbol{\mu}, \quad (12.5)$$

$$B = \boldsymbol{\mu}' \Sigma^{-1} \boldsymbol{\mu}, \quad (12.6)$$

$$C = \mathbf{i}' \Sigma^{-1} \mathbf{i}, \quad (12.7)$$

$$D = BC - A^2, \quad (12.8)$$

can be written in terms of the mean-VaR frontier as

$$\frac{[(\text{VaR}^m + \bar{r})/z_\alpha]^2}{1/C} - \frac{(\bar{r} - A/C)^2}{D/C^2} = 1, \quad (12.9)$$

in which (12.2) has been solved for σ_W .

The shapes of the efficient frontiers are analyzed in Alexander and Baptista (2002) at the limits of the confidence level α , that is, $\alpha \rightarrow 1/2$ and $\alpha \rightarrow 1$. In the former case, the efficient frontier is a straight downward sloping line through the origin with slope coefficient equal to -1 in the (expected return, mean-VaR) space. This is due to the fact that z_α approaches zero and hence, according to (12.2), the mean VaR equals minus the expected portfolio return. In the latter case, the efficient frontier converges to the traditional mean standard deviation hyperbola. In this case, $z_\alpha \rightarrow \infty$ and hence $\text{VaR}^m/z_\alpha = \sigma_W$, because $\bar{r}/z_\alpha \rightarrow 0$. The general formula for the mean-VaR boundary in the (expected return, mean-VaR) plane is given by

$$\text{VaR}^m = -\bar{r} + z_\alpha \sqrt{\frac{1}{D}(c\bar{r}^2 - 2a\bar{r} + b)}. \quad (12.10)$$

The band of efficient frontiers between these two extreme mean-VaR levels, which is a straight line for $\alpha = 0.5$, becomes increasingly hyperbolic as the confidence level increases. This is illustrated in Figure 12.1. The mean-VaR boundaries in the (expected portfolio return, standard deviation) space are shown in Figure 12.2 for the 90%, 95%, and 99% confidence levels, respectively.

As can be discerned from Figures 12.1 and 12.2, a proper efficient frontier for mean-VaR portfolios does not result for all mean-VaR confidence levels, but only for values of α that are sufficiently large. The reason for this can be gleaned from (12.2). The right-hand side consists of a standard deviation term $z_\alpha \sigma_W$ and the mean effect term \bar{r} . If the former does not outweigh the latter, then the minimization of the mean VaR has no solution. If the condition $\alpha > \Phi(\sqrt{D/C})$ is met, than a valid mean-VaR optimization problem results and the weights of the minimum-VaR portfolio at the α confidence level can be determined in closed form as

$$\boldsymbol{\omega}_{\text{VaR}_\alpha^m} = g + h \left[\frac{A}{C} + \sqrt{\frac{D}{C} \left(\frac{z_\alpha^2}{C(z_\alpha)^2 - D} - \frac{1}{C} \right)} \right], \quad (12.11)$$

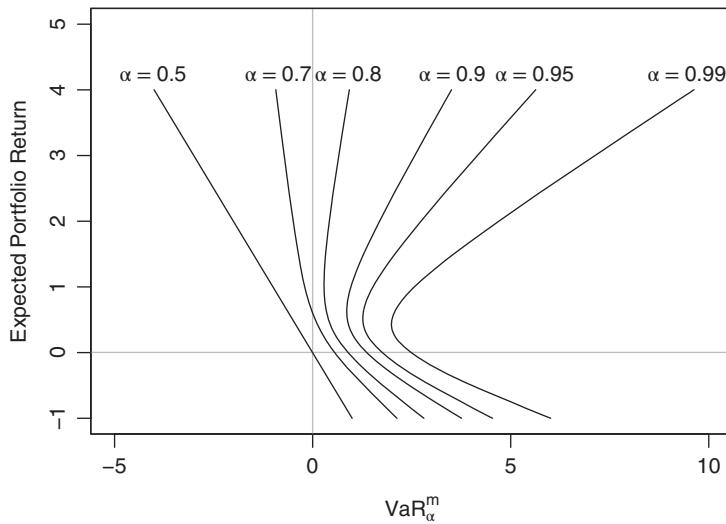


Figure 12.1 Boundaries of mean-VaR portfolios in the mean-VaR plane.

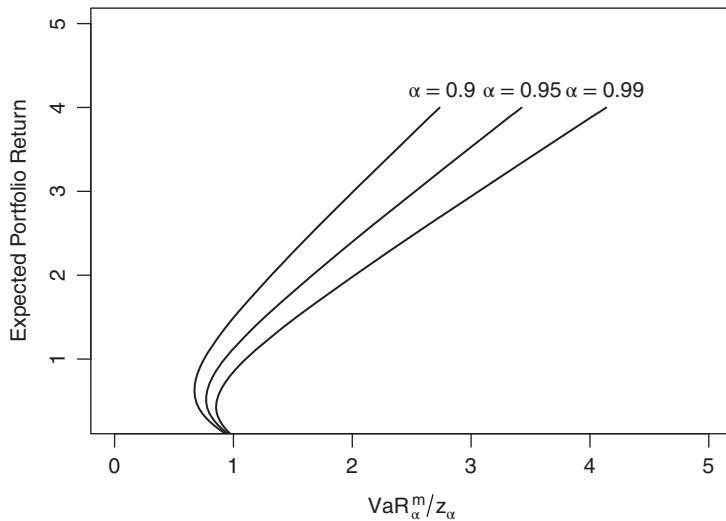


Figure 12.2 Boundaries of mean-VaR portfolios in the mean standard deviation plane.

where the $(N \times 1)$ vectors \mathbf{g} and \mathbf{h} are defined as

$$\mathbf{g} = \frac{1}{D} [B\Sigma^{-1}\mathbf{i} - A\Sigma^{-1}\boldsymbol{\mu}], \quad (12.12)$$

$$\mathbf{h} = \frac{1}{D} [C\Sigma^{-1}\boldsymbol{\mu} - A\Sigma^{-1}\mathbf{i}]. \quad (12.13)$$

The value of VaR_α^m corresponding to the weight vector of the minimum-VaR portfolio α for a given confidence level is then

$$\text{VaR}_\alpha^m = z_\alpha \sqrt{\frac{z_\alpha^2}{C(z_\alpha)^2 - D}} - \left[\frac{A}{C} + \sqrt{\frac{D}{C} \left(\frac{z_\alpha^2}{C(z_\alpha)^2 - D} - \frac{1}{C} \right)} \right]. \quad (12.14)$$

Because a minimum-VaR portfolio is on the efficient frontier of mean-variance portfolios, if it exists, by reverse conclusion all portfolios on the mean-variance efficient frontier except for the minimum-variance portfolio are also minimum-VaR portfolios for a certain confidence level. The values of these confidence levels α_{MV} are equal to

$$\alpha_{MV} = \Phi \left(\sqrt{\frac{D}{C} + \frac{D^2/C^3}{(\bar{r} - A/C)^2}} \right). \quad (12.15)$$

Furthermore, each mean-variance efficient portfolio except for the minimum-variance portfolio is mean-VaR inefficient for confidence levels lower than expressed in (12.15) and mean-VaR efficient for greater confidence levels. In addition, the minimum-variance portfolio is mean-VaR inefficient. Finally, it is also worth highlighting that the minimum-VaR portfolio solution approaches the minimum-variance portfolio as $\alpha \rightarrow 1$.

Next, the relation between the mean-variance and mean-VaR efficient frontiers will be presented when a risk-free security, $r_f > 0$, is included as an investment opportunity. Therefore, the scope of the assets is extended and the set of admissible $N+1$ weights is now given as $\Omega = \{\omega \in \mathbb{R}^{N+1} : \sum_{i=1}^{N+1} \omega_i = 1\}$, where ω_{N+1} denotes the share invested in the risk-free asset. The problem formulation as in (12.3) remains unchanged and hence a portfolio that belongs to the mean-VaR boundary is also on the efficient frontier for mean-variance portfolios for which an investment opportunity in a safe asset exists. In Merton (1972) the condition to be met for this kind of portfolio is stated as

$$\sigma_W^2 = (\bar{r} - r_f)^2 / H, \quad (12.16)$$

where the scalar H is defined as $H = Cr_f^2 - 2Ar_f + B$ and the constants A , B , and C are given by (12.5)–(12.7). Similar to the reformulation of the condition for mean-VaR portfolios without a risk-free asset, (12.14) can be expressed in terms of VaR_α^m :

$$[(\text{VaR}_\alpha^m + \bar{r})/z_\alpha]^2 = (\bar{r} - r_f)^2 / H. \quad (12.17)$$

Similar to the enclosing asymptotes of the mean-variance hyperbola as shown in Section 5.2, the asymptotes for the mean-VaR boundary are given by

$$\text{VaR}_\alpha^m = \begin{cases} \frac{\bar{r}(z_\alpha - \sqrt{H}) - z_\alpha r_f}{\sqrt{H}} & \text{if } \bar{r} \geq r_f, \\ \frac{\bar{r}(-z_\alpha - \sqrt{H}) + z_\alpha r_f}{\sqrt{H}} & \text{if } \bar{r} < r_f, \end{cases} \quad (12.18)$$

after solving (12.17) for VaR_α^m . The qualitative difference between these asymptotes and those for the mean-variance efficient frontier is that the slopes of the former differ in absolute terms.

For portfolios with an investment opportunity in a risk-free asset, efficient mean-VaR portfolios exist only for sufficiently large confidence levels α . The condition now becomes $\alpha > \Phi(\sqrt{H})$ for a mean-VaR efficient portfolio that is an element of the mean-variance efficient frontier, too. In the case of a risk-free asset the two frontiers coincide, whereas when a risk-less security is excluded from the set of investment opportunities, the mean-VaR boundary is only a subset of the mean-variance efficient frontier.

Recall from Section 5.2 the derivation of the capital market line for mean-variance portfolios, in particular (5.7) and (5.8). For mean-VaR portfolios

with a risk-less asset, the CML can be deduced from the definition of its VaR:

$$\begin{aligned} \mathbb{P}\{\mathbb{E}[R] = r_p \leq -\text{VaR}_\alpha^m\} &= 1 - \alpha, \\ \mathbb{P}\{r_f + \gamma(\bar{r} - r_f) \leq -\text{VaR}_\alpha^{mp}\} &= 1 - \alpha, \end{aligned} \quad (12.19)$$

where VaR_α^{mp} now refers to the VaR of the total portfolio and not only the VaR of the risky assets. The probability statement in the curly braces can be transformed to

$$\mathbb{P}\{\bar{r} \leq r_f - (\text{VaR}_\alpha^{mp} + r_f)/\gamma\} = 1 - \alpha. \quad (12.20)$$

The relation between the portfolio's VaR and that of the risky assets only is then given by

$$\text{VaR}_\alpha^{mp} = -r_f + \gamma(\text{VaR}_\alpha^m + r_f). \quad (12.21)$$

The share to be invested in the risky assets is thus given as a function of the ratio between the two adjusted VaR measures: $\gamma = (\text{VaR}_\alpha^{mp} + r_f)/(\text{VaR}_\alpha^m + r_f)$. Finally, α in the formula for the CML can be expressed similarly to (5.8):

$$\mathbb{E}[R] = r_p = r_f + \frac{\bar{r} - r_f}{\text{VaR}_\alpha^m + r_f}(\text{VaR}_\alpha^{mp} + r_f), \quad (12.22)$$

but now the relevant mean-VaR measures adjusted by the risk-free rate are used, rather than the portfolio dispersion. This mean-VaR CML is applicable for $\text{VaR}_\alpha^{mp} \geq -r_f$ and hence the minimal mean-VaR portfolio is the trivial solution of being invested 100% in the risk-free asset. The mean-VaR efficient frontier with a risk-free asset and the lower and upper asymptotes, that is, the CML, are depicted in Figure 12.3.

The point P_T represents the tangency portfolio which is defined as the locus where the slope of the CML is equal to the slope of the efficient frontier. Akin to the maximum Sharpe-ratio portfolio in the mean standard deviation space, at this point the reward-to-VaR ratio is maximized. The origin of the two asymptotes is defined by $(-r_f, r_f)$, that is, the VaR of a 100% investment is equal to the negative of its rate of return.

Finally, the point P_{MV} corresponding to a minimum-variance portfolio is marked on the efficient frontier, highlighting the fact that minimum-variance portfolios are in

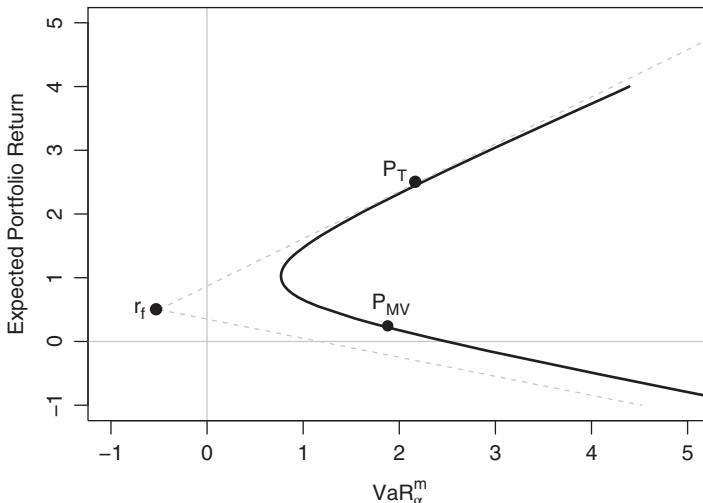


Figure 12.3 Tangency mean-VaR portfolio.

principle not mean-VaR efficient, that is, there are portfolio solutions that are characterized by a higher return and less risk.

12.3 Optimal CVaR portfolios

In a series of papers by Uryasev and Rockafellar portfolio optimization with respect to the conditional VaR (CVaR) is advocated (see Rockafellar and Uryasev 2000, 2001; Uryasev and Rockafellar 1999). The market risk measure CVaR is defined as the expected loss exceeding the VaR for a given confidence level. Synonyms for CVaR are “mean excess loss,” “mean shortfall,” and “tail VaR.” For continuous loss distributions, this risk measure is identical to the expected shortfall as introduced in Section 4.2.

As elucidated in Chapter 4, the VaR is not a coherent risk measure, but both ES and CVaR are coherent. From an optimization point of view, Rockafellar and Uryasev (2001) show that CVaR is a convex function, but VaR may be non-convex, which can yield portfolio solutions pertinent to local and not global optima. However, an outright optimization with respect to CVaR/ES is complicated from a numerical point of view due to the dependence of the ES on the VaR—see (4.6). To cope with this issue, the authors introduced more granular risk measures, and then defined CVaR as a weighted average. This step enables the derivation of a convex function with respect to the portfolio weights. In addition, the objective function can be optimized by utilizing linear programming techniques and is also applicable for non-normal loss distributions.

The following paragraphs elucidate the approach taken by the authors in optimizing portfolios with respect to CVaR and how portfolios of this kind relate to

mean-variance and mean-VaR portfolios as shown in the previous section. The latter topic has been investigated by Alexander and Baptista (2004).

The following risk measures are needed in order to define CVaR (see Uryasev 2005):

- VaR_α , the α -quantile of a loss distribution;
- CVaR^+ , the expected losses strictly exceeding VaR (i.e., mean excess loss or expected shortfall);
- CVaR^- , expected losses that are weakly exceeding VaR (i.e., losses that are either equal to or exceed VaR)—this risk measure is also known as “tail VaR.”

The CVaR is then defined as a weighted average between VaR and CVaR^+ :

$$\text{CVaR} = \lambda \text{VaR} + (1 - \lambda) \text{CVaR}^+, \quad (12.23)$$

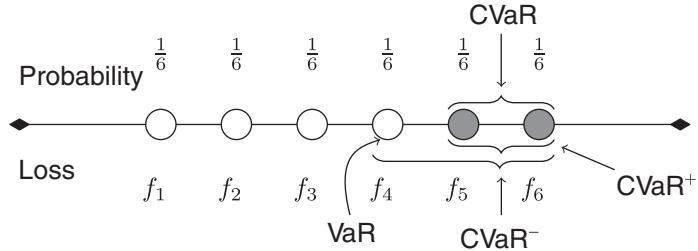
where the weight $0 \leq \lambda \leq 1$ is given by $\lambda = (\Psi(\text{VaR}) - \alpha)/(1 - \alpha)$ and $\Psi(\text{VaR})$ denotes the probability that losses do not exceed or are equal to VaR for a given confidence level.

The relations between the risk measures for a discrete loss distribution are shown in Figure 12.4, adapted from Uryasev (2005). In this figure six fictional and equal likely losses, f_1, \dots, f_6 , are portrayed on the real line. The first case considered in the figure refers to a confidence level $\alpha = 4/6$ that coincides with the VaR quantile, hence the weight λ takes a value of zero. The risk measures CVaR and CVaR^+ are then identical and equal to the average of the losses f_5 and f_6 . The VaR is given by the loss f_4 and the risk measure CVaR^- is the probability-weighted average of f_4, f_5, f_6 . In this instance the relation between the risk measures is $\text{VaR} < \text{CVaR}^- < \text{CVaR} = \text{CVaR}^+$. Now, consider a confidence level $\alpha = 7/12$, which is depicted in the lower part of Figure 12.4. The value of CVaR^+ is the same as implied in the panel above. However, the value of λ now becomes $\lambda = 1/5$, and hence $\text{CVaR} = 1/5\text{VaR} + 4/5\text{CVaR}^+$, which translates into loss terms as $\text{CVaR} = 1/5f_4 + 2/5f_5 + 2/5f_6$. In the second case, the relation between the risk measures reads: $\text{VaR} < \text{CVaR}^- < \text{CVaR} < \text{CVaR}^+$.

Having defined the targeted risk measure CVaR, one can then derive the objective function. Though the existence of a (continuous) multivariate distribution function for the portfolio returns, $p(\mathbf{r}|\boldsymbol{\omega})$, and hence the losses is not critical, its existence is assumed for the sake of notational convenience. Generally, the loss function in the context of portfolios can be expressed as $f(\boldsymbol{\omega}, \mathbf{r})$, where the dimension of $\boldsymbol{\omega}$ is N and \mathbf{r} is M . The difference in dimensions is due to the possibility of non-random returns, that is, those that are deemed to be fixed and hence risk-less. The loss function then maps $\mathbb{R}^N \times \mathbb{R}^M \rightarrow \mathbb{R}$ onto the real line. The probability function for losses less than a threshold value z can then be expressed as:

$$\Psi(\boldsymbol{\omega}, z) = \int_{f(\boldsymbol{\omega}, \mathbf{r}) \leq z} p(\mathbf{r}|\boldsymbol{\omega}) \, d\mathbf{r}. \quad (12.24)$$

Case: $\alpha = \frac{4}{6} \rightarrow \lambda = 0$



Case: $\alpha = \frac{7}{12} \rightarrow \lambda! = 0$

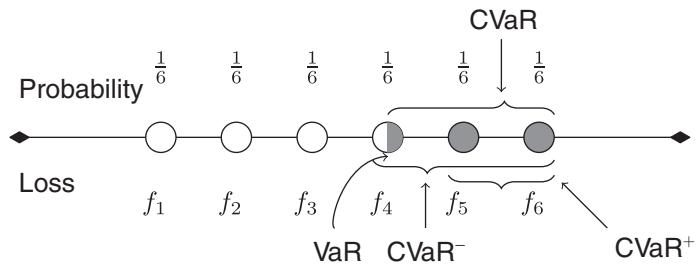


Figure 12.4 Discrete loss distribution: VaR, CVaR, CVaR⁺, and CVaR⁻.

If one substitutes for z the VaR for a given confidence level α , and introduce a performance function for CVaR as $\frac{1}{1-\alpha}\Phi(\omega)$,

$$\Phi(\omega) = \int_{f(\omega, r) \geq \text{VaR}(\omega, \alpha)} f(\omega, r) p(r|\omega) dr, \quad (12.25)$$

then the minimization of the excess loss function $\frac{1}{1-\alpha}\Phi(\omega)$ can be reduced to the minimization of

$$F(\omega, z) = (1 - \alpha)z + \int_{f(\omega, r) \geq z} (f(\omega, r) - z) p(r|\omega) dr, \quad (12.26)$$

which resembles the definition of CVaR introduced above. It is proved in Rockafellar and Uryasev (2001) that the minimization of (12.26) with respect to (ω, z) yields the same outcome as the minimization of (12.25) with respect to ω . The optimal value obtained for z is equal to the VaR. Hence, if one optimizes (12.26) one directly obtains a portfolio weights solution for VaR and minimal CVaR. In order to pave the way to showing how portfolio optimization with respect to CVaR can be expressed in terms of a linear program, the objective function can be expressed equivalently as

$$F(\omega, z) = (1 - \alpha)z + \int_{r \in \mathbb{R}^m} (f(\omega, r) - z)^+ p(r|\omega) dr, \quad (12.27)$$

where $(f(\boldsymbol{\omega}, \mathbf{r}) - z)^+ = b^+$ is defined as $b^+ = \max\{0, b\}$. Therefore, expressed in discrete terms, the portfolio problem that yields a minimal CVaR can be written as

$$\begin{aligned} P_{\text{CVaR}} &= \arg \min_{\boldsymbol{\omega} \in \Omega, \zeta \in \mathbb{R}} \zeta + \nu \sum_{j=1}^J y_j, \\ y_j &\geq f(\boldsymbol{\omega}, \mathbf{r}_j) - \zeta, \\ y_j &\geq 0, \\ \boldsymbol{\omega}' \boldsymbol{\mu} &= \bar{r}, \\ \boldsymbol{\omega}' \mathbf{i} &= 1, \end{aligned} \tag{12.28}$$

where $\nu = \frac{1}{(1-\alpha)J}$ is a constant factor used in the calculation of the mean excess losses. The portfolio problem can be amended by further constraints, such as box or group constraints and/or the exclusion of short positions. In the context of a portfolio, the loss function $f(\cdot)$ and hence the inequality constraints are linear with respect to the weights and thus can be solved by linear programming techniques. The optimal solution contains the value for VaR, ζ^{opt} , and the weights that produce a portfolio with a minimum CVaR, $\boldsymbol{\omega}^{\text{opt}}$. When an optimal solution for the portfolio weights has been found, these values can be inserted into the performance function $\frac{1}{1-\alpha} F(\boldsymbol{\omega} = \boldsymbol{\omega}^{\text{opt}}, z = \zeta^{\text{opt}})$ to compute the CVaR. In practice, the question arises as to which data should be employed for \mathbf{r}_j . Two approaches are ordinarily encountered in the literature: either one utilizes historic returns or one conducts a Monte Carlo simulation and draws random samples from a fitted multivariate distribution. A CVaR constraint, that is, a constraint such that the CVaR of a portfolio does not exceed a certain value C , can easily be included in the linear program as shown in (12.28) by including an additional set of linear constraints: $\zeta + \nu \sum_{j=1}^J y_j \leq C$.

We conclude this section by highlighting the relation between mean-variance, mean-VaR, and mean-CVaR portfolios in the model framework of the previous section, drawing on the work of Alexander and Baptista (2004). It is assumed throughout that the returns of the financial instruments are elliptically distributed (e.g., multivariate normal). Recall (4.7) and (4.8) in which the VaR and ES were defined for normally distributed losses. The terms $\Phi^{-1}(\alpha) = z_\alpha$ and $\phi(\Phi^{-1}(\alpha))/(1 - \alpha) = k_\alpha$ in those equations are the slope coefficients in the (standard deviation, risk measure) space, and the slope coefficient for ES is greater than that for VaR. Hence, *ceteris paribus*, the ES yields the more conservative risk measure, as should also be evident from Figures 4.1 and 12.4. Furthermore, it was stated in Chapter 4 that VaR and ES are both quantile values pertinent to the loss distribution. Therefore, a confidence level α^* exists with $\alpha^* < \alpha$, such that $\text{ES}_{\alpha^*} = \text{VaR}_\alpha$ (e.g., $\text{ES}_{0.975} = \text{VaR}_{0.99}$). In Section 12.2 it was shown that mean-VaR portfolios are on the mean-variance efficient frontier if the former exists. Because the value of the expected shortfall can be made coincident with VaR, but for a smaller confidence level, portfolios that are optimized with respect to ES/CVaR will be elements on the mean-variance efficient frontier, too. Furthermore, the following relation of the expected portfolio returns

exists between minimum-variance, minimum-VaR, and minimum-CVaR portfolios:

$$E[\mu_{VaR}] > E[\mu_{CVaR}] > E[\mu_{MV}]. \quad (12.29)$$

This is because, first, $k_\alpha > z_\alpha$, and second, one can simultaneously decrease the expected return and the standard deviation by moving in a south-west direction along the mean-variance efficient frontier starting at the point of the minimum-VaR portfolio. The reason why the expected portfolio return in the case of a CVaR portfolio is greater than that for a minimum-variance portfolio lies in the definition of the former. The latter does not depend on μ and hence the derivative with respect to the standard deviation is zero. Matters are different for a minimum-CVaR portfolio; here, if one moves in a north-east direction along the efficient frontier, the mean increases by a greater amount relative to the standard deviation, which implies a lower CVaR. Therefore, minimum-variance portfolios are also CVaR inefficient for any confidence level $\alpha < 1$. It was stated in the previous section that a mean-VaR efficient portfolio exists at the α confidence level if $z_\alpha > \sqrt{D/C}$, where the scalars C and D are as defined in (12.7) and (12.8), respectively. Furthermore, the mean-VaR boundary is a proper subset of the mean-variance boundary, that is, $E[\mu_{MV}] \geq E[\mu_{VaR}]$. A similar statement for CVaR is proved in the appendix of Alexander and Baptista (2004), if the quantile z_α is replaced by k_α and the portfolio return of the mean-VaR portfolio by μ_{CVaR} . In addition, if a risk-free asset is included, then the proposition that a portfolio belongs to the mean-VaR efficient frontier if $z_\alpha > \sqrt{H}$ for a given confidence level α and that it is an element on the mean-variance efficient frontier is also true for mean-CVaR portfolios when z_α is replaced with k_α .

12.4 Optimal draw-down portfolios

Closely related to CVaR-optimal portfolios are portfolio optimization problems that try to achieve weight solutions with respect to the portfolio's draw-down. This kind of optimization was proposed by Chekhlov et al. (2000, 2005) and Pardalos et al. (2004). A comparison between draw-down optimal portfolios and other risk-measure-related optimization techniques is made in Krokmal et al. (2002). The task of finding optimal portfolio allocations with respect to draw-down is of considerable interest to asset managers. In an attempt to limit the chances of business termination, asset managers are eager to avoid either large portfolio draw-downs and/or draw-downs over an extended period of time. A client might be tempted to withdraw his mandate when this occurs, resulting in the loss of management fees. This point has been exemplified by Chekhlov et al. (2005) for a commodity trading adviser.

The draw-down of a portfolio at time t is defined as the difference between the maximum uncompounded portfolio value prior to t and its value at t . More formally, denote by $W(\omega, t) = \mathbf{y}_t' \omega$ the uncompounded portfolio value at time t , with ω the portfolio weights for the N assets included in it and \mathbf{y}_t the cumulated returns. Then the draw-down, $\mathbb{D}(\omega, t)$, is defined as

$$\mathbb{D}(\omega, t) = \max_{0 \leq \tau \leq t} \{ W(\omega, \tau) \} - W(\omega, t). \quad (12.30)$$

The authors cited above deduced three functional risk measures from this definition: maximum draw-down (MaxDD), average draw-down (AvDD), and conditional draw-down at risk (CDaR). Similar to CVaR, CDaR is dependent on the chosen confidence level α , so that this concept constitutes a family of risk functions. The definition of CDaR is akin to CVaR—see (12.27)—for a data set in the time interval $[0, T]$:

$$\text{CDaR}(\boldsymbol{\omega})_\alpha = \min_{\zeta} \left\{ \zeta + \frac{1}{(1-\alpha)T} \int_0^T [\mathbb{D}(\boldsymbol{\omega}, t) - \zeta]^+ dt \right\}, \quad (12.31)$$

where ζ is a threshold value for the draw-downs, such that only $(1-\alpha)T$ observations exceed this value. If the number of observations is not an integer, then ζ is the weighted average of draw-downs that strictly exceed this threshold value and its next lower bounded value, hence a computation similar to CVaR is utilized, but now CDaR is a risk functional and not a risk measure as in the case of CVaR, and it is expressed in nominal rather than in return space. The limiting cases of this family of risk functions are MaxDD and AvDD. For $\alpha \rightarrow 1$, CDaR approaches the maximum draw-down: $\text{CDaR}(\boldsymbol{\omega})_{\alpha \rightarrow 1} = \text{MaxDD}(\boldsymbol{\omega}) = \max_{0 \leq t \leq T} \{\mathbb{D}(\boldsymbol{\omega}, t)\}$. The AvDD results for $\alpha = 0$, that is, $\text{CDaR}(\boldsymbol{\omega})_{\alpha=0} = \text{AvDD}(\boldsymbol{\omega}) = 1/T \int_0^T \mathbb{D}(\boldsymbol{\omega}, t) dt$.

With respect to the optimization of a portfolio's draw-down, these risk functionals can be implemented as inequality constraints for a fixed share of wealth at risk. For instance, one could require that the MaxDD is at most 10% of the capital: $\text{MaxDD}(\boldsymbol{\omega}) \leq \nu_1 C$, where $0 \leq \nu_1 \leq 1$ and C is the available capital/wealth. Similarly, this draw-down constraint can be specified for AvDD, $\text{AvDD}(\boldsymbol{\omega}) \leq \nu_2 C$, and CDaR, $\text{CDaR}(\boldsymbol{\omega}) \leq \nu_3 C$, as can a linear combination of these three risk functionals, with $0 \leq \nu_1, \nu_2, \nu_3 \leq 1$. The portfolio optimization is then expressed in discrete terms and the objective is defined as maximizing the annualized average portfolio return:

$$R(\boldsymbol{\omega}) = \frac{1}{dC} \mathbf{y}'_T \boldsymbol{\omega}, \quad (12.32)$$

where d is the number of years in the time interval $[0, T]$. More precisely, the following three linear program formulations are stated in Chekhlov et al. (2005) for optimizing a portfolio such that its MaxDD, AvDD, or CDaR is bounded by a certain wealth fraction. The task of maximizing the average annualized portfolio return with respect to limiting the maximum draw-down is given by

$$\begin{aligned} P_{\text{MaxDD}} = \arg \max_{\boldsymbol{\omega} \in \Omega, \boldsymbol{u} \in \mathbb{R}} R(\boldsymbol{\omega}) &= \frac{1}{dC} \mathbf{y}'_T \boldsymbol{\omega}, \\ u_k - \mathbf{y}'_k \boldsymbol{\omega} &\leq \nu_1 C, \\ u_k &\geq \mathbf{y}'_k \boldsymbol{\omega}, \\ u_k &\geq u_{k-1}, \\ u_0 &= 0, \end{aligned} \quad (12.33)$$

where \mathbf{u} denotes a $(T + 1 \times 1)$ vector of slack variables in the program formulation, that is, the maximum portfolio values up to time period k with $1 \leq k \leq T$.

When the portfolio is optimized with respect to limiting the average draw-down, only the first set of inequality constraints needs to be replaced with the discrete analogue of the mean draw-down expressed in continuous time as stated above, leading to

$$\begin{aligned} P_{\text{AvDD}} = \arg \max_{\omega \in \Omega, \mathbf{u} \in \mathbb{R}} R(\omega) &= \frac{1}{dC} \mathbf{y}'_T \omega, \\ \frac{1}{T} \sum_{k=1}^T (u_k - \mathbf{y}'_k \omega) &\leq v_2 C, \\ u_k &\geq \mathbf{y}'_k \omega, \\ u_k &\geq u_{k-1}, \\ u_0 &= 0. \end{aligned} \tag{12.34}$$

The setup for the CDaR linear program is slightly more cumbersome compared to the previous two. In addition to the slack variables, \mathbf{u} , that represent the maximum portfolio wealth levels, two additional auxiliary variables need to be introduced: first, the threshold draw-down value ζ dependent on the confidence level α ; and second, the $(T \times 1)$ vector \mathbf{z} representing the weak threshold exceedances. Hence, the linear programming problem is given by

$$\begin{aligned} P_{\text{CDaR}} = \arg \max_{\omega \in \Omega, \mathbf{u} \in \mathbb{R}, \mathbf{z} \in \mathbb{R}, \zeta \in \mathbb{R}} R(\omega) &= \frac{1}{dC} \mathbf{y}'_T \omega, \\ \zeta + \frac{1}{(1-\alpha)T} \sum_{k=1}^T z_k &\leq v_3 C, \\ z_k &\geq u_k - \mathbf{y}'_k \omega - \zeta, \\ z_k &\geq 0, \\ u_k &\geq \mathbf{y}'_k \omega, \\ u_k &\geq u_{k-1}, \\ u_0 &= 0. \end{aligned} \tag{12.35}$$

These linear programs can be amended by further restrictions on ω such as budget, non-negativity, and/or box constraints.

A word of caution concerning these concepts is in order. The MaxDD as well as the AvDD might capture or reflect portfolio losses inadequately. Recall that historic return trajectories are employed in the linear program. Therefore, the portfolio allocation that obeys a maximum draw-down constraint can be severely impacted by a single worst draw-down. On the other hand, the average draw-down concept is inconclusive about the size of the maximum portfolio's draw-down, but this can become quite an important issue in terms of risk management and control. The CDaR measure

circumvents these pitfalls to some extent, but all measures assume that the historic path dependencies will somehow prevail in the subsequent periods.

12.5 Synopsis of R packages

12.5.1 The package **fPortfolio**

The package **fPortfolio** has already been discussed in Section 10.4.2. Here, the package's capabilities with respect to CVaR portfolios as presented in Section 12.3 are highlighted along with the graphics functions for producing risk surface plots. A detailed description with worked examples is provided in Würtz et al. (2014). In order to get started, an object of class `fPFOLIOSPEC` must be generated with the `portfolioSpec()` function. If the default settings of this function are used, then the CVaR characteristics must be explicitly specified. That is, the type of portfolio is set to "CVaR" with the function `setType()`. Next, the user must decide the confidence level at which the optimization is carried out. This is accomplished with the function `setAlpha()`. Incidentally, the confidence level is pertinent to returns and not losses (i.e., the left tail of the distribution). The minimum required settings for a CVaR portfolio optimization problem are complete once the solver to be used has been provided. At the time of writing, only the GNU Linear Programming Kit (GLPK) solver has been implemented for solving the linear program and hence the right-hand side of the assignment reads "`solveRglpk`" in the call to `setSolver()`. Incidentally, constraints on the weights are defined later, when a given portfolio specification is utilized in the call to the optimizing function.

An efficient CVaR portfolio solution for a given target return is returned by the function `efficientPortfolio()`. The desired target return can be assigned to an object of class `fPFOLIOSPEC` with the function `setTargetReturn()`. The arguments of `efficientPortfolio()` are the return data (argument `data`), the `fPFOLIOSPEC` object (argument `spec`), and the constraints (argument `constraints`). The solution of a global minimum-CVaR portfolio is returned by calling `minriskPortfolio()`, and the portfolio solution that maximizes the return-to-CVaR ratio by the function `maxratioPortfolio()`. The risk-free rate is assigned to the portfolio specification object with the function `setRiskFreeRate()`. The portfolio frontier of a CVaR portfolio can be computed with the function `portfolioFrontier()`, which returns an object of formal class `fPORTFOLIO`. The efficient frontier can then be plotted with the functions `frontierPlot()` and/or `tailoredFrontierPlot()`. The efficient frontier can also be displayed by calling the `plot()` method; this brings up a menu by which the plotting of the frontier can be controlled and additional particular portfolio solutions can be marked on the frontier as points.

The package provides four functions for analyzing the risk surface of feasible long-only, fully invested allocations and one function for producing a risk surface plot. The convex hull of Markowitz-type optimization is returned by the function `markowitzHull()` (see also Figure 5.1). The function is specified with two

arguments: `data` and `nFrontierPoints`. The former expects a `timeSeries` object of the asset's returns and the latter defines the count of points on the frontier. The function returns a `matrix` object that contains the return–risk coordinates of the entire convex hull, that is, a polygon. The returned object is endowed with attributes for the used data ("`data`"), the convex hull ("`hull`"), and the frontier ("`frontier`"). The latter attribute is an object of class `fPORTFOLIO` itself. These attributes can be accessed for an object `foo` created by calling `markowitzHull()` with `attr(foo, "data")`, `attr(foo, "hull")`, `attr(foo, "frontier")`, respectively.

The grid points within the convex hull are determined by calling the function `feasibleGrid()`. Its first argument is `hull` and it expects an object created by invoking `markowitzHull()`. From the provided `hull` object the attributes "`hull`" and "`data`" are extracted. Next, a grid ranging from the minimum to maximum values of targeted returns and risks is determined, along with which grid points are located in the hull. The progress of this check is visualized if one sets the second argument `trace = TRUE`. The function returns a `list` object of informal class `feasibleGrid` that extends the `list` class. The named elements of this `list` object are `x`, `y`, and `z`. The former two contain the grid coordinates in the risk–return space and the latter is a `matrix` with elements set to 1 for points lying in the hull. Objects of the informal class `feasibleGrid` are furthermore endowed with the attributes "`data`", "`polygon`", and "`hull`" as discussed above.

Next, given the grid points within the convex hull, the best diversifying allocations can be determined by calling `bestDiversification()`. This function is endowed with three arguments. The first, `grid`, expects an object of informal class `feasibleGrid`. The second argument `FUN` requires the name of a function for which the optimizations of the feasible risk–return combinations should be carried out. By default, its value is set to "`var`". The optimal allocations for each grid coordinate are determined by calling the function `donlp2NLP()` with long-only and budget constraints. The progression of these optimizations can be visualized by setting the argument `trace = TRUE` in the call to `bestDiversification()`. The function returns an object of informal class `bestDiversification` that extends the `list` class. The named `list` elements are "`x`", "`y`", and "`z`". The former two elements carry the targeted risk and return levels for the grid coordinates and the latter the values of the objective function (by default, the portfolio variance). Objects of this informal class are furthermore endowed with the attributes "`data`", "`polygon`", and "`weights`". The former two are identically defined as in the previous function. The latter is a `matrix` object whereby the first two columns indicate the grid coordinates of the feasible allocations within the convex hull and the weights associated with each feasible allocation are stored in the remaining columns.

In order to create a risk surface plot, the function `riskSurface()` has to be employed. Its first argument `diversification` expects an object of informal class `bestDiversification`. Within the function's body the weight vectors for the feasible allocations are extracted and the values associated with these for the desired risk measure are computed. The user can provide a function for computing the risk measure as argument `FUN` in the call to `riskSurface()`. By default, its value is set

to `NULL` and the variance of the weights as an entropy-like risk measure is computed. Additional arguments for the function that returns the risk measure can be provided and are passed down by means of the ellipses argument in the function's closure. The function `riskSurface()` returns an object of informal class `riskSurface`, which extends the `list` class. The named `list` elements are `"x"`, `"y"`, and `"z"`. Similar to the previously discussed functions, the elements `"x"` and `"y"` contain the targeted risk and return levels for the grid coordinates. The element `"z"` is a `matrix` object that has the values for the risk measure as its elements for the coordinates within the convex hull and otherwise `NA`. Objects of `S3` class `riskSurface` have attributes `"data"`, `"polygon"`, and `"weights"`, defined as for objects of informal class `bestDiversification`.

A risk surface plot can then be created by calling `surfacePlot()`. The function's first argument is `surface`, which expects an object of informal class `riskSurface`. The values for the `x`, `y`, `z` coordinates are extracted from this `list` object and the risk surface is created by calling the `image()` function within the function's body. Whether these values are drawn as an image (the default) or as a filled contour is controlled by the argument `type` that takes either the value `"image"` for the former or `"filled.contour"` for the latter. The count of contour levels to be drawn is determined by the argument `nlevels` with a default value of 11. In addition to these parameters, the user can provide a color palette for the image. By default, the colors for the contours are taken from the `topo.colors()` palette. The logical arguments `addContour`, `addGrid`, and `addHull` with obvious meanings can further be employed for fine-tuning the appearance of the risk surface plot (all default values are set to `TRUE`).

12.5.2 The package **FRAPO**

The package **FRAPO** accompanying this book implements functions for computing portfolio solutions with draw-down constraints. `S4` classes and methods are employed. All functions, classes, and methods are exported via directives in its `NAMES-PACE`. The `S4` class structure is hierarchically ordered such that the classes defined for draw-down optimal portfolios inherit from the general portfolio class `PortSol`. The virtual class `PortDD` is a union of the particular draw-down classes in which the common characteristics of this kind of portfolio are brought together. The methods available for further analyzing the solution of draw-down optimal portfolios are defined either for the `S4` class `PortSol` or for the virtual class `PortDD`.

The linear programs as shown in (12.33)–(12.35) for the constrained maximum draw-down, average draw-down, and conditional draw-down at risk portfolios are solved with `GLPK` by utilizing the function `Rglpk_solve_LP()` of the package `Rglpk`.

A portfolio solution for a maximum draw-down constrained portfolio is returned by the function `PMaxDD()`. This function is specified with a total of four arguments. The first element of the function's closure is `PriceData` for holding the historic asset prices. The only requirement for `PriceData` is that the object is rectangular and, hence, has dimension 2. All time-series-related classes are

supported as well as `matrix` and `data.frame` objects. The second argument is `MaxDD` by which the upper bound of the draw-down is specified. The user has the option with the logical argument `softBudget` to decide whether the budget constraint is implemented as a strict equality constraint (fully invested) or whether a partial allocation (i.e., the sum of weights is less than 1), is allowed. The final item in the function's closure is the ellipsis argument; its content, if any, is passed down to the function `Rglpk_solve_LP()`. The function returns an object of formal class `PortMdd` with the slots `weights`, `opt`, `type`, `call`, `MaxDD`, and `DrawDown`. The first slot carries the optimal weight vector. It is displayed with some further information by the `show()` method or can be queried by the `Weights()` method, both defined for objects of class `PortSol`. The slot `opt` is a `list` object and carries the output from the solver. The solution returned from the optimizer can be queried by the `Solution()` method defined for objects of class `PortSol`. The next slot `type` is a character string containing information about the kind of portfolio optimization, in this case "maximum draw-down". The slot `call` is of class `call` and contains the call of the object's creating function. By providing this information, one can provide an `update()` method for swiftly changing the values of the arguments. So far, all slots are pertinent to the general class of portfolio solutions, `PortSol`. The last two slots contain information peculiar to maximum draw-down portfolios. The first is `MaxDD` in which the maximum draw-down of the portfolio solution is returned. This value need not coincide with the specified upper bound, that is, the specified upper bound in the call of `PMaxDD()` is a not a binding constraint. The historic portfolio draw-downs are contained in the slot `DrawDown`. In addition to the methods described above, a `DrawDowns()` and a `plot()` method are defined for objects of virtual class `PortDD`. The former returns the historic portfolio draw-downs as a `timeSeries` object and the latter plots the portfolio's draw-downs. The maximum draw-down is superimposed on this time series chart.

Portfolio solutions constrained by the average draw-down can be computed by means of the function `PaveDD()`. The function's argument specification differs only by one argument from `PMaxDD()`. The upper bound of the draw-down constraint is now `AveDD` instead of `MaxDD`. The function returns an object of formal class `PortAdd` and differs only with respect to the slot that carries the constraint information: `AveDD` for the average historical portfolio draw-down. The methods described above are also available through inheritance. With respect to the `plot()` method, the average draw-down is now superimposed on the time series chart of the draw-downs.

Portfolio optimizations that refer to the conditional draw-down concept are available in two flavors: first, a solution for a portfolio that is constrained by the CDaR can be computed with the function `PCDaR()`; second, a minimum CDaR portfolio is returned by calling `PMinCDaR()`. Both functions return an object of class `PortCdd`, which inherits from the class `PortSol` and is a member of the class union `PortDD`. The object's definition differs from the previous ones by having slots `CDaR` and `thresh` instead of `MaxDD` for the maximum draw-down and `AveDD` for the average draw-down portfolios. The slot `CDaR` carries the value of the conditional draw-down at risk for the specified confidence level `alpha` in the call to either `PCDaR()`

or `PMinCDaR()`, and the slot `thresh` is the threshold value for the conditional draw-downs. In the linear program specified in (12.35), ζ denotes this value. If a constrained CDaR portfolio solution is computed by calling `PCDaR()`, the permitted upper bound of the CDaR is specified via the argument `bound` (default value 0.05). The confidence level in both functions is set by the argument `alpha` with a default value of 0.95. The plot of the draw-downs is now superimposed by the value of CDaR.

12.5.3 Packages for linear programming

In this subsection **R** packages for solving linear programs are presented. The focus is on package implementations based on freely available (i.e., open source) solvers, which are ordinarily covered by the GPL or a similar license. However, it should be noted that **R** packages that interface to commercial optimizers are also hosted on CRAN and/or R-Forge. The packages **Rcplex** and **cplexAPI** provide interfaces to the CPLEX solver package (see Bravo et al. 2014 and Gelius-Dietrich 2015a, respectively) and the package **Rmosek** provides an interface to the commercial MOSEK optimizer (see Friberg 2014).¹ For more information, consult the vendors' internet sites: <http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/> and <http://www.mosek.com/>, respectively. At the time of writing, both vendors offered free trial versions and/or free academic license agreements. Finally, it should be mentioned that the **R** package **rneos** (see Pfaff 2016) provides user access to the Network-Enabled Optimization System (NEOS; see Czyzyk et al. 1998; Dolan 2001; Groppe and Moré 1997) by means of the XML-RPC (eXtensible Markup Language remote procedure call) application programming interface (API). Incidentally, the MOSEK solver is available in NEOS. For more information about NEOS, see the project's internet site at <https://neos-server.org/neos/>.

The packages **glpkAPI** and **Rglpk**

The packages **glpkAPI** (see Gelius-Dietrich 2015b) and **Rglpk** (see Theussl and Hornik 2015) both provide access to the **GLPK**. This software package is organized in the form of a callable library written in ANSI C. With the GLPK, large-scale linear programs as well as mixed integer programs can be solved. As optimization techniques, primal/dual simplex, primal/dual interior-point, and branch-and-cut methods have been implemented. The GLPK software is shipped with an API and a standalone linear program / mixed integer linear program (LP/MILP) solver. Problems can be formulated in the GNU MathProg modelling language (GMPL) which is a subset of the AMPL language. More information about GLPK can be found on the project's internet site, <http://www.gnu.org/s/glpk/>.

The two **R** packages differ in terms of how access between **R** and GLPK is established and the layer of abstraction. An implementation of the C API is made available in the package **glpkAPI**. This package is hosted on CRAN. It requires a

¹ All three packages are only available as source packages.

GLPK installation, including libraries and header files. Directives for installing the package are provided in an `INSTALL` file, located in the root directory of the source package. The package is shipped with a vignette in which the usage of the cornerstone functions for setting up and solving a linear program are shown.

A different route is followed for accessing GLPK from `R` within the package **Rglpk**. Instead of offering the user a low-level entry point to the C API, as is the case with the previous package, the package **Rglpk** offers high-level access. The package is hosted on CRAN and R-Forge. It is considered to be a core package in the CRAN “Optimization” Task View. The package is shipped with a `NAMESPACE` file in which the two main functions `Rglpk_read_file()` and `Rglpk_solve_LP()` are exported, as well as a `print()` method for files that have been imported by the GLPK file reader. The package is dependent on the package **slam** for handling sparse arrays and matrices. Linear programs in either the fixed or free Mathematical Programming System (MPS) format or expressed in the CPLEX LP macro language can be imported and brought into memory with the function `Rglpk_read_file()`. This is accomplished with the GLPK file reader. The function is endowed with four arguments. The argument `file` expects a character string which specifies the absolute or relative path to the model file. With the argument `type` the format of the model file is selected. The remaining two arguments are of type `logical` and the user can indicate whether the first row should be ignored by setting `ignore_first_row` to `FALSE` (the default) and whether additional solver output should be printed while the model file is parsed with the `verbose` argument (the default is set to `FALSE`). The function returns a `list` object of informal class `MP_data_from_file` for which a `print()` method is available. The relevant elements of this kind of object could then be used in the call of `Rglpk_solve_LP()`, the cornerstone function of the package. The arguments of this function are:

- `obj` for the coefficients of the objective function;
- `mat` for the constraint coefficients (either a vector or a matrix object);
- `dir` for indicating the kind of inequality/equality constraint specified as a character vector (possible directions are "`<`", "`<=`", "`>`", "`>=`", or "`==`");
- `rhs` for the right-hand-side values of the constraints;
- `types` for determining whether the variables of the objective function can take continuous ("`C`"), integer ("`I`"), or binary ("`B`") values (the default setting is that all variables are defined as continuous variables);
- `max` for indicating whether the objective is to be maximized or minimized (the default value is `FALSE`, which means that the objective will be minimized);
- `bounds` for defining allowable ranges of the variables;
- `verbose` which serves as a logical switch for controlling the printing of additional output from the solver.

The function returns a list object with the elements "solution", "objval", and "status". The first list element contains the solution found for the variables, the second element is the value of the objective function, and the last element is a status flag indicating whether an optimal solution was determined (0) or not.

The package **linprog**

For solving linear programs, in the package **linprog** (see Henningsen 2012) the function `lp()` contained in the package **IpSolve** can be employed; otherwise, the linear program is solved by the simplex algorithm which is directly implemented as R code. The package **IpSolve** is discussed below. The package **linprog** is hosted on CRAN and contained in the “Optimization” Task View. It is shipped with a NAMESPACE file, and S3 classes and methods are employed.

Linear programs in the MPS format can be brought into memory with the function `readMps()`. In contrast to the package **Rglpk** in which the GLPK file reader is employed, the parsing of the MPS file is accomplished entirely in R. In addition to the argument `file`, which points to the MPS file to be read, the user can specify whether the linear program should immediately be solved by setting the logical switch `solve` to TRUE (the default value is FALSE), and whether the objective function should be maximized or minimized by the argument `maximum` (the default value is FALSE, which means that the objective function will be minimized). The function `readMps()` will return a list of named objects with elements containing the relevant information such that these can be employed in a call to `solveLP()`. These are `name` for carrying the name of the problem and the elements `cvec`, `bvec`, and `Amat` which refer to coefficients of the objective function, the right-hand-side variables, and the left-hand-side constraint matrix, respectively. In the case of direct optimization, where `solve=TRUE` has been used, the function will return an additional list element, `res`, which is an object of informal class `solveLP`.

The user can specify the direction of the inequality constraints by means of the argument `const.dir`. The default value is to treat all constraints as less-than-or-equal-to constraints. At the time of writing, equality constraints are not implemented. The next group of arguments are intended for controlling the algorithm. The maximum number of iterations can be set with the argument `maxiter`, a threshold value for floating point numbers to be considered as equal is determined by the argument `zero`, and the convergence tolerances for the primal and dual specification of the linear program can be specified with the arguments `tol` and `dualtol`, respectively. Whether the problem is solved by means of the simplex algorithm or by calling `lp()` is controlled by the logical switch `lpSolve`. The default is `lpSolve=FALSE`. With the last two arguments the user can also specify whether the linear program in its dual form is solved (the default is FALSE) and/or whether any intermediate results are returned to the console (and, if so, in how much detail) by assigning an integer value between 0 and 4 to the argument `verbose`. The function returns a named list object of S3 class `solveLP` for which `print()` and `summary()` methods are available.

Finally, the function `writeMps()` enables the user to save a linear program in MPS format. In addition to the arguments `cvec`, `bvec`, and `Amat` for specifying the coefficient, a name argument is used to assign a name to the linear program (the default is set to "LP"). The main purpose of this function is to cross-compare results between different solvers, given that these can digest MPS files.

The packages `lpSolve` and `lpSolveAPI`

Akin to GLPK, the software `lp_solve` is in the public domain and its purpose is to provide routines for solving linear (integer) programs. This suite supports not only LP and MILP, but also binary, semi-continuous, and special ordered set problems. These routines are collected into a library that can be accessed through an API written in ANSI C. The project is hosted on the Sourceforge platform, and more information is available on the project's internet site at <http://lpsolve.sourceforge.net/>.

Two R packages are available that give users access to `lp_solve`: **lpSolve** (see Berkelaar 2015) and **lpSolveAPI** (see Konis 2016). They differ from each other with respect to the level of interfacing with `lp_solve`. In the former package a rather high-level interface is implemented in the form of a driver routine, whereas the latter package provides low-level access by means of the C API functions.

The package **lpSolve** is hosted on CRAN and is considered to be a core package in the "Optimization" Task View. S3 classes and methods have been employed, and the package ships with a NAMESPACE file with export directives for the functions and methods defined. The cornerstone function for solving LP or MILP is `lp()`. The function returns a named list object of informal class `lp`, for which a `print()` method has been defined. The elements consist of the elements that have been passed to the solver when `lp()` has been called—the direction of the optimization, the coefficients of the objective function, the right-hand-side matrix, and the left-hand-side constraints. In addition, objects of class `lp` contain as list elements the value of the objective function and the associated solution vector. Furthermore, the number of (integer) variables, their positions in the vector of variables, and the number of constraints are returned. Finally, the list element `status` indicates successful completion of the algorithm if it takes the value 0; if no feasible solution is found, a 2 is returned instead. Incidentally, the package **limSolve** (see Soetaert et al. 2009; Van den Meersche et al. 2009) provides the wrapper function `linp()`, in which the constraints are separated according to their direction. The remaining functions `lp.assign()` and `lp.transport()` contained in the package **lpSolve** are intended for specific types of linear program, namely assignment and transportation problems, respectively.

Hosted on CRAN and R-Forge, the package **lpSolveAPI** is contained in the CRAN "Optimization" Task View. Development versions of the package can be obtained from <http://lpsolve.r-forge.r-project.org>. A vignette is available in which the definition, handling, solving, and solution retrieval of a linear program model object (LPMO) are elucidated by a worked-through example linear program. Therefore, a typical workflow would consist of first defining a linear program with

respect to its dimension by the function `make.lp()` and then filling the LPMO object (an external pointer to a C structure) with data and specifying the type of variables and the directions of the constraints. When this is accomplished, the model can be solved by a call to the function `solve()`. The results can then be retrieved by means of access methods. Methods for LPMO printing and plotting (two-dimensional only) are available, as well as R functions for reading a linear program specification (`read.lp()`) that has been expressed in either LP (a macro language pertinent to `lp_solve`) or fixed/free MPS format. Finally, a linear program specification can be written to a file in the same formats with the function `write.lp()`.

The package **Rsymphony**

The package **Rsymphony** provides a high-level interface to the MILP solver **SYMPHONY** (see Harter et al. 2016). The package is hosted on CRAN and R-Forge. It is contained in the “Optimization” Task View and has the status of a core package. The solver SYMPHONY is part of the COIN-OR project (see <http://www.coin-or.org/SYMPHONY/index.htm> for more information). The solver implements the branch, cut, and price approach for solving MILP problems. As such the approach taken is more powerful than using the branch and bound method for solving linear programs alone. The solver itself is written in the C language.

The package exports the function `Rsymphony_solve_LP()` for solving MILP problems. The target vector of the objective function has to be provided as argument `obj` to this function. The coefficient matrix, the relation operator between this and the right-hand-side variables, and the right-hand-side variables themselves have to be specified as arguments `mat`, `dir`, and `rhs`, respectively. The following relation operators can be used as characters for `dir`: "`<`", "`<`", "`<=`", "`>`", "`>=`", "`==`", and "`!=`". Bounds for the target variables can be specified as a `list` object with elements `lower` and `upper` for the respective lower and upper bounds. The type of target variables is determined by the argument `type`. Here, the user can specify whether a target variable is treated as continuous ("C"), integer ("I"), or binary ("B"). If this argument is left `NULL` then all targets are taken as continuous variables. Finally, the logical argument `max` controls whether the objective function is maximized or minimized (the default). The function `Rsymphony_solve_LP()` returns a `list` object with elements `solution`, `objval`, and `status` for the solved target variables, the value of the objective function, and an integer that indicates the status of the solver’s execution, respectively.

The package is further endowed with functions that convert the coefficient matrix into the sparse column major order used by SYMPHONY as well as for preparing and checking the specified bounds. These functions are not exported via directives in the NAMESPACE file of the package.

12.5.4 The package **PerformanceAnalytics**

The focus of the package **PerformanceAnalytics** is on the provision of functions and methods for assessing the performance of a given portfolio allocation and the

associated risks (see Peterson and Carl 2014). The package is contained in the CRAN “Finance” Task View and is considered to be a core package. The latest development versions can be obtained from both CRAN and R-Forge. The user is guided by means of six vignettes through the utilization and application of the cornerstone functions. A feature of the package is the support of the most commonly encountered time series classes—time series objects of classes `xts`, `timeSeries`, and `zoo` are supported.

Due to the large number of functions contained in the package, the following presentation concentrates on the facilities directly related to the topics discussed in this chapter as well as in Chapter 4.

The risk measures VaR and CVaR/ES are implemented as functions `VaR()` and `ES()`, respectively. Both functions can be applied to either a univariate stream of returns or an object that contains the returns of the portfolio constituents (function argument `R`). If the latter is chosen, a corresponding weight vector has to be provided for the argument `weight`. The confidence level is determined by the argument `p`, with default value 0.95. With respect to the estimation method of these risk measures, the user can choose between a Cornish–Fisher modified type (`method = "modified"`), a calculation based on the assumption of normally distributed returns (`method = "gaussian"`), the use of historic returns (`method = "historical"`), or a kernel density estimate (`method = "kernel"`). A feature of these functions is the option to “clean” the returns by means of the argument `clean`. Two ways of making the results robust are implemented: a return adjustment according to the first-order autocorrelation coefficient (`clean="geltner"`), or a scheme proposed in Boudt et al. (2008) by which the returns are trimmed while preserving the tail characteristics (`clean="boudt"`). If a multivariate return series object is passed to the function, the user can determine whether the VaR or ES is computed as a single figure for the implied portfolio returns for a given weight vector or whether in addition the component and/or marginal contributions of the assets to the overall risk figures are also computed by means of the `portfolio_method` argument. Finally, it should be noted that instead of passing an object with the assets’ returns to the function `ES()` or `VaR()`, the user can directly provide moment estimates by means of the arguments `mu`, `sigma`, `m3`, and `m4` for the mean(s), the variance-covariance matrix, the (co-)skewness, and the (co-)kurtosis, respectively.

The functions that relate to the draw-down concept are `CDD()`, `chart.Drawdown()`, `Drawdowns()`, `findDrawdowns()`, `max.Drawdown()`, `sort.Drawdowns()`, and `table.Drawdowns()`, in alphabetical order. The `CDD()` function is an implementation of the conditional draw-down at risk concept. In addition to the returns object, `R`, the user can specify the confidence level by the argument `p` and provide a weight vector, in the case of a multivariate return object, by means of the argument `weights`. In addition, one has to specify whether the returns are discrete or continuous by the logical argument `geometric`. The draw-down measure can be inverted to a positive scale by employing the logical argument `invert`. Incidentally, this last argument is also part of the functions’ closures in `VaR()` and `ES()`. The implied draw-downs of a return stream can be visualized with the function `chart.Drawdown()`. For a multivariate return object, the draw-down curves are plotted on the same panel and the user can specify

where a legend is placed with the argument `legend.loc` and which colors are used by means of `colorset`. The functions `Drawdowns()` and `findDrawdowns()` can be utilized for inspecting the draw-downs in more detail. Both functions return a `list` object with elements the size of draw-down, the starting, trough, and ending periods, the length of these episodes in total, and the length from peak to trough and from trough until the next peak. The function `Drawdowns()` returns the draw-downs expressed as a percentage from the equity levels and is employed in the function `chart.Drawdown()`. The user must specify whether the return data is pertinent to discrete or continuous returns. Incidentally, the function `Drawdowns()` has become a non-exported function compared to previous releases. Hence, in order to invoke this function at the `R` prompt, the user must employ the triple-colon operator, that is, `PerformanceAnalytics::Drawdowns()`. The maximum draw-down of a return stream is returned by the function `maxDrawdown()`. Again, the user has to specify by means of the argument `geometric` the kind of return data. If the function is employed for returning the maximum draw-down of a portfolio, the weights have to be supplied by `weights`. The draw-downs can be sorted by their sizes with the function `sortDrawdowns()`. The function has one argument `runs`, which is the `list` object returned by the function `findDrawdowns()`. Finally, the historic draw-downs can be returned in a tabular format by the function `table.Drawdowns()`. In addition to the returns (provided through the argument `R`), the user can specify how many draw-downs to include in the tabular structure by the argument `top` and the number of decimal places for the returns via the argument `digits` (the default is to print the draw-downs to four decimal places).

12.6 Empirical applications

12.6.1 Minimum-CVaR versus minimum-variance portfolios

In this first application, a comparison between a global minimum-CVaR and a global minimum-variance portfolio is conducted in the form of a back-test for an index-based equity portfolio. The monthly returns of the following stock indexes are considered: the S&P 500, the Nikkei 225, the FTSE 100, the French CAC 40, the German DAX, and the Hang Seng. The sample period of the index data runs from 31 July 1991 to 30 June 2011 and comprises a total of 240 observations. The `R` code for conducting the back-test is shown in Listings 12.1 and 12.2.

First, the packages are brought into memory. The data set `StockIndex` is contained in the package **FRAPO** and the routines of the package **fPortfolio** for optimizing the minimum-CVaR and minimum-variance portfolios are utilized. In line 4 of Listing 12.1, the data set is loaded into the workspace and in line 5 the discrete percentage returns are computed.

The commands in which the portfolio specifications are defined are given on lines 8–14. First, a default specification is assigned to the object `pspec` which can be used directly for the global minimum-variance portfolios (object `gmv`). In order to define a CVaR portfolio (object `cvar`) some amendments to the default portfolio

R code 12.1 Minimum-CVaR versus minimum-variance portfolio: back-test.

```

library(FRAPO)                                     1
library(fPortfolio)                                2
## Retrieving data and calculating returns        3
data(StockIndex)                                   4
StockReturn <- na.omit(timeSeries(returnseries(StockIndex,           5
                                              method = "discrete"),           6
                                              charvec = rownames(StockIndex)))  7
                                              8
## Specifying portfolio                         9
pspec <- portfolioSpec()                         10
gmv <- pspec                                     11
cvar <- pspec                                     12
setType(cvar) <- "CVaR"                         13
setAlpha(cvar) <- 0.1                            14
setSolver(cvar) <- "solveRglpk.CVAR"            15
## Conducting back-test                         16
end <- time(StockReturn)[60:239]                 17
from <- time(StockReturn)[1:length(end)]          18
wGMV <- matrix(NA, ncol = ncol(StockReturn), nrow = length(end)) 19
wCVAR <- wGMV                                     20
for(i in 1:length(end)){                         21
  series <- window(StockReturn, start = from[i], end = end[i]) 22
  gmvpf <- minvariancePortfolio(data = series, spec = gmv,      23
                                  constraints = "LongOnly") 24
  wGMV[i, ] <- c(getWeights(gmvpf))              25
  cvarpf <- minriskPortfolio(data = series, spec = cvar,        26
                             constraints = "LongOnly") 27
  wCVAR[i, ] <- c(getWeights(cvarpf))            28
}
                                         29

```

definition have to be made. The type of the portfolio has to be set to "CVaR" with the `setType()` function. A confidence level has to be assigned. Here, a 10% level is assigned to the portfolio specification by the `setAlpha()` function. Incidentally, and as stated previously, the confidence level must be specified in return and not in loss space, hence a value of 0.1 coincides with the 90% confidence level of the loss distribution. Finally, the solver to be used is GLPK, hence the character string "solveRglpk.CVAR" is assigned to `cvar` by the `setSolver()` function. That is all it takes to define the two approaches for optimizing the portfolios in the following.

The back-test itself is carried out in lines 15–28. The **portfolios will be optimized with respect to subsamples of size 60, that is, a prior data history of 5 years** is employed. **The start and end points for the sliding data windows are assigned to the objects `from` and `end`.** Next, two matrices are defined which are designated to store the portfolio weight vectors as rows. Hence, sufficient memory is allotted which will not hamper the speed of execution in the ensuing for loop. **Executing the back-test in**

R code 12.2 Plotting wealth trajectory.

```

## Compute Portfolio values (subsequent returns)          1
wGMVLI <- lag(timeSeries(wGMV, charvec = end), k = 1)  2
colnames(wGMVLI) <- colnames(StockReturn)             3
wCVARLI <- lag(timeSeries(wCVAR, charvec = end), k = 1) 4
colnames(wCVARLI) <- colnames(StockReturn)             5
## Return factors and portfolio values               6
GMVRetFac <- 1 + rowSums(wGMVLI *                   7
                         StockReturn[time(wGMVLI), ]) / 100 8
GMVRetFac[1] <- 100                                9
GMVPort <- timeSeries(cumprod(GMVRetFac),           10
                         charvec = names(GMVRetFac))        11
CVARRetFac <- 1 + rowSums(wCVARLI *                  12
                         StockReturn[time(wCVARLI), ]) / 100 13
CVARRetFac[1] <- 100                                14
CVARPort <- timeSeries(cumprod(CVARRetFac),          15
                         charvec = names(CVARRetFac))        16
## Plotting of portfolio values                     17
ylims <- range(cbind(GMVPort, CVARPort))           18
plot(GMVPort, ylim = ylims, xlab = "",             19
      ylab = "Portfolio Value (Index)")           20
lines(CVARPort, col = "blue")                      21
legend("topleft",                                22
       legend = c("Minimum–Variance", "Minimum–CVaR"),
       col = c("black", "blue"), lty = 1)           23
## Relative performance                         24
RelOutPerf <- (CVARPort - GMVPort) / GMVPort * 100 25
plot(RelOutPerf, type = "h", col = "blue", xlab = "", 26
      ylab = "Percent",                         27
      main = "Relative Out–Performance Min–CVaR vs. 28
              Min–Variance")                      29
abline(h = 0, col = "grey")                      30
  
```

a `for` loop might be considered as not R-like and one could have put the back-test in terms of the `fapply()` function which is contained in the package `timeSeries`, for instance, or one could have created an object for the returns with class attribute `zoo` in the first place and utilized the `rollapply()` function of the package `zoo` instead. However, the aim of putting the back-test in a `for` loop is threefold. First, it shows that the prejudice against `for` loops is unjustified (if they are properly set up). Second, by looking at the body of the `for` loop, the user sees exactly what is happening. Finally, the definition of a function that must be called from either `fapply()` or `rollapply()` is avoided. Within the `for` loop, the relevant data window from `StockReturn` is assigned to the object `series`. This object, together with the specification `gmv` and the long-only constraint, is then used next in the determination of the solution for global minimum-variance portfolios, which is accomplished by the `minvariancePortfolio()` function. The corresponding weight vector

is retrieved by the access function `getWeights()` and stored in the i th row of the matrix object `wGMV`. The same is done in lines 25–27 for the global minimum-CVaR portfolios. Here, the function `minriskPortfolio()` is called, and as arguments the series object and the long-only constraint together with the portfolio specification object `cvar` are employed. The optimal portfolio weight vector is then assigned into the i th row of the object `wCVAR`.

In order to compute the portfolios' equity lines, the inner products between the lagged portfolio weights and the subsequent returns have to be computed for each of the two portfolio approaches. This is accomplished in the first part of Listing 12.2.

From these portfolio return streams, the return factors can swiftly be computed by means of the `cumprod()` function, given that the returns have previously been calculated as discrete percentage changes and the initial wealth position has been assumed to be equal to 100. The portfolio values for the back-test period are assigned to the objects `GMVPort` and `CVARPort` for the global minimum-variance portfolios and the global minimum-CVaR portfolios, respectively.

These two equity curves are then displayed by means of lines 17–24 (see Figure 12.5) and the relative performance with respect to the equity line of the GMV portfolio as a bar-plot in the final lines of the listing (see Figure 12.6).

For this back-test design, the CVaR portfolio approach outperforms the GMV portfolio solutions. However, the sharp portfolio draw-downs witnessed during the financial market crisis during 2007 and 2008 could not be averted by either of the two approaches.

12.6.2 Draw-down constrained portfolios

In this example, the solution of a global minimum-variance portfolio is again utilized as a benchmark allocation for long-only investments. The characteristics of this

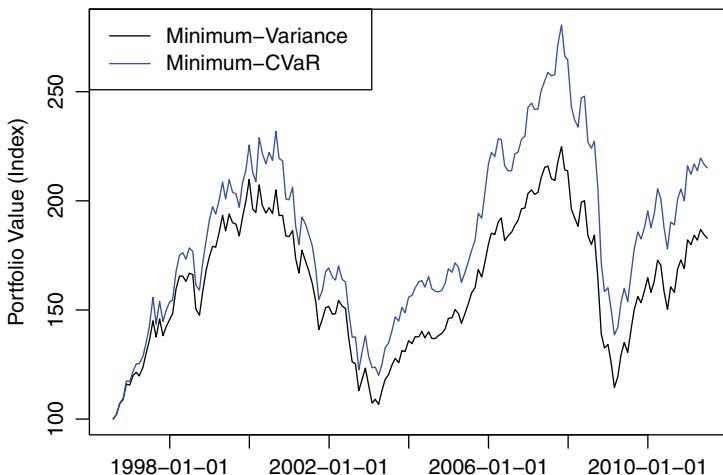


Figure 12.5 Trajectory of minimum-CVaR and minimum-variance portfolio values.

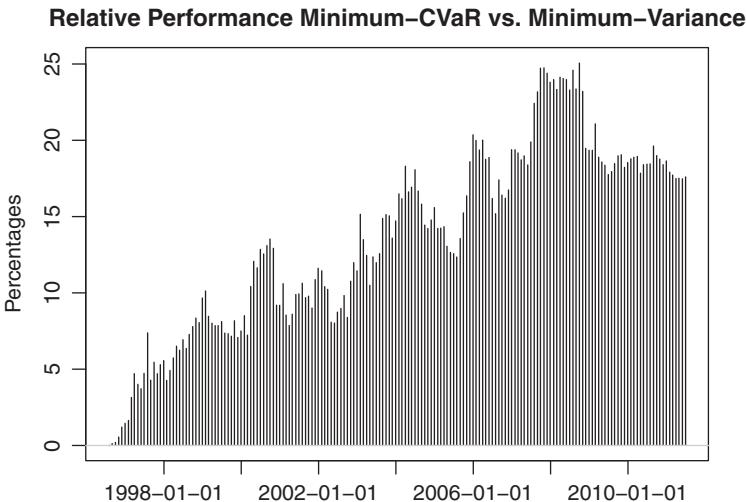


Figure 12.6 *Relative performance of minimum–CVaR and minimum–variance portfolios.*

solution are compared with the allocations of portfolios that are restricted by their maximum, average, and conditional draw-downs and with a minimum conditional draw-down at risk portfolio. The spectrum of assets covers the major equity and bond markets as well as investment in gold. The R code is presented in Listing 12.3.

First, the relevant R packages are brought into memory. Then the `MultiAsset` data set is loaded, which is part of the `FRAPO` package. The data set covers the month's-end prices of a number of equity indexes (S&P 500, Russell 3000, DAX (XETRA), FTSE 100, Nikkei 225 and iShares MSCI Emerging Markets) and fixed income indexes (Dow Jones CBOT Treasury, German REX Performance, and United Kingdom gilts—all maturities) as well as the price of the SPDR Gold Shares exchange traded fund from 30 November 2004 to 30 November 2011. In lines 5–8 the discrete returns are computed and converted to a `timeSeries` object. Commencing in line 11 the solution of the global minimum-variance portfolio is determined and the historic draw-downs are computed with the function `Drawdown()` contained in the package `PerformanceAnalytics`. Plots of these are displayed through the commands in lines 17–21. The time series plot is created from scratch in order to make its appearance the same as the draw-down plots for the ensuing portfolio plots. The result is provided in Figure 12.7. Incidentally, one could have used the function `chart.Drawdown()` instead, as mentioned in Section 12.5.4.

In the following lines the various draw-down portfolio solutions are computed, with the maximum draw-down of the GMV portfolio serving as anchor value. The conditional draw-down at risk portfolios are computed for a confidence level of 95%. Next, the trajectory of the four historic draw-down series is depicted in the form of a (2×2) plot where for a better cross-comparison the ordinates share the same scale except for the average draw-down constrained portfolio (see Figure 12.8).

R code 12.3 Comparison of draw-down and GMV portfolios.

```

library(fPortfolio)                                     1
library(FRAPO)                                       2
library(PerformanceAnalytics)                         3
data(MultiAsset)                                     4
## Return calculation                                5
Rets <- returnseries(MultiAsset, method = "discrete", 6
                      percentage = FALSE, trim = TRUE) 7
Rets <- timeSeries(Rets, charvec = rownames(Rets)) 8
## Benchmark portfolio: GMV                         9
gmvspec <- portfolioSpec()                         10
GMV <- minvariancePortfolio(data = Rets, spec = gmvspec, 11
                             constraints = "LongOnly") 12
GMVret <- timeSeries(Rets %*% getWeights(GMV), 13
                      charvec = time(Rets)) 14
GMVDD <- PerformanceAnalytics:::Drawdowns(GMVret) 15
## Plot of draw downs for GMV                      16
ylims <- c(-6, 0)                                    17
plot(GMVDD * 100, xlab = "", ylab = "Draw Downs (percentage)", 18
     main = "Draw Downs of Global Minimum Variance", 19
     ylim = ylims)                                    20
abline(h = 0, col = "grey")                          21
grid()                                              22
## Max DD of GMV                                    23
GMVMaxDD <- max(-1.0 * GMVDD)                      24
## Draw Down Portfolios                            25
MaxDD <- PMaxDD(MultiAsset, MaxDD = GMVMaxDD)      26
AveDD <- PAveDD(MultiAsset, AveDD = GMVMaxDD)      27
CDaR95 <- PCDaR(MultiAsset, alpha = 0.95, bound = GMVMaxDD) 28
CDaRMin95 <- PCDaR(MultiAsset, alpha = 0.95)        29
## Plot of draw downs                            30
oldpar <- par(no.readonly = TRUE)                   31
par(mfrow = c(2, 2))                                32
plot(AveDD, main = "(a) AveDD")                     33
plot(MaxDD, ylim = ylims, main = "(b) MaxDD")       34
plot(CDaR95, ylim = ylims, main = "(c) CDaR")        35
plot(CDaRMin95, ylim = ylims, main = "(d) Minimum CDaR") 36
par(oldpar)                                         37

```

As already hinted in Section 12.4, the solution of the average draw-down constrained portfolio can imply the occurrence of large draw-downs. In this case the maximum draw-down of the average draw-down constrained solution is greater by roughly a factor of four than the maximum historic draw-down of the minimum conditional draw-down at risk portfolio. By cross-comparing the draw-downs with those implied by the GMV solution, the latter seems to fare reasonable well, at first glance.

Draw-downs of global minimum variance

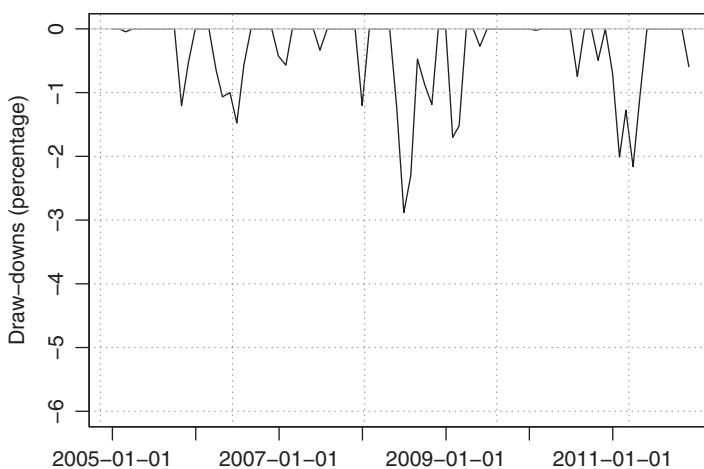


Figure 12.7 Draw-downs of GMV portfolio.

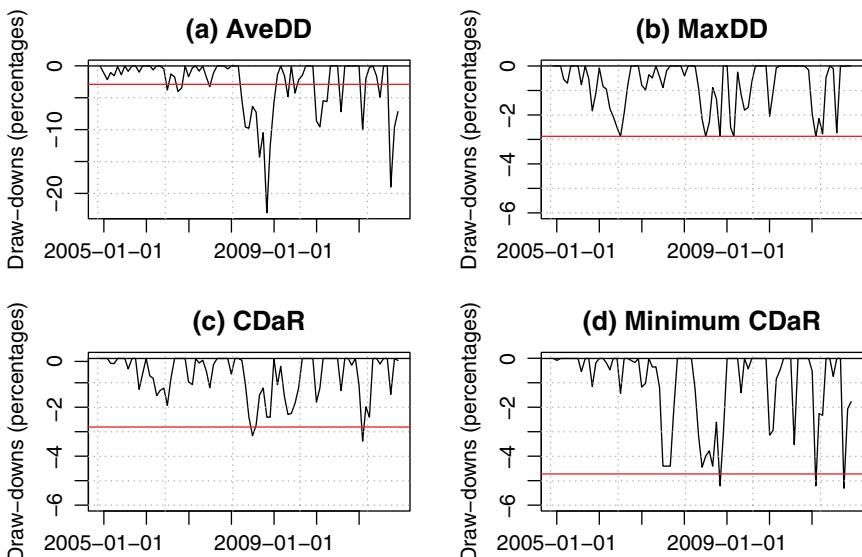


Figure 12.8 Comparison of draw-downs.

The characteristics of the five portfolio solutions are next analyzed with respect to the different allocations and their associated risk contributions and diversification ratios. The calculation of these statistics is provided in Listing 12.4.

In line 2 of this listing, the names for the five portfolio types are defined; these will be used as column names. Next, the portfolio weights are extracted from the objects

R code 12.4 Analysis of portfolio solutions.

```

## Portfolio Names
Pnames <- c("GMV", "MaxDD", "AveDD", "CDaR95", "CDaRMin95") 1
## Portfolio allocations
WeightMatrix <- cbind(getWeights(GMV), 2
                      Weights(MaxDD), 3
                      Weights(AveDD), 4
                      Weights(CDaR95), 5
                      Weights(CDaRMin95)) 6
colnames(WeightMatrix) <- Pnames 7
## Expected Shortfall and components
tmp <- apply(WeightMatrix, 2, function(x) ES(Rets, weights = x, 8
                                         method = "gaussian", 9
                                         portfolio_method = "component")) 10
## ES 95%
PES <- unlist(lapply(tmp, function(x) x[[1]])) * 100 11
## Marginal Contributions to ES
PMES <- matrix(unlist(lapply(tmp, function(x) x[[3]])), 12
                nrow = ncol(Rets)) * 100 13
rownames(PMES) <- colnames(Rets) 14
colnames(PMES) <- Pnames 15
## Marginal Contributions to StdDev
V <- cov(Rets) 16
PMRC <- apply(WeightMatrix, 2, mrc, Sigma = V) 17
rownames(PMRC) <- colnames(Rets) 18
## Diversification ratio
PDR <- apply(WeightMatrix, 2, dr, Sigma = V) 19

```

created earlier and assigned to the matrix object `WeightMatrix`. This object can then be utilized to compute the portfolios' ES at the 95% level and the percentage contributions of the underlying assets with the function `ES()`. The intermediate result in the form of a `list` object is assigned to `tmp`. The relevant items can then be extracted easily by employing `lapply()` and using the `unlist()` function to assemble the results as a matrix. The marginal contributions to the portfolio risk are computed in line 20 onwards with the function `mrc()` which is included in a call to `apply()`. In a similar manner the diversification ratios for the five portfolios are computed using the function `dr()` in line 25. The results are summarized in Table 12.1.

The GMV portfolio is characterized by a high share allotted to German government bonds, due to the low volatility of this asset. Exposure to equities is roughly 10% and investments in UK gilts and gold amount to less than 1% in each case. According to the GMV solution, the capital is split between six of the ten possible assets. Overall, the empirical stylized fact of highly concentrated solutions for GMV portfolios is vindicated. This becomes even more apparent when the marginal contributions to the 95% ES and the overall portfolio risk are taken into account. However, when

Table 12.1 Comparison of portfolio allocations and characteristics.

Analysis	GMV	MaxDD	AveDD	CDaR95	CDaRMin95
S&P 500					
Weight	4.89	0.00	0.00	0.00	0.00
MES	5.58	0.00	0.00	0.00	0.00
MPR	4.89	0.00	0.00	0.00	0.00
Russell 3000					
Weight	0.00	3.00	0.00	6.40	0.00
MES	0.00	-0.93	0.00	-3.68	0.00
MPR	0.00	-0.44	0.00	-2.10	0.00
DAX (XETRA)					
Weight	4.34	3.62	0.00	0.00	0.00
MES	3.46	-3.51	0.00	0.00	0.00
MPR	4.34	-1.83	0.00	0.00	0.00
FTSE 100					
Weight	0.00	0.00	0.00	0.00	0.00
MES	0.00	0.00	0.00	0.00	0.00
MPR	0.00	0.00	0.00	0.00	0.00
Nikkei 225					
Weight	1.73	0.91	0.00	0.00	0.00
MES	2.37	-0.50	0.00	0.00	0.00
MPR	1.73	-0.45	0.00	0.00	0.00
MSCI EM					
Weight	0.00	0.00	0.00	0.00	0.00
MES	0.00	0.00	0.00	0.00	0.00
MPR	0.00	0.00	0.00	0.00	0.00
CBOT Treasury					
Weight	0.00	48.03	0.00	16.62	0.00
MES	0.00	75.01	0.00	25.73	0.00
MPR	0.00	69.75	0.00	23.39	0.00
German REX					
Weight	87.87	42.83	55.07	72.49	85.73
MES	87.47	28.09	3.73	67.97	54.50
MPR	87.87	30.21	6.93	66.94	54.97
UK Gilts					
Weight	0.96	0.00	0.00	0.00	0.00
MES	1.12	0.00	0.00	0.00	0.00
MPR	0.96	0.00	0.00	0.00	0.00
Gold					
Weight	0.21	1.62	44.93	4.49	14.27
MES	-0.01	1.83	96.27	9.97	45.50
MPR	0.21	2.76	93.07	11.76	45.03
Overall					
ES 95%	1.31	1.82	4.36	1.53	1.98
DivRatio	1.86	1.63	1.17	1.67	1.34

one compares the diversification ratio of this portfolio with the values taken by the draw-down portfolios, the conclusion would be that the GMV yields the most favorable asset allocation in terms of the degree of diversification. This artifact can again be primarily attributed to the low volatility of the REX, which is mirrored by a low figure for the ES.

The number of assets invested in is the same for the constrained maximum draw-down portfolio. However, this solution differs in two major points with respect to the asset allocation: first, the allotment to government bonds is now roughly equally split between US Treasuries and German Bunds; and second, the equity exposure is characterized by negative marginal contributions to the ES and the overall portfolio risk. Hence, from a risk perspective this outcome is more akin to a general understanding of risk diversification, even though the diversification ratio is less than that of the GMV and the ES is half a percentage point greater.

The solutions for the average draw-down and minimum conditional draw-down at risk constrained portfolios imply an asset allocation in the REX and gold. However, these two solutions are quite distinct from each other. Whereas for the average draw-down portfolio the capital is shared almost equally between the two assets, resulting in a high marginal contribution of gold to the portfolio's risk, for the minimum conditional draw-down portfolio the pattern is reversed. Here, the lion's share of capital would be invested in German Bunds and the implied marginal risk contributions are almost equal.

The asset allocation for the constrained conditional draw-down at risk portfolio yields an intermediate solution between the maximum draw-down constrained and the other draw-down portfolios discussed above. Its equity exposure is limited to the Russell 3000 only, which contributes negatively to the ES and the portfolio's risk, on the margin. It is further characterized by bond exposure to US Treasuries and German Bunds, where the latter constitute almost three quarters of the capital invested. Compared to the GMV allocation, the risk contributions are more evenly spread between only four investments. The ES and the diversification ratio is pretty close to the corresponding figures for the GMV portfolio.

12.6.3 Back-test comparison for stock portfolio

In this example, the results of a conditional draw-down at risk strategy are compared to the outcome of a minimum-variance allocation. The portfolio solutions consist of the constituents of the Euro Stoxx 50 index. The purpose of this example is to show that a global minimum-variance allocation is not a shelter from losses per se. The first section of the code is provided in Listing 12.5.

First, the necessary packages are brought into the workspace. The package **FRAPO** will be employed for the data set used (EuroStoxx50) and to conduct the CDaR optimizations. The data set is also used in Cesarone et al. (2011). It is included in the package as a data frame with 265 weekly observations of 48 members of the index. The sample starts on 3 March 2003 and ends on 24 March 2008. The solutions of the global minimum-variance portfolios are determined by means of the functions contained in the package **fPortfolio**, and the analysis of the simulations is carried out

R code 12.5 Back-test: GMV versus CDaR portfolio optimization.

```

library(FRAPO)                                     1
library(fPortfolio)                                2
library(PerformanceAnalytics)                     3
## Loading of data set                           4
data(EuroStoxx50)                                 5
## Creating timeSeries of prices and returns     6
pr <- timeSeries(EuroStoxx50, charvec = rownames(EuroStoxx50)) 7
NAssets <- ncol(pr)                                8
RDP <- na.omit((pr / lag(pr, k = 1) - 1) * 100) 9
## Backtest of GMV vs. CDaR                      10
## Start and end dates                           11
to <- time(RDP)[208:nrow(RDP)]                   12
from <- rep(start(RDP), length(to))              13
## Portfolio specifications                     14
## CDaR portfolio                                15
DDbound <- 0.10                                    16
DDalpha <- 0.95                                    17
## GMV portfolio                                 18
mvspec <- portfolioSpec()                         19
BoxC <- c("minsumW[1:NAssets] = 0.0" , "maxsumW[1:NAssets] = 1.0") 20
## Initialising weight matrices                 21
wMV <- wCD <- matrix(NA, ncol = ncol(RDP), nrow = length(to)) 22
## Conducting backtest                         23
for(i in 1:length(to)){                         24
  series <- window(RDP, start = from[i], end = to[i]) 25
  prices <- window(pr, start = from[i], end = to[i]) 26
  mv <- minvariancePortfolio(data = series,          27
                               spec = mvspec,          28
                               constraints = BoxC) 29
  cd <- PCDaR(prices, alpha = DDalpha, bound = DDbound, 30
               softBudget = TRUE) 31
  wMV[i, ] <- c(getWeights(mv)) 32
  wCD[i, ] <- Weights(cd) 33
}
## Lagging optimal weights and sub-sample of returns 35
wMV <- rbind(rep(NA, ncol(RDP)), wMV[-nrow(wMV), ]) 36
wMVL1 <- timeSeries(wMV, charvec = to) 37
colnames(wMVL1) <- colnames(RDP) 38
wCD <- rbind(rep(NA, ncol(RDP)), wCD[-nrow(wCD), ]) 39
wCDL1 <- timeSeries(wCD, charvec = to) 40
colnames(wCDL1) <- colnames(RDP) 41
RDPback <- RDP[to,] 42
colnames(RDPback) <- colnames(RDP) 43

```

using the package **PerformanceAnalytics**. After the packages have been loaded, the data set is brought into memory and converted to a `timeSeries` object, and the discrete returns of the stock prices are assigned to the object `RDP`. The back-test is then carried out in the form of a recursive window with an initial length of 208 observations, which coincides with a span of four years. Lines 10–22 define the parameters and portfolio specifications employed. The `CDaR` portfolio will be optimized for a conditional draw-down as high as 10% for a confidence level of 95%. The `GMV` portfolio is defined with long-only and soft-budget constraints, such that the sum of weights is in the interval $[0, 1]$. This restriction is created as a two-element character vector which defines the lower and upper bounds. In the last line of this chunk

R code 12.6 Back-test: evaluation of results, part one.

```

## Portfolio equities of strategies
MVRetFac <- 1 + rowSums(wMVL1 * RDPback) / 100
MVRetFac[1] <- 100
MVPort <- timeSeries(cumprod(MVRetFac),
                      charvec = names(MVRetFac))
CDRetFac <- 1 + rowSums(wCDL1 * RDPback) / 100
CDRetFac[1] <- 100
CDPort <- timeSeries(cumprod(CDRetFac),
                      charvec = names(CDRetFac))

## Progression of wealth
ylims <- range(cbind(MVPort, CDPort))
plot(CDPort, main = "", ylim = ylims, ylab = "Index values",
     xlab = "")
lines(MVPort, col = "darkgrey")
legend("topleft", legend = c("CDaR", "GMV"),
       col = c("black", "darkgrey"),
       lty = 1)

## Portfolio returns
MVRet <- returns(MVPort, method = "discrete",
                  percentage = FALSE, trim = TRUE)
CDRet <- returns(CDPort, method = "discrete",
                  percentage = FALSE, trim = TRUE)

## Draw down table
table.Drawdowns(MVRet)
table.Drawdowns(CDRet)

## Plot of draw down curves
MVD <- 100 * PerformanceAnalytics:::Drawdowns(MVRet)
CDD <- 100 * PerformanceAnalytics:::Drawdowns(CDRet)
plot(CDD, main = "", ylab = "Percentages", xlab = "",
     ylim = c(min(c(MVD, CDD)), 0))
lines(MVD, col = "blue")
abline(h = 0, col = "lightgrey")
abline(h = -10, col = "lightgrey", lty = 2)
legend("bottomleft", legend = c("CDaR", "GMV"),
       col = c("black", "blue"), lty = 1)

```

of code the `matrix` objects `wMV` and `wCD` are created so that the necessary memory is reserved in advance. The simulation itself is then carried out with the ensuing `for` loop. Within this closure, the subsamples of the returns and prices are extracted and each data set is then passed into the calls of `minvariancePortfolio()` for the GMV and `PCDaR()` for the CDaR solutions, respectively. The optimal portfolio weights are then assigned to the i th row of the `matrix` objects `wMV` and `wCD` where the relevant methods for extracting the weight vectors have been utilized. Next, the weights are lagged by one period, such that the portfolio equities can be computed by direct multiplication with the returns for the back-test period.

In the first lines of Listing 12.6, the return factors for each strategy are assigned to the objects `MVRetFac` and `CDRetFac`, where an initial wealth of 100 monetary units has been assumed. The trajectory of the portfolios' equity is then given as the cumulative product of the return factors. These equity values are stored as objects `MVPort` for the GMV strategy and `CDPort` for the CDaR solution.

The R code for the evaluation of the two simulations is shown in the bottom part of Listing 12.6 and in Listing 12.7. First, the wealth trajectories are depicted in Figure 12.9.

The hump-shaped pattern during the financial crises is eye-catching and much more pronounced for the GMV than for the CDaR strategy. The final wealth of the GMV portfolio is slightly greater than that of the CDaR allocation. It is noteworthy that during the back-test period the GMV strategy yielded a fully invested solution, whereas according to the CDaR strategy not all capital was allocated to stocks at each rebalancing date. In particular, during the heyday of the financial crisis in 2007 and 2008 only between roughly 55% and 85% would have been allotted to stocks. This also partly explains the rather shallow wealth trajectory at the end of the back-test period. Accrued interest from the wealth not invested in stocks was not considered.

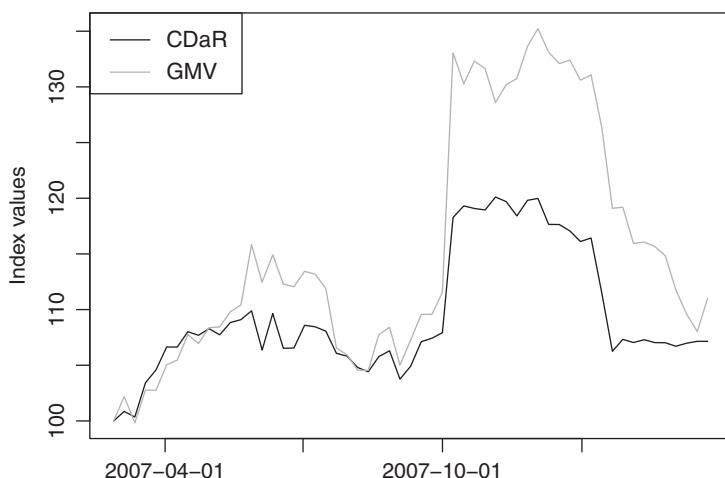


Figure 12.9 Comparison of wealth trajectories.

Table 12.2 Overview of draw-downs (positive, percentages).

Portfolio	From	Trough	To	Depth	→	↖	↗
GMV							
1	2007-12-10	2008-03-17		20.11	17	15	
2	2007-06-04	2007-08-13	2007-10-08	9.75	19	11	8
3	2007-10-15	2007-11-05	2007-11-26	3.34	7	4	3
4	2007-03-12	2007-03-12	2007-03-19	2.30	2	1	1
5	2007-04-23	2007-04-23	2007-04-30	0.76	2	1	1
CDaR							
1	2007-11-12	2008-01-21		11.53	21	11	
2	2007-06-04	2007-09-03	2007-10-08	5.58	19	14	5
3	2007-05-07	2007-05-07	2007-05-14	0.51	2	1	1
4	2007-03-12	2007-03-12	2007-03-19	0.49	2	1	1
5	2007-10-22	2007-10-29	2007-11-05	0.30	3	2	1

In lines 16–20 of Listing 12.6 the portfolio returns for each strategy are assigned to the objects `MVRet` and `CDRet`, respectively. These objects are utilized in the computation of characteristic portfolio statistics. The first of these is an overview of the five greatest draw-downs within each strategy (see the results in Table 12.2).

As is evident from this table, the portfolio draw-downs are all larger for the GMV strategy than for the CDaR allocation. The conditional draw-down level for the CDaR strategy has been violated in one instance, albeit marginally. The draw-downs are displayed in lines 24–33, now expressed as negative percentages (see Figure 12.10).

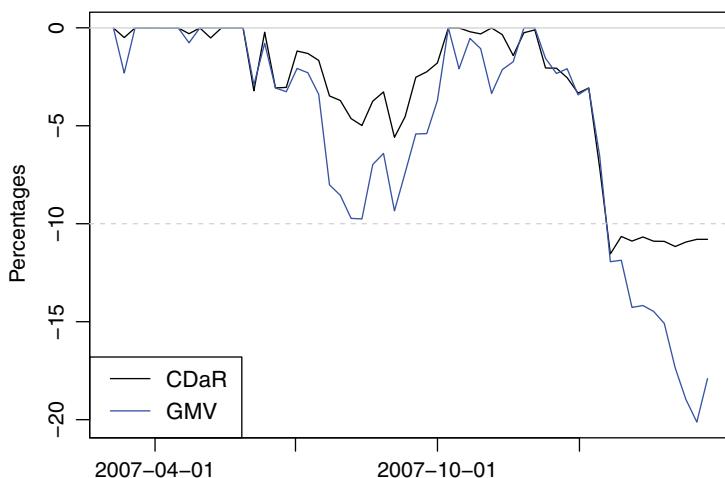


Figure 12.10 Comparison of draw-down trajectories.

R code 12.7 Back-test: evaluation of results, part two.

```

## Portfolio Statistics
## VaR
MVVAR <- -100 * VaR(MVRet, p = 0.95, method = "gaussian") 1
CDVAR <- -100 * VaR(CDRet, p = 0.95, method = "gaussian") 2
## ES
MVES <- -100 * ES(MVRet, p = 0.95, method = "gaussian") 3
CDES <- -100 * ES(CDRet, p = 0.95, method = "gaussian") 4
## Sharpe
MVSR <- SharpeRatio(MVRet) 5
CDSR <- SharpeRatio(CDRet) 6
## Annualised returns
MVRA <- Return.annualized(MVRet, scale = 52) 7
CDRA <- Return.annualized(CDRet, scale = 52) 8
## Draw downs
MVDD <- -100 * findDrawdowns(MVRet)$return 9
MVDD <- MVDD[MVDD!=0.0] 10
length(MVDD) 11
summary(MVDD) 12
CDDD <- -100 * findDrawdowns(CDRet)$return 13
CDDD <- CDDD[CDDD!=0.0] 14
length(CDDD) 15
summary(CDDD) 16

```

In Listing 12.7 some key portfolio performance statistics are computed. A summary of these is provided in Table 12.3.

In retrospect, the Gaussian tail risks VaR and ES for the GMV portfolio are both greater than the respective numbers for the CDaR strategy at the 95% level. Because the higher riskiness of the former strategy is roughly balanced against the more favorable annualized return for the GMV, the portfolios' Sharpe ratios are almost equal. The summary statistics for the portfolio draw-downs all indicate a less risky allocation of the CDaR strategy, except for the number of draw-downs. As stated at the beginning of this section, the purpose of this example is to remind the reader that a GMV optimization is not a panacea against large losses per se, and the often encountered corner solutions of such a strategy can turn into the opposite of what was initially intended.

12.6.4 Risk surface plots

In this chapter and the previous one alternative measures for defining portfolio diversification and portfolio risk in contrast to the classical mean-variance approach have been put forward. In this section a graphical tool—namely, risk surface plots—for inspecting all feasible asset allocations with respect to the implied diversification/risk characteristics will be presented. The functions for creating this kind of plot are contained in the package **fPortfolio**, which has been discussed in Sections 10.4.2

Table 12.3 Performance statistics.

Statistics	GMV	CDaR
Risk/return		
VaR 95%	5.16	3.05
ES 95%	6.54	3.86
Sharpe ratio	0.07	0.07
Return annualized %	10.20	6.62
Draw-down		
Count	6	7
Minimum	0.02	0.00
First quartile	1.14	0.30
Median	2.82	0.49
Mean	6.05	2.67
Third quartile	8.15	3.05
Maximum	20.11	11.53

and 12.5.1. In this example, a risk surface plot will be created in which the standard deviations of the marginal risk contributions are portrayed as a contour plot within the convex hull of all feasible, fully invested, long-only allocations of a multi-asset portfolio (see also Figure 5.1). In addition, a line for the most diversified allocations per portfolio standard deviation risk level is superimposed on this chart. That is, by means of risk surface plots, one is able to depict at least three different types of portfolio risk measures for the allocations that lie in the convex hull: (i) for a particular portfolio solution the portfolio standard deviation risk can be taken directly from the abscissa, (ii) the degree of risk concentration can be read from the contour level lines, and (iii) the most diversified portfolio solutions per risk level are indicated by the superimposed line.

In Listing 12.8 the required packages **FRAPO** and **fPortfolio** are brought into the workspace. Next, the data set `MultiAsset` is loaded and converted to a `timeSeries` object. The discrete percentage returns of the ten assets contained in this data set are computed next by calling the `returns()` function. In lines 8–12, parameters and constants are assigned to objects that will be employed in the remaining part of this listing. The creation of a risk surface plot consists of several consecutive steps. Commencing in line 14, the convex hull for a long-only portfolio is computed by means of the function `markowitzHull()`. The resultant object can then be used for computing the grid of feasible portfolio solutions by invoking `feasibleGrid()`. The object `grid` is then used next for determining the portfolio characteristics by calling the function `bestDiversification()`. This function returns a named `list` object with elements `x`, `y`, and `z` for the targeted risks, the targeted returns, and the values of the argument "FUN" for the associated grid point allocations, respectively. By default, the portfolio standard deviation risk is returned for the `list` element `z`. Objects returned by calls to `bestDiversification()` are

R code 12.8 Risk surface plot of multi-asset portfolio, part one.

```

library(FRAPO)
library(fPortfolio)
## Loading of data set
data(MultiAsset)
## Creating timeSeries of prices and returns
pr <- timeSeries(MultiAsset, charvec = rownames(MultiAsset))
data <- returns(pr, method = "discrete",
                 percentages = TRUE, trim = TRUE)
## Parameters / constant
NAssets <- ncol(pr)
ANames <- colnames(pr)
Sigma <- cov(data)
mu <- colMeans(data)
## Risk surface plot
hull <- markowitzHull(data, nFrontierPoints = 50)
grid <- feasibleGrid(hull, trace = FALSE)
divers <- bestDiversification(grid, trace = FALSE)
## Standard deviation of marginal risk contributions
mrc.sd <- function(data, weights){
  Sigma <- cov(data)
  a <- mrc(weights, Sigma)
  sd(a)
}
surf <- riskSurface(divers, FUN = "mrc.sd")
## Feasible portfolios with highest diversification ratio
allWeights <- attr(divers, "weights")
idx <- sort(unique(allWeights[, 1]))
dropt <- matrix(0, nrow = length(idx), ncol = 2)
idxRow <- 1:length(idx)
for(j in idx){
  w <- matrix(allWeights[, 1] == j, -c(1, 2)],
  ncol = NAssets)
  divm <- vector()
  length(divm) <- nrow(w)
  for(i in 1:nrow(w)){
    divm[i] <- dr(w[i, ], Sigma)
  }
  divmidx <- which.max(divm)
  wopt <- w[divmidx, ]
  dropt[idxRow[j], ] <- c(crossprod(wopt, mu),
                           sqrt(crossprod(wopt, Sigma))
                           %*% wopt)
}

```

furthermore endowed with attributes. The attributes "weights" and "polygon" can be employed for retrieving the weights of the feasible allocations and the plotting of the convex hull, respectively.

Commencing in line 17, the function `mrc.sd()` for computing the standard deviation of the marginal risk contributions for all feasible long-only allocations is defined. This function is then used in line 23 to compute this risk measure for all grid allocations by calling the function `riskSurface()`. It should be mentioned that risk surfaces can be created for any kind of scalar risk measure, and in this R code example the standard deviation of the marginal risk contributions have been chosen for assessing the degree of potential risk concentration implied by each of the feasible allocations. These values are contained in the named `list` element `z`. Similar to objects returned by `bestDiversification()`, the assigned object `surf` is also endowed with the same set of attributes.

The last piece that needs to be computed are the most diversified portfolio allocations per level of portfolio standard deviation risk, before the risk surface plot can be produced. These allocations (points within the convex hull) are computed in Listing 12.8 from line 24 onward. With this information the associated weight vector can be extracted and the implied portfolio return and the portfolio standard deviation risk is computed. These data pairs are assigned by row to the `matrix` object `dropt`. Of course, instead of superimposing the line of the most diversified allocations per risk level, one could also use a different scalar risk measure, for instance a one-sided one. By now, all objects for creating the desired risk surface plot have been created. The creation of the plot is provided in Listing 12.9.

The function for producing the risk surface plot is named `surfacePlot()`. It expects as its first argument an object with class attribute `riskSurface`, which is a `list` object itself with elements `x`, `y`, and `z` for creating the contour plot. The remaining arguments can be used for shaping the graph's appearance. In line 7, the line of allocations with the greatest diversification ratios for each risk level is superimposed on this chart. In the remaining lines, additional points of interest are marked on the plot, namely the solutions pertinent to the minimum-variance portfolio (MVP), the tangency allocation (TGP), the equal risk-contributed (ERC) portfolio, the equal-weight portfolio (EWP), and the risk-return pairs for each of the single assets. The resultant risk surface plot is provided in Figure 12.11.

By inspection of this risk surface plot, one could observe that the solutions of the minimum-variance and equal-risk contributions portfolios are located next to each other in the return standard deviation risk space. The locus of the equally weighted allocation is further to the right. These relations with respect to the portfolio standard deviation are a confirmation of a result derived in Maillard et al. (2009, 2010), namely that $\sigma_{MVP} \leq \sigma_{ERC} \leq \sigma_{EWP}$ holds. It is further evident from Figure 12.11 that as one moves along the efficient frontier, the portfolio solutions are characterized by a higher degree of risk concentration compared to feasible solutions that are located toward the center of the hull. Hence, if the degree of risk clustering within an asset allocation is considered more important than an efficient allocation, an investor might be better off not implementing a solution on the efficient frontier. Furthermore, it is

R code 12.9 Risk surface plot of multi-asset portfolio, part two.

```

## Surface plot with superimposed mdp solutions
## per standard deviation risk level
surfacePlot(surf, type = "filled.contour",
            palette = topo.colors, addHull = TRUE,
            addGrid = FALSE, addAssets = FALSE,
            xlab = "Target Risk", ylab = "Target Return",
            main = "Convex Hull with Risk Surface:\nStd.Dev.
            of MRC and MDP-line")
lines(x = dropt[, 2], y = dropt[, 1], col = "blue", lwd = 2)
box()
## Computing special points and plotting
frontier <- portfolioFrontier(data)
MVP <- minvariancePoints(frontier)
TGP <- tangencyPoints(frontier)
sap <- singleAssetPoints(frontier)
wewp <- rep(1/NAssets, NAssets)
mewp <- crossprod(mu, wewp)
sewp <- sqrt(crossprod(wewp, Sigma) %*% wewp)
ERC <- PERC(Sigma)
werc <- Weights(ERC) / 100.0
merc <- crossprod(mu, werc)
serc <- sqrt(crossprod(werc, Sigma) %*% werc)
points(sap, col = "darkgreen", pch = 19, cex = 0.8)
text(sap, ANames, col = "darkred", cex = 0.6, pos = 4)
points(TGP, col = "tan", pch = 19, cex = 2.5)
text(TGP[1], TGP[2], "TGP", col = "purple", cex = 0.5)
points(x = sewp, y = mewp, col = "tan", pch = 19, cex = 2.5)
text(sewp, mewp, "EWP", col = "purple", cex = 0.5)
points(x = serc, y = merc, col = "tan", pch = 19, cex = 2.5)
text(serc, merc, "ERC", col = "purple", cex = 0.5)
points(MVP, col = "tan", pch = 19, cex = 2.5)
text(MVP[1], MVP[2], "MVP", col = "purple", cex = 0.5)

```

quite revealing how different the asset allocations can be if one moves along a certain contour line—that is, by comparing the portfolio weights that have the same standard deviation of the marginal risk contributions. Such an analysis is provided in Listing 12.10. In the first line of this example the standard deviations of the marginal risk contributions are assigned to the object `sdmrc`. Next, the grid points of the feasible solutions that are located close to the 10.4 standard deviation contour line are determined (object `c104`). The associated weights can then be swiftly recovered by using these index pairs on the first two columns of the `matrix` object `allWeights` that was created in Listing 12.8. Commencing in line 11, the standard deviations of the marginal risk contributions and the portfolio standard deviation risk are computed for the allocations that are close to the 10.4 contour line. In the last part of this Listing

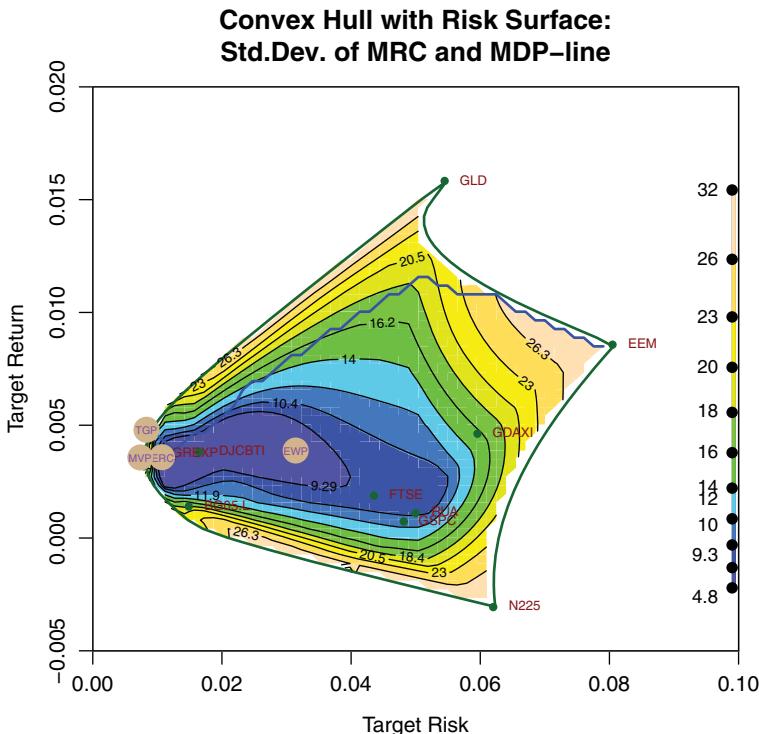


Figure 12.11 Risk surface plot of marginal risk contributions with most diversified portfolio line.

the allocations are aggregated by the asset classes “Equity,” “Bonds,” and “Gold” and then combined with their associated risk measures.

The result (object `ans`) is provided in Table 12.4. In total, 12 feasible solutions are located in the proximity of the 10.4 contour line. For instance, solely assessed by the degree of risk clustering, a 100% allocation to equities (allocation in row five) is roughly the same as an investment of 27% in equities, 56% in bonds, and 17% in gold (allocation in row nine). Certainly, the associated levels of the portfolio standard deviation risks do differ. However, if one considers allocations with the same risk level, the allocations can be markedly different. That is, a portfolio standard deviations risk of 3.38% intersects the 10.4 contour line twice. The first allocation is solely in equities and bonds (roughly two thirds and one third, respectively, as given in row two), whereas in the case of the second feasible portfolio roughly 62% would have been allotted to equities, 21% to bonds, and 17% to gold (see allocation in row 11). Hence, an investor who takes the degree of risk concentration and the level of the portfolio standard deviation risk into account, should be indifferent between these two feasible allocations.

R code 12.10 Risk surface plot of multi-asset portfolio, part three.

```

1 sdmrc <- surf$z
2 c104 <- which((sdmrc >= 10.35) & (sdmrc <= 10.45) ,
3           arr.ind = TRUE)
4 w104 <- matrix(NA, nrow = nrow(c104), ncol = NAssets)
5 colnames(w104) <- ANames
6 for(i in 1:nrow(c104)){
7   gidx <- which((allWeights[, 1] == c104[i, 1]) &
8                 (allWeights[, 2] == c104[i, 2]),
9                 arr.ind = TRUE)
10  w104[i, ] <- allWeights[gidx, -c(1, 2)]
11}
12## Computing standard deviations of mrc and standard deviation
13## risk
14 sdmrc104 <- apply(w104, 1, function(x)
15                   sd(mrc(x, Sigma = Sigma)))
16 sdr104 <- apply(w104, 1, function(x) sqrt(crossprod(x, Sigma)
17                   %*% x)) * 100
18## Grouping by asset class
19 wEquity <- w104[, 1:6]
20 wBonds <- w104[, 7:9]
21 wGold <- w104[, 10]
22 wEquity <- rowSums(wEquity)
23 wBonds <- rowSums(wBonds)
24 wAsset <- cbind(wEquity, wBonds, wGold) * 100
25 ans <- cbind(wAsset, sdmrc104, sdr104)
26 colnames(ans) <- c("Equity", "Bonds", "Gold", "StdDev .
27                   of MRC", "StdDev. Risk")
28 rownames(ans) <- 1:nrow(ans)

```

Table 12.4 Feasible asset allocations with similar degree of risk concentration.

Allocations	Equity	Bonds	Gold	Std dev. MRC	Std dev. risk
1	85.00	15.00	0.00	10.43	4.29
2	68.92	31.08	0.00	10.38	3.38
3	57.83	40.74	1.43	10.41	2.78
4	36.74	60.54	2.72	10.35	1.72
5	100.00	0.00	0.00	10.41	5.49
6	99.93	0.00	0.07	10.37	5.49
7	93.32	1.66	5.01	10.43	5.19
8	75.22	12.55	12.24	10.43	4.29
9	27.31	55.65	17.04	10.38	1.72
10	68.90	16.46	14.64	10.39	3.98
11	62.38	20.59	17.03	10.35	3.68
12	55.58	25.01	19.41	10.40	3.38

References

- Alexander G. and Baptista A. 1999 Value at risk and mean-variance analysis. Working Paper 9804, University of Minnesota, Minneapolis.
- Alexander G. and Baptista A. 2002 Economic implications of using a mean-VaR model for portfolio selection: A comparison with mean-variance analysis. *Journal of Economic Dynamics & Control* **26**, 1159–1193.
- Alexander G. and Baptista A. 2004 A comparison of VaR and CVaR constraints on portfolio selection with the mean-variance model. *Management Science* **50**(9), 1261–1273.
- Alexander G. and Baptista A. 2008 Active portfolio management with benchmarking: Adding a value-at-risk constraint. *Journal of Economic Dynamics & Control* **32**, 779–820.
- Berkelaar M. 2015 *lpSolve: Interface to “lp_solve” v. 5.5 to Solve Linear/Integer Programs*. R package version 5.6.11.
- Boudt K., Peterson B., and Croux C. 2008 Estimation and decomposition of downside risk for portfolios with non-normal returns. *The Journal of Risk* **11**(2), 79–103.
- Bravo H. C., Theussl S., and Hornik K. 2014 *Rcplex: R interface to CPLEX*. R package version 0.3-2.
- Cesarone F., Scozzari A., and Tardella F. 2011 Portfolio selection problems in practice: a comparison between linear and quadratic optimization models. *Quantitative Finance Papers* 1105.3594, arXiv.org.
- Chekhlov A., Uryasev S., and Zabarankin M. 2000 Portfolio optimization with drawdown constraints. Research report 2000-5, Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL.
- Chekhlov A., Uryasev S., and Zabarankin M. 2005 Drawdown measure in portfolio optimization. *International Journal of Theoretical and Applied Finance* **8**(1), 13–58.
- Czyzyk J., Mesnier M., and Moré J. 1998 The NEOS server. *IEEE Journal on Computational Science and Engineering* **5**, 68–75.
- De Giorgi E. 2002 A note on portfolio selection under various risk measures. Working Paper 9, National Centre of Competence in Research, Financial Valuation and Risk Management, Zürich, Switzerland.
- Dolan E. 2001 The NEOS server 4.0 administrative guide. Technical memorandum ANL/MCS-TM-250, Mathematics and Computer Science Division, Argonne National Laboratory.
- Durand R., Gould J., and Miller R. 2011 On the performance of the minimum VaR portfolio. *The European Journal of Finance* **17**(7), 553–576.
- Friberg H. 2014 *Rmosek: The R-to-MOSEK Optimization Interface*. R package version 1.2.5.1.
- Gelius-Dietrich G. 2015a *cplexAPI: R Interface to C API of IBM ILOG CPLEX*. R package version 1.3.1.
- Gelius-Dietrich G. 2015b *glpkAPI: R Interface to C API of GLPK*. R package version 1.3.0.
- Gropp W. and Moré J. 1997 *Approximation Theory and Optimization* Cambridge University Press chapter 9, pp. 167–182.
- Harter R., Hornik K., and Theussl S. 2016 *Rsymphony: SYMPHONY in R*. R package version 0.1-22.
- Henningsen A. 2012 *linprog: Linear Programming / Optimization*. R package version 0.9-2.

- Konis K. 2016 *lpSolveAPI: R Interface for lp_solve version 5.5.2.0*. R package version 5.5.2.0-17.
- Krokhmal P., Uryasev S., and Zrazhevsky G. 2002 Risk management for hedge fund portfolios. *Journal of Alternative Investments* 5(1), 10–29.
- Maillard S., Roncalli T., and Teiletche J. 2009 On the properties of equally-weighted risk contributions portfolios. Working paper, SGAM Alternative Investments and Lombard Odier and University of Paris Dauphine.
- Maillard S., Roncalli T., and Teiletche J. 2010 The properties of equally weighted risk contribution portfolios. *The Journal of Portfolio Management* 36(4), 60–70.
- Merton R. 1972 An analytical derivation of the efficient portfolio frontier. *Journal of Financial and Quantitative Analysis* 7, 1851–1872.
- Pardalos P., Migdalas A., and Baourakis G. 2004 *Supply Chain and Finance* vol. 2 of *Series on Computers and Operations Research* World Scientific Publishing Singapore chapter 13, pp. 209–228.
- Peterson B. and Carl P. 2014 *PerformanceAnalytics: Econometric tools for performance and risk analysis*. R package version 1.4.3541.
- Pfaff B. 2016 *rneos: XML-RPC Interface to NEOS*. R package version 0.3-1.
- Rockafellar R. and Uryasev S. 2000 Optimization of conditional value-at-risk. *The Journal of Risk* 2(3), 21–41.
- Rockafellar R. and Uryasev S. 2001 Conditional value-at-risk for general loss distributions. Research Report 2001-5, Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL.
- Soetaert K., Van den Meersche K., and van Oevelen D. 2009 *limSolve: Solving Linear Inverse Models*. R package 1.5.1.
- Theussl S. and Hornik K. 2015 *Rglpk: R/GNU Linear Programming Kit Interface*. R package version 0.6-1.
- Uryasev S. 2005 Conditional value-at-risk (CVaR): Algorithms and applications. Master in finance and risk management (FINARM), Universita degli Studi di Milano, Milano, Italy.
- Uryasev S. and Rockafellar T. 1999 Optimization of conditional value-at-risk. Research Report 99-4, Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL.
- Van den Meersche K., Soetaert K., and Van Oevelen D. 2009 xsample(): An R function for sampling linear inverse problems. *Journal of Statistical Software, Code Snippets* 30(1), 1–15.
- Würtz D., Setz T. and Chalabi Y. 2014 *fPortfolio: Rmetrics – Portfolio Selection and Optimization*. R package version 3011.81.

13

Tactical asset allocation

13.1 Overview

The previous chapters were concerned with the modelling of portfolio risk and how this can be directly incorporated in the process of portfolio optimization. The focus now shifts to the other side of the return/risk coin. The following sections will first outline how directional and relative forecasts can be obtained and then how portfolio allocations can be derived from a set of quantitatively derived trades/signals. Of course, the techniques portrayed are not confined to tactical asset allocation (TAA), but can also be applied to wealth allocations other than tactically shifting assets in and out of a portfolio. The domain of TAA is rather wide. Trading strategies can be derived from purely technical indicators, by subjective reasoning, and from statistical/econometric models. The first and second categories will not be covered in this chapter; rather, the focus will be on the description of selected time series methods for deriving forecasts of asset prices. Incidentally, even though technical analysis will not be covered, the reader is referred to the CRAN packages **fTrading** (see Würtz 2013b) and **TTR** (see Ulrich 2016) in which numerous technical indicators are implemented. There are also quite a few different ways of deriving TAA allocations, the Black–Litterman model probably being the most widely known and almost synonymous with TAA-driven portfolio allocations. However, TAA can also be combined with risk-overlay and/or high-watermark strategies.

Given the quite huge variety of applications and combinations of TAA, it is quite surprising that the literature directly related to TAA is rather sparse. To the author's knowledge, the monograph of Lee (2000) is the only book devoted entirely to the theory and practice of TAA. Also, the number of articles explicitly covering TAA is rather small. Although this chapter can by no means completely fill this gap, the aim is to provide as thorough an account of the topic as space permits.

13.2 Survey of selected time series models

13.2.1 Univariate time series models

AR(p) time series process

We start by considering a simple first-order autoregressive (AR) process. The value of y at the current time t is explained by its value at time $t - 1$, a constant c , and a white noise error process $\{\varepsilon_t\}$:

$$y_t = c + \phi y_{t-1} + \varepsilon_t. \quad (13.1)$$

Basically, (13.1) is a first-order inhomogeneous difference equation. The path of this process depends on the value of ϕ . If $|\phi| \geq 1$, then shocks accumulate over time and the process is non-stationary. Incidentally, if $|\phi| > 1$ the process grows without bounds, and if $|\phi| = 1$ it has a unit root. The latter will be discussed in more detail in the subsection on multivariate time series modelling. For now, only the covariance-stationary case, $|\phi| < 1$, is considered. With the lag operator \mathbf{L} , (13.1) can be rewritten as

$$(1 - \phi \mathbf{L})y_t = c + \varepsilon_t. \quad (13.2)$$

The stable solution to this process is given by an infinite sum of past errors with decaying weights:

$$y_t = (c + \varepsilon_t) + \phi(c + \varepsilon_{t-1}) + \phi^2(c + \varepsilon_{t-2}) + \phi^3(c + \varepsilon_{t-3}) + \dots \quad (13.3a)$$

$$= \left[\frac{c}{1 - \phi} \right] + \varepsilon_t + \phi \varepsilon_{t-1} + \phi^2 \varepsilon_{t-2} + \phi^3 \varepsilon_{t-3} + \dots \quad (13.3b)$$

The expected value and the second-order moments of the AR(1) process in (13.1) are given by

$$\mu = \mathbb{E}[y_t] = \frac{c}{1 - \phi}, \quad (13.4a)$$

$$\gamma_0 = \mathbb{E}[(y_t - \mu)^2] = \frac{\sigma^2}{1 - \phi^2}, \quad (13.4b)$$

$$\gamma_j = \mathbb{E}[(y_t - \mu)(y_{t-j} - \mu)] = \left[\frac{\phi^j}{1 - \phi^2} \right] \sigma^2. \quad (13.4c)$$

From (13.4c), the geometrically decaying pattern of the auto-covariances is evident.

The AR(1) process can be generalized to an AR(p) process:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \dots + \phi_p y_{t-p} + \varepsilon_t. \quad (13.5)$$

As with (13.1), (13.5) can be rewritten as

$$(1 - \phi_1 \mathbf{L} - \phi_2 \mathbf{L}^2 - \dots - \phi_p \mathbf{L}^p)y_t = c + \varepsilon_t. \quad (13.6)$$

It can be shown that such an AR(p) process is stationary if all roots z_0 of the polynomial

$$\phi_p(z) = 1 - \phi_1 z - \phi_2 z^2 - \cdots - \phi_p z^p \quad (13.7)$$

have a modulus greater than one. The modulus of a complex number $z = z_1 + iz_2$ is defined as $|z| = \sqrt{z_1^2 + z_2^2}$. Viewing the stationarity condition from that angle, it turns out that in the case of an AR(1) process, as in (13.1), $|\phi| < 1$ is required because the only solution to $1 - \phi z = 0$ is given for $z = 1/\phi$ and $|z| = |1/\phi| > 1$ when $|\phi| < 1$.

If the error process $\{\varepsilon_t\}$ is normally distributed, (13.5) can be consistently estimated by the ordinary least-squares (OLS) method. Furthermore, the OLS estimator for the unknown coefficient vector $\beta = (c, \phi)'$ is asymptotically normally distributed. Alternatively, the model parameters can be estimated by the principle of maximum likelihood. However, one problem arises in the context of AR(p) models and this holds true for the more general class of ARMA(p, q) models discussed later. For iid random variables with probability density function $f(y_t; \theta)$ for $t = 1, \dots, T$ and parameter vector θ , the joint density function is the product of the marginal densities:

$$f(\mathbf{y}; \theta) = f(y_1, \dots, y_T; \theta) = \prod_{t=1}^T f(y_t; \theta). \quad (13.8)$$

This joint density function can, in line with the ML principle, be interpreted as a function of the parameters θ given the data vector \mathbf{y} ; that is, the likelihood function is given by

$$\mathfrak{L}(\theta|\mathbf{y}) = \mathfrak{L}(\theta|y_1, \dots, y_T) = \prod_{t=1}^T f(y_t; \theta). \quad (13.9)$$

The log-likelihood function then has the simple form

$$\ln \mathfrak{L}(\theta|\mathbf{y}) = \sum_{t=1}^T \ln f(y_t; \theta). \quad (13.10)$$

Because our model assumes that the time series $\{y_t\}$ has been generated from a covariance-stationary process, the iid assumption is violated and hence the log-likelihood cannot be derived as swiftly as in (13.8)–(13.10). That is, y_t is modelled as a function of its own history, and therefore y_t is not independent of y_{t-1}, \dots, y_{t-p} given that $\{\varepsilon_t\}$ is normally distributed with expectation $\mu = 0$ and variance σ^2 . In order to apply the ML principle, one therefore has two options: either estimate the full-information likelihood function or derive the likelihood function from a conditional marginal factorization. The derivation of the log-likelihood for both options is provided, for instance, in Hamilton (1994). Here, we will focus on the second option. The idea is that the joint density function can be factored as the product of the conditional density function given all past information and the joint density function of the initial values:

$$f(y_T, \dots, y_1; \theta) = \left(\prod_{t=p+1}^T f(y_t | \mathcal{I}_{t-1}, \theta) \right) \cdot f(y_p, \dots, y_1; \theta), \quad (13.11)$$

where \mathcal{I}_{t-1} denotes the information available at time t . This joint density function can then be interpreted as the likelihood function with respect to the parameter vector θ given the sample y , and therefore the log-likelihood is given by

$$\ln \mathfrak{L}(\theta|y) = \sum_{t=p+1}^T \ln f(y_t|\mathcal{I}_{t-1}, \theta) + \ln f(y_p, \dots, y_1; \theta). \quad (13.12)$$

The log-likelihood consists of two terms. The first term is the conditional log-likelihood and the second term the marginal log-likelihood for the initial values. Whether one maximizes the exact log-likelihood, as in (13.12), or only the conditional log-likelihood, that is, the first term of the exact log-likelihood, is asymptotically equivalent. Both are consistent estimators and have the same limiting normal distribution. Bear in mind that in small samples the two estimators might differ by a non-negligible amount, in particular if the roots are close to unity. Because a closed-form solution does not exist, numerical optimization methods are used to derive optimal parameter values.

MA(q) time series process

It was shown above that a finite stable AR(p) process can be inverted to a moving average (MA) of current and past shocks. It is now considered how a process can be modelled as a finite moving average of its shocks. Such a process is called MA(q), where the parameter q refers to the highest lag of shocks included in such a process. An MA(1) process is given by

$$y_t = \mu + \varepsilon_t + \theta \varepsilon_{t-1}, \quad (13.13)$$

where $\{\varepsilon_t\}$ is a white noise process and μ, θ can be any constants. This process has moments

$$\mu = \mathbb{E}[y_t] = \mathbb{E}[\mu + \varepsilon_t + \theta \varepsilon_{t-1}], \quad (13.14a)$$

$$\gamma_0 = \mathbb{E}[(y_t - \mu)^2] = (1 + \theta^2)\sigma^2, \quad (13.14b)$$

$$\gamma_1 = \mathbb{E}[(y_t - \mu)(y_{t-1} - \mu)] = \theta\sigma^2. \quad (13.14c)$$

The higher auto-covariances γ_j with $j > 1$ are zero. Neither the mean nor the auto-covariance are functions of time, hence an MA(1) process is covariance-stationary for all values of θ . Incidentally, this process also has the characteristic of ergodicity.

The MA(1) process can be extended to the general class of MA(q) processes:

$$y_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q} \quad (13.15)$$

With the lag operator \mathbb{L} , this process can be rewritten as

$$y_t - \mu = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q} \quad (13.16a)$$

$$= (1 + \theta_1 \mathbb{L} + \dots + \theta_q \mathbb{L}^q) \varepsilon_t = \theta_q(\mathbb{L}) \varepsilon_t. \quad (13.16b)$$

Much as a stable AR(p) process can be rewritten as an infinite MA process, an MA(q) process can be transformed into an infinite AR process as long as the roots of the characteristic polynomial, the z -transform, have modulus greater than 1, that is, are outside the unit circle:

$$\theta_q z = 1 + \theta_1 z + \cdots + \theta_q z^q. \quad (13.17)$$

The expected value of an MA(q) process is μ and hence invariant with respect to its order. The second-order moments are

$$\gamma_0 = E[(y_t - \mu)^2] = (1 + \theta_1^2 + \cdots + \theta_q^2)\sigma^2, \quad (13.18a)$$

$$\gamma_j = E[(\varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}) \times (\varepsilon_{t-q} + \theta_1 \varepsilon_{t-j-1} + \cdots + \theta_q \varepsilon_{t-j-q})]. \quad (13.18b)$$

Because the $\{\varepsilon_t\}$ are uncorrelated with each other by assumption, (13.18b) can be simplified to

$$\gamma_j = \begin{cases} (1 + \theta_{j+1}\theta_1 + \theta_{j+2}\theta_2 + \cdots + \theta_q\theta_{q-j})\sigma^2 & \text{for } j = 1, 2, \dots, q, \\ 0 & \text{for } j > q. \end{cases} \quad (13.19)$$

That is, empirically an MA(q) process can be detected by its first q significant autocorrelations and a slowly decaying or alternating pattern of its partial autocorrelations. For large sample sizes T , a 95% significance band can be calculated as

$$\left(\rho_j - \frac{2}{\sqrt{T}}, \rho_j + \frac{2}{\sqrt{T}} \right), \quad (13.20)$$

where ρ_j refers to the j th-order autocorrelation.

It has been stated above that a finite AR process can be inverted to an infinite MA process. Before we proceed further, let us first examine the stability condition of such an MA(∞) process,

$$y_t = \mu + \sum_{j=0}^{\infty} \psi_j \varepsilon_{t-j}. \quad (13.21)$$

The coefficients for an infinite process are denoted by ψ instead of θ . It can be shown that such an infinite process is covariance-stationary if the coefficient sequence $\{\psi_j\}$ is either square summable,

$$\sum_{j=0}^{\infty} \psi_j^2 < \infty, \quad (13.22)$$

or absolute summable,

$$\sum_{j=0}^{\infty} |\psi_j| < \infty, \quad (13.23)$$

where absolute summability is sufficient for square summability—that is, the former implies the latter, but not vice versa.

ARMA(p, q) time series process

The previous subsections have shown how a time series can be explained either by its history or by current and past shocks. Furthermore, the moments of these data-generating processes have been derived and the mutual convertibility of these model classes has been stated for parameter sets that fulfill the stability condition. These two time series processes are now put together and a more general class of ARMA(p, q) processes is investigated.

In practice, it is often cumbersome to detect a pure AR(p) or MA(q) process by the behaviors of its empirical ACF and PACF because neither one tapers off with increasing lag order. In these instances, the time series might have been generated by a mixed autoregressive moving average process.

For a stationary time series $\{y_t\}$, such a mixed process is defined as

$$y_t = c + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}. \quad (13.24)$$

By assumption, $\{y_t\}$ is stationary, that is, the roots of the characteristic polynomial lie outside the unit circle. Hence, with the lag operator, (13.24) can be transformed to:

$$y_t = \frac{c}{1 - \phi_1 L - \cdots - \phi_p L^p} + \frac{1 + \theta_1 L + \cdots + \theta_q L^q}{1 - \phi_1 L - \cdots - \phi_p L^p} \varepsilon_t \quad (13.25a)$$

$$= \mu + \psi(L) \varepsilon_t. \quad (13.25b)$$

The condition of absolute summability for the lag coefficients $\{\psi_j\}$ must hold. Put differently, the stationarity condition depends only on the AR parameters and not on the MA ones.

We now briefly touch on the Box–Jenkins approach to time series modelling (see Box and Jenkins 1976). This approach consists of three stages: identification, estimation, and diagnostic checking. As a first step, the series is visually inspected for stationarity. If an investigator has doubts that this condition is met, he/she has to suitably transform the series before proceeding. Such transformations could involve the removal of a deterministic trend or taking first differences with respect to time. Furthermore, variance instability such as higher fluctuations as time proceeds can be coped with by using the logarithmic values of the series instead. By inspecting the empirical ACF and PACF, a tentative ARMA(p, q) model is specified. The next stage is the estimation of a preliminary model. The ML principle allows one to discriminate between different model specifications by calculating information criteria and/or applying likelihood-ratio tests. Thus, one has a second set of tools to determine an appropriate lag order for ARMA(p, q) models compared with the order decision that is derived from ACF and PACF. Specifically, the Akaike, Schwarz, and/or Hannan–Quinn information criteria can be utilized in the determination of an appropriate model structure (see Akaike 1981, Hannan and Quinn 1979, Quinn 1980, Schwarz 1978):

$$AIC(p, q) = \ln(\hat{\sigma}^2) + \frac{2(p + q)}{T}, \quad (13.26a)$$

$$\text{BIC}(p, q) = \ln(\hat{\sigma}^2) + \frac{\ln T(p+q)}{T}, \quad (13.26\text{b})$$

$$\text{HQ}(p, q) = \ln(\hat{\sigma}^2) + \frac{\ln(\ln(T))(p+q)}{T}, \quad (13.26\text{c})$$

where $\hat{\sigma}^2$ denotes the estimated variance of an ARMA(p, q) process. The lag order (p, q) that minimizes the information criteria is then selected. As an alternative, a likelihood-ratio test can be computed for an unrestricted and a restricted model. The test statistic is defined as

$$2[\mathfrak{L}(\hat{\theta}) - \mathfrak{L}(\tilde{\theta})] \sim \chi^2(m), \quad (13.27)$$

where $\mathfrak{L}(\hat{\theta})$ denotes the estimate of the unrestricted log-likelihood and $\mathfrak{L}(\tilde{\theta})$ that of the restricted log-likelihood. This test statistic is distributed as χ^2 with m degrees of freedom, which corresponds to the number of restrictions. Next, one should check the model's stability as well as the significance of its parameters. If one of these tests fails, the econometrician has to start again by specifying a more parsimonious model with respect to the ARMA order. In the last step, diagnostic checking, one should then examine the residuals for lack of correlation and for normality and conduct tests for correctness of the model order, (i.e., over- and under-fitting). Incidentally, by calculating pseudo *ex ante* forecasts, the model's suitability for prediction can be examined.

Once a stable (in the sense of covariance-stationary) ARMA(p, q) model has been estimated, it can be used to predict future values of y_t . These forecasts can be computed recursively from the linear predictor:

$$y_T(h) = \phi_1 \bar{y}_{T+h-1} + \cdots + \phi_p \bar{y}_{T+h-p} + \varepsilon_t + \theta_1 \varepsilon_{t-T-1} + \cdots + \theta_q \varepsilon_{t-T-q} + \cdots, \quad (13.28)$$

where $\bar{y}_t = y_t$ for $t \leq T$ and $\bar{y}_{T+j} = y_T(j)$ for $j = 1, \dots, h-1$. Using the Wold representation of a covariance-stationary ARMA(p, q) process (see (13.25a) and (13.25b)), this predictor is equivalent to

$$y_T(h) = \mu + \psi_h \varepsilon_t + \psi_{h+1} \varepsilon_{t-1} + \psi_{h+2} \varepsilon_{t-2} + \cdots \quad (13.29)$$

It can be shown that this predictor is minimal with respect to the mean squared error criterion based on the information set \mathcal{I}_t —see, for instance, Judge et al. (1985, Chapter 7) and Hamilton (1994, Chapter 4). Incidentally, if the forecast horizon h is greater than the moving average order q , the forecasts are determined solely by the autoregressive terms in (13.28).

If $\{\varepsilon_t\}$ is assumed to be standard normally distributed, then it follows that the h -steps-ahead forecast is distributed as

$$y_{t+h} | \mathcal{I}_t \sim \mathcal{N}(y_{t+h|t}, \sigma^2(1 + \psi_1^2 + \cdots + \psi_{h-1}^2)), \quad (13.30)$$

where the $\psi_i, i = 1, \dots, h-1$, denote the coefficients from the Wold representation of a covariance-stationary ARMA(p, q) process. The 95% forecast confidence band

can then be computed as

$$y_{t+h} | \mathcal{I}_t \pm 1.96 \cdot \sqrt{\sigma^2(1 + \psi_1^2 + \dots + \psi_{h-1}^2)}. \quad (13.31)$$

13.2.2 Multivariate time series models

Structural multiple equation models

It almost goes without saying that today's capital markets are highly interdependent. This interdependence occurs across countries and/or across assets. Structural multiple equation models allow the modelling of explicit interdependencies as observed in financial markets. In addition, exogenous variables (e.g., macroeconomic data) can be included in this model type. Textbook accounts of this model class are included, for example, in Judge et al. (1985, 1988) and Greene (2008). Structural multiple equation models (SMEMs) can be utilized for forecasting, scenario analysis, and risk assessment as well as for multiplier analysis, and they are therefore ideally suited for tactical asset allocation. The origins of the SMEM can be traced back to the 1940s and 1950s, when this model type was proposed by the Cowles Foundation. At that time the SMEM was primarily applied to large-scale macroeconomic modelling. Prominent examples of this application were the Klein model for the US economy and the models run by most central banks and other organizations and international institutions, such as the Deutsche Bundesbank. However, this type of model can also be applied to financial markets and forecasts for a set of assets can be obtained for a TAA as shown in Pfaff (2007).

The structural form of a multiple equation model is

$$Ay_t + Bz_t = u_t \text{ for } t = 1, \dots, T, \quad (13.32)$$

where A is the $(N \times N)$ coefficient matrix of the endogenous variables, y_t is the $(N \times 1)$ vector of the endogenous variables, B is the $(N \times K)$ coefficient matrix of the predetermined variables, z_t is the $(K \times 1)$ vector of the predetermined variables (i.e., exogenous and lagged endogenous variables), and u_t is the $(N \times 1)$ vector of white noise disturbances. If applicable, one can simplify the model's structure by concentrating out the irrelevant endogenous variables. This yields the revised form of the model. The reduced form of the model can be obtained from the revised form if it is expressed explicitly in terms of the endogenous variables:

$$y_t = -A^{-1}Bz_t + A^{-1}u_t \text{ for } t = 1, \dots, T. \quad (13.33)$$

If the reduced form is solved in terms of starting values for the endogenous and exogenous variables, one obtains the final form. This representation of the model can be utilized for stability and multiplicand analysis. An SMEM can be characterized as either a recursive or an interdependent model and as either dynamic or static. If the matrix A can be rearranged so that it is either upper or lower diagonal, then the multiple equation model is recursive, otherwise interdependencies between the n endogenous variables exist. This is equivalent to $\det(A) = \prod_{i=1}^n a_{i,i}$. If the matrix Z does

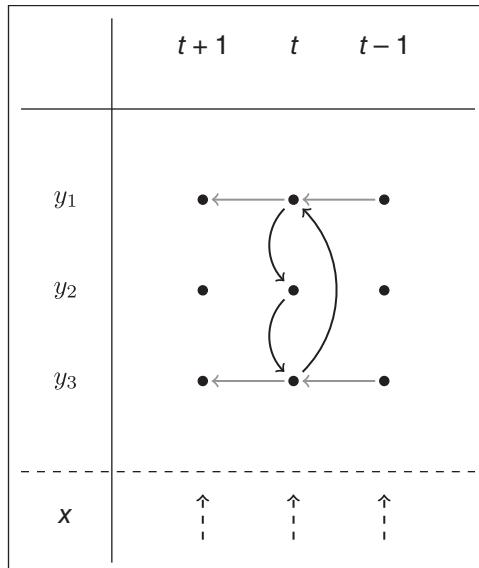


Figure 13.1 Tinbergen arrow diagram.

not contain lagged endogenous variables, the model is said to be static, otherwise it is dynamic. Both of the above can be visualized in a Tinbergen arrow diagram. An example is shown in Figure 13.1.

This example shows a fictitious SMEM structure for three endogenous variables (y_1, y_2, y_3) and a set of model exogenous variables, x . An interdependent structure is obtained if the curved arrows indicate a loop on a per-period basis, and a dynamic structure is indicated when arrows are drawn from one period to the next for at least one endogenous variable. As can be seen from this figure, although there is no interdependency between y_1 and y_3 a closed loop exists via the indirect links through y_2 . It can further be concluded that y_2 is dynamically affected by the trajectories of y_1 and y_3 through the inclusion of lagged endogenous variables in the reaction equations for these two variables. Thus, the structure of the SMEM in Figure 13.1 is dynamic and interdependent.

In addition to these model characteristics, the degree of identification with respect to the structural form parameters is important. Because ordinarily only the reduced-form parameters are estimated, it might not be feasible to infer the structural parameters. This problem is illustrated by a simple partial market model.

Consider a partial market model for a good, with an upward-sloping supply curve and downward-sloping demand curve. The intersection of the two lines is a point of equilibrium (see Figure 13.2). We observe only the pairs (p_i, q_i) . The simple partial market model (light gray lines, not identified) can be written as:

$$q_d = \alpha_1 + \alpha_2 p, \quad (13.34a)$$

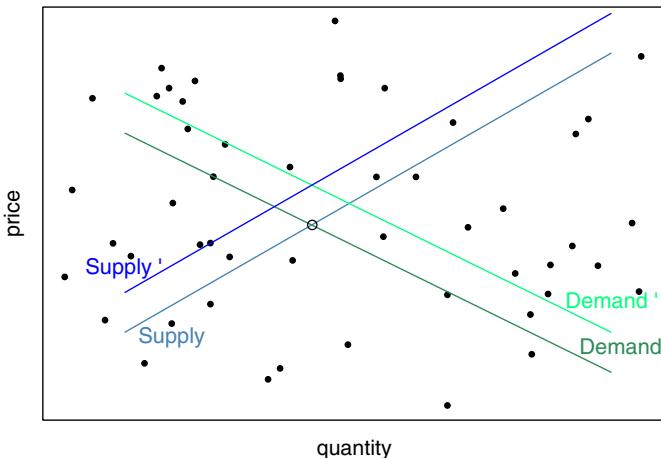


Figure 13.2 Identification: partial market model.

$$q_s = \beta_1 + \beta_2 p, \quad (13.34b)$$

$$q_s = q_d. \quad (13.34c)$$

A model in which the demand and supply curve are identified is obtained by introducing shift variables—for example, a cost variable for supply and a wealth variable for demand (dark gray lines)—such that the structural parameters can be determined, and hence a need arises for the inclusion of exogenous variables, in contrast to the vector autoregressive and vector error correction models discussed later in this subsection, which are ordinarily expressed solely in terms of past values for the endogenous variables:

$$q_d = \alpha_1 + \alpha_2 p + \alpha_3 w, \quad (13.35a)$$

$$q_s = \beta_1 + \beta_2 p + \beta_3 c, \quad (13.35b)$$

$$q_s = q_d. \quad (13.35c)$$

An interdependent multiple equation model is identified if a unique solution for its structural coefficients exists. A structural equation is said to be identified if it is not possible to obtain another structural equation with the same statistical characteristics by a linear combination of the remaining equations. As a general rule, a structural equation is identified if a sufficient number of predetermined variables are excluded from this equation. Two criteria for determining the degree of identification are the number (necessary) and rank (necessary and sufficient) criteria. If K^- denotes the number of excluded predetermined variables in a particular equation and H^+ denotes the number of included endogenous variables in a particular equation, then according to the number criterion, an equation is not identified if $K^- < H^+ - 1$, is just identified if $K^- = H^+ - 1$, and is over-identified if $K^- > H^+ - 1$. Consider the matrix

$P = -A^{-1}B$ and let P_{K^-, H^+} be the sub-matrix for K^- and H^+ of a particular equation. According to the rank criteria, an equation is said to be identified if

$$rg(P_{K^-, H^+}) = H^+ - 1. \quad (13.36)$$

The unknown coefficients can be estimated by either a two- or three-stage least squares method (2SLS or 3SLS) or by full-information maximum likelihood (FIML). The former two methods are implemented in the package **systemfit** (see Henningsen and Hamann 2007) and the latter method can be embedded in a call to a numeric optimization routine, such as `optim()`. However, in practice, the reaction equations of a reduced form are estimated by OLS and the model solution for the endogenous variables can be determined by iterative methods. Iterative methods can be expressed as $\mathbf{y}^{(k)} = B\mathbf{y}^{(k-1)} + \mathbf{c}$, with k denoting the iteration number. An iterative method is said to be stationary if neither B nor \mathbf{c} depends on k . An iterative method is deemed to have converged if some measure is smaller than a predefined threshold, for instance $\|\mathbf{y}^{(k)} - \mathbf{y}^{(k-1)}\| < \epsilon$, with ϵ being chosen suitably small (e.g., $\epsilon < 0.001$). An example of such a method is the Gauss–Seidel algorithm, which can be applied on a per-period basis. The Gauss–Seidel algorithm is a stationary iterative method for linear models of the form $A\mathbf{y}_t = \mathbf{b}_t$, where A is an $(n \times n)$ matrix, \mathbf{y}_t is an $(n \times 1)$ vector of endogenous variables, and \mathbf{b}_t is an $(n \times 1)$ vector of the right-hand-side expressions. The algorithm is defined as

$$y_{t,i}^{(k)} = (b_{t,i} - \sum_{j < i}^n a_{i,j}y_{t,j}^{(k)} - \sum_{j > i}^n y_{t,j}^{(k-1)})/a_{i,i}, \quad (13.37)$$

or, in matrix notation,

$$\mathbf{y}_t^{(k)} = (D - L)^{-1}(U\mathbf{y}_t^{(k-1)} + \mathbf{b}), \quad (13.38)$$

where D, L, U represent the diagonal, lower-, and upper-triangular parts of A . The latter approach is implemented, for instance, in the freely available Fair–Parke program—see <http://fairmodel.econ.yale.edu/> and Fair (1984, 1998, 2004) for more information. The appropriateness of an SMEM should then be evaluated in terms of its dynamic *ex post* forecasts. This is the strictest test for evaluating the overall stability of the model. The fitted values in each period are used as values for the lagged endogenous variables in the subsequent periods. Hence, forecast errors can accumulate over time and the forecasts can diverge. Thus, a small root-mean-square error between the actual and fitted values of the dynamic *ex post* forecasts is sought.

Vector autoregressive models

Since the critique of Sims (1980), multivariate data analysis in the context of vector autoregressive models (VAR) has evolved as a standard instrument in econometrics. Because statistical tests are frequently used in determining interdependencies and dynamic relationships between variables, this methodology was soon enriched by incorporating non-statistical *a priori* information. VAR models explain the endogenous variables solely in terms of their own history, apart from deterministic regressors. In contrast, structural vector autoregressive (SVAR) models allow the explicit modelling of contemporaneous interdependence between the left-hand-side

variables. Hence, these types of models try to bypass the shortcomings of VAR models. At the same time as Sims challenged the paradigm of multiple structural equation models laid out by the Cowles Foundation in the 1940s and 1950s, Granger (1981) and Engle and Granger (1987) gave econometricians a powerful tool for modelling and testing economic relationships, namely, the concept of co-integration. Nowadays these branches of research are unified in the form of vector error correction models (VECMs) and structural vector error correction (SVEC) models. A thorough theoretical exposition of all these models is provided in the monographs of Banerjee et al. (1993), Hamilton (1994), Hendry (1995), Johansen (1995), and Lütkepohl (2006).

In its basic form, a VAR consists of a set of K endogenous variables $\mathbf{y}_t = (y_{1t}, \dots, y_{kt}, \dots, y_{Kt})$ for $k = 1, \dots, K$. The VAR(p) process is then defined as¹

$$\mathbf{y}_t = A_1 \mathbf{y}_{t-1} + \dots + A_p \mathbf{y}_{t-p} + \mathbf{u}_t, \quad (13.39)$$

where the A_i are $(K \times K)$ coefficient matrices for $i = 1, \dots, p$ and \mathbf{u}_t is a K -dimensional process with $E(\mathbf{u}_t) = \mathbf{0}$ and time-invariant positive definite covariance matrix $E(\mathbf{u}_t \mathbf{u}_t^\top) = \Sigma_u$ (white noise).

One important characteristic of a VAR(p) process is its stability. This means that it generates stationary time series with time-invariant means, variances, and covariance structure, given sufficient starting values. One can check this by evaluating the characteristic polynomial

$$\det(I_K - A_1 z - \dots - A_p z^p) \neq 0 \quad \text{for } |z| \leq 1. \quad (13.40)$$

If the solution of the above equation has a root for $z = 1$, then either some or all variables in the VAR(p) process are integrated of order 1, $I(1)$. It might be the case that co-integration between the variables exists. This is better analyzed in the context of a VECM.

In practice, the stability of an empirical VAR(p) process can be analyzed by considering the companion form and calculating the eigenvalues of the coefficient matrix. A VAR(p) process can be written as a VAR(1) process,

$$\xi_t = A \xi_{t-1} + v_t, \quad (13.41)$$

with:

$$\xi_t = \begin{bmatrix} \mathbf{y}_t \\ \vdots \\ \mathbf{y}_{t-p+1} \end{bmatrix}, A = \begin{bmatrix} A_1 & A_2 & \cdots & A_{p-1} & A_p \\ I & 0 & \cdots & 0 & 0 \\ 0 & I & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & I & 0 \end{bmatrix}, v_t = \begin{bmatrix} \mathbf{u}_t \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}, \quad (13.42)$$

where the stacked vectors ξ_t and v_t have dimension $(Kp \times 1)$ and the matrix A has dimension $(Kp \times Kp)$. If the moduli of the eigenvalues of A are less than 1, then the VAR(p) process is stable.

¹ Without loss of generality, deterministic regressors are suppressed in the following notation. Furthermore, vectors are assigned by bold lower-case letters and matrices by capital letters. Scalars are written as lower-case letters, possibly subscripted.

For a given sample of the endogenous variables $\mathbf{y}_1, \dots, \mathbf{y}_T$ and sufficient pre-sample values $\mathbf{y}_{-p+1}, \dots, \mathbf{y}_0$, the coefficients of a VAR(p) process can be estimated efficiently by least squares applied separately to each of the equations.

Once a VAR(p) model has been estimated, further analysis can be carried out. A researcher might—indeed, should—be interested in diagnostic tests, such as testing for the absence of autocorrelation, heteroscedasticity, or non-normality in the error process. He/she might be interested further in causal inference, forecasting, and/or diagnosing the empirical model's dynamic behavior—impulse response functions (IRFs) and forecast error variance decomposition. The latter two are based upon the Wold moving average decomposition for stable VAR(p) processes, which is defined as

$$\mathbf{y}_t = \Phi_0 \mathbf{u}_t + \Phi_1 \mathbf{u}_{t-1} + \Phi_2 \mathbf{u}_{t-2} + \dots \quad (13.43)$$

with $\Phi_0 = I_K$; Φ_s can be computed recursively from

$$\Phi_s = \sum_{j=1}^s \Phi_{s-j} A_j \text{ for } s = 1, 2, \dots \quad (13.44)$$

with $A_j = 0$ for $j > p$.

Finally, forecasts for horizons $h \geq 1$ of an empirical VAR(p) process can be generated recursively from

$$\mathbf{y}_{T+h|T} = A_1 \mathbf{y}_{T+h-1|T} + \dots + A_p \mathbf{y}_{T+h-p|T}, \quad (13.45)$$

where $\mathbf{y}_{T+j|T} = \mathbf{y}_{T+j}$ for $j \leq 0$. The forecast error covariance matrix is given as:

$$\text{COV} \left(\begin{bmatrix} \mathbf{y}_{T+1} - \mathbf{y}_{T+1|T} \\ \vdots \\ \mathbf{y}_{T+h} - \mathbf{y}_{T+h|T} \end{bmatrix} \right) = \begin{bmatrix} I & 0 & \dots & 0 \\ \Phi_1 & I & & 0 \\ \vdots & & \ddots & 0 \\ \Phi_{h-1} & \Phi_{h-2} & \dots & I \end{bmatrix} (\Sigma_u \otimes I_h) \begin{bmatrix} I & 0 & \dots & 0 \\ \Phi_1 & I & & 0 \\ \vdots & & \ddots & 0 \\ \Phi_{h-1} & \Phi_{h-2} & \dots & I \end{bmatrix}^T, \quad (13.46)$$

and the matrices Φ_i are the empirical coefficient matrices of the Wold moving average representation of a stable VAR(p) process as shown above. The operator \otimes is the Kronecker product.

Structural vector autoregressive models

Recall the definition of a VAR(p) process, in particular (13.39). A VAR(p) process can be interpreted as a reduced-form model. An SVAR model is its structural form and is defined as

$$A\mathbf{y}_t = A_1^* \mathbf{y}_{t-1} + \dots + A_p^* \mathbf{y}_{t-p} + B\boldsymbol{\epsilon}_t. \quad (13.47)$$

It is assumed that the structural errors, ϵ_t , are white noise and the coefficient matrices $A_i^*, i = 1, \dots, p$, are structural coefficients that differ in general from their reduced form counterparts. To see this, consider the equation that results from left-multiplying (13.47) by the inverse of A :

$$\begin{aligned} \mathbf{y}_t &= A^{-1}A_1^*\mathbf{y}_{t-1} + \dots + A^{-1}A_p^*\mathbf{y}_{t-p} + A^{-1}B\epsilon_t \\ \mathbf{y}_t &= A_1\mathbf{y}_{t-1} + \dots + A_p\mathbf{y}_{t-p} + \mathbf{u}_t. \end{aligned} \quad (13.48)$$

An SVAR model can be used to identify shocks and trace these out by employing impulse response analysis and/or forecast error variance decomposition to impose restrictions on the matrices A and/or B . Incidentally, although an SVAR model is a structural model, it departs from a reduced-form VAR(p) model and only restrictions for A and B can be added. It should be noted that the reduced-form residuals can be retrieved from an SVAR model by $\mathbf{u}_t = A^{-1}B\epsilon_t$, and its variance-covariance matrix by $\Sigma_u = A^{-1}BB^\top A^{-1}$.

Depending on the restrictions imposed, three types of SVAR models can be distinguished:

- In the A model, B is set to I_K
(minimum number of restrictions for identification is $K(K - 1)/2$).
- In the B model, A is set to I_K
(minimum number of restrictions to be imposed for identification is the same as for the A model).
- In the AB model, restrictions can be placed on both matrices
(minimum number of restrictions for identification is $K^2 + K(K - 1)/2$).

The parameters are estimated by minimizing the negative of the concentrated log-likelihood function:

$$\begin{aligned} \ln \mathfrak{L}_c(A, B) &= -\frac{KT}{2} \ln(2\pi) + \frac{T}{2} \ln |A|^2 - \frac{T}{2} \ln |B|^2 \\ &\quad - \frac{T}{2} \text{tr}(A^\top B^{-1} B^{-1} A \tilde{\Sigma}_u), \end{aligned} \quad (13.49)$$

where $\tilde{\Sigma}_u$ denotes an estimate of the reduced-form variance-covariance matrix for the error process.

Vector error correction models

Consider again the VAR in (13.39):

$$\mathbf{y}_t = A_1\mathbf{y}_{t-1} + \dots + A_p\mathbf{y}_{t-p} + \mathbf{u}_t. \quad (13.50)$$

There are two vector error correction specifications. The first is given by

$$\Delta\mathbf{y}_t = \alpha\beta^\top \mathbf{y}_{t-p} + \Gamma_1\Delta\mathbf{y}_{t-1} + \dots + \Gamma_{p-1}\Delta\mathbf{y}_{t-p+1} + \mathbf{u}_t \quad (13.51)$$

with

$$\Gamma_i = -(I - A_1 - \cdots - A_i), i = 1, \dots, p - 1, \quad (13.52)$$

and

$$\Pi = \alpha\beta^\top = -(I - A_1 - \cdots - A_p). \quad (13.53)$$

The Γ_i matrices contain the cumulative long-run impacts, hence this VECM specification is referred to as the long-run form. The other specification is given as follows and is in common use:

$$\Delta y_t = \alpha\beta^\top y_{t-1} + \Gamma_1 \Delta y_{t-1} + \cdots + \Gamma_{p-1} y_{t-p+1} + u_t \quad (13.54)$$

with

$$\Gamma_i = -(A_{i+1} + \cdots + A_p) \quad i = 1, \dots, p - 1. \quad (13.55)$$

Equation (13.53) applies to this specification too. Hence, the Π matrix is the same as in the first specification. Since the Γ_i matrices in (13.54) now measure transitory effects, this specification is referred to as the transitory form. In the case of co-integration the matrix $\Pi = \alpha\beta^\top$ is of reduced rank. The dimensions of α and β are $K \times r$, and r is the co-integration rank, denoting how many long-run relationships exist between the elements of y_t . The matrix α is the loading matrix, and the coefficients of the long-run relationships are contained in β .

Structural vector error correction models

Consider again the VECM in (13.54). It is possible to apply the same SVAR model reasoning to SVEC models, in particular when the equivalent level VAR representation of the VECM is used. However, the information contained in the co-integration properties of the variables is not then used for identifying restrictions on the structural shocks. Hence, typically a B model is assumed when an SVEC model is specified and estimated:

$$\Delta y_t = \alpha\beta^\top y_{t-1} + \Gamma_1 \Delta y_{t-1} + \cdots + \Gamma_{p-1} y_{t-p+1} + B\epsilon_t, \quad (13.56)$$

where $u_t = B\epsilon_t$, and $\epsilon_t \sim \mathcal{N}(\mathbf{0}, I_K)$. In order to exploit this information, one considers the Beveridge–Nelson moving average representation of the variables y_t if they follow the VECM process as in (13.54):

$$y_t = \Xi \sum_{i=1}^t u_i + \sum_{j=0}^{\infty} \Xi_j^* u_{t-j} + y_0^*. \quad (13.57)$$

The variables contained in y_t can be decomposed into a part that is integrated of order 1 and a part that is integrated of order 0. The first term on the right-hand side of (13.57) is referred to as the “common trends” of the system, and this term drives the system y_t . The middle term is integrated of order 0 and it is assumed that the infinite sum is bounded, that is, the Ξ_j^* converge to zero as $j \rightarrow \infty$. The j initial values are captured by y_0^* . For the modelling of SVEC the interest centers on the common trends

in which the long-run effects of shocks are captured. The matrix is of reduced rank $K - r$, where r is the number of stationary co-integration relationships. The matrix is defined as

$$\Xi = \beta_{\perp} \left[\alpha_{\perp}^T \left(I_K - \sum_{i=1}^{p-1} \Gamma_i \right) \beta_{\perp} \right]^{-1} \alpha_{\perp}^T. \quad (13.58)$$

Because of its reduced rank, only $K - r$ common trends drive the system. Therefore, knowing the rank of Π , one can conclude that at most r of the structural errors can have a transitory effect. This implies that at most r columns can be set to zero. One can combine the Beveridge–Nelson decomposition with the relationship between the VECM error terms and the structural innovations. The common trends term is then $\Xi B \sum_{t=1}^{\infty} \epsilon_t$, and the long-run effects of the structural innovations are captured by the matrix B . The contemporaneous effects of the structural errors are contained in the matrix B . As in the case of SVAR models of type B , for locally just-identified SVEC models one needs $\frac{1}{2}K(K - 1)$ restrictions. The co-integration structure of the model provides $r(K - r)$ restrictions on the long-run matrix. The remaining restrictions can be placed on either matrix; at least $r(r - 1)/2$ of them must be imposed directly on the current matrix B .

13.3 The Black–Litterman approach

In Chapter 10 it was stated that the weight solution is sensitive to the input parameters for Markowitz solutions. In that chapter the focus was on robust estimation techniques for the dispersion matrix to rectify this problem. The sensitivity of the weight solution to the assumed return vector was neglected, but references to the literature were provided.

In this and the next section, approaches are presented that deal indirectly with this issue. The first of these is the Black–Litterman (BL) model (see Black and Litterman 1990, 1991, 1992; He and Litterman 2002).

This consists of five building blocks, with a Bayesian estimation of equilibrium and expected returns at its core. The latter lets a portfolio manager directly include his return expectations into the portfolio solution and it is this characteristic that has led to the ubiquitous application of this method for tactical asset allocation. It will be assumed that the returns are joint normally distributed. The five building blocks are

- the capital asset pricing model (CAPM; Sharpe 1964),
- reverse optimization (Sharpe 1974),
- mixed (Bayesian) estimation,
- the concept of a universal hedge ratio (Black 1989),
- mean-variance optimization (Markowitz 1952);

these will be discussed in turn.

The CAPM is used as an equilibrium model, where the supply and demand of assets are equated. Recall Figure 5.1, in which this point is depicted as the locus where the capital market line is tangent to the curve of efficient portfolios. At equilibrium the efficient market portfolio consists, therefore, of an efficient allocation of risky assets and the risk-free asset. This point is further characterized by the market capitalization of each asset. The relative market capitalizations represent the assets' weights at equilibrium and are denoted by ω_{MKT} . If one assumes that the unconstrained portfolio optimization problem can be stated as

$$\arg \max_{\omega \in \Omega} \omega' \mu - \frac{\lambda}{2} \omega' \Sigma \omega, \quad (13.59)$$

where the expected excess returns, μ , are balanced against the portfolio's riskiness, $\omega' \Sigma \omega$, for a risk aversion parameter, $\lambda > 0$, then the optimal (equilibrium) weight vector is given by

$$\omega_{OPT} = [\lambda \Sigma]^{-1} \mu. \quad (13.60)$$

For arbitrary expectations for the assets' excess returns, the allocation ω_{OPT} will generally differ from ω_{MKT} , only coinciding if μ is equal to the excess return expectations implied by the relative market capitalizations. By reverse optimization, that is, left-multiplying (13.60) by $\lambda \Sigma$, one obtains the implied excess returns, π , for a given market capitalization structure:

$$\pi = \lambda \Sigma \omega_{MKT}. \quad (13.61)$$

The implied equilibrium returns, the vector elements of π , are employed as the market neutral starting point in the BL model, that is, the returns are a priori distributed with an expected mean of π . In the absence of any views on the future values of the assets, an investor is therefore best advised to allocate his wealth according to ω_{MKT} , thus implicitly sharing the market expectation for the excess returns as stated in (13.61). The question of how the investor's expectations about the excess returns can be included in this model arises next. In the BL model the inclusion of these "views" is accomplished by deriving the Bayesian posterior distribution for the returns. Let A denote the (multivariate) return expectation(s) and B the implied excess returns at equilibrium. The joint likelihood function $P(A, B)$ can then be expressed by Bayes' theorem as

$$P(A, B) = P(A|B)P(B) \quad (13.62a)$$

$$= P(B|A)P(A), \quad (13.62b)$$

and because one is interested in the conditional return distribution in terms of the equilibrium returns, one obtains, from the right-hand equality in (13.62),

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \quad (13.63)$$

If the expected excess returns are stated as $\mathbf{E}(\mathbf{r})$, then (13.63) is written as

$$P(\mathbf{E}(\mathbf{r})|\boldsymbol{\pi}) = \frac{P(\boldsymbol{\pi}|\mathbf{E}(\mathbf{r}))P(\mathbf{E}(\mathbf{r}))}{P(\boldsymbol{\pi})}. \quad (13.64)$$

A feature of the BL model is that expectations do not have to be supplied for all market assets and/or that these return expectations can be either expressed as absolute or relative return targets. Therefore, the views can be modelled as:

$$P\mathbf{E}(\mathbf{r}) \sim \mathcal{N}(\mathbf{q}, \Omega), \quad (13.65)$$

where P denotes a $(K \times N)$ pick matrix, \mathbf{q} is the $(K \times 1)$ vector of absolute/relative return expectations, and Ω a $(K \times K)$ diagonal matrix expressing the uncertainties about each view. Given the normality assumption, the latter assumption implies independence between the expressed views. The implied independence of views is awkward. It does not make much sense to assume that return forecasts for various assets are formed independently of each other, in particular when these are derived from a multivariate statistical model, so this assumption will be relaxed in the next section on copula opinion pooling, but retained for the time being. Finally, the conditional equilibrium returns for the expected returns are distributed as

$$\boldsymbol{\pi}|\mathbf{E}(\mathbf{r}) \sim \mathcal{N}(\mathbf{E}(\mathbf{r}), \tau \Sigma), \quad (13.66)$$

where a homogeneity assumption for the market participants has been exploited such that $\mathbf{E}(\boldsymbol{\pi}) = \mathbf{E}(\mathbf{r})$ and τ is a scalar for scaling the uncertainty around the implicit market equilibrium excess returns.

The posterior probability density function of the conditional random variable $\mathbf{r} = \mathbf{E}(\mathbf{r})|\boldsymbol{\pi}$ is multivariate normally distributed with parameters

$$\mathbf{E}(\mathbf{r}) = [(\tau \Sigma)^{-1} + P' \Omega^{-1} P]^{-1} [(\tau \Sigma)^{-1} \boldsymbol{\pi} + P' \Omega^{-1} \mathbf{q}], \quad (13.67)$$

$$\text{VAR}(\mathbf{r}) = [(\tau \Sigma)^{-1} + P' \Omega^{-1} P]^{-1}. \quad (13.68)$$

In essence, (13.67) is a risk-scaled weighted average of the market equilibrium returns and the view returns. The latter two are scaled by the inverse of the variance-covariance matrix, $(\tau \Sigma)^{-1}$, and by the confidence of the views, $P' \Omega^{-1}$, respectively. The relative importance of the market equilibrium return and the view return expressed in the k th row of P is determined by the ratio $\Omega_{kk}/\tau = \mathbf{p}'_k \Sigma \mathbf{p}_k$. The term on the right-hand side of this equation is the variance of the view portfolio. This equation can also be used to determine implicit confidence levels Ω_{kk} for a given value of τ . If this route is followed, then the point estimates for $\mathbf{E}(\mathbf{r})$ are unaffected by the chosen value for τ (see He and Litterman 2002). In the limiting case of no expressed views ($P = 0$), the point estimates for $\mathbf{E}(\mathbf{r})$ are the equilibrium returns. At the other extreme, when the views are expressed without prediction error, the point estimates for $\mathbf{E}(\mathbf{r})$ are identical to the view expectations if specified and otherwise are equal to the market returns. If these point estimates are used in an unconstrained Markowitz-type portfolio optimization, then only the

weights will differ from the relative market capitalization ones, for which a return expectation has been formulated. To summarize, the BL method, in its original form, consists of a Bayesian-derived vector of expected returns, which is then utilized in a Markowitz-type portfolio optimization. Of course, the Bayesian estimates can also be used in any other kind of portfolio problem formulation, for example, with respect to optimizing a downside-risk measure.

13.4 Copula opinion and entropy pooling

13.4.1 Introduction

Copula opinion pooling (COP) and entropy pooling (EP) can both be viewed as extensions to the original BL model. These methods are due to Meucci (2006a, b, 2010a, b). The BL model and the two extensions have in common that a synthesis between an a priori distribution and a prior view distribution is accomplished in order to yield a posterior distribution. The major difference between these two methods and the BL model is in how these two ingredients are supplied or specified, and that ordinarily the posterior distributions according to the COP and EP models are retrieved by means of a Monte Carlo simulation. In the rest of this section the COP and EP models are presented.

13.4.2 The COP model

In the BL model the a priori distribution for the returns is derived from an equilibrium model, namely the CAPM, and the views are expressed as either absolute or relative return forecasts. Furthermore, the BL model assumes that the returns follow a multivariate normal distribution. These model constituents are too restrictive, and the stylized facts of financial returns conflict with the normal assumption. In contrast, the COP method is much more flexible and less demanding with respect to its underlying assumptions. First, the a priori distribution need not be defined in terms of the returns on a set of securities, but can also be given in terms of risk factors or a set of any other random variables, and it is not assumed that these random variables follow a multivariate normal distribution function. Instead, the prior distribution can be represented by a copula and the marginal distributions can be of any kind. Second, the views are expressed as cumulative distribution functions and as such are not confined to absolute or relative return forecasts. However, confidence levels, $c_k, k = 1, \dots, K$, for the views have to be provided by the user, similar to the diagonal elements of the uncertainty matrix in the BL model. In addition, a “pick” matrix P is a required input. The cost of this flexible approach is that in general no closed-form solutions can be provided, but the synthesis between the two ingredients is accomplished by means of a Monte Carlo simulation.

Deriving the posterior distribution by the COP method involves the following five steps (see Meucci 2006a):

1. A rotation of the prior distribution into the views' coordinates.
2. Computation of the views' cumulative distribution function and the market implied prior copula.
3. Computation of the marginal posterior cumulative distribution functions of each view.
4. Computation of the joint posterior distribution of the views.
5. Computation of the joint posterior realizations of the market distributions.

Each of these five steps is now described in more detail. It is assumed that a Monte Carlo simulation, \mathcal{M} , for the N securities of size J is given. These simulated values can be drawn, for instance, from a Student's t copula, and any of the proposed distributions in Chapter 6 could have been utilized as marginal models. But the Monte Carlo values contained in \mathcal{M} could also have been generated from a multiple time series model as outlined in the previous sections of this chapter.

In the first step, the simulated values contained in the $(J \times N)$ matrix \mathcal{M} are mapped into the coordinate system pertinent to the views, \mathcal{V} . This is accomplished by means of a simple matrix multiplication, $\mathcal{V} = \mathcal{M}\bar{P}'$, where \bar{P} denotes the (invertible) pick matrix. There is a qualitative difference between this pick matrix and the one used in the BL model. In the latter approach, the matrix P consists of K rows in which the securities for which views are expressed are selected, hence its dimension is $(K \times N)$. In the COP model, however, the dimension is $(N \times N)$. The first K rows of (\bar{P}) are the views P , and the remaining $N - K$ rows are filled by its matrix complement, P^\perp with dimension $((N - K) \times N)$, hence $\bar{P} = (P|P^\perp)'$.

In the second step, the first K columns of \mathcal{V} are sorted in ascending order, which results in the $(J \times K)$ matrix \mathcal{W} , the prior cumulative distribution functions. The copula of the views, \mathcal{C} , are the order statistics of \mathcal{W} , that is, $\mathcal{C}_{j,k} = F_k(\mathcal{W}_{j,k}) = j/(J + 1)$.

Next, the marginal posterior cumulative distribution functions, \mathcal{F} , for each view are computed as a weighted average,

$$\mathcal{F}_{j,k} = c_k \hat{F}_k(\mathcal{W}_{j,k}) + (1 - c_k) \frac{j}{J + 1}. \quad (13.69)$$

In the fourth step, the joint posterior distribution of the views $\tilde{\mathcal{V}}$ (the quantile function) is retrieved by means of interpolation between the grid points $(\mathcal{F}_{\cdot,k}, \mathcal{W}_{\cdot,k})$.

In the final step, the joint posterior simulated realizations of the market distribution are recovered by inversion of the equation in the first step. Because the dimension of $\tilde{\mathcal{V}}$ is only $(J \times K)$, the matrix is amended from the right by the rightmost $(N - K)$ columns of \mathcal{V} , and hence $\tilde{\mathcal{M}} = \tilde{\mathcal{V}}(\bar{P}')^{-1}$. This set of J joint posterior realizations for the N assets can then be utilized in the portfolio optimization problem at hand.

13.4.3 The EP model

The EP model is a further generalization of the BL and COP models. It has in common with the COP model an arbitrary model for the market prior distribution and that the

COP and EP models are based on a Monte Carlo simulation of the market's prior distribution. A qualitative difference between the COP and EP models is the kind of views expressed. In the former model class views on the dispersion, volatility, or correlations of and between securities and/or the expression of nonlinear views were not feasible. This limitation is rectified for EP models. A concise description of how to formulate these views is provided in Meucci (2010b, Section 2 and Appendix A.2). A further material difference between the COP and EP models is the form of the posterior distribution. In the latter model, the same simulated values used in the modelling of the market's prior distribution are utilized, but with altered probabilities assigned to these random variables.

In the following, the steps for deriving the posterior distribution are presented. As stated in the previous subsection, it is assumed that the market for N assets can be modelled by a set of random variables X which follow an a priori joint distribution, f_M . In its simplest form, X represents a sample of the securities' returns, but is not limited to these market factors. When the data has been fitted to the assumed distribution model, simulated values for this market distribution can be obtained by means of Monte Carlo, and hence one obtains a matrix \mathcal{M} of dimension $(J \times N)$ as in the COP model. The columns in \mathcal{M} are the marginal prior distributions and the rows are simulated outcomes for the market factors. Associated with each of these outcomes $\mathcal{M}_{j, \cdot}$ is a probability p_j , and most easily these J probabilities are set equal to $1/J$, that is, each Monte Carlo draw is treated as being equally likely. Hence, the $(J \times 1)$ probability vector \mathbf{p} has as elements the reciprocal of the size of the Monte Carlo simulation.

The second input is the formulation of the views. This is now a $(K \times 1)$ vector of function values $V = \mathbf{g} = (g_1(X), \dots, g_k(X), \dots, g_K(X))'$ with joint distribution f_v , a priori. The functions g_k , $k = 1, \dots, K$, can be nonlinear in nature. The implied distribution of these views is then empirically approximated by the market simulations \mathcal{M} according to

$$\mathcal{V}_{j,k} = g_k(\mathcal{M}_{j,1}, \dots, \mathcal{M}_{j,N}) \quad (13.70)$$

for $k = 1, \dots, K$ and $j = 1, \dots, J$, such that a $(J \times K)$ matrix results containing the empirical distribution of the views implied by the Monte Carlo simulation for the market.

Two panels \mathcal{M} and \mathcal{V} have now been created, and the question is how these can be combined to retrieve the posterior distribution of the market that obeys the views. This problem is resolved in three steps. First, the views are expressed in terms of a set of linear inequality constraints, $\mathbf{a}_{\text{lower}} \leq A\bar{\mathbf{p}} \leq \mathbf{a}_{\text{upper}}$, where now the probability vector $\bar{\mathbf{p}}$ is treated as the objective variable in the ensuing optimization. The lower and upper bounds $(\mathbf{a}_{\text{lower}}, \mathbf{a}_{\text{upper}})$ and the matrix A are deduced from \mathcal{V} .

In the second step, the relative entropy—loosely speaking, a distance measure between distributions—is minimized.² The (discrete) objective function is defined as

$$\text{RE}(\bar{\mathbf{p}}, \mathbf{p}) = \sum_{j=1}^J \bar{p}_j [\log(\bar{p}_j) - \log(p_j)], \quad (13.71)$$

² A detailed exposition of the entropy concept is given in Golan et al. (1996).

and hence the probabilities of the posterior distribution under the assumption of perfect foresight are obtained by evaluating

$$\bar{\mathbf{p}} = \arg \min_{a_{\text{lower}} \leq A\bar{x} \leq a_{\text{upper}}} \text{RE}(\mathbf{x}, \mathbf{p}). \quad (13.72)$$

That is, the probabilities $\bar{\mathbf{p}}$ under the assumption of perfect foresight are determined such that the resulting posterior distribution is least distorted by the distribution of the views, or, put differently, the view distribution is made most akin to the reference model. At first sight, the optimization stated in (13.72) seems to be a numerically demanding exercise, given the dimension of the target variable. However, in Meucci (2010b, Appendix A.3) the dual form of this mathematical program is derived from the Lagrangian function, which results in a convex program with linear constraints and with the size of the objective variable equal to the number of views K .

In the third step, the empirical confidence-weighted posterior distribution, $(\mathcal{M}, \mathbf{p}_c)$, is determined as in the COP model according to

$$\mathbf{p}_c = (1 - c)\mathbf{p} + c\bar{\mathbf{p}}. \quad (13.73)$$

The empirical posterior distribution thus constructed can then be utilized in the portfolio optimization problem at hand.

13.5 Synopsis of R packages

13.5.1 The package BLCOP

The package **BLCOP** was briefly discussed in Section 9.4.1 with regard to its capabilities in the modelling of copulae. We will now present its functions and procedures for the BL method.

To handle the views in the BL model an S4 class **BLViews** is available. The class definition consists of four slots: `p` for the pick matrix, `qv` for the expressed returns (absolute and relative), `confidences` for the expressed subjective confidences of the views, and `assets` for the names of the assets contained in the BL model. Objects of this class can be created by the generating function `BLViews()`. They have methods for displaying the views (method `show()`), for deleting views (method `deleteViews()`), and for adding views (method `addViews()`). A convenient function for setting up the pick matrix is `newPMatrix()`. This function will return by default a zero-filled matrix with number of rows equal to the value of the argument `numViews` and number of columns equal to the length of `assetNames`, this argument being a character vector of the assets included in the universe. The values contained in the pick matrix or in the vector of returns and confidences can be altered with the assignment functions `PMatrix()`, `qv()`, and `confidences()`, respectively. Furthermore, functions for inspecting certain aspects from objects of class `BLViews()` are: `assetSet()` for the asset names contained in the universe; `viewMatrix()` for the pick matrix amended on the right by the absolute/

relative return expectations of each view; `PMatrix()` for the pick matrix only; and `confidences()` for the associated confidence levels of each view.

The cornerstone function for estimating the posterior distribution in the BL model is `posteriorEst()`. This function has as arguments `views` for the `BLViews` object, `mu` for the equilibrium returns, `tau` for the scaling parameter of the equilibrium distribution (default value 0.5), `sigma` for the variance-covariance matrix of returns, and `kappa` with default value zero (if set greater than zero, then the confidence levels are replaced by $\kappa P \Sigma P'$). The function returns an object of `S4` class `BLResult`. Alternatively, this kind of object can be created by calling the wrapper function `BLPosterior()`. The difference between the two functions is that the latter computes the prior distribution parameters internally. The user has to provide the matrix of asset returns, the returns for the market index, and the value of the risk-free rate. A function other than `cov()` for estimating the dispersion matrix can be set by the argument `covEstimator()`. Of course, values for the other arguments pertinent to the BL model must also be provided—`views`, `tau`, and `kappa`. Objects of class `BLResult` consist of seven slots: `views`, which contains the `BLViews` object provided; `tau` for the indeterminacy of the equilibrium distribution; four slots for the prior and posterior distribution parameters, `priorMean`, `priorCovar`, `posteriorMean`, and `posteriorCovar`; and a logical indicator for whether `kappa` has been set to a value greater than zero. The parameters of the posterior distribution can be extracted as a `list` from objects of this class with the function `posteriorMeanCov()`. In addition to this extractor function, the package provides methods for displaying the prior and posterior densities (method `densityPlots()`) and for using the estimated BL posterior distribution parameters directly in portfolio optimization (method `optimalPortfolios.fPort()`).

In **BLCOP**, two functions/methods for determining an optimal portfolio allocation for given prior/posterior distributions are provided. The function `optimalPortfolios()` takes as argument `result` an object of class `BLResult`, and as argument `optimizer` the name of the optimizing function. By default, this argument is set equal to the internal function `.optimalWeights.simpleMV()`, which performs a Markowitz optimization by calling the function `solve.QP()` contained in the package **quadprog**. The the body of the internal function can be displayed by utilizing the `:::` operator. In addition to the estimates for the expected returns and their variance-covariance matrix, it is specified with an additional argument `constraints` which is designed to hold a named list object with elements pertinent to the arguments of `solve.QP()`. If this argument is left unspecified, then a portfolio with a budget and non-negativity constraints is assumed. If a different function than the default is to be employed, then the first two arguments must be the mean and the variance-covariance matrix (in that order). Additional arguments can be specified and are passed down to the call of `optimizer` by means of the ellipsis argument in `optimalPortfolios()`. Whether and how bar-plots of the prior and posterior allocations are returned can be controlled by the logical arguments `doPlot` and `beside`. As stated in the previous paragraph, optimal portfolio allocations can also be determined by the method `optimalPortfolio.fPort()`. This method is a wrapper around the portfolio optimization facilities contained in the package

fPortfolio. In addition to the `BLResult` object, this function has arguments for the portfolio specification (class `fPFOLIOSPEC` with generating function `portfolioSpec()`, both defined and contained in the package `fPortfolio`), a valid character string for the constraints on the portfolio weights (e.g., "LongOnly"), and the kind of portfolio optimization to be carried out. The latter argument is set by default to "minriskPortfolio", but any other optimizing function in `fPortfolio` can be provided. Both optimization functions will return a named `list` with two elements for the results according to the prior (`priorPfolioWeights`) and posterior (`postPfolioWeights`) distributions. If the `S4` method has been employed the list elements will be objects of formal class `fPORTFOLIO`, and otherwise vectors containing the optimal allocations.

With respect to the COP model a set of `S4` classes and methods similar to the BL model is provided. This implementation is aided by the functions `mvdistribution()` and `distribution()` for modelling the market and the view distributions, respectively. Pertinent to both functions is the argument `RName` for specifying the `R` suffix name corresponding to the desired distribution, for example "mt" for the multivariate Student's *t* distribution or "pois" for the Poisson distribution. The former function returns an object of `S4` class `marketDistribution` and the latter an `S4` class object `distribution`. Both classes have two slots: `RName` for the name of the assumed distribution and `parameters` for the provided (estimated) parameters of the distribution. The views in the COP model are created by invoking the function `COPViews()`. Here, the pick matrix must be provided as argument `pickMatrix`, the views as a `list` object with elements of `S4` class `distribution`, a vector containing the confidence of each view (argument `confidences`), and a character vector of the asset names included in the data set (argument `assetNames`). This creator function returns an object of formal class `COPViews`. Similar to the BL model, methods for manipulating this kind of object are available: `createCOPViews()`, `addCOPViews()`, and `deleteViews()`. An estimate of the posterior distribution is returned by the function `COPPosterior()` (`S4` class object `COPResult`). The simulations drawn from the posterior distribution are contained in the slot `posteriorSims`, which can be used in an ensuing portfolio optimization. For visualizing the posterior densities, the function `densityPlots()` is made available.

13.5.2 The package `dse`

The focus of the package `dse` is on model implementations for univariate and multivariate time series (see Gilbert 1993, 1995, 2000, 2009).³ The model classes implemented are primarily ARMA models and dynamic systems of equations expressed in state-space form. The package is hosted on CRAN and contained in the "Econometrics," "Environmetrics," "Finance," and "TimeSeries" Task Views. Development

³ The package `EvalEst` complements `dse` by providing methods for analyzing the properties of different estimators (see Gilbert 2015a).

versions are hosted on R-Forge. It employs S3 classes and methods. The more demanding computations with respect to ARMA modelling, Kalman filtering, and the estimation and simulation of these models are interfaced from FORTRAN routines. A vignette is shipped with it to illustrate the utilization of these model classes and methods. Furthermore, demo files are available to elucidate data handling and the definition of model structure, as well as estimation, simulation, and forecasting.

At the center of the package are the three S3 classes `TSdata`, `TSmodel`, and `TSestModel`. The first is for handling the data set, the second for defining the model structure, and the last for containing information on an estimated model. The generating function for objects of class `TSdata` is `TSdata()`. This function can also be used to extract the data part from objects of class `TSestModel`. The data class is based on the time series class `tframe` contained in the package of the same name (see Gilbert 2015b). Objects of this class contain either the time series as such or the user can provide data sets to be used as input (ordinarily the data to be used on the right-hand side of a time series model) and output (ordinarily the data to be used on the left-hand side of a time series model). The time series used as either the input or output series can also be queried from a `TSdata` object or assigned to it by the methods `inputData()` or `outputData()`. Methods defined for these objects are `print()`, `plot()`, `tfplot()`, `summary()`, `as.TSdata()`, and `is.TSdata()`, with obvious meanings. It is also possible to assign different names to the series than the column names or query the series names by means of the methods `seriesNames()`, `seriesNamesInput()`, and `seriesNamesOutput()`. The column location of a series used in either the input or output data set can be retrieved by either of the `nseriesInput()` or `nseriesOutput()` methods. Information on when the sample starts and ends, the number of observations, and the frequency of the time series can be queried by the methods:

- `start()`, `startInput()`, `startOutput()`;
- `end()`, `endInput()`, `endOutput()`;
- `Tobs()`, `TobsInput()`, `TobsOutput()`;
- `frequency()`, `frequencyInput()`, `frequencyOutput()`.

The equality of two objects with class attribute `TSdata` can be assessed by the method `testEqual()`, and whether the dimensions agree by the method `checkConsistentDimensions()`. To manipulate the sample span or bind two objects together, the methods defined in the package `tframe`—`tframed()`, `tfwindow()`, `tbond()`, and `trimNA()`—are available, as well as a `combine()` method for joining two `TSdata` objects. Further, the `window()` method defined in the base package `stats` can also be applied to `TSdata` objects. Incidentally, the `acf()` method for returning the autocorrelations defined in package `stats` can also be applied to `TSdata` objects, and the scaling of the `TSdata` object can be accomplished by a method definition for the generic function `scale()` defined in the base R distribution. Percentage changes of time series data can be

swiftly computed by the `percentChange()` method. This is a rather detailed listing of methods defined for the time series class employed in the package `dse`. However, it should be noted that these methods are also available for objects of class `TSestModel`, where applicable.

Before giving a more detailed description of this class, the second cornerstone class, `TSmodel`, and its associated methods will be discussed. First, objects of this class can be created by calling the generating function `TSmodel()`, which takes as argument the definition of either an ARMA or a state-space model. The two supported model classes, which themselves return objects inheriting from `TSmodel`, are `ARMA()` (class `ARMA`) and `SS()` (class `SS`), respectively. The coefficients can be recovered and/or assigned by the `coef()` method; the generic definition is imported from the base package `stats`. The function `TSmodel()` can also be used to extract the fitted model from an object of class `TSestModel`. For such objects, `print()` and `summary()` methods are available. Whether an object has one of the class attributes `TSmodel`, `ARMA`, or `SS` can be checked by the `is.foo()` methods, where `foo` is a placeholder for the class name. The equality of two time series models can be tested by means of `testEqual()`. To assess the validity of a model, the methods `stability()` and `roots()` can be used. A `plot()` method for objects returned by the latter function is also available, and with the function `addPlotRoots()` these can be superimposed on an existing graphics device. The representation of a certain model structure can be transformed from an ARMA to a state-space specification, and vice versa, by means of the methods `toSS()` and `toARMA()`, respectively. A model structure can be evaluated for a given time series object `TSdata` by the method `l()`. A state-space model can also be evaluated by applying Kalman filtering to a given state-space structure and time series object by calling the `smoother()` method. The states of a state-space model can also be returned by calling `state()` and setting the argument `smoother` to `TRUE`, otherwise this function will return the state information for a fitted state-space model. Finally, simulated values for a given ARMA or state-space model structure can be generated by calling the `simulate()` method.

So far, classes and methods for handling time series models with a specified structure have been presented. This paragraph discusses the final class, `TSestModel`, and how estimates for the unknown coefficients can be determined. A given data set is fitted to a time series model by calling the function `estfoo()`, where `foo` is a placeholder for the model class and/or the estimation method. These functions return an object with class attribute `TSestModel`, for which numerous evaluation, analytical, and extraction methods are provided. The currently implemented models and estimators are:

- `estVARXar()`, which estimates the coefficients of a VAR with optional exogenous variables by means of the Yule–Walker equations;
- `estVARXls()`, which estimates the coefficients of a VAR with optional exogenous variables by means of OLS;
- `estSSMittnik()`, which estimates the coefficients of a nested state-space model with Markov switching;

- `estMaxLik()`, for ML-based estimation of `TSmodel` objects;
- `estSSfromVARX()`, which converts an estimated VAR model with optional exogenous variables to its state-space representation.

In addition to these functions, first, a generic function `estimateModels()` is available where the kind of model/estimation method is provided by the argument `estimation.methods`, and second, functions `estBlackBoxn()` with $n = 1, \dots, 4$ are available as wrapper functions for `estVARXls()` where certain arguments and restrictions on the parameter space are preset. The function `estBlackBox()` will call each consecutively and selects the model specification that is most appropriate. Similar to this function is `bestTSEstModel()`, which selects the best model from a list of fitted models. Objects of class `TSEstModel` can then be further analyzed by methods `residuals()` for retrieving the residuals, `coef()` for displaying the estimates, `stability()` and `roots()` for assessing the model's empirical stability, and `simulate()` for simulating trajectories based on the estimates. The fitted values can be recovered by applying the `l()` method and the state information can be recovered by the `smoother()` method. The residuals can be checked for their white noise characteristic by calling `checkResiduals()` and the information criteria are returned by the functions `informationTests()` and `informationTestsCalculations()`. Further functions and methods, albeit of less importance, are available and the reader is referred to the package's manual for a description of these.

Finally, forecasts can be created by calling `forecast()`, `horizonForecasts()`, or `featherForecasts()`. The latter two functions can be used to generate multiple-step-ahead forecasts. Each of these functions returns a named `list` object with class attribute set equal to the function's name. For visual inspection of the forecast accuracy, `plot()` methods are available. A feature of these methods is that the in-sample forecasts are aligned to the actual data, such that the actual values and the n -ahead forecasts for that time period can be directly compared.

13.5.3 The package `fArma`

The package `fArma` is part of the `Rmetrics` bundle of `R` packages and is hosted on CRAN (see Würtz 2013a). Within the package, `S4` classes and methods are employed. The computationally intensive calculations are interfaced from routines written in `FORTRAN`. In particular, estimation and inference of the degree of fractional integration are conducted by calling the underlying functions of the `fracdiff` package (see Fraley et al. 2012). The package is shipped with a `NAMESPACE` file containing the relevant export and import directives, but some functions are internal.

The strength of the package is the provision of a unified interface for specifying the different types of ARMA models—AR, MA, ARMA, ARIMA, and ARFIMA. This is accomplished by providing the order of a time series model as a formula in the cornerstone function `armaFit()`. The other important arguments to this function are `data` for the univariate time series, `method` for determining whether the coefficients

are estimated by ML or OLS, `include.mean` as a logical argument for indicating whether a constant is to be included, and `fixed` where the user can provide a vector of preset values for the coefficients. The remaining arguments `title`, `description`, and `...` can be used for descriptions of the model object returned, which is of formal class `fARMA`. For objects of this class, `show()`, `plot()`, `summary()`, and `predict()` methods are available. Furthermore, the estimated coefficients can be extracted by means of a defined `coef()` method, as can the fitted values with the method `fitted()` and the residuals by `residuals()`.

Model trajectories can be simulated by means of the function `armaSim()`. This function takes as input a parameter specification in the form of a `list` object as argument `model` and the length of the simulated series as argument `n`. The user can control the generation of the random variables by providing a time series of innovations to be used (argument `innov`) where by default these will be drawn from a normal population. In addition, the number of pre-sample innovations is controlled by either of the arguments `n.start` or `start.innov`. A seed for random number generation can be provided as argument `rseed`, and the kind of distribution from which the innovations are drawn is specified by the argument `rand.gen`, which should be the name of the function which returns samples from the distribution in question (the default is to use `rnorm` as indicated above). The function returns an object of class `timeSeries`.

The roots of an ARMA model and its associated ACF can be inspected by the functions `armaRoots()` and `armaTrueacf()`, respectively. The former function takes a vector of coefficient values as input and the latter a specification of the ARMA model as a `list` object as in the function `armaSim()`.

13.5.4 The package **forecast**

The purpose of the package **forecast** is to provide implementations of univariate time series models and filter/decomposition methods from which forecasts can be deduced (see Hyndman 2016). The package is hosted on CRAN and listed in the “Econometrics,” “Environmetrics,” “Finance,” and “TimeSeries” Task Views; it is considered a core package in the latter.⁴ Within the package S3 classes and methods are employed, with corresponding export directives contained in its `NAMESPACE` file. Certain computationally burdensome models are interfaced from C or C++ routines. In particular, the estimation of an exponential smoothing state-space model with a Box–Cox transformation, ARMA errors, and with trend/seasonal components (BATS) is interfaced from C++ routines (single and parallel processing). Five data sets are included in the package to demonstrate the use of the various functions and methods available.

The univariate statistical time series models implemented are ARFIMA, ARIMA, BATS, and exponential smoothing state-space models. The methods implemented that can be considered as filters, transformations, or time series decompositions are Box–Cox transformation, Holt–Winters with extensions, moving averages, and cubic

⁴ Incidentally, development versions of the package are hosted on GitHub and can be cloned from <https://github.com/robjhyndman/forecast.git>.

splines. The focus here is on the functions and methods pertinent to the time series models outlined in Section 13.2.1.

The classical Box–Jenkins ARIMA model is implemented as function `Arima()`, which is a wrapper for the function `arima()` in the package **stats**. A tentative order for the ARIMA model can be deduced from the autocorrelation and partial autocorrelation functions of the series. These are implemented as functions `Acf()` and `Pacf()`, which are wrapper functions for `acf()` and `pacf` contained in the base package **stats**, respectively. Furthermore, for inspecting the characteristics of a time series, the function `tsdisplay()` is available. This function plots the trajectory of the series with its ACF and, additionally, either the PACF or a scatter-plot of the series itself against its one-period lag or its spectrum. If exogenous regressors have been included in the ARIMA model, except for the drift term, the residuals of this model can be retrieved with the function `arima.errors()`. The ARIMA can also be automatically determined by calling `auto.arima()`. Here, the AR and MA order and how often the series must be differenced to achieve stationarity are determined using either the Akaike or Schwarz’s Bayesian information criterion. The user can restrict the search by providing upper limits for orders of the ARIMA model.

ARFIMA models can be fitted with the function `arfima()`. The degree of fractional integration is first estimated from an autoregressive model of order 2, where by default the fractional parameter is restricted to yield a stationary series. The filtered series according to its degree of fractional integration is then used in the Box–Jenkins approach. Unless otherwise specified by the user, the automatic procedure `auto.arima()` for selecting the AR and MA order is executed in order to fit the ARMA model and determine optimal orders. After the ARMA order has been determined the ARFIMA model is re-estimated by calling the function `fracdiff()` from the package of the same name.

For these models, and also for the others implemented, a `forecast()` method is available. The user can specify the number of forecast steps and the confidence level. The object returned is of informal class `forecast`, and `print()`, `plot()`, and `summary()` methods are available. If pseudo *ex ante* forecasts have been produced, that is, there are actual values of the series to compare the forecasts against, the function `accuracy()` can be used to obtain a set of accuracy measures (mean error, root mean square error, mean absolute error, mean percentage error, mean absolute percentage error, mean absolute scaled error, Theil’s inequality coefficient, and the ACF). To cross-compare forecasts the test proposed by Diebold and Mariano (1995) is implemented as function `dm.test()`. Finally, in this rubric of forecast evaluation, the functions `rwf()` and `naive()` should be mentioned. Both can be used to generate benchmark forecasts from a random walk model (with drift) or simply to produce naive forecasts.

13.5.5 The package **MSBVAR**

The package **MSBVAR** enables the user to estimate classical and Bayesian (structural) VAR models and carry out inferential analysis on them (see Brandt 2015). The

package is available on CRAN and is contained in the “Bayesian,” “Econometrics,” “Finance,” “Multivariate,” and “TimeSeries” Task Views. The package employs S3 classes and methods, and the computationally burdensome calculations for Monte Carlo simulations and Markov chain Monte Carlo (MCMC) based estimations are interfaced from routines written in C++, in particular the Gibbs sampler implemented for the latter model class. The `NAMESPACE` file contains the relevant export directives for the defined classes, methods, and functions. Two data sets are shipped with the package, which are both used in the example sections of the manual. Given the large number of model classes, methods, and functions implemented, the focus here is on multiple time series models, from which out-of-sample forecasts can be deduced.

A VAR model as described in (13.39) can be estimated with the function `reduced.form.var()` by means of the seemingly unrelated regression (SUR) principle, which yields the same estimates as an OLS estimator applied equation by equation. This function takes three arguments: `Y` for the matrix of endogenous variables, `p` for the lag order of the $\text{VAR}(p)$ model, and `z` for the matrix of exogenous (deterministic) variables. A suitable lag order can be determined from χ^2 tests and the Akaike, Schwarz, and/or Hannan–Quinn information criteria with the function `var.lag.specification()` for a maximum lag order set by the argument `lag.max`. The former function returns a named `list` object with class attribute `VAR`. Its elements are: `intercept` for the estimated intercepts of the VAR equations, `ar.coefs` for an array of the coefficients for the lagged endogenous variables, `Bhat` for a combination of the former two sets of estimates, `exog.coefs` for the estimated coefficients of the exogenous variables (if `z` was non-null), `vcv` and `mean.S` for the (posterior) variance-covariance matrix of the residuals, `hstar` for the cross-product of the right-hand-side variables, and the objects `X`, `Y`, and `y` for the left- and right-hand-side variables. A `summary()` method for objects of this S3 class is available. The package next provides functions for producing out-of-sample forecasts (`forecast()`) and for the density of unconditional forecasts by means of MCMC with the function `uc.forecast()`, computing the impulse response (`irf()` or, based on Monte Carlo methods, `mc.irf()`), and the forecast error decomposition (`dfev()`). Time series plots of the forecasts can be created with the function `plot.forecast()`, and the impulse responses can be visually inspected by means of the functions `plot.irf()` and `plot.mc.irf()`. The forecast accuracy can be assessed by the mean absolute error (`mae()`) or by the root mean squared errors (`rmse()`). Both functions expect two `matrix` objects, one for the forecasts and the other for actual or reference values.

For Bayesian estimation of the coefficients for the reduced-form VAR model as proposed by Sims and Zha (1998) the function `szbvar()` is available. It returns a named `list` object with class attribute `BVAR`. Bayesian estimation of an SVAR model as described in Waggoner and Zha (2003) can be carried out with the function `szbsvar()`. The $(1, 0)$ matrix used for identification has to be provided as argument `ident`. This matrix places zero restrictions on the coefficient matrix for the current dependencies between the variables. The function returns a named `list` object with class attribute `BSVAR`. The posterior distribution of the structural parameters

of this Bayesian SVAR model can then be determined by applying a Gibbs sampler to objects of this class by means of the function `gibbs.A0()`. The densities for each of the structural parameters can then be visually inspected by the function `plot.gibbs.A0()`. The same set of methods and functions that are applicable to objects of class `VAR` can also be applied to these objects.

13.5.6 The package `PortfolioAnalytics`

The **PortfolioAnalytics** package has already been introduced in Section 11.5.4. With respect to the entropy pooling approach, the function `EntropyProg()` is made available. The function's closure has a total of six arguments. The vector of initial probabilities has to be provided as argument `p`. In case of equality constrained views, the arguments `Aeq` and `beq` take the left-hand-side matrix and right-hand-side vector. Similarly, the arguments for casting inequality constraints are termed `A` and `b`, with default values set to `NULL`. Additional information with respect to the optimization is provided, if the logical argument `verbose` is set to `TRUE`. The default value is `FALSE`.

For minimizing the Kullback–Leibler divergence, the numerical implementation as proposed in Meucci (2010b) is followed; the optimization algorithms provided in `NLOpt` (see Johnson 2015) are employed. These algorithms are interfaced from the **R** package **nloptr** (see Ypma et al. 2014) and hence **nloptr** is a suggested package for **PortfolioAnalytics**; indeed, one must install it to use `EntropyProg()`. In particular, for equality constrained views only, `NLOPT_LD_LBFGS` is used (see Luksan and Vlcek 1998, Luksan et al. 2004), and for view specifications that include inequality constraints a nested optimization approach has been implemented; the algorithms `NLOPT_LD_SLSQP` for the sequential quadratic programming part (see Kraft 1988, 1994) and `NLOPT_LD_AUGLAG` (see Birgin and Martinez 2008; Conn et al. 1991) are invoked. The function `EntropyProg()` does return a named `list` object with elements `p_` (containing the revised probabilities) and `optimizationPerformance` (a `list` object itself which includes as elements the status of the optimization, the value of the objective, the count of iterations, and the sum of revised probabilities).

13.5.7 The packages `urca` and `vars`

In this last subsection the packages **urca** and **vars** are presented. The focus of the former is the analysis of integrated and co-integrated time series, whereas the emphasis of the latter is on the implementation of multivariate time series models and inferences associated with these as described in Section 13.2.2. Thus, the packages are mutually complementary. A detailed description is provided in Pfaff (2008a, c). Both packages are hosted on CRAN and are contained in the “Econometrics,” “Finance,” and “TimeSeries” Task Views. The package **urca** is considered a core package in the first two Task Views.

In the package **urca** S4 classes and methods are employed. To assess the degree of integration of a time series, the following unit root/stationary tests are implemented (in alphabetical order of the authors' names):⁵

- (augmented) Dickey–Fuller (ADF) (see Dickey and Fuller 1979, 1981), `ur.df()`;
- Elliot–Rothenberg–Stock (ERS) (see Elliott et al. 1996), `ur.ers()`;
- Kwiatkowski–Phillips–Schmidt–Shin (see Kwiatkowski et al. 1992), `ur.kpss()`;
- Phillips–Perron (see Phillips and Perron 1988), `ur.pp()`;
- Schmidt–Phillips (see Schmidt and Phillips 1992), `ur.sp()`;
- Zivot–Andrews (see Zivot and Andrews 1992), `ur.za()`.

Each of these functions returns an S4 object with class name the same as the function name. For objects of these classes, `show()`, `summary()`, and `plot()` methods are available. Functions to compute the distribution of unit root/co-integration tests by means of critical surface responses as proposed in MacKinnon (1991, 1996) are available as `punitroot()`, `qunitroot()`, and `unitrootTable()`. These functions interface with the original FORTRAN routines, which were kindly provided by MacKinnon.

To investigate the long-run relationship between integrated time series, the procedures proposed by Johansen (1988, 1991, 1995) and Johansen and Juselius (1990, 1992) (as function `ca.jo()`) and Phillips and Ouliaris (1990) (as function `ca.po()`) can be used. Both functions return S4 objects of the same name as the generating functions, and `show()`, `summary()`, and `plot()` methods are defined for these kinds of object. Numerous functions are also available to handle the ML approach to estimation and inference of/from VECM models. Unrestricted and restricted estimates for the left-hand-side coefficients are returned by the functions `cajools()` and `cajorls()`, respectively. The VECM residuals can be inspected graphically by the function `plotres()`. Inference on the kind of long-run relationship(s) and the validity of restrictions imposed can be conducted with the functions `alrtest()`, `blrtest()`, `ablrtest()`, `bh5lrtest()`, and `bh6lrtest()`. All of these return an S4 object `cajo.test` for which `show()` and `summary()` methods are available. A test of the co-integration rank by allowing a level shift at an unknown point in time, as proposed by Lütkepohl et al. (2004), can be conducted with the function `cajolst()`. Finally, the package is shipped with 15 data sets for replicating results in the references cited.

In contrast to **urca**, S3 classes and methods have been employed in the package **vars**. An overview of the informal classes, methods, and functions implemented is provided in Table 13.1.

⁵ Facilities for testing the degree of integration are also implemented in the packages **tsseries** (see Trapletti and Hornik 2015) and **fUnitRoots** (see Würtz 2013c).

Table 13.1 Structure of package **vars**.

Function/method	Class	Methods for class	Functions for class
VAR()	varest	coef(), fevd(), Acoef(), fitted(), arch.test(), irf(), Bcoef(), BQ(), logLik(), causality(), Phi(), plot(), normal- predict(), ity.test(), print(), Psi(), restrict(), resid(), roots(), summary(), serial.test(), stability()	
SVAR()	svarest	fevd(), irf(), logLik(), Phi(), print(), summary()	
SVEC()	sveccest	fevd(), irf(), logLik(), Phi(), print(), summary()	
vec2var()	vec2var	fevd(), arch.test(), fitted(), normal- irf(), ity.test(), logLik(), serial.test(), Phi(), predict(), print(), Psi(), resid()	
fevd()	varfevd	plot(), print()	
irf()	varirf	plot(), print()	
predict()	varprd	plot(), print()	fanchart()
summary()	varsum, svarsum, svecsum	print()	
arch.test()	varcheck	plot(), print()	
normality.test()	varcheck	plot(), print()	
serial.test()	varcheck	plot(), print()	
stability()	varstabil	plot(), print()	

The cornerstone functions for estimating VAR, SVAR, or SVEC models are `VAR()`, `SVAR()`, and `SVEC()`, respectively. The link between `urca` and `vars` is established by converting a VECM model (object of S4 class `ca.jo`) into its level form by means of the function `vec2var()`. For all of these returned model objects at least `summary()`, `logLik()`, and `plot()` methods are available. Furthermore, impulse response functions and the forecast error variance decomposition are returned by the methods `irf()` and `fevd()`, respectively. For objects with class attributes `varest` or `vec2var`, statistical tests for assessing whether the error term is spherical and/or normally distributed or not can be accomplished by the functions `serial.test()`, `arch.test()`, and `normality.test()`, with obvious meanings. These functions return an object with class attribute `varcheck` for which a `plot()` method is available. In addition to the statistical testing procedures for these two classes, extractor methods for the fitted values (`fitted()`) and residuals (`residuals()`) are available. To obtain forecasts from a VAR or VECM-in-level model, `predict()` methods can be used. These produce point forecasts for `n.ahead` steps with associated confidence bands at the level determined by the argument `ci` (the default is 95%). Finally, the data set `Canada` for replicating the results of Lütkepohl and Krätzig (2004) as shown in the package's vignette is included in `vars`.

13.6 Empirical applications

13.6.1 Black–Litterman portfolio optimization

This first example shows how the BL approach can be applied to the one-step-ahead forecasts for European equity markets derived from a VECM model. The estimated parameters of the posterior distributions are then used to obtain a weight vector that maximizes the portfolio's Sharpe ratio—that is, the solution to the empirical tangency portfolio is sought. The example involves a back-test where, in addition to the BL approach, portfolio weights based upon the prior distribution and a naive allocation are computed and the resulting portfolios' equities are contrasted with each other.

The first step of this analysis is shown in Listing 13.1. The packages `urca` and `vars` are brought into memory and the `EuStockMarkets` data set (see Section 9.5.1) is loaded. To summarize, this data set consists of the daily levels for the DAX, FTSE, CAC, and SMI indexes and has class attribute `mts`. In line 5 the data set is converted to a `zoo` object and in line 7 the month's-end values are extracted. The sample starts in June 1991 and ends in August 1998, comprising a total of 87 observations. The first 61 entries will be used as a subsample for unit root and co-integration testing, as is achieved by the `window()` function in line 10. The remaining data is kept back for executing the back-test. In lines 12–18 the degree of integration is determined by applying the ADF and ERS tests to the level and first differences of the series. The outcome of these tests is shown in Table 13.2.

Given critical values for the tests at the 5% level of -2.89 (with constant) and -1.95 (without constant), it can be concluded that all series are difference stationary.

R code 13.1 Integration and co-integration analysis of equity indexes.

```

library(urca)
library(vars)
## Loading data set and converting to zoo
data(EuStockMarkets)
Assets <- as.zoo(EuStockMarkets)
## Aggregating as month's-end series
AssetsM <- aggregate(Assets, as.yearmon, tail, 1)
head(AssetsM)
## Applying unit root tests for sub-sample
AssetsMsub <- window(AssetsM, start = start(AssetsM),
                      end = "Jun 1996")
## Levels
ADF <- lapply(AssetsMsub, ur.df, type = "drift",
              selectlags = "AIC")
ERS <- lapply(AssetsMsub, ur.ers)
## Differences
DADF <- lapply(diff(AssetsMsub), ur.df, selectlags = "AIC")
DERS <- lapply(diff(AssetsMsub), ur.ers)
## VECM
VEC <- ca.jo(AssetsMsub, ecdet = "none", spec = "transitory")
summary(VEC)

```

Table 13.2 Test statistics for unit root tests.

Unit roots	ADF		ERS	
	Level	Diff	Level	Diff
DAX	-0.33	-5.75	-0.05	-2.82
SMI	0.33	-4.74	0.36	-2.76
CAC	-2.56	-6.62	-1.64	-3.09
FTSE	-0.66	-6.10	0.26	-2.49

Next, the subsample is fitted to a VECM model by means of the expression in line 20. The default lag order of 2 (expressed in level form) and the transitory specification of the VECM have been used. The results of the maximum eigenvalue test are shown in Table 13.3. The null hypothesis of no long-run relationship is rejected, hence the four equity markets are co-integrated with rank 1. This model specification will be maintained throughout the back-test.

Having established the structure of the working model, the one-step-ahead forecasts and the return expectations deduced from them are computed as shown in Listing 13.2. First, the date stamps for the extending back-test windows are assigned to the object `idx` and the names and sizes of the equity indexes are stored in the objects `ANames` and `NAssets`, respectively. In the body of function `f1()` a simple

Table 13.3 Results of maximum eigenvalue test for VECM.

Co-integration	Max. eigenvalue	Critical values		
		Statistic	10%	5%
$r \leq 3$	0.10	6.50	8.18	11.65
$r \leq 2$	5.68	12.91	14.90	19.19
$r \leq 1$	16.62	18.90	21.07	25.75
$r = 0$	33.33	24.78	27.14	32.14

R code 13.2 Generating views derived from VAR model of assets.

```

## Index of time stamps in back-test (extending window) 1
idx <- index(AssetsM)[-c(1:60)] 2
ANames <- colnames(AssetsM) 3
NAssets <- ncol(AssetsM) 4
## Function for return expectations 5
f1 <- function(x, ci, percent = TRUE){ 6
  data <- window(AssetsM, start = start(AssetsM), end = x) 7
  Lobs <- t(tail(data, 1)) 8
  vec <- ca.jo(data, ecdet = "none", spec = "transitory") 9
  m <- vec2var(vec, r = 1) 10
  fcst <- predict(m, n.ahead = 1, ci = ci) 11
  LU <- matrix(unlist(fcst$fcst), 12
    ncol = 4, byrow = TRUE)[, c(2, 3)] 13
  RE <- rep(0, NAssets) 14
  PView <- LU[, 1] > Lobs 15
  NView <- LU[, 2] < Lobs 16
  RE[PView] <- (LU[PView, 1] / Lobs[PView, 1] - 1) 17
  RE[NView] <- (LU[NView, 1] / Lobs[NView, 1] - 1) 18
  names(RE) <- ANames 19
  if(percent) RE <- RE * 100 20
  return(RE) 21
}
ReturnEst <- lapply(idx, f1, ci = 0.5) 22
qv <- zoo(matrix(unlist(ReturnEst), 23
  ncol = NAssets, byrow = TRUE), idx) 24
colnames(qv) <- ANames 25
tail(qv) 26

```

forecasting rule is implemented—only those forecasts deemed sufficiently large in absolute terms will enter into a view for a certain equity market. This function is used in the call to `lapply()` in line 23, so that the views for the entire back-test window are computed by one line of code. Within the function, first the subsample for recursive estimation of the VECM is extracted from the object `AssetsM` and the last row of this observation is assigned to the object `Lobs`. This object will be used later to determine whether a directional view warrants a certain return for a given confidence level. In lines 9–10 the VECM is estimated and converted into its level representation by utilizing the function `vec2var()`, where a co-integration rank of 1 is assumed. This step is necessary because the one-step-ahead forecasts together with the lower and upper bounds for a given confidence level `ci` can then be swiftly computed by means of the available `predict()` method for this kind of object. Because the `predict()` method returns a `list` object, the lower and upper bounds for the one-step-ahead forecasts are extracted and put in the matrix `LU`. The vector of return expectations is initialized to zero for all four markets. A positive (negative) view on a market is given when the lower (upper) bound is above (below) the latest observed price. Whether this is the case is determined by the logical vectors `PView` and `NView`, respectively. The return expectations are then replaced at the relevant entries in `RE` accordingly. Hence, the absolute return should be at least as high as in `RE` with $1 - ci/2$ confidence. The `list` object of the views, where each element contains the view for the subsequent period, is assigned to the object `ReturnEst`. In lines 24–27 this `list` object is converted to a `zoo` object (`qv`) where the rows, with their date information, carry the views formed at that period for the subsequent period.

Having determined the views for each period, the estimates for the parameters of the posterior distributions have to be computed for each period in the back-test sample, and these can then be employed in portfolio optimizations. These tasks are accomplished in Listing 13.3.

First, the necessary packages **BLCop**, **fCopulae** and **fPortfolio** for conducting these computations are loaded into the workspace. In line 5 the discrete one-period percentage returns are calculated and assigned to the object `R`. In the following lines the pick matrix `P` is created; because only directional views for each asset will be implemented, this pick matrix is diagonal. In lines 9–21 an auxiliary function `BL()` is created which returns an object of class `BLResult`. This `S4` object contains the parameters of the prior and posterior distributions. Similar to the previous function for generating views, `BL()` includes an argument `x` that tells it when the extending sample ends. Within the function the view of a certain period is recovered from `qv` and the subsample of the returns is assigned to the object `Rw`. Next, the parameters of the prior distribution are estimated. The confidences in the views can take any value, because in the estimation of the posterior distribution they will be replaced by the sample variances of the returns, enforced by calling `BL()` with the argument `kappa=1`. In the final lines, an object view of class `BLViews` is created, which is then employed in the estimation of the posterior parameters. This function is then called in line 23 by applying `lapply()` to the time index `idx`. The result will be a `list` object of length 27 and each element contains the objects of class `BLResult`. In the following block of statements, two functions, `EstPrior()` and `EstBL()`,

R code 13.3 Maximum Sharpe ratio portfolio specifications.

```

library (BLCOP)                                1
library (fCopulae)                             2
library (fPortfolio)                           3
## Computing returns                         4
R <- (AssetsM / lag(AssetsM, k = -1) - 1.0) * 100 5
## Pick matrix for directional views          6
P <- diag(NAssets)                            7
colnames(P) <- ANames                         8
## Function for BL posterior distribution      9
BL <- function(x, tau, kappa){                10
  q <- qv[time(qv) == x, ]                     11
  q <- c(coredata(q))                         12
  Rw <- window(R, start = start(R), end = x) 13
  mu <- colMeans(Rw)                           14
  cov <- cov(Rw)                             15
  clevel <- rep(1, NAssets)                   16
  views <- BLViews(P = P, q = q, confidences = clevel, 17
                    assetNames = ANames)          18
  post <- posteriorEst(views, mu = mu, tau = tau, 19
                        sigma = cov, kappa = kappa) 20
  return(post)
}                                              21
PostDist <- lapply(idx, BL, tau = 1, kappa = 1) 22
## Defining portfolio specifications          23
EstPrior <- function(x, spec = NULL, ...){      24
  list(mu = BLR@priorMean, Sigma = BLR@priorCovar) 25
}
EstBL <- function(x, spec = NULL, ...){          26
  list(mu = BLR@posteriorMean, Sigma = BLR@posteriorCovar) 27
}
## Prior specification                         28
MSPrior <- portfolioSpec()                    29
setEstimator(MSPrior) <- "EstPrior"           30
## BL specification                           31
MSBI <- portfolioSpec()                      32
setEstimator(MSBI) <- "EstBL"                 33
## Constraints                               34
BoxC <- c("minW[1:NAssets] = -0.8", "maxW[1:NAssets] = 0.8") 35

```

are defined which will be used to extract the estimated parameters of the prior and posterior distributions, respectively. These functions are then assigned as estimators in the portfolio specifications employed for determining the solutions of the maximum Sharpe ratio portfolios derived from the estimates for the parameters of the prior and posterior distributions, respectively. A box constraint where the weights are allowed to vary in the interval $[-0.8, 0.8]$ completes the specification.

R code 13.4 Maximum Sharpe ratio portfolio back-test.

```

## Back-test function
BT <- function(DataSub, BLR){
  DataSub <- as.timeSeries(DataSub)
  PPrior <- tangencyPortfolio(data = DataSub, spec = MSPrior,
                                constraints = BoxC)
  PBl <- tangencyPortfolio(data = DataSub, spec = MSBl,
                            constraints = BoxC)
  Weights <- rbind(getWeights(PPrior), getWeights(PBl))
  colnames(Weights) <- ANames
  rownames(Weights) <- c("Prior", "BL")
  return(Weights)
}
## Conducting back-test
Backtest <- list()
for(i in 1:length(idx)){
  DataSub <- window(R, start = start(AssetsM), end = idx[i])
  BLR <- PostDist[[i]]
  Backtest[[i]] <- BT(DataSub, BLR)
}

```

In Listing 13.4 an auxiliary function `BT()` is created with the purpose of returning a matrix containing the optimal weights for the prior- and posterior-based maximum Sharpe ratio portfolios. This function is then employed in a `for` loop where these matrices are stored as elements in the `list` object `Backtest`. Note that in this loop the object `BLR` is created which will be used by means of lexical scoping in the call to `tangencyPortfolio()`. To that end, the optimal allocations in each period of the back-test sample are determined and can then be used for calculating the portfolio wealth trajectories.

As a final step, the weights have to be recovered from the `list` object and the portfolio equities calculated as shown in Listing 13.5. In the first two blocks of statements the first and second rows are extracted from each list element and arranged as `zoo` objects for the prior and posterior weights. The weights are lagged by one period such that the allocations can then be directly multiplied with the corresponding returns. These calculations are accomplished in lines 12–18. First, the subsample of returns pertinent to the back-test period is extracted from the object `R` which holds the returns for the entire sample period. These returns are then multiplied by the optimal allocations, the row sums are computed, and 1 is added, to arrive at the return factors. The seed wealth is set to 100 monetary units and the cumulated product of initial wealth and the return factors equals the portfolio wealth trajectories for the maximum Sharpe ratio allocations according to the prior and posterior distributions. For better comparison, the wealth trajectory of a naive (i.e., equal-weighted) allocation is computed to be used as a benchmark.

The paths of the portfolio equities and the relative wealth shares between the `BL`-based portfolio allocations vis-à-vis those determined from the prior distribution

R code 13.5 Comparison of portfolio strategies.

```

## Extracting weights based on prior
WPrior <- matrix(unlist(lapply(Backtest, function(x) x[1, ])),      1
                  ncol = NAssets, byrow = TRUE)                         2
WPrior <- zoo(WPrior, order.by = idx)                                3
WPriorL1 <- lag(WPrior, k = -1, na.pad = TRUE)                         4
## Extracting weights based on BL
WB1 <- matrix(unlist(lapply(Backtest, function(x) x[2, ])),      5
                  ncol = NAssets, byrow = TRUE)                         6
WB1 <- zoo(WB1, order.by = idx)                                7
WB1L1 <- lag(WB1, k = -1, na.pad = TRUE)                         8
## Compute portfolio equities
Rsub <- R[time(WB1L1), ] / 100                                     9
RetFacPrior <- rowSums(Rsub * WPriorL1) + 1                      10
RetFacPrior[1] <- 100                                              11
RetFacBL1 <- rowSums(Rsub * WB1L1) + 1                           12
RetFacBL1[1] <- 100                                              13
EquityPrior <- zoo(cumprod(RetFacPrior), index(Rsub))           14
EquityBL <- zoo(cumprod(RetFacBL1), index(Rsub))                  15
## Equal-weight strategy
EW <- matrix(1 / NAssets, ncol = NAssets, nrow = nrow(WB1L1))    16
RetFacEw <- rowSums(Rsub * EW) + 1                                17
RetFacEw[1] <- 100                                              18
EquityEw <- zoo(cumprod(RetFacEw), index(Rsub))                  19

```

and the equal-weighted allocations are displayed in Figure 13.3; the associated R code is shown in Listing 13.6.

The MSR portfolio allocation well outperforms the equal-weighted approach, and the BL approach adds value compared to the allocation based on the prior distribution. Finally, the distributions of the allocations derived from the prior and posterior parameter estimates are shown as box-plots in Figure 13.4. The weights applied to the equity indexes according to the prior distribution are much more concentrated compared to the portfolio solution derived from the BL posterior distribution. Qualitatively, only short positions would have been taken in the former approach with respect to the French stock market, and only long positions for the remaining three markets. In contrast to this, the allocations according to the BL posterior span a wider interval, and long positions for the French stock market as well as short positions for the UK stock market would have been entered during the back-test, reflecting the impact of the diverging return expectations between the prior and posterior estimates.

13.6.2 Copula opinion pooling

To continue the previous example, we will now employ the latest return forecasts in the copula opinion pooling framework. The relevant commands are shown in Listing 13.7. In contrast to Listing 13.3, a multivariate skewed Student's t distribution

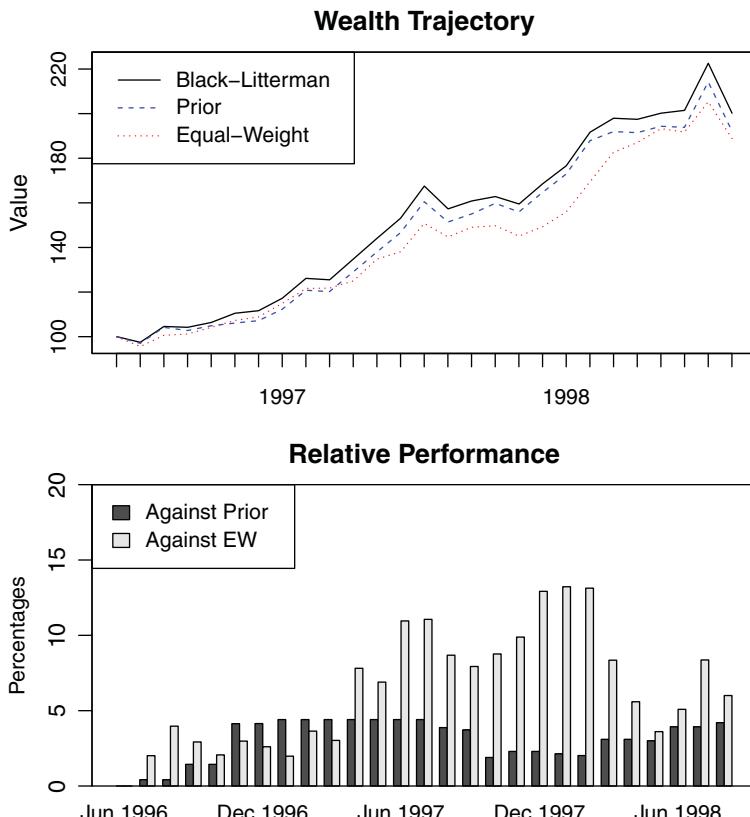


Figure 13.3 Equity for BL, prior, and equal-weighted portfolios.

is now assumed for the market returns. The unknown parameters are estimated with the function `mvFit()`, which is contained in the package **fCopula**. The estimated distribution parameters are then extracted from the fitted model object and passed to the function `mvdistribution()` of the package **BLCOP**. The object `CopPrior` now contains the relevant information on the assumed market distribution.

In the next block of R statements, the latest return forecast is extracted from the previously created list object `ReturnEst` and assigned to `RetEstCop`. For that time period return views different from zero at the 50% confidence level were recovered for the DAX, SMI, and FTSE indexes. Hence, the pick matrix will consist of three rows and four columns and is filled with 1s at the relevant row entries for these indexes. After the pick matrix `PCop` has been created, the return forecasts have to be expressed in terms of a distributional argument. The return variances are used as a measure of dispersion and the point forecasts are assumed to be normally distributed. These three distribution specifications are included in the list object `RetViews` which is utilized in the subsequent assignment of copula views. So far, objects pertinent to the prior and the views distributions have been

R code 13.6 Display of portfolio strategies.

```

## Graphical display of back-test result
1
par(mfrow = c(2, 1))
2
## Plotting of equity curves
3
plot(EquityBL, main = "Wealth Progression",
4      ylab = "Value", xlab = "")
5
lines(EquityPrior, col = "blue", lty = 2)
6
lines(EquityEw, col = "red", lty = 3)
7
legend("topleft",
8       legend = c("Black-Litterman", "Prior", "Equal-Weight"),
9       col = c("black", "blue", "red"), lty = 1:3)
10
## Relative performance
11
RelOut <- cbind((EquityBL / EquityPrior - 1) * 100,
12                  (EquityBL / EquityEw - 1) * 100)
13
barplot(RelOut, xlab = "", ylab = "Percentages",
14        main = "Relative Performance",
15        ylim = c(0, 20), beside = TRUE,
16        legend.text = c("Against Prior", "Against EW"),
17        args.legend = list(x = "topleft"))
18
box()
19
## Boxplots of weights
20
par(mfrow = c(2, 1))
21
boxPR <- coredata(WPriorL1)
22
colnames(boxPR) <- ANames
23
boxplot(boxPR, ylim = c(-0.8, 0.8),
24        main = "Based on Prior Distribution")
25
abline(h = 0, col = "grey")
26
boxBL <- coredata(WBIL1)
27
colnames(boxBL) <- ANames
28
boxplot(boxBL, ylim = c(-0.8, 0.8),
29        main = "Based on Posterior Distribution")
30
abline(h = 0, col = "grey")
31

```

defined. The posterior distribution is then obtained by the pooling of the random variates thereof. The simulation size is set to 10 000 random draws and the simulated random values of the posterior distributions are computed by invoking the function `COPPosterior()`. These values are contained in the slot `posteriorSims` and the prior distribution and its parameters are contained in the slot `marketDist`.

As shown in Listing 13.8, these simulated random variates can then be employed to derive location and dispersion estimates. The listing concludes with an overview of the density plots for the prior and posterior distributions for each of the four markets considered. The outcome is shown in Figure 13.5.

Clearly, the prior and posterior densities for the CAC index pretty much overlap, given that no view has been expressed for this market aggregate. The differences in shape of the prior and posterior densities for the other aggregates primarily reflect the differences between the sample means and the stated return forecasts.

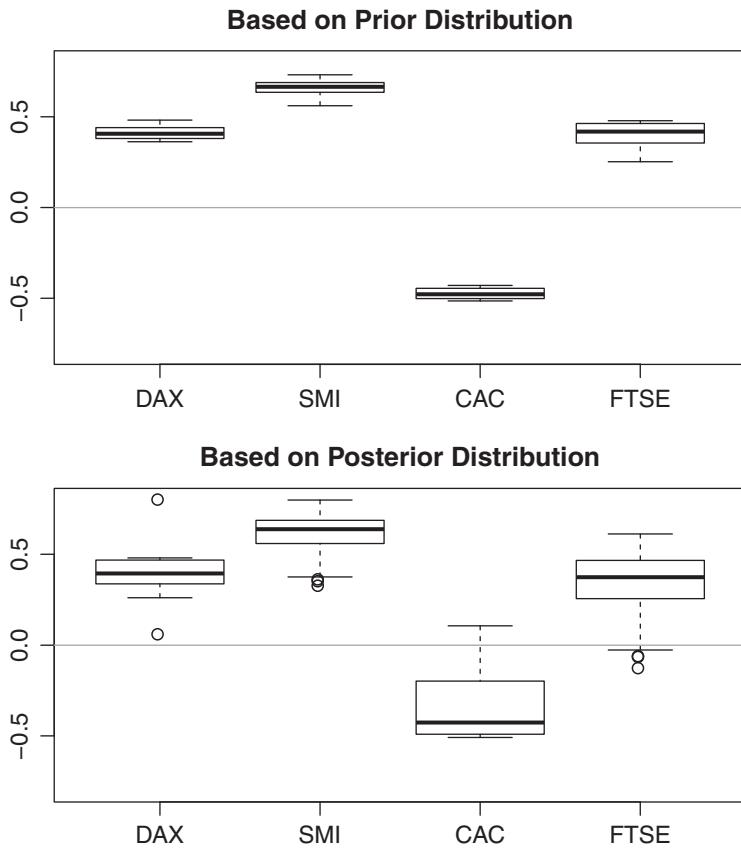


Figure 13.4 Box-plots of weights based on prior and BL distributions.

Listing 13.9 shows how the allocations according to the MSR objective for the Gaussian and skewed Student's t prior distributions as well as according to the BL and COP models are determined. The aim here is to convince the reader that quite differing portfolio allocations can result, given the same set of inputs. The portfolio optimizations are again conducted by means of the function `tangencyPortfolio()`. Similar to the specifications for the Gaussian and BL MSR portfolio solutions shown earlier, two functions for recovering the moment estimates are defined first. The function `SSTPrior()` returns the estimated location and dispersion value as given by the object `MSTfit`. The corresponding moment estimates for the simulated random variates of the COP model are returned by the function `BLCopPost()`. These two functions are then defined as estimators in the portfolio specifications for the MSR portfolios based on the skewed Student's t distribution, `MSPriorSST`, and on the opinion pooling framework, `MSBLCop`, respectively. Having created these two remaining portfolio specifications, the solutions to the tangency portfolios can be swiftly determined by including them in the list object `PSpecs` and employing

R code 13.7 Copula opinion pooling.

```

## Prior distribution
## Fitting of skewed Student 's t distribution
MSTfit <- mvFit(R, method = "st")
mu <- c(MSTfit@fit$estimated [[ "beta" ]])
S <- MSTfit@fit$estimated [[ "Omega" ]]
skew <- c(MSTfit@fit$estimated [[ "alpha" ]])
df <- MSTfit@fit$estimated [[ "nu" ]]
CopPrior <- mvdistribution("mvst", dim = NAssets, mu = mu,
                           Omega = S, alpha = skew, df = df)
## Pick matrix and view distributions for last forecast
RetEstCop <- ReturnEst[[27]]
RetEstCop
PCop <- matrix(0, ncol = NAssets, nrow = 3)
colnames(PCop) <- ANames
PCop[1, ANames[1]] <- 1
PCop[2, ANames[2]] <- 1
PCop[3, ANames[4]] <- 1
Sds <- apply(R, 2, sd)
RetViews <- list(distribution ("norm", mean = RetEstCop [1],
                           sd = Sds [1]),
                  distribution ("norm", mean = RetEstCop [2],
                           sd = Sds [2]),
                  distribution ("norm", mean = RetEstCop [4],
                           sd = Sds [4]))
)
CopViews <- COPViews(pick = PCop, viewDist = RetViews,
                      confidences = rep(0.5, 3),
                      assetNames = ANames)
## Simulation of posterior
NumSim <- 10000
CopPost <- COPPosterior(CopPrior, CopViews,
                         numSimulations = NumSim)
slotNames(CopPost)

```

`lapply()`. The object `POpt` is again a list object with elements of formal class `fPORTFOLIO`. The allocations can then be extracted by using `lapply()` and invoking the `getWeights()` method. Finally, the allocations are casted in a `matrix` object and expressed as percentages. The outcome is shown in Table 13.4.

From a qualitative point of view, all allocations coincide. That is, long positions would have been entered for the German, Swiss, and British equity markets and a short position is indicated for the French stock aggregate. However, the position sizes do differ materially, between each of the two prior/posterior pairs and also across the BL and COP models. Given that the BL and COP results were derived from the same set of return forecasts, the latter observation shows the importance of specifying the

R code 13.8 Copula opinion pooling: densities.

```

## Plotting of densities
CopPriorSim <- sampleFrom(CopPost@marketDist , NumSim) 1
CopPostSim <- CopPost@posteriorSims 2
oldpar <- par(no.readonly = TRUE) 3
par(mfrow = c(2, 2)) 4
for(i in 1:NAssets){ 5
  plot(density(CopPostSim[, i]), main = ANames[i], 6
        ylab = "density", ylim = c(0, 0.12), xlab = "") 7
  lines(density(CopPriorSim[, i]), col = "blue", lty = 2) 8
  abline(v = mean(CopPostSim[, i])) 9
  abline(v = mean(CopPriorSim[, i]), col = "blue", lty = 2) 10
  legend("topleft", legend = c("Posterior", "Prior"), 11
         lty = c(1, 2), col = c("black", "blue"), bty = "n") 12
}
par(oldpar) 13

```

Table 13.4 Comparison of portfolio allocations

Model	DAX	SMI	CAC	FTSE
Gauss	40.00	78.18	-34.02	15.83
Skewed Student's t	51.31	30.18	-42.88	61.39
BL	60.47	59.70	-22.39	2.22
BLCop	63.07	16.85	-12.47	32.55

marginal distributions and dependence structure appropriately, as has been stressed in Chapters 3 and 9.

13.6.3 Entropy pooling

Overview

In the following example the EP approach will be applied to selected currencies; the weekly euro reference rates with respect to Australian, Canadian, Hong Kong, and US dollars are used as well as the spot exchange rates of Switzerland and Japan vis-à-vis the euro (Wednesday settlements).

As market distribution, a skewed Student's t distribution will be assumed and the subjective views are formed with respect to the conditional volatilities for each of these currency pairs; one-step ahead forecasts deduced from GARCH(1, 1) models are employed.

The full sample will be split into two sub-periods; the data points in the latter will be used in a portfolio back-test, where optimal currency allocations are determined based on the EP posteriors, the market distributions, and—as a base-case allocation—according to a multivariate normal distribution assumption. The outcome of the three portfolio strategies is analyzed at the end of this section.

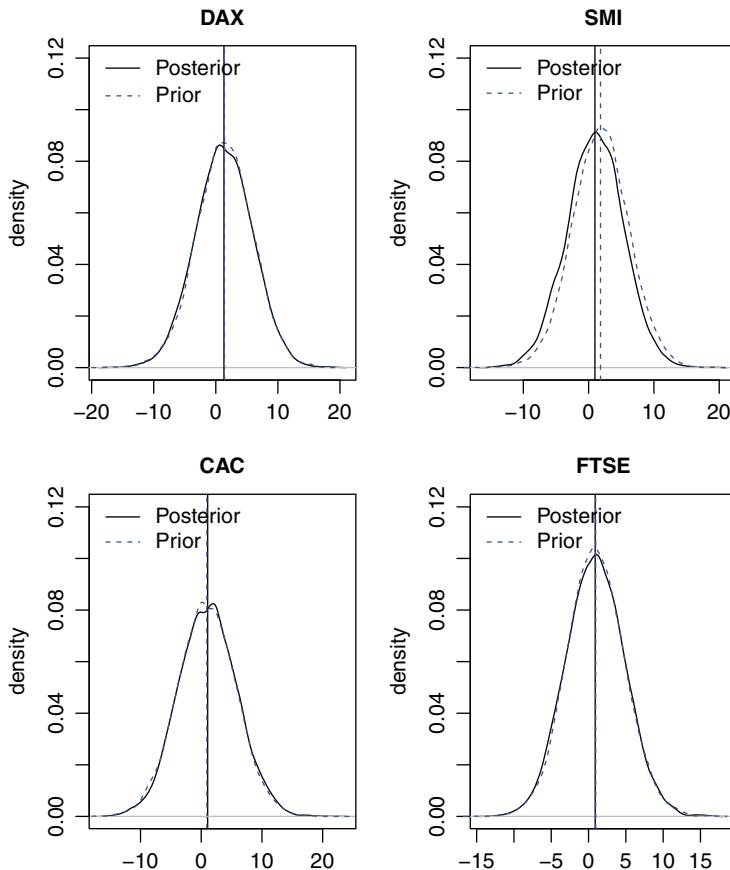


Figure 13.5 Prior and posterior densities.

Data preparation

In Listing 13.10 the data for conducting the back-test is prepared. First, the necessary packages are loaded into the workspace. The package **FRAPo** contains the data set ESCBFX with a selection of the daily ESCB reference rates. The facilities of the package **fGarch** (see Würtz and Chalabi 2013) will be used for estimating the GARCH models and computing the one-step-ahead forecasts with respect to the conditional volatilities. The **fMultivar** (see Würtz and Setz 2014) and **sn** (see Azzalini 2015) packages are utilized for fitting the skewed Student's *t* distribution to the data and drawing random samples, respectively. Finally, the packages **fPortfolio** (see Würtz et al. 2014) and **PerformanceAnalytics** (see Peterson and Carl 2014) are employed for portfolio optimization and evaluation of the back-test results.

The computation of the weekly FX returns commences on line 7. First, the data set with daily observations is brought into memory and transformed into a `timeSeries` object `FXDaily`. The sample starts on 4 January 1999 and ends on 4 April

R code 13.9 Comparison of portfolio allocations.

```

## Defining portfolio specifications
SSTPrior <- function(x, spec = NULL, ...){
  list(mu = c(MSTfit@fit$estimate[["beta"]]),
       Sigma = MSTfit@fit$estimate[["Omega"]])
}

B1CopPost <- function(x, spec = NULL, ...){
  Sim <- CopPost@posteriorSims
  list(mu = colMeans(Sim), Sigma = cov(Sim))
}

## Skewed Student's t
MSPriorSST <- portfolioSpec()
setEstimator(MSPriorSST) <- "SSTPrior"
## BLCOP specification
MSB1Cop <- portfolioSpec()
setEstimator(MSB1Cop) <- "B1CopPost"
## Tangency portfolios
R <- as.timeSeries(R)
BLR <- PostDist[[27]]
PSpecs <- list(MSPrior, MSB1, MSPriorSST, MSB1Cop)
POpt <- lapply(PSpecs, function(x)
  tangencyPortfolio(data = R, spec = x,
                     constraints = BoxC))
)

PWeights <- unlist(lapply(POpt, getWeights))
Weights <- matrix(PWeights, ncol = NAssets, nrow = 4,
                   byrow = TRUE) * 100
colnames(Weights) <- ANames
rownames(Weights) <- c("Gauss", "Skewed Student's t",
                       "BL", "BLCOP")
Weights

```

2012, comprising a total of 3427 observations. In the subsequent lines a `timeDate` object `AllWedDays` is created, which holds all dates pertinent to Wednesdays. This `timeDate` object is then used for creating the weekly spot exchange rate observations, `FXWeekly`. Because the euro reference rates are following the sterling notation, the reciprocals are computed to reflect the stance of a euro-oriented investor. The object `FXRet` holds the discrete percentage returns thereof.

Listing 13.10 ends with parameter settings for the entropy pooling and the back-test design. The latter starts after an initial period of 520 observations—a moving data window of that size will be used. The panels created in the back-test will consist of 1000 random draws and this number is assigned to object `J`. The objects for storing the allocations (objects `WEP`, `WMD`, and `WND`), the start values used in the EP optimizations (object `x0`), and the prior probabilities `pprior` are defined next.

R code 13.10 Data preparation.

```

library (FRAPO) 1
library (fGarch) 2
library (fMultivar) 3
library (sn) 4
library (fPortfolio) 5
library (PerformanceAnalytics) 6
## Preparing FX-data / returns 7
data (ESCBFX) 8
FXDaily <- timeSeries (ESCBFX, charvec = rownames (ESCBFX)) 9
DDates <- time (FXDaily) 10
WedDays <- isWeekday (DDates, wday = 3) 11
FirstWed <- head (which (WedDays, arr.ind = TRUE), 1) 12
LastWed <- tail (which (WedDays, arr.ind = TRUE), 1) 13
AllWedDays <- timeSequence (from = DDates [FirstWed], 14
                           to = DDates [LastWed], 15
                           by = "week") 16
DumWed <- timeSeries (rep (1, length (AllWedDays)), 17
                      charvec = AllWedDays) 18
FXWeekly <- interpNA (cbind (DumWed, FXDaily), 19
                      method = "before") [AllWedDays, -1] 20
assetsNames <- Anames <- FxNames <- colnames (FXWeekly) 21
FxPrice <- 1 / FXWeekly 22
FxRet <- returns (FxPrice, percentage = TRUE, 23
                  type = "discrete", trim = TRUE) 24
FxRetSub <- window (FxRet, 25
                     start = start (FxRet), end = time (FxRet) [520]) 26
## Setting parameters / initializing objects 27
J <- 1000 28
N <- ncol (FxPrice) 29
Eperiods <- time (FxRet) [-c (1:519)] 30
Speriods <- time (FxRet) [1:length (Eperiods)] 31
LengthBack <- length (Speriods) 32
WEP <- WMD <- WND <- matrix (NA, nrow = LengthBack, ncol = N) 33
x0 <- rep (0, N + 1) 34
pprior <- matrix (rep (1 / J, J), ncol = 1) 35

```

Forecasts and entropy pooling

In Listing 13.11 a function for computing the one-step-ahead predictions for the conditional volatilities (function `CondVolafcst()`) and functions relating to the EP optimization are defined. Within the body of function `f0` the objective is defined and `gf()` returns the gradient evaluated at a particular solution thereof. Note that the objective is defined for the case of equality constrained views, only. The function `ep()` is a wrapper for computing the probabilities that minimize the divergence measure. The optimization is carried out by calling the function `optim()`,

R code 13.11 Definition of forecast and entropy pooling functions.

```

## Defining function for GARCH-forecast
CondVolaFcst <- function(x){
  m <- garchFit(formula = ~ garch(1,1),
                 data = x, trace = FALSE)
  mp <- as.numeric(predict(m, n.ahead = 1))[3]
  mp
}

## Defining functions for entropy pooling
f0 <- function(v, p, Aeq, beq){
  x <- exp(log(p) - 1 - crossprod(Aeq, v))
  x = apply(cbind(x, 0), 1, max)
  L = t(x) %*% (log(x) - log(p) + crossprod(Aeq, v)) -
    crossprod(beq, v)
  -L
}

gf <- function(v, p, Aeq, beq){
  x <- exp(log(p) - 1 - crossprod(Aeq, v))
  beq - Aeq %*% x
}

ep <- function(x0, Aeq, beq, pprior){
  vopt <- try(optim(par = x0, fn = f0, gr = gf,
                    Aeq = Aeq, beq = beq, p = pprior, method = "BFGS"))
  if(class(vopt) == "try-error"){
    return(pprior)
  } else {
    v <- vopt$par
    pbar <- exp(log(pprior) - 1 - crossprod(Aeq, v))
    return(pbar / sum(pbar))
  }
}

```

with "BFGS" as the optimization method. The final line within the body of function `ep()` is normalization to ensure that the sum of the returned probabilities equals one.

Back-test

The back-test is cast in a `for` loop as shown in Listing 13.12. Within the body of the loop, the data set according to the i th moving window is extracted from the `timeSeries` object `FxRet` and assigned to the object `fp`. Based on this sub-sample, the skewed Student's t distribution is fitted next and a random panel of size 1000 is generated and assigned to object `M`.⁶ This $(J \times N)$ matrix contains the simulated values according to the assumed market distribution. These values together with

⁶ For replication purposes, a seed is defined at the beginning of the loop body and its value is incremented in each step.

the one-step-ahead predictions are utilized for defining the equality constraints with respect to the volatility views of the form

$$\sum_{j=1}^J \tilde{p}_j \mathcal{V}_{j,k}^2 = \hat{m}_k^2 + \sigma_k^2, \quad (13.74)$$

with \hat{m}_k defining the first moment of the k th asset. Note that the matrices `Aeq` and `beq` are amended by one row, such that the sum of the probabilities equates to one.

Commencing in line 19, the probabilities are computed for a pooling parameter of $c = 0.5$ and are assigned to object `EpH`. To determine the solution of the tangency portfolio, a specification object (`EPspec`) is defined next. This specification is endowed with a custom function `EPmom()` such that the probability weighted moments will be used. These are swiftly returned by calling `cov.wt()`. The MSR optimizations are then conducted by invoking `tangencyPortfolio()`, providing the previously defined portfolio specification. The weights are returned by invoking the `getWeights()` method, and these MSR allocations are stored in the i th row of the matrix `WEP`.

Similarly, the allocations implied by the market and normal distributions only are determined and stored in the i th rows of objects `WMD` and `WND`, respectively.

Analysis of results

An analysis of the back-test is provided in Listing 13.13. First, the three weight matrices are stored as a `list` object `W`. This helps the computations of the wealth trajectories and the portfolio performance measures. Next, the subset of returns pertinent to the back-test window is assigned to the object `FxRetBack`. The wealth progressions can be calculated with the function `Equity()`, defined next. The calculations are then carried out by invoking `lapply()` on that function in line 11.

The result is depicted as a time series plot in Figure 13.6.

Clearly, the allocations according to the EP model in terms of the final wealth values achieved exceed those based on either the market or normal distributions. If one compares the wealth trajectories based on the skewed Student's t and normal distributions, the latter is below the former. As such, the skewed Student's t distribution seems to capture the return characteristics better. Comparing the result of the market distribution with the blended views it is pretty obvious that the inclusion of a conditional risk measure did yield a better risk assessment.

The performance of the three test cases as depicted in Figure 13.6 is also mirrored by the performance measures as computed within the function `PerfM()` (see lines 24ff in Listing 13.13). These are shown in Table 13.5.

The annualized return and the Sharpe ratios are more favorable for the allocations based on the skewed Student's t distribution compared to the strategy that was based on the normal distribution assumption; the EP-based portfolio incarnation fairs best. This picture stays the same for the maximum draw-downs. Only with respect to the CVaR does the outcome derived under the normality assumption take an intermediate position. Judged by this measure, the EP approach yielded the riskiest progression.

R code 13.12 Back-testing portfolio strategies.

```

## Backtest
for(i in 1:LengthBack){
  cat(paste("Backtestperiod:", Eperiods[i], "\n"))
  set.seed(i + 12345)
  fp <- window(FxRet, start = Speriods[i], end = Eperiods[i])
  ## Update Market Model
  MarketMod <- mstFit(series(fp))
  par <- MarketMod@fit$estimated
  M <- rmst(J, xi = c(par$beta), Omega = par$Omega,
             alpha = par$alpha, nu = par$nu)
  colnames(M) <- Anames
  mu2 <- colMeans(M)^2
  Vt <- t(M)
  Aeq <- rbind(Vt^2, rep(1, J))
  ## GARCH-model
  mfcst <- lapply(fp, CondVolafcst)
  fcst <- matrix(unlist(mfcst), ncol = 1, byrow = TRUE)
  beq <- matrix(c(mu2 + fcst[, 1]^2, 1), ncol = 1)
  ## EP-optimization
  Ep <- ep(x0, Aeq, beq, pprior)
  ## EP for fixed tau = 0.5
  EpH <- 0.5 * Ep + 0.5 * pprior
  EPspec <- portfolioSpec()
  EPmom <- function(x, spec = NULL, ...){
    m <- cov.wt(x, drop(EpH))
    list("mu" = m$center, "Sigma" = m$cov)
  }
  setEstimator(EPspec) <- "EPmom"
  PSol <- tangencyPortfolio(data = as.timeSeries(M),
                             spec = EPspec)
  WEP[i, ] <- getWeights(PSol)
  ## Portfolio based on market distribution
  PSolM <- tangencyPortfolio(data = as.timeSeries(M),
                             spec = portfolioSpec())
  WMD[i, ] <- getWeights(PSolM)
  ## Portfolio based on normality assumption
  PSolN <- tangencyPortfolio(data = fp, spec = portfolioSpec())
  WND[i, ] <- getWeights(PSolN)
}

```

13.6.4 Protection strategies

Overview

Wealth protection strategies have been known for a long time. Their origins can probably be traced back to the work of Leland and Rubinstein (1976), where the authors

R code 13.13 Analysis of results.

```

## Evaluation of backtest
W <- list(WEP, WMD, WND)
FxRetBack <- FxRet[Eperiods, ]
Equity <- function(x, Eperiods, FxRet){
  WTs <- timeSeries(x, charvec = Eperiods)
  WTsL1 <- lag(WTs, k = 1)
  RetFacPR <- rowSums(FxRetBack / 100 * WTsL1) + 1
  RetFacPR[1] <- 100
  timeSeries(cumprod(RetFacPR), Eperiods)
}
WealthBack <- lapply(W, Equity, Eperiods = Eperiods,
                      FxRet = FxRet)
## plot of wealth trajectories
ypretty <- unlist(lapply(WealthBack, function(x)
                          pretty(range(x))))
ylims <- c(min(ypretty), max(ypretty))
plot(WealthBack[[1]], lwd = 2,
      xlab = "", ylab = "Index",
      main = "",
      ylim = ylims)
lines(WealthBack[[2]], lty = 2, lwd = 2, col = "blue")
lines(WealthBack[[3]], lty = 3, lwd = 2, col = "red")
legend("topleft", legend = c("EP", "Market", "Normal"),
       lty = 1:3, lwd = 2, col = c("black", "blue", "red"))
## portfolio performance/risk measures
PerfM <- function(x){
  EPRet <- returns(x, method = "discrete", percentage = FALSE,
                    trim = TRUE)
  EPRA <- Return.annualized(EPRet[, 1], scale = 52) * 100
  EPSD <- StdDev.annualized(EPRet[, 1], scale = 52) * 100
  EPSRA <- SharpeRatio.annualized(EPRet[, 1], scale = 52)
  EPES <- ES(EPRet[, 1], p = 0.95,
             method = "modified", clean = "boudt") * -100
  EPMDD <- maxDrawdown(EPRet[, 1]) * 100
  EPPerfW <- rbind(EPRA, EPSD, EPSRA, EPES, EPMDD)
  rownames(EPPerfW) <- c("Return (annual)",
                         "Risk (annual, SD)", "Sharpe Ratio",
                         "CVaR (modified, 95%)",
                         "Max Draw Down")
  EPPerfW
}
PerfBack <- lapply(WealthBack, PerfM)
PerfM <- matrix(unlist(PerfBack), ncol = 3)
rownames(PerfM) <- c("Return (annual)", "Risk (annual, SD)",
                      "Sharpe Ratio", "CVaR (modified, 95%)",
                      "Maximum Draw Down")
colnames(PerfM) <- c("EP", "Market", "Normal")
PerfM

```

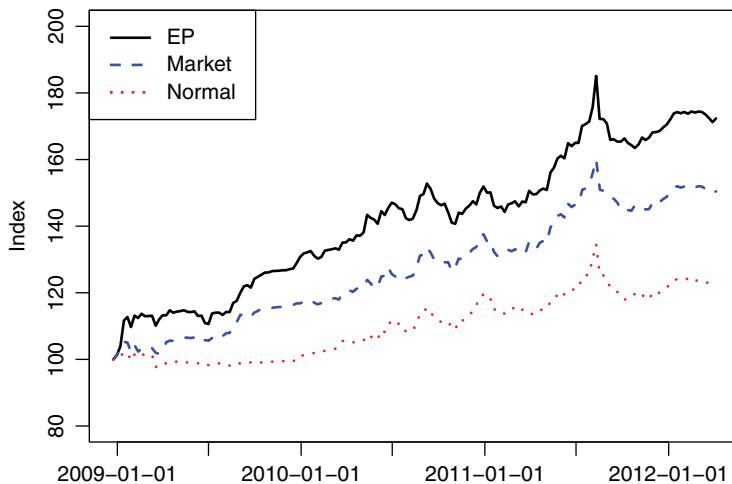


Figure 13.6 Wealth trajectories.

Table 13.5 Key portfolio performance measures.

Measures	Entropy pooling	Market	Normal
Return (annual)	17.99	13.24	6.38
Risk (annual, SD)	10.54	8.16	8.04
Sharpe ratio	1.71	1.62	0.79
CVaR (modified, 95%)	2.17	1.93	2.00
Maximum draw-down	11.68	9.61	12.32

introduced the idea of option-based portfolio insurance (OBPI). In the 1980s the concept of constant proportion portfolio insurance (CPPI) was advocated by Perold (1986) and Perold and Sharpe (1988). Extensions and modifications of the latter approach were put forward by Black and Perold (1992) and Black and Jones (1987). A comparison of these strategies was conducted, among others, by Black and Rouhani (1989) and Bookstaber and Langsam (2000). It is fair to say that both of these approaches were proposed during financial crisis episodes in the aftermath of oil price shocks and the ensuing deterioration of financial wealth positions, primarily in stocks.

This final subsection, based on Pfaff (2008b), shows how a TAA-related protection strategy can be implemented as an alternative to the OBPI/CPPI approaches. Similarly to the insurance strategies, a portfolio floor value is defined that may not be violated. The difference between the current wealth and this floor is then used as a risk buffer, where the risk budget is allocated between the assets according to their downside risk. This constraint can be expressed in terms of a simple linear program with

R code 13.14 Data preparation.

```

library(FRAPO) 1
library(evir) 2
library(forecast) 3
## Data preparation 4
data(StockIndexAdjD) 5
data(ESCBFX) 6
PDaily <- timeSeries(StockIndexAdjD, 7
                      charvec = rownames(StockIndexAdjD)) 8
FXDaily <- timeSeries(ESCBFX, charvec = rownames(ESCBFX)) 9
FXDaily <- window(FXDaily, start = start(FXDaily), 10
                  end = end(PDaily)) 11
DDates <- time(FXDaily) 12
WedDays <- isWeekday(DDates, wday = 3) 13
FirstWed <- head(which(WedDays, arr.ind = TRUE), 1) 14
LastWed <- tail(which(WedDays, arr.ind = TRUE), 1) 15
AllWedDays <- timeSequence(from = DDates[FirstWed], 16
                            to = DDates[LastWed], 17
                            by = "week") 18
DumWed <- timeSeries(rep(1, length(AllWedDays)), 19
                      charvec = AllWedDays) 20
PWeekly <- interpNA(cbind(DumWed, PDaily), 21
                     method = "before")[AllWedDays, -1] 22
FXWeekly <- interpNA(cbind(DumWed, FXDaily), 23
                      method = "before")[AllWedDays, -1] 24
PEWeekly <- PWeekly 25
PEWeekly[, "SP500"] <- PWeekly[, "SP500"] / FXWeekly[, "USD"] 26
PEWeekly[, "N225"] <- PWeekly[, "N225"] / FXWeekly[, "JPY"] 27
PEWeekly[, "FTSE100"] <- PWeekly[, "FTSE100"] / FXWeekly[, "GBP"] 28
PEWeekly[, "HSI"] <- PWeekly[, "HSI"] / FXWeekly[, "HKD"] 29
REDWeekly <- (PEWeekly / lag(PEWeekly, k = 1) - 1) 30
PELWeekly <- log(PEWeekly) 31
## Defining moving window and indexes 32
epoints <- time(PELWeekly)[-c(1:259)] 33
size <- length(epoints) 34
spoints <- time(PELWeekly)[1:size] 35
idx <- 1:size 36
NAssets <- ncol(PEWeekly) 37
## Time series chart of euro-denominated indexes 38
plot(PEWeekly, main = "", xlab = "", col = "black") 39

```

return maximization as its target. In addition to the risk budget, further constraints such as budget, box, and/or group constraints can be included in the linear program. The latter can be derived from a forecasting model, as will be shown in due course. Hence, the three pieces that are put together are a forecasting model (univariate models or multivariate), a risk model, and a linear program.

Data preparation

In Listing 13.14 the data used in the simulation is prepared. It is assumed that the TAA part of the portfolio can be allocated to the stock markets of the US, Great Britain, France, Germany, Japan, and Hong Kong (China) represented by the S&P 500, FTSE 100, CAC 40, DAX, Nikkei 225, and Hang Seng (HSI) indexes, respectively. The focus is on a non-hedged, euro-based, long-only investor.

First, the packages required for conducting the simulation are brought into memory. The package **FRAPO** contains the data sets and loads the packages **timeSeries** and **Rglpk**. The package **evir** will be used to assess the downside risk, and the package **forecast** to derive the one-step-ahead return expectations. Next, the data sets **StockIndexAdjD** and **ESCBFX** are brought into the workspace. The former contains the adjusted daily closing prices of the indexes and the latter the ESCB reference rates, which will be employed to denominate the market values in euros. Both data sets are converted **timeSeries** objects. The FX data set is curtailed to the sample end of the stock data and the daily time stamps thereof are assigned to the object **DDates**. These dates are then used to extract the dates that fall on Wednesdays. It is also necessary to create a time/date series of Wednesdays for the entire sample period, given that some public holidays coincide with this day of the week. If this happens, the last observed daily observation is carried forward, such that finally the objects **PWeekly** and **FXWeekly** consist of consecutive regular time series data sets for the equity and currency markets. In lines 25–31 the levels of the foreign stock indexes are converted to euro-based values (object **PEWeekly**), the discrete decimal returns thereof are computed and assigned to the object **REDWeekly**, and the log levels are stored in **PELWeekly**. In the last block of statements the dates of the moving window utilized in the forthcoming simulation are created, and the size thereof and the names of the stock indexes are assigned to the objects **size** and **NAssets**, all used in the ensuing examples. The moving window has been set to a length of roughly five years. Finally, the trajectory of the euro-denominated index values is created (see Figure 13.7).

Forecasting model

The first piece of the protection strategy is a forecasting model. In this example the return expectations are taken as the one-step-ahead predictions from a simple ARIMA(1, 1, 1) model. To produce these forecasts for the back-test period, two functions are defined in Listing 13.15. The first, **Forecast()**, is designed to take a log-level series as argument **x** and an order specification as argument **order**. In the function body, the coefficients of the ARIMA model are estimated and the one-step-ahead point forecast is produced. The return expectation is then formed as the difference between the latest log value and the point forecast. The functions and methods of package **forecast** are employed. **RetExp()** is a wrapper function for **Forecast()**, in the sense that the latter is applied per column if a multivariate object is used as input. The return forecasts are then swiftly returned by utilizing **fapply()** on the weekly log prices.

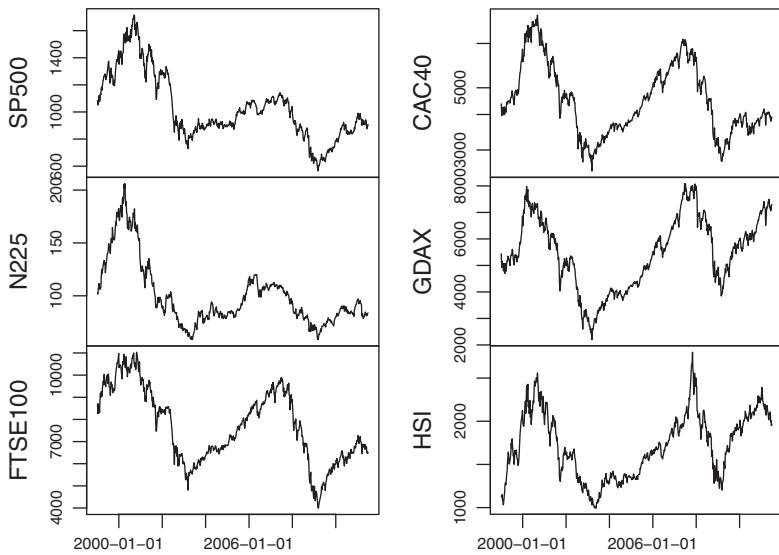


Figure 13.7 Trajectories of stock index values in euros.

R code 13.15 Forecasting model.

```

## Producing one-step ahead forecasts
Forecast <- function(x, order = c(1, 1, 1), ...) {
  mod <- arima(x, order = order, ...)
  f1 <- forecast(mod, h = 1)$mean
  re <- (f1 - tail(x, 1)) * 100
  re
}
RetExp <- function(x, order = c(1, 1, 1), ...) {
  ans <- apply(x, 2, Forecast, order = order, ...)
  ans
}
## Computing return forecasts
RE <- fapply(PELWeekly, from = spoints, to = epoints,
             FUN = RetExp)
head(RE)
tail(RE)

```

Risk model

The second piece consists of a market price risk model. In Listing 13.16 the ES for a confidence level of 95% is used. This downside risk measure is derived from the generalized Pareto distribution. Because the riskiness is determined for each loss series alone, the weighted sum thereof will form an upper bound for the overall portfolio downside risk. That is, a perfect concordance of the stock losses is assumed.

R code 13.16 Risk model.

```

## Computing market risk measures
## Long/Short risk
RiskFac <- -1.0 * sign(RE)
ES <- matrix(0, ncol = NAssets, nrow = size)
GpdEs <- function(x, nextremes = 30, method = "pwm",
                   level = 0.95, RiskFac) {
  x <- RiskFac * x
  mod <- gpd(data = x, nextremes = nextremes, method = method)
  GpdEs <- riskmeasures(mod, level)[3]
  GpdEs
}
for(i in idx){
  DatSub <- window(REDWeekly, start = spoints[i],
                    end = epoints[i])
  FacSub <- window(RiskFac, start = epoints[i],
                    end = epoints[i])
  for(j in 1:6){
    x <- na.omit(DatSub[, j])
    ES[i, j] <- GpdEs(x, RiskFac = c(FacSub[, j]))
  }
}
ES <- timeSeries(ES, charvec = epoints)
colnames(ES) <- colnames(REDWeekly)
head(ES)
tail(ES)

```

Given that losses are expressed as positive numbers, in line 3 the sign factors are determined from the return expectations to be used in altering the signs of the actual returns such that these comply with the pertinent losses from long or short positions. In line 4 a `matrix` object `ES` is defined to hold the relevant downside risk estimates. Next, the function `GpdEs()` is created. In its body, the 30 largest losses are considered in the estimation of the GPD parameters. The EVT-based risk measures are then recovered from the output of the function `riskmeasures()`. The elements of the matrix `ES` are next filled by means of a double `for` loop, where in the outer loop the subsample of the discrete returns is extracted from `REDWeekly` and the trade/loss direction from the object `RiskFac`. In the inner loop, the risk measures are determined for each stock market. In the last block of statements the results are converted to an object of class `timeSeries`.

Formulation of linear program

The third and final piece is the formulation of the linear program. This is contained in the function `Lp()` shown in Listing 13.17. The function is specified with four arguments. The first, `RE`, takes the return expectations which are the parameters of

R code 13.17 Formulation of the linear program.

```

## Creating LP
Lp <- function(RE, ES, Buffer, ub = 0.4){
  obj <- as.vector(RE)
  ## Initialise LHS matrix and RHS vector
  nvar <- length(obj)
  ## Wealth constraint
  a1 <- rep(1, nvar)
  b1 <- 1
  d1 <- "<="
  ## Risk constraint
  a2 <- as.vector(ES)
  b2 <- Buffer
  d2 <- "<="
  ## Upper bound
  a3 <- diag(nvar)
  b3 <- rep(ub, nvar)
  d3 <- rep("<=", nvar)
  ## Combining
  A <- rbind(a1, a2, a3)
  b <- c(b1, b2, b3)
  d <- c(d1, d2, d3)
  ans <- Rglpk_solve_LP(obj, mat = A, dir = d, rhs = b,
                        max = TRUE)
  ans
}

```

the objective function to be maximized. The second, `ES`, holds the downside risk `ES` estimates as derived from the GPD. The object `Buffer` is for the risk budget, such that the portfolio's downside risk as a weighted sum of its components shall not exceed this limit. The last argument, `ub`, will be used as an upper bound on the portfolio weights in order to circumvent concentrated portfolio solutions.

In the first line of the function's body, the return expectations are coerced into a vector and assigned to the object `obj`, which will be employed later in the call to `Rglpk_solve_LP()`. The number of portfolio constituents is determined next. The following three blocks of statements contain: (i) the budget constraint, such that the sum of weights shall not exceed unity; (ii) the risk constraint, such that the portfolio risk shall not exceed the buffer; and (iii) the upper bound constraints. In each of these three blocks the row-wise entries of the constraint matrix, the inequalities, and the right-hand-side values are specified. These items are then collected into the `matrix` object `A`, the inequality vector `d`, and the binding constraint values `b`. These objects are then used in the call to the linear program function of the package `Rglpk`. The function returns a `list` object with the solution and further output as described in Section 12.5.3.

Portfolio simulation

The previously created objects are now used in a portfolio simulation. Given the path dependency of the risk budget, the back-test is cast in a `for` loop as shown in Listing 13.18. But first, an object `LO` of class `timeSeries` is created in which the portfolio equity is stored. It is initialized to 100 monetary units. Next, the portfolio's floor level is defined and the returns pertinent to the back-testing period are extracted. It is further assumed that the funds can be invested at a money market rate of 1% per annum. Within the `for` loop, the 90% current wealth level is determined and compared to the portfolio floor to be protected. Only if the former is greater than the latter is the optimization carried out. Otherwise, the remaining funds earn interest at the money market rate until the initial floor level is exceeded again. Given that the forecasts for all markets can be negative at the same time, and hence a long-only investor would lose from exposure to any of these markets, the wealth would also earn interest only. This situation is checked in the inner `if/else` clause. In the final lines of the listing,

R code 13.18 Portfolio simulation.

```

## Recursive back-test
## Initialising variables
LO <- timeSeries(rep(NA, size), charvec = epooints)
LO[1] <- 100
PLevel <- 0.9
FLO <- LO[1, ] * PLevel
Returns <- REDWeekly[epooints, ]
MoneyRate <- 1.01^(1/52) - 1
## Simulation
for(i in 2:size){
  BLO <- c(PLevel * LO[i - 1, ])
  if(BLO < FLO){
    LO[i, ] <- LO[i - 1, ] * (1 + MoneyRate)
  } else {
    re <- c(RE[i - 1, ])
    if(all(re <= 0)){
      LO[i, ] <- LO[i - 1, ] * (1 + MoneyRate)
    } else {
      es <- c(ES[i - 1, ])
      r <- c(Returns[i, ])
      B <- c(LO[i - 1, ]) / c(FLO) - 1
      ans <- Lp(RE = re, ES = es, Buffer = B, ub = 0.4)
      w <- ans$solution
      LO[i, ] <- LO[i - 1, ] * (1 + t(w) %*% c(r))
    }
  }
}

```

the solution of the linear program is recovered and the proceeds from the current allocation are expressed in terms of the portfolio wealth.

Analysis of results

The simulated wealth trajectory is now contained in the object `LO`. The outcome of this strategy is compared with the solution of an equal-weighted strategy. In Listing 13.19 the portfolio equity according to the equal-weighted strategy is computed first and the two equity curves are plotted in Figure 13.8.

Clearly, the trajectory according to the TAA strategy leads to a higher wealth level at the end of the simulation period than the equal-weighted strategy. Furthermore, the draw-down witnessed during the financial market crisis is less pronounced in the case of the former strategy. These two characteristics are also mirrored by the

R code 13.19 Comparison of portfolio allocations.

```

## Equal-weighted long-only strategy
EwRetfac <- 1 + rowMeans(Returns)
EwRetfac[1] <- 100
EW <- timeSeries(cumprod(EwRetfac), epoints)
## Plot of portfolio wealth
ylims <- range(cbind(LO, EW))
plot(LO, ylim = ylims, xlab = "", ylab = "Index")
lines(EW, col = "blue", lty = 2)
legend("topleft",
       legend = c("TAA long-only", "EW long-only"),
       lty = 1:2, col = c("black", "blue"))
## Portfolio analytics
library(PerformanceAnalytics)
## Portfolio returns
LORet <- returns(LO, method = "discrete", percentage = FALSE,
                  trim = TRUE)
EWRet <- returns(EW, method = "discrete", percentage = FALSE,
                  trim = TRUE)
## VaR
LOVAR <- -100 * VaR(LORet, p = 0.95, method = "gaussian")
EWVAR <- -100 * VaR(EWRet, p = 0.95, method = "gaussian")
## ES
LOES <- -100 * ES(LORet, p = 0.95, method = "gaussian")
EWES <- -100 * ES(EWRet, p = 0.95, method = "gaussian")
## Sharpe
LOSR <- SharpeRatio(LORet)
EWSR <- SharpeRatio(EWRet)
## Annualised returns
LORA <- Return.annualized(LORet, scale = 52)
EWRA <- Return.annualized(EWRet, scale = 52)

```

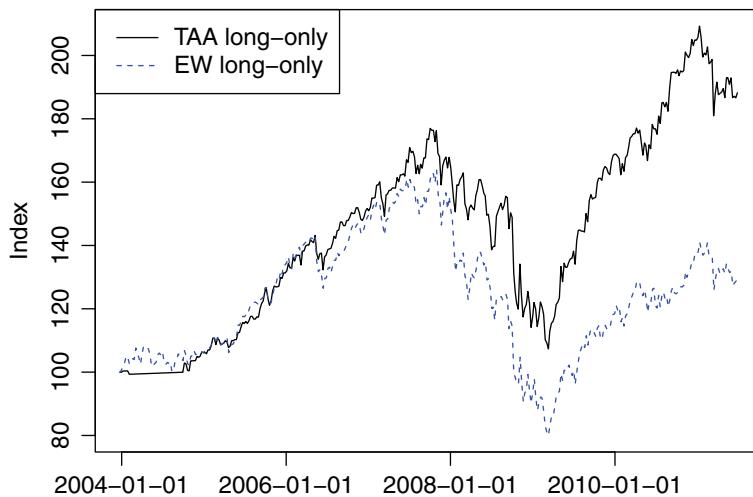


Figure 13.8 Progression of portfolio wealth.

Table 13.6 Key portfolio performance measures.

Measures	TAA long-only	EW long-only
VaR 95%	3.37	3.64
VaR 99%	4.27	4.59
Sharpe ratio	0.09	0.04
Return annualized %	8.76	3.50

performance and risk-related characteristics as computed in the final part of Listing 13.19 and shown in Table 13.6.

References

- Akaike H. 1981 Likelihood of a model and information criteria. *Journal of Econometrics* **16**, 3–14.
- Azzalini A. 2015 *The R package sn: The skew-normal and skew-t distributions (version 1.2-1)* Università di Padova, Italia.
- Banerjee A., Dolado J., Galbraith J., and Hendry D. 1993 *Co-Integration, Error-Correction, and the Econometric Analysis of Non-Stationary Data*. Oxford University Press, New York.
- Birgin E. and Martinez J. 2008 Improving ultimate convergence of an augmented Lagrangian method. *Optimization Methods and Software* **23**(2), 177–195.
- Black F. 1989 Universal hedging: Optimizing currency risk and reward in international equity portfolios. *Financial Analysts Journal* **51**(1), 16–22.
- Black F. and Jones R. 1987 Simplifying portfolio insurance. *The Journal of Portfolio Management* pp. 48–51.

- Black F. and Litterman R. 1990 *Asset allocation: Combining investor views with market equilibrium*. Technical report, Goldman Sachs Fixed Income Research.
- Black F. and Litterman R. 1991 *Global asset allocation with equities, bonds, and currencies*. Technical report, Goldman Sachs Fixed Income Research.
- Black F. and Litterman R. 1992 Global portfolio optimization. *Financial Analysts Journal* **48**(5), 28–43.
- Black F. and Perold A. 1992 Theory of constant proportion portfolio insurance. *Journal of Economics, Dynamics & Control* **16**(3/4), 403–426.
- Black F. and Rouhani R. 1989 *Institutional Investor Focus on Investment Management* Ballinger Cambridge, MA pp. 695–708.
- Bookstaber R. and Langsam J. 2000 Portfolio insurance trading rules. *The Journal of Futures Markets* **8**, 15–31.
- Box G. and Jenkins G. 1976 *Time Series Analysis: Forecasting and Control* revised edn. Holden-Day, San Francisco.
- Brandt P. 2015 *MSBVAR: Markov-Switching, Bayesian, Vector Autoregression Models*. R package version 0.9-2.
- Conn A., Gould N., and Toint P. 1991 A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal of Numerical Analysis* **28**(2), 545–572.
- Dickey D. A. and Fuller W. A. 1979 Distributions of the estimators for autoregressive time series with a unit root. *Journal of the American Statistical Association* **74**, 427–431.
- Dickey D. A. and Fuller W. A. 1981 Likelihood ratio statistics for autoregressive time series with a unit root. *Econometrica* **49**, 1057–1072.
- Diebold F. and Mariano R. 1995 Comparing predictive accuracy. *Journal of Business & Economic Statistics* **13**, 253–263.
- Elliott G., Rothenberg T., and Stock J. 1996 Efficient tests for an autoregressive unit root. *Econometrica* **64**(4), 813–836.
- Engle R. and Granger C. 1987 Co-integration and error correction: Representation, estimation and testing. *Econometrica* **55**, 251–276.
- Fair R. 1984 *Specification, Estimation, and Analysis of Macroeconomic Models*. Harvard University Press, Cambridge, MA.
- Fair R. 1998 *Testing Macroeconometric Models*. Harvard University Press, Cambridge, MA.
- Fair R. 2004 *Estimating How the Macroeconomy Works*. Harvard University Press, Cambridge, MA.
- Fraley C., Leisch F., Mächler M., Reisen V., and Lemonte A. 2012 *fracdiff: Fractionally differenced ARIMA aka ARFIMA(p,d,q) models*. R package version 1.4-2.
- Gilbert P. 1993 *State space and ARMA models: An overview of the equivalence*. Working Paper 93-4, Bank of Canada, Ottawa, Canada.
- Gilbert P. 1995 Combining VAR estimation and state space model reduction for simple good predictions. *Journal of Forecasting: Special Issue on VAR Modelling* **14**, 229–250.
- Gilbert P. 2000 A note on the computation of time series model roots. *Applied Economics Letters* **7**, 423–424.
- Gilbert P. 2009 *Brief User's Guide: Dynamic Systems Estimation*. CRAN.
- Gilbert P. 2015a *EvalEst: Dynamic Systems Estimation – Extensions*. R package version 2015.4-2.

- Gilbert P. 2015b *tframe: Time Frame Coding Kernel*. R package version 2015.12-1.
- Golan A., Judge G., and Miller D. 1996 *Maximum Entropy Econometrics* Series in Financial Economics and Quantitative Analysis. John Wiley & Sons Ltd., Chichester.
- Granger C. W. J. 1981 Some properties of time series data and their use in econometric model specification. *Journal of Econometrics* **16**, 121–130.
- Greene W. 2008 *Econometric Analysis* 7th edn. Prentice Hall, Upper Saddle River, NJ.
- Hamilton J. D. 1994 *Time Series Analysis*. Princeton University Press, Princeton.
- Hannan E. and Quinn B. 1979 The determination of the order of an autoregression. *Journal of the Royal Statistical Society B* **41**, 190–195.
- He G. and Litterman R. 2002 *The intuition behind Black-Litterman model portfolios*. Working paper, Goldman Sachs Group, Inc. Quantitative Strategy Group, <http://ssrn.com/abstract=334304>.
- Hendry D. F. 1995 *Dynamic Econometrics*. Oxford University Press, Oxford.
- Henningsen A. and Hamann J. 2007 systemfit: A package for estimating systems of simultaneous equations in R. *Journal of Statistical Software* **23**(4), 1–40.
- Hyndman R. 2016 *forecast: Forecasting functions for time series and linear models*. R package version 7.1.
- Johansen S. 1988 Statistical analysis of cointegration vectors. *Journal of Economic Dynamics and Control* **12**, 231–254.
- Johansen S. 1991 Estimation and hypothesis testing of cointegration vectors in Gaussian vector autoregressive models. *Econometrica* **59**(6), 1551–1580.
- Johansen S. 1995 *Likelihood-Based Inference in Cointegrated Vector Autoregressive Models* Advanced Texts in Econometrics. Oxford University Press, Oxford.
- Johansen S. and Juselius K. 1990 Maximum likelihood estimation and inference on cointegration, with applications to the demand for money. *Oxford Bulletin of Economics and Statistics* **52**(2), 169–210.
- Johansen S. and Juselius K. 1992 Testing structural hypothesis in a multivariate cointegration analysis of the PPP and the UIP for UK. *Journal of Econometrics* **53**, 211–244.
- Johnson S. 2015 *The NLOpt nonlinear-optimization package* Massachusetts Institute of Technology Cambridge, MA. URL <http://ab-initio.mit.edu/nlopt>.
- Judge G., Hill R., Griffiths W., Lütkepohl H., and Lee T. 1985 *The Theory and Practice of Econometrics* 2nd edn. John Wiley & Sons Inc., New York, NY.
- Judge G., Hill R., Griffiths W., Lütkepohl H., and Lee T. 1988 *Introduction to the Theory and Practice of Econometrics* 2nd edn. John Wiley & Sons Inc., New York, NY.
- Kraft D. 1988 *A software package for sequential quadratic programming*. Technical Report DFVLR-FB 88-28, Institut für Dynamik der Flugsysteme, Oberpfaffenhofen, Germany.
- Kraft D. 1994 Algorithm 733: TOMP-Fortran modules for optimal control calculations. *ACM Transactions on Mathematical Software* **20**(3), 262–281.
- Kwiatkowski D., Phillips P., Schmidt P., and Shin Y. 1992 Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root?. *Journal of Econometrics* **54**, 159–178.
- Lee W. 2000 *Theory and Methodology of Tactical Asset Allocation*. Frank J. Fabozzi Associates, New Hope, Pennsylvania.
- Leland H. and Rubinstein M. 1976 *Portfolio insurance: A guide to dynamic hedging* John Wiley & Sons chapter 2.

- Luksan L. and Vlcek J. 1998 *Sparse and partially separable test problems for unconstrained and equality constrained optimization*. Research Report V-767, Academy of Sciences of the Czech Republic, Institute of Computer Science, Prague, Czech Republic. URL <http://www.cs.cas.cz/luksan/subroutines.html>.
- Luksan L., Matonoha C., and Vlcek J. 2004 *LSA: Algorithms for large-scale unconstrained and box constrained optimization*. Research Report V-896, Academy of Sciences of the Czech Republic, Institute of Computer Science, Prague, Czech Republic. URL <http://www.cs.cas.cz/luksan/subroutines.html>.
- Lütkepohl H. 2006 *New Introduction to Multiple Time Series Analysis*. Springer-Verlag, New York.
- Lütkepohl H. and Krätzig M. 2004 *Applied Time Series Econometrics*. Cambridge University Press, Cambridge.
- Lütkepohl H., Saikkonen P., and Trenkler C. 2004 Testing for the cointegrating rank of a VAR with level shift at unknown time. *Econometrica* **72**(2), 647–662.
- MacKinnon J. 1991 Critical values for cointegration tests In *Long-Run Economic Relationships: Readings in Cointegration* (ed. Engle R. F. and Granger C. W. J.) Advanced Texts in Econometrics Oxford University Press Oxford, UK chapter 13.
- MacKinnon J. 1996 Numerical distribution functions for unit root and cointegration tests. *Journal of Applied Econometrics* **11**, 601–618.
- Markowitz H. 1952 Portfolio selection. *The Journal of Finance* **7**(1), 77–91.
- Meucci A. 2006a Beyond Black–Litterman in practice: A five-step recipe to input views on non-normal markets. *Risk* **19**(9), 114–119.
- Meucci A. 2006b Beyond Black–Litterman: Views on non-normal markets. *Risk* **19**(2), 96–102.
- Meucci A. 2010a The Black–Litterman approach: Original model and extensions In *The Encyclopedia of Quantitative Finance* (ed. Cont R.) vol. I John Wiley & Sons Chichester, UK p. 196.
- Meucci A. 2010b *Fully flexible views: Theory and practice*. Working paper, Symmys, <http://ssrn.com/abstract=1213325>.
- Perold A. 1986 *Constant portfolio insurance* Harvard Business School, unpublished manuscript.
- Perold A. and Sharpe W. 1988 Dynamic strategies for asset allocation. *Financial Analyst Journal* pp. 16–27.
- Peterson B. and Carl P. 2014 *PerformanceAnalytics: Econometric tools for performance and risk analysis*. R package version 1.4.3541.
- Pfaff B. 2007 *UseR in the financial sector International Workshop on Computational and Financial Econometrics*, Geneva, Switzerland.
- Pfaff B. 2008a *Analysis of Integrated and Cointegrated Time Series with R* 2nd edn. Springer, New York.
- Pfaff B. 2008b *Tactical asset allocation: Putting the pieces together* 2nd International R/Rmetrics User and Developer Workshop, Meielisalp, Lake Thune, Switzerland.
- Pfaff B. 2008c VAR, SVAR and SVEC models: Implementation within R package vars. *Journal of Statistical Software* **27**(4), 1.
- Phillips P. and Ouliaris S. 1990 Asymptotic properties of residual based tests for cointegration. *Econometrica* **58**, 165–193.

- Phillips P. and Perron P. 1988 Testing for a unit root in time series regression. *Biometrika* **75**(2), 335–346.
- Quinn B. 1980 Order determination for multivariate autoregression. *Journal of the Royal Statistical Society B* **42**, 182–185.
- Schmidt P. and Phillips P. 1992 LM tests for a unit root in the presence of deterministic trends. *Oxford Bulletin of Economics and Statistics* **54**(3), 257–287.
- Schwarz H. 1978 Estimating the dimension of a model. *The Annals of Statistics* **6**, 461–464.
- Sharpe W. 1964 Capital asset prices: A theory of market equilibrium. *Journal of Finance* pp. 425–442.
- Sharpe W. 1974 Imputing expected security returns from portfolio composition. *Journal of Financial and Quantitative Analysis* pp. 463–472.
- Sims C. A. 1980 Macroeconomics and reality. *Econometrica* **48**, 1–48.
- Sims C. and Zha T. 1998 Bayesian methods for dynamic multivariate models. *International Economic Review* **39**(4), 949–968.
- Trapletti A. and Hornik K. 2015 *tseries: Time Series Analysis and Computational Finance*. R package version 0.10-34.
- Ulrich J. 2016 *TTR: Technical Trading Rules*. R package version 0.23-1.
- Waggoner D. and Zha T. 2003 A Gibbs sampler for structural vector autoregressions. *Journal of Economic Dynamics & Control* **28**, 349–366.
- Würtz D. 2013a *fArma: ARMA Time Series Modelling*. R package version 3010.79.
- Würtz D. 2013b *fTrading: Technical Trading Analysis*. R package version 3010.78.
- Würtz D. 2013c *fUnitRoots: Trends and Unit Roots*. R package version 3010.78.
- Würtz D. and Chalabi Y. 2013 *fGarch: Rmetrics – Autoregressive Conditional Heteroskedastic Modelling*. R package version 3010.82.
- Würtz D. and Setz T. 2014 *fMultivar: Rmetrics – Analysing and Modeling Multivariate Financial Return Distributions*. R package version 3011.78.
- Würtz D., Setz T., and Chalabi Y. 2014 *fPortfolio: Rmetrics – Portfolio Selection and Optimization*. R package version 3011.81.
- Ypma J., Borchers H., and Eddelbüttel D. 2014 *nloptr: R interface to NLOpt*. R package version 1.0.4.
- Zivot E. and Andrews D. 1992 Further evidence on the Great Crash, the Oil-Price Shock, and the Unit-Root Hypothesis. *Journal of Business & Economic Statistics* **10**(3), 251–270.

14

Probabilistic utility

14.1 Overview

In Chapter 5, a link between Markowitz-type portfolio optimization and the maximization of expected utility (MEU) was established (see Section 5.2). Throughout the remainder of this book, the Markowitz model served as a starting point and amendments/refinements have been exhibited. Financial returns and/or loss factors have been interpreted as random variables and one quest was how to address the riskiness most appropriately for an investor. As such, various statistical distributions and concepts have been introduced, but also risk measures that differ from the ubiquitously encountered variance-covariance matrix. It was then shown how these more broadly defined risk concepts can be employed in the formulation of portfolio optimization problems. The “optimal” solutions obtained for these portfolio optimizations have then been used as fixed, deterministic wealth allocations to the risk assets. Here, the concept of probabilistic utility (PU) begins on the one side and closes the circle to utility-maximizing portfolio choices on the other side.

In the next section the concept of probabilistic utility will be presented. As will become evident, applying this method requires the evaluation of a high-dimensional unknown density. This is accomplished by means of Markov chain Monte Carlo (MCMC) techniques. Hence, in the subsequent section two approaches for doing so will be briefly discussed. The section on the available **R** packages will focus on MCMC-related facilities. The chapter ends with a comparison of probabilistic and maximal utility portfolio optimizations.

14.2 The concept of probabilistic utility

The concept of probabilistic utility was introduced into the literature by Rossi et al. (2002) and Marschinski et al. (2007). It is applicable to portfolio selection problems that are derived from utility functions, for instance in the case of mean-variance optimization in the form of

$$U = \lambda \boldsymbol{\omega}' \boldsymbol{\mu} - (1 - \lambda) \boldsymbol{\omega}' \boldsymbol{\Sigma} \boldsymbol{\omega}, \quad (14.1)$$

where $\lambda \in [0, 1]$ denotes the risk aversion parameter, $\boldsymbol{\mu}$ the $(N \times 1)$ vector of expected returns, $\boldsymbol{\Sigma}$ the $(N \times N)$ variance-covariance matrix, and $\boldsymbol{\omega}$ the $(N \times 1)$ vector of portfolio allocations for the N assets to be considered in the portfolio optimization problem. Hence, the solution to this utility-based portfolio selection problem is sensitive to its input parameters, namely λ , $\boldsymbol{\mu}$, and $\boldsymbol{\Sigma}$.

As exhibited in Chapter 10, the empirically encountered problem of concentrated portfolio solutions and/or wealth allocations that are fairly sensitive with respect to the input parameters can be ameliorated by either applying robust estimation techniques and/or employing robust optimization methods. This is one approach for coping with the problems encountered by directly optimizing Markowitz-type portfolios. Marschinski et al. approached this issue from a different and novel angle. The parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ that enter into the formulation of the optimization problem are random variables, because they are estimates for the unknown population moments of the returns of financial instruments, which are themselves random variables. As such, the allocation vector $\boldsymbol{\omega}$ is a random variable, too. An implication of this reasoning is a probabilistic interpretation of utility functions in this context as advocated by Marschinski et al. A utility function is now interpreted as the logarithm of the probability density for a portfolio. The optimal allocation is defined as the expected value of the portfolio's weights with respect to that probability, that is, the weights are viewed as parameters of this distribution. This concept can be stated more formally: one is now considering a function

$$u = u(\boldsymbol{\omega}, U, \boldsymbol{\theta}), \quad (14.2)$$

where $\boldsymbol{\omega}$ is weight vector, U the assumed utility function (e.g., as in (14.1)), and $\boldsymbol{\theta}$ a catch-all parameter vector (e.g., expected returns, dispersion, risk sensitivity). Expected utility is proportional to the logarithm of a probability measure,

$$\boldsymbol{\omega} \sim \mathsf{P}(\boldsymbol{\omega}|U, \boldsymbol{\theta}) = Z^{-1}(v, U, \boldsymbol{\theta}) \exp(vu(\boldsymbol{\omega}, U, \boldsymbol{\theta})), \quad (14.3)$$

where Z is a normalizing constant defined as

$$Z(v, U, \boldsymbol{\theta}) = \int_{\mathfrak{D}(\boldsymbol{\omega})} \exp(vu(\boldsymbol{\omega}, U, \boldsymbol{\theta})) \, d\boldsymbol{\omega} \quad (14.4)$$

such that the area below the density is one. The hyper-parameter v influences the rate of asymptotic convergence. It can be defined as $v = pT^\gamma$, where p and γ are constants and T is the sample size. For instance, setting $p = 1$ and $\gamma = \frac{1}{2}$ would

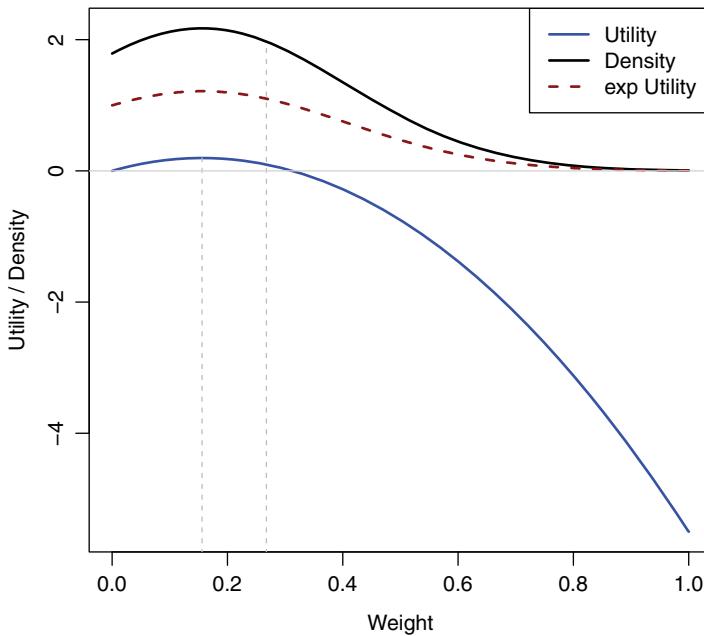


Figure 14.1 Concept of probabilistic utility: one risky asset.

imply a \sqrt{T} conversion rate. That is, for $v \rightarrow \infty$ the solution converges to the “true” utility-maximizing wealth allocation. On the other hand, for $v \rightarrow 0$ an equal-weighted (naive) allocation would result. The portfolio solution of the probabilistic utility concept is then defined as the expected value:

$$\bar{\omega}(U, \theta) = Z^{-1}(v, U, \theta) \int_{\mathfrak{D}(\omega)} \omega \exp(vu(\omega, U, \theta)) d\omega. \quad (14.5)$$

The concept of probabilistic utility is now elucidated by a simple example. Consider the case of two assets—one risk-less and one risky asset. The risky asset has an assumed excess return of 5% and a standard deviation risk of 4%. The shape of the quadratic utility function with a risk aversion parameter of $\lambda = \frac{1}{2}$ is shown as a dark gray solid line in Figure 14.1.

The maximum utility is achieved by a wealth allocation of 15.62% in the risky asset. The exponentiated utility levels are shown as a black dashed line. Though the function values are now all positive, this curve would not qualify as a density, because the integral is not equal to one (area enclosed below the curve and the abscissa). The latter requirement and the qualification as a proper density is met once the normalization of the exponentiated utilities has been applied. The resulting continuous density curve is shown as a black solid line in Figure 14.1. The expected value and hence the share invested in the risky asset is indicated as a light gray dashed line and takes a value of 26.74%. The two allocations are quite distinct from each other, which is partly explained by the setting of $v = 1$ in this example.

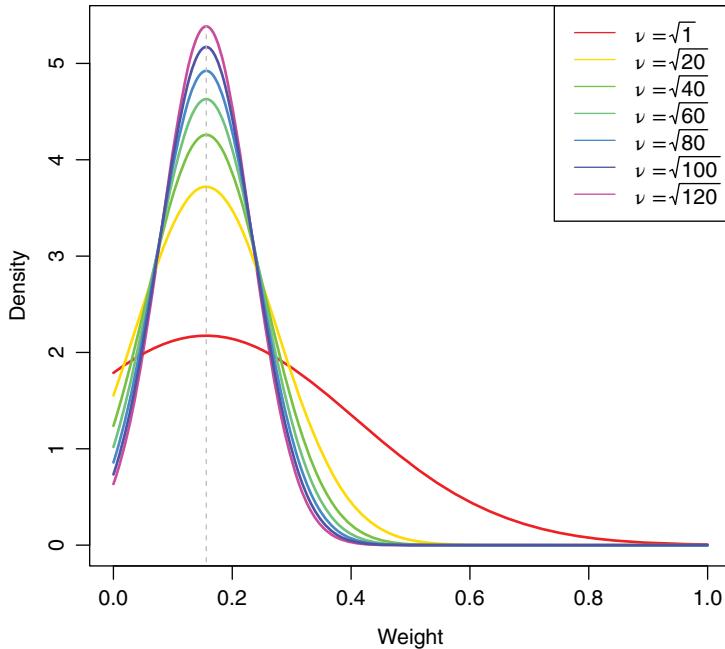


Figure 14.2 Probabilistic utility: asymptotic property for $v = \sqrt{T}$.

Staying within this two-asset example, the convergence behavior is depicted in Figure 14.2. Here, the hyper-parameter v is set according to $v = \sqrt{T}$ for $T = \{1, 20, 40, 60, 80, 100, 120\}$.

As is evident from this graph, as $T \rightarrow \infty$ the probability mass collapses over the “true” utility-maximizing allocation.

As is evident from (14.5), the application of the probabilistic utility concept requires the evaluation of an N -dimensional integral, that is, the count of financial instruments considered in a utility-based portfolio selection problem. Ordinarily, these integrals cannot be evaluated in closed form, and hence a researcher has to resort to numerical techniques for determining the expected value of this expression. One means of doing so is by employing Markov chain Monte Carlo methods, which are briefly described in the following section.

14.3 Markov chain Monte Carlo

14.3.1 Introduction

Recall (14.3)–(14.5) from the previous section. Applying the concept of the probabilistic reinterpretation of utility functions requires the evaluation of a high-dimensional density function, that is, the count of assets considered in the

portfolio. In general, a closed-form estimator for computing the expected value of $\bar{\omega}(U, \theta)$ as in (14.5) cannot be provided. Hence, to obtain an estimate for the portfolio solution one has to resort to techniques for evaluating the integral numerically; Markov chain Monte Carlo (MCMC) is one such method. MCMC techniques are covered in numerous textbook expositions and articles. It is far beyond the scope of this brief section to provide the reader with a full exposition, but the aim is rather to provide a concise treatment and foster his/her understanding such that the content in the following sections can be better comprehended. For additional expositions of this topic, the reader is referred to Brandimarte (2014), Brooks et al. (2011), Gilks et al. (1995), and Robert and Casella (2010).

14.3.2 Monte Carlo approaches

The notion of Markov chain Monte Carlo consists of two terms, namely “Markov chains” and “Monte Carlo.” We will address the latter term now and the former in the following subsection.

In a classical Monte Carlo integration setting the following integral for computing the expected value of a (density) function $h(\mathbf{x})$ is considered:

$$\mathbb{E}[h(\mathbf{x})] = \int_{\mathfrak{D}(\mathbf{x})} f(\mathbf{x})h(\mathbf{x})d\mathbf{x}, \quad (14.6)$$

where the random vector \mathbf{x} is distributed according to the density $f()$. An estimate for the expected value of $h(\mathbf{x})$ can then be computed as

$$\bar{h}_N = \frac{1}{N} \sum_{i=1}^N h(x_i) \quad (14.7)$$

for a given random sample $\mathbf{x} = \{x_1, \dots, x_N\}$ drawn from the distribution $f()$. The variance of this estimator is given by

$$\text{VAR}(\mathbb{E}[h(\mathbf{x})]) = \sigma_N^2 = \frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{h}_N)^2. \quad (14.8)$$

If the sequence $\{\sigma_1, \sigma_2, \sigma_3, \dots\}$ is bounded, then the variance decreases asymptotically to zero with rate $1/N$. Because of the central limit theorem, the estimation error,

$$\frac{\bar{h}_N - \mathbb{E}[h(\mathbf{x})]}{\sqrt{(\text{VAR}(\mathbb{E}[h(\mathbf{x})]))}}, \quad (14.9)$$

is approximately distributed as $\mathcal{N}(0, 1)$. Hence, the user can control the width of the confidence bounds by setting N to a sufficiently high number.

To elucidate the classical approach of Monte Carlo integration an example is provided in Listing 14.1. First, the utility function for one risky asset as in (14.1) is defined as object `U1()`. Next, the density function as in (14.3) and (14.4) is created as `U1DU()` with normalizing constant `Z`, whereby the formerly defined function `U1`

is invoked in the function body of `U1DU()`. The normalizing constant Z , that is, the surface below the exponential function of the product between the convergence parameter v and the utility function $u(\omega, U, \theta)$, is calculated by the `integrate()` function. Commencing on line 12 an estimate for $\hat{\omega}$ as in (14.5) is computed. Thus, the size of the Markov Chain has been set to $N = 1000$ and the normalizing constant Z has been factored out of the integral. The random variables are drawn from the uniform distribution with supporting interval $[0, 1]$. The estimate is 27.23%, which is fairly close to the theoretically computed value of 26.74%.

Starting from line 16 a graphical presentation of the progression of the estimates for the allocation with a two-sided error band is produced (see Figure 14.3).

With respect to the application of classical Monte Carlo integration to probabilistic utility functions, the problem arises that the normalizing constant for this density is in general unknown and difficult to compute. Methods that circumvent the computation of this normalizing constant are therefore sought. The concept of “accept–reject” sampling introduced by von Neumann (1951) into the literature is one means of doing so. This is an indirect method whereby random variables are drawn from a so-called “instrumental” or “candidate” distribution $G(\cdot)$ and are accepted as a random draw from the target distribution $F(\cdot)$ if they pass a test (to distinguish between density and distribution functions of a distribution, the latter are indicated by capitalized letters). Hence, this method can be applied if only the shape of the density $f(x)$ is known—see Brandimarte (2014, Chapter 14) or Robert and Casella (2010, Chapter 2), for instance.

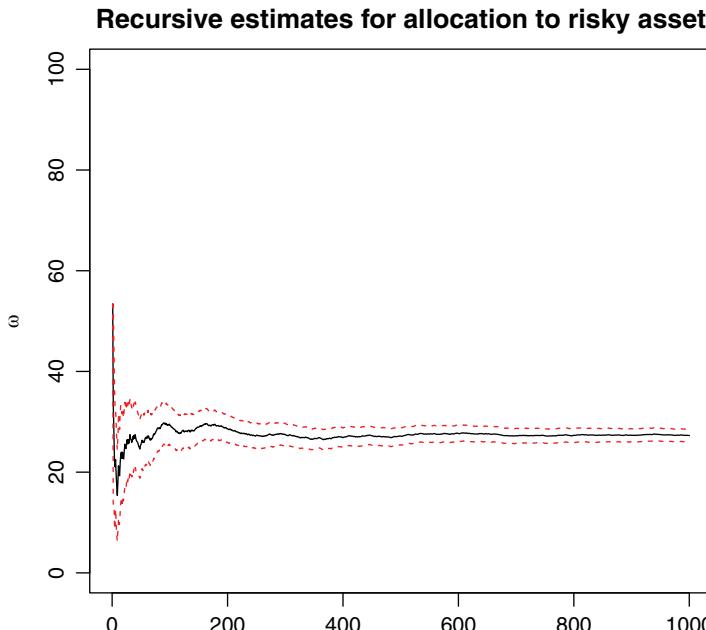


Figure 14.3 Monte Carlo integration: progression of estimates with two-sided error bands.

R code 14.1 Monte Carlo integration methods for probabilistic utility function.

```

## Utility function
U1 <- function(x, mu, risk, lambda){
  lambda * x * mu - (1 - lambda) * risk * x^2
}
## Unscaled density
UIDU <- function(x, mu = 5, risk = 16, lambda = 0.5, nu = 1){
  exp(nu * U1(x = x, mu = mu, risk = risk, lambda = lambda))
}
## Constant of density
Z <- integrate(UIDU, lower = 0, upper = 1)$value
## Classical MC integration for expected value of scaled UIDU
MCsize <- 1e3
u1 <- runif(MCsize)
omegal <- u1 * UIDU(u1) / Z * 100
omegalHat <- mean(omegal)
## Graphical presentation of recursive means with bands
omegalRecMean <- cumsum(omegal) / (1:MCsize)
omegalRecErr <- sqrt(cumsum((omegal - omegalRecMean)^2)) /
  (1:MCsize)
plot(omegalRecMean, type = "l",
      ylim = c(0, 100), ylab = expression(omega), xlab = "",
      main = "Recursive estimates for allocation to risky
              asset")
lines(omegalRecMean + 2 * omegalRecErr, col = "red", lty = 2)
lines(omegalRecMean - 2 * omegalRecErr, col = "red", lty = 2)
## Random draws from UIDU by accept-reject method
## Candidate distribution is a truncated Exponential with
## lambda = 2 and k = 1.5
library(truncdist)
k <- 1.5
y2 <- rtrunc(MCsize, spec = "exp",
              a = 0.0, b = 1.0, rate = 2)
u2 <- runif(MCsize)
AcceptRejectIdx <- which(u2 <= (UIDU(y2) /
  (k * dtrunc(y2, spec = "exp",
              a = 0, b = 1, rate = 2))))
omega2 <- y2[AcceptRejectIdx]
omega2Hat <- mean(omega2) * 100

```

The only constraints for applying this method are (i) that the candidate and target densities have the same domain ($g(x) > 0$ when $f(x) > 0$) and (ii) the ratio of the densities evaluated for all x is bounded by a constant: $f(x)/g(x) \leq k$. The method consists of three steps:

1. Generate a random draw from the candidate distribution $Y \sim G(\cdot)$.

2. Generate a random draw from the uniform distribution $U \sim \mathcal{U}_{[0,1]}$.
3. Evaluate $U \leq \frac{f(Y)}{kg(Y)}$.

If the test in item 3 is passed then Y is accepted as a random draw from $F(\cdot)$, that is, $X = Y$, otherwise the random draws Y and U are dropped. This process can be repeated N times. It should be noted that the count of acceptances is a binomial random variable with a probability of $1/k$. Therefore, choosing a candidate distribution akin to the target distribution will yield a less wasteful simulation compared to an ill-chosen density. In Figure 14.4, a graphical presentation of the accept–reject method is provided, which elucidates the last statement.

In this figure, the unscaled target density as given in Listing 14.1 is plotted in the interval $[0, 1]$. As candidate densities, a truncated exponential distribution with parameter $\lambda = 2$ and the uniform distribution have been chosen, signified by “Candidate 1 Density” and “Candidate 2 Density” in the figure, respectively. The scaling factors k_1 and k_2 have been set to 1 for the truncated exponential and 1.5 for the uniform distribution. From each candidate and from the uniform distribution, 20 random draws have been generated. The accepted and rejected points for each of the candidates are marked. These pairs are given in terms of (x, y) coordinates by $\{x, y\} = \{y_i, u_i \cdot k_i \cdot g_i(y_i)\}$ for $i = 1, 2$. Points that lie between the target and the candidate density are discarded and only those below the target density are considered. In this instance, 9 out of 20 draws from the truncated exponential distribution can be taken as samples from the target distribution; the corresponding value for the uniform candidate is 7.

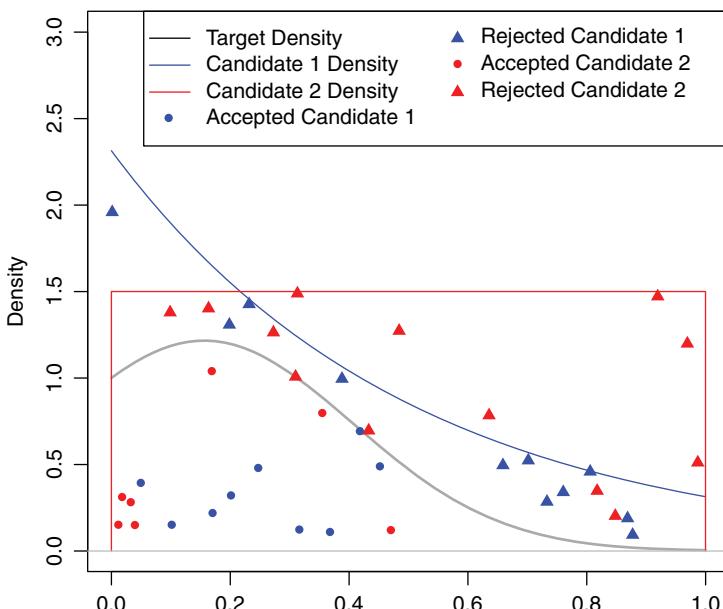


Figure 14.4 Accept–reject sampling: graphical representation.

In the final part of Listing 14.1 the method of accept–reject sampling is applied to the unscaled density, where a truncated exponential distribution in the interval $[0, 1]$ with parameter $\lambda = 2$ has been used as candidate. To produce random draws from the truncated exponential distribution and compute the value of the density, the facilities made available in the package **truncdist** have been employed (see Novomestky and Nadarajah, 2012). The estimate for the allocation into the risky asset can be computed as the mean of the random draws, and equals 24.5% in this instance.

A caveat of both approaches is that an almost prohibitively large number of random draws have to be created in cases of high-dimensional densities, and this still does not guarantee that the entire state space of distribution parameters is covered. This matters even more in the case of the accept–reject method because of discarded random draws. This problem can be ameliorated by modelling the search of the parameter’s state space as a specific stochastic process, which is the topic of the following section.

14.3.3 Markov chains

A Markov chain is a stochastic process governing the behavior of a sequence of dependent random variables. For a fuller treatment of this subject, the reader is referred to the monographs cited in Section 14.3.1 and also Robert and Casella (2004, Chapter 6) and Meyn and Tweedie (1993). As encountered and used in MCMC, a first-order Markov chain for a sequence of indexed random variables $X = \{X^{(t)}\}_{t \in T}$ and $T \subset [0, \infty)$ and $0 \in T$ is governed by the probability statement:¹

$$\mathbb{P}(X^{(t+1)}|x^{(t)}, x^{(t-1)}, \dots) = \mathbb{P}(X^{(t+1)}|x^{(t)}). \quad (14.10)$$

Hence, in a first-order Markov chain the distribution of the subsequent value of the random variable $X^{(t+1)}$ only depends on the most recent observation. In other words, the progression from one state to the next is only governed by the former. The conditional probability distribution $\mathbb{P}(\cdot|\cdot)$ is referred to as the “transition kernel” or “Markov kernel.”

A key property of a Markov chain is that it will converge to a stationary (invariant) distribution $f(\cdot)$, that is, $X^{(t)} \sim f(\cdot)$ then $X^{(t+1)} \sim f(\cdot)$ for $t \rightarrow \infty$. In other words, a Markov chain will “forget” its initial state $X^{(0)}$. The unique existence of this stationary distribution necessitates a constraint on the transition kernel, which is termed *irreducibility*. This attribute means that any set of states can be reached from any other state in a finite number of steps, more formally stated as $\mathbb{P}(x|\cdot) > 0$. Implied by the irreducibility attribute is the chain’s characteristic of *recurrency*. Given that the chain can reach any state in its allowable space as it progresses, it can return to any arbitrary non-negligible set of states an infinite number of times. Under these conditions, the realizations of a Markov chain of length T , discarding the first m observations as burn-in period, can be employed for calculating an estimate of the expected value for

¹ Note that the index is now superscripted in parenthesis to better distinguish it from a time series and avoid any ambiguities with former usages of the time subscript t .

the stationary distribution $\mathbf{E}_f[h(X)]$ by an ergodic average

$$\bar{h}_T = \frac{1}{T-m} \sum_{t=m+1}^T h(X^{(t)}) \rightarrow \mathbf{E}_f[h(X)] \quad \text{as } T \rightarrow \infty. \quad (14.11)$$

If the variance for this process is bounded, that is, $\sigma_h^2 = \text{VAR}_f[h(X)] < \infty$, then the central limit theorem holds and the estimator converges geometrically to the expected value. Hence, a (numerical) standard error is given by $\sqrt{\text{VAR}_f(\bar{h}_N)}$. If the Markov chain $\{h(X^{(t)})\}$ can be approximated by a first-order autoregressive process with AR(1) parameter θ_1 , then a (numerical) standard error for the estimator in (14.11) is given as $\sqrt{\sigma_h^2/N \cdot (1 + \theta_1)/(1 - \theta_1)}$. The first factor of the radicand is the variance in the case of an independent sample; the second is a correction factor that takes the size of the autocorrelation into account. This term is for a stable positive autocorrelation greater than one and hence error bands will be inflated (become wider) as a function of the autocorrelation. This can be viewed as a price to be paid for running a Markov chain compared to direct sampling approaches.

In Listing 14.2, the concept of Markov chains and their properties are elucidated. Within this listing it is assumed that the transition kernel is an AR(1) process: $X^{(t+1)} = \theta_0 + \theta_1 X^{(t)} + \theta^{(t+1)}$, with $\theta^{(t+1)} \sim \mathcal{N}(0, 1)$, $\forall t$. That is, the chain is simulated as $X^{(t+1)}|x^{(t)} \sim \mathcal{N}(\theta_0 + \theta_1 x^{(t)}, 1)$. This recursion is cast in the function `MCsim()`. Next, the required parameters are initialized and two Markov Chains are created, each with a size of 10 000 observations, but with differing starting values. A plot of the progression for the first 100 observations for each chain is created next (see Figure 14.5). Even though the starting values have been set to 14 and -10 , both chains pretty rapidly converge to a common path.

In the remaining part of Listing 14.2, estimates for the parameters of the stationary distribution, namely its expected value and variance, are computed and compared to the—in this example, known—population counterparts. For a stable AR(1) model with normally distributed innovations, these are given as $\mu = \theta_0/(1 - \theta_1)$ and $\sigma_\theta^2/(1 - \theta_1)$. In addition, the numerical standard error for the estimate of the expected value of the stationary distribution is computed. These computations are all conducted with the output of the first generated Markov chain. Furthermore, the first 5000 realizations of this chain have been discarded from the computations, that is, these observations have been treated as the burn-in period.

A caveat of this example is the reversed reasoning and hence it should be viewed only as an example. That is, a specific transition kernel has been assumed that does yield the implied stationary distribution of an AR(1) process. Matters are different in MCMC applications. Here, the target density, for example (14.3), is set as the stationary (limiting) distribution and the transition kernel that will yield this limiting distribution is unknown. Hence, the quest is to find a transition kernel that has a stationary distribution as its outcome equal to the target density—the distribution of interest. A solution to this problem is discussed in the next section, where the concept of the indirect accept–reject sampling approach presented in Section 14.3.2 will be revisited.

R code 14.2 Simulation of Markov chains.

```

MCsim <- function(x0 = 0, n, theta0 , theta1){
  ans <- vector()
  length(ans) <- n + 1
  ans[1] <- x0
  for(i in 2:(n + 1)){
    ans[i] <- theta0 + theta1 * ans[i-1] + rnorm(1)
  }
  ans
}

## Parameter settings
theta0 <- 2
theta1 <- 0.5
N <- 10000
x01 <- 14
x02 <- -10
## Markov Chains
mc1 <- MCsim(x0 = x01 , n = N, theta0 = theta0 ,
               theta1 = theta1)
mc2 <- MCsim(x0 = x02 , n = N, theta0 = theta0 ,
               theta1 = theta1)
## Progression of Markov Chains
plot(mc1[1:100], type = "l", ylim = range(cbind(mc1, mc2)),
      xlab = "", ylab = "X", main = "Progression of first-order
      Markov Chain")
lines(mc2[1:100], col = "red")
## Expected value of stationarity distribution and estimates
EfPop <- theta0 / (1 - theta1)
m <- trunc(N / 2) ## burn-in
EfEst1 <- mean(mc1[-c(1:m)])
c(EfPop, EfEst1)
## Standard error of estimate for first MC
ar1Est <- ar(mc1, order.max = 1)$ar
se1 <- sqrt(var(mc1) / N * (1 + ar1Est) / (1 - ar1Est))
c(EfEst1 - 2 * se1, EfEst1, EfEst1 + 2 * se1)
## Variance of stationarity distribution and estimate
VarfPop <- 1 / (1 - theta1^2)
VarfEst1 <- 1 / (1 - ar1Est^2)
c(VarfPop, VarfEst1)

```

14.3.4 Metropolis–Hastings algorithm

One means for deriving a transition kernel that yields as its stationarity distribution the target density f has been proposed by Hastings (1970) based on the seminal paper of Metropolis et al. (1953). The key to this method is a conditional distribution $q(y|x)$ from which random samples can easily be produced. The requirements with respect

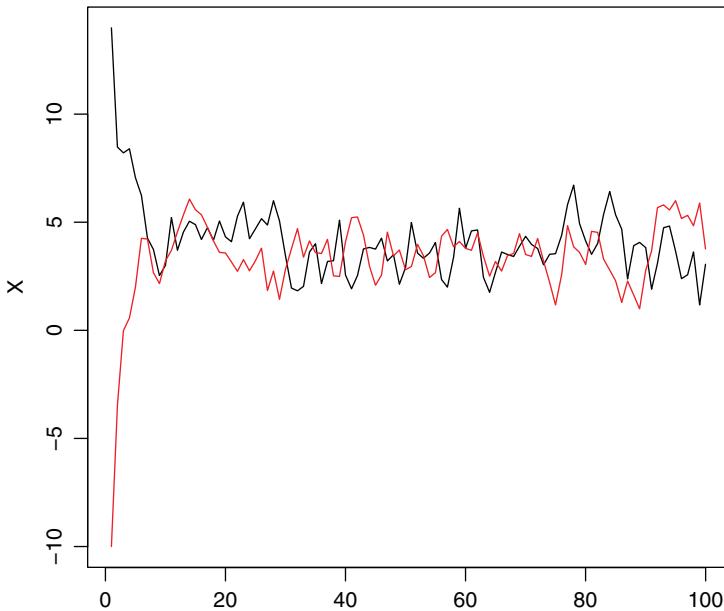


Figure 14.5 Progression of first 100 observations of a Markov chain process.

to $q(y|x)$ are: (i) the ratio $f(y)/q(y|x)$ is known up to a constant value, which must not depend on x ; and (ii) the dispersion of the candidate distribution $q(y|x)$ should be wide enough to cover the entire domain of f , such that the parameter's state space can be explored sufficiently. The former requirement implies that only the shape of the target density has to be known, and hence this algorithm can be applied to densities for which the normalizing constant cannot be provided. For the t iterate of a Markov chain, the algorithm consists of two steps:

1. Generate a random variable $Y_t \sim q(y|x^{(t)})$.
2. Use as the subsequent element of the chain:

$$X^{(t+1)} = \begin{cases} Y_t & \text{with probability } \alpha(x^{(t)}, Y_t) \\ x^{(t)} & \text{with probability } 1 - \alpha(x^{(t)}, Y_t) \end{cases} \quad (14.12)$$

with

$$\alpha(x^{(t)}, Y_t) = \min \left\{ 1, \frac{f(y)}{f(x)} \frac{q(x|y)}{q(y|x)} \right\}. \quad (14.13)$$

The probability term $\alpha(x, y)$ is called the Metropolis–Hastings (MH) acceptance probability. Note that the denominator in the probability expression, $f(x)q(y|x)$, cannot be negative as long as the chain is started from a positive point of $f(\cdot)$ and $q(y|x)$ will always be positive. Recall that $f(\cdot)$ must not qualify as a density such that the area below its curve equals one, but the shape of the density suffices and hence negative

values for $f(\cdot)$ can be possible. As such, this algorithm is similar to the accept–reject approach. Namely, both are a form of rule-based indirect sampling and the corresponding accept–reject steps are both derived by random draws from a uniformly distributed variable $U \sim \mathcal{U}[0, 1]$. In contrast to accept–reject sampling, the sequence $\{X^{(t)}\}$ is not independent and the knowledge of a scaling k or an upper bound is not required in the case of the Metropolis–Hastings algorithm. Furthermore, rejected steps must not be discarded from the Markov chain, which is not the case for the accept–reject sampling as outlined in Section 14.3.2.

Two special cases of the candidate distribution have been proposed in the literature: the independence and the random walk Metropolis–Hastings samplers.

In the former case the candidate distribution does not depend on the chain's state, that is $q(y|x) = q(y)$. Therefore, the MH acceptance probability simplifies to $\alpha(x^{(t)}, Y_t) = \min \left\{ 1, \frac{f(y)}{f(x)} \frac{q(x)}{q(y)} \right\}$. This implies that if the ratio of the target densities for the newly proposed and the realized state increases more than the values evaluated by the candidate densities, the proposed value will be accepted as a new element in the chain. If the increase in the ratio of the target densities is less, then whether the proposed state is accepted depends on the realization of a uniform draw. When an independence sampler is used, the candidate density should be close in shape to the target density and it should have thicker tails (a wider support) than the target density. The latter requirement ensures that in principle the entire parameter state space of the target density is explored, and hence $\frac{f(y)}{f(x)} \frac{q(x)}{q(y)}$ can take values greater than one. The name “*independent* Metropolis–Hastings sampler” might appear at first glance a bit irritating in the sense that it is not the Markov chain that has the attribute of independence, but rather only that the candidate distribution does not depend on the previous chain's state. The characteristic of the chain's dependence is ensured because the MH acceptance probabilities depend on the previous draws.

In the second special case of the candidate distributions, the increments from one state to the next behave like a random walk: $Y^{(t)} = x^{(t)} + \sigma \theta^{(t)}$, where $\theta^{(t)}$ is an independent error term with an expected value of zero. The scalar σ is scaling factor for this error term which influences how aggressively the state space is searched; this characteristic is also referred to as the mixing property of a Markov chain. The proposal distribution is then given by $g(y|x) = g(|y - x|)$, and the MH acceptance probability by $\alpha(x^{(t)}, Y_t) = \min \left\{ 1, \frac{f(y)}{f(x)} \right\}$.

Examples of the two special Metropolis–Hastings cases with respect to the unscaled density of a quadratic utility function as outlined in Section 14.2 are provided in Listing 14.3. In the first lines, objects for the length of the Markov chain (MCMCSIZE), the count of burn-in samples (BurnIn), the parameters (omega3ID, omega3RW1, omega3RW2), and the acceptances (acID, acRW1, acRW2) are defined. The Markov chains are initialized with a value of 0.5. In the following `for` loop a Markov chain for the unknown parameter, that is, the allocation into the risky asset, is populated by means of the independence sampler. As in Listing 14.1, a truncated exponential distribution is used as the candidate/proposal distribution. The MH acceptance probability is computed in line 11 and assigned to the object alpha. Its value is compared against a random draw from the uniform distribution and the i th

element of the chain `omega3ID` is set accordingly. In addition, the counter `acID` is incremented by one, if the proposal was accepted. The empirical rate of acceptance is assigned to the object `acrID` and takes a value of 76.7%. The first-order autocorrelation is computed next, with a value of 0.13. Finally, an estimate for the allocation into the risky asset is determined by discarding the first 500 samples, and the assigned object `omega3IDHat` takes a value of 27.68%. The progression of this chain is plotted in the upper panel of Figure 14.6.

In the second `for` loop two random walk samplers are populated. Both employ standard normal innovations, but differ with respect to the scaling factor σ . In the first $\sigma = 0.5$, whereas in the second the steps are much more limited by a setting of $\sigma = 0.01$. For both chains, the weight constraint $\omega \in [0, 1]$ is enforced by setting $y1 < -\text{Inf}$ or $y2 < -\text{Inf}$, if a proposal violates this condition. In these instances, the unscaled density takes a value of zero and a proposed step would be rejected. Similar to the example of the independence sampler the counts of acceptances are stored in the objects `acRW1` and `acRW2` for the two chains `omega3RW1Hat` and `omega3RW2Hat`, respectively. The acceptance rate for the chain that is allowed to search the state space in greater steps is 36% and the acceptance rate for the less volatile chain is 99.7%. At first sight, the second chain with the higher acceptance rate might seem superior compared to the other, but the opposite is true. Observe the first-order autocorrelations of the two chains. These are given as 0.67 for the first and

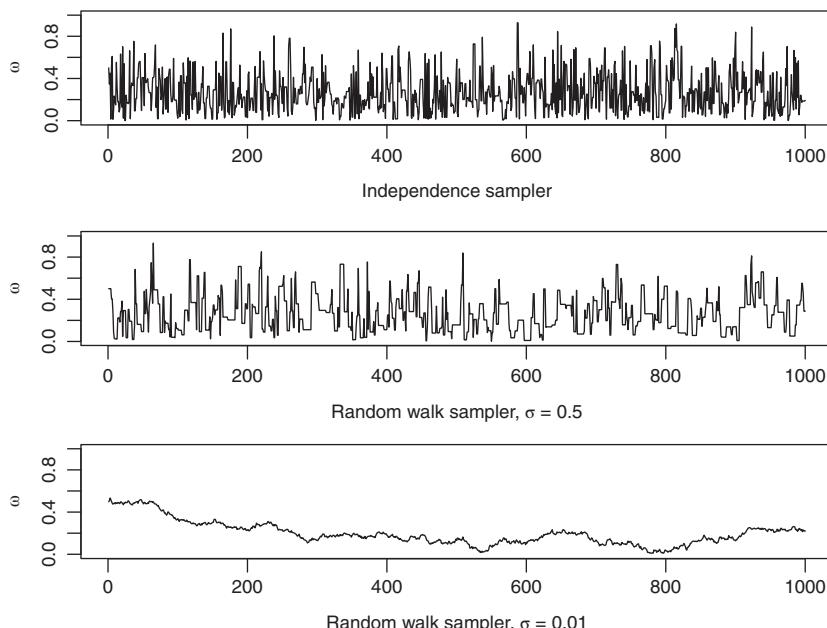


Figure 14.6 Metropolis–Hastings: comparison of independence and random walk sampler.

R code 14.3 Metropolis–Hastings: comparison of independence and random walk sampler.

```

## Initialization
MCMCSIZE <- 1e3
BURNIN <- trunc(MCMCSIZE / 2)
omega3ID <- omega3RW1 <- omega3RW2 <- rep(NA, MCMCSIZE)
omega3ID <- omega3RW1 <- omega3RW2 <- 0.5
acID <- acRW1 <- acRW2 <- 0
## MH: independence sampler
for(i in 2:MCMCSIZE){
  y <- rtrunc(1, spec = "exp",
  a = 0.0, b = 1.0, rate = 2)
  alpha <- min(1, (UIDU(y) / UIDU(omega3ID[i - 1])) *
  (dtrunc(omega3ID[i - 1], spec = "exp",
  a = 0.0, b = 1.0, rate = 2) /
  dtrunc(y, spec = "exp",
  a = 0.0, b = 1.0, rate = 2)))
  u <- runif(1)
  if(u < alpha){
    omega3ID[i] <- y
    acID <- acID + 1
  } else {
    omega3ID[i] <- omega3ID[i - 1]
  }
}
acrID <- acID / MCMCSIZE * 100
ar1ID <- ar(omega3ID, order.max = 1)$ar
omega3IDHat <- mean(omega3ID[-c(1:BURNIN)]) * 100
## MH: random walk sampler
for(i in 2:MCMCSIZE){
  y1 <- omega3RW1[i - 1] + 0.5 * rnorm(1)
  if(y1 > 1 || y1 < 0) y1 <- -Inf
  y2 <- omega3RW2[i - 1] + 0.01 * rnorm(1)
  if(y2 > 1 || y2 < 0) y2 <- -Inf
  alpha1 <- min(1, UIDU(y1) / UIDU(omega3RW1[i - 1]))
  alpha2 <- min(1, UIDU(y2) / UIDU(omega3RW2[i - 1]))
  u <- runif(1)
  if(u < alpha1){
    omega3RW1[i] <- y1
    acRW1 <- acRW1 + 1
  } else {
    omega3RW1[i] <- omega3RW1[i - 1]
  }
  if(u < alpha2){
    omega3RW2[i] <- y2
    acRW2 <- acRW2 + 1
  } else {
    omega3RW2[i] <- omega3RW2[i - 1]
  }
}
acrRW1 <- acRW1 / MCMCSIZE * 100
ar1RW1 <- ar(omega3RW1, order.max = 1)$ar
omega3RW1Hat <- mean(omega3RW1[-c(1:BURNIN)]) * 100
acrRW2 <- acRW2 / MCMCSIZE * 100
ar1RW2 <- ar(omega3RW2, order.max = 1)$ar
omega3RW2Hat <- mean(omega3RW2[-c(1:BURNIN)]) * 100

```

0.99 for the second. Taking both measures for each chain into account, the interpretation is that the less volatile series does not mix as well, that is, searches through the entire support of f , as the first—more volatile—random walk sampler does. This reasoning becomes obvious in the middle and lower panels of Figure 14.6. The poor mixing property of the second chain yields 13.95% as an allocation into the risky asset, which is far from the theoretical value. In contrast, the Markov chain that has been populated from the more volatile random walk sampler yields an allocation as high as 25.21%, which is much more in line with the optimal weight.

Certainly, the last example was an extreme setting, but the reader should be alerted and made aware of the pitfalls in drawing inferences from MCMC with bad mixing and/or high persistence characteristics.

In this respect, enhancements to the classical Metropolis–Hastings algorithm have been put forward in the literature. Namely, Hamiltonian Monte Carlo (HMC: see Duane et al., 1987, Neal, 2011, Brooks et al., 2011) and the no-U-turn sampler (NUTS: see Hoffman and Gelman, 2014). It is beyond this book’s scope to provide a detailed account of these extensions and the reader is referred to the cited literature.

14.4 Synopsis of R packages

Quite a few R packages are hosted on CRAN for conducting and analyzing MCMC. In this synopsis only a selective presentation is provided and the reader is encouraged to visit the listing of packages on CRAN and/or the Task View on “Bayesian Inference.”

14.4.1 Packages for conducting MCMC

The package **adaptMCMC**

In the package **adaptMCMC** an implementation of the robust adaptive Metropolis sampler as proposed by Vihola (2012) is provided (see Scheidegger, 2012). The package is hosted on CRAN and is purely written in R. A class/method scheme has not been utilized; the package is a collection of R functions only.

The main function is `MCMC()`. The function for the log probability density has to be provided as argument `p`. The chain length is set by the argument `n` and initial values for the parameters of the density have to be provided by the argument `init`. Whether adaptive or the classic Metropolis sampling will be conducted is controlled by the logical argument `adapt`: if set to `TRUE`, the user can define a desired acceptance rate by means of the argument `acc.rate`. Achievement of the desired acceptance rate is accomplished by a self-adapting jump distribution for the parameters. By default, its variance-covariance matrix is initialized as an identity matrix, but differing initial elements for the main diagonal can be provided by the argument `scale`. Furthermore, the user can control the adaption speed by the argument `gamma`. The range is $\gamma \in [0.5, 1]$, whereby lower values lead to a faster adaption. A caveat of this MCMC implementation is that it is only available for distributions with at least two parameters.

Aside from the cornerstone function `MCMC()`, within the package there are utility functions for adding more samples to a chain (`MCMC.add.samples()`), populating several independent chains in parallel (`MCMC.parallel()`), and converting the object returned by `MCMC()` to an object of either class `mcmc` or `mcmc.list` with the function `convert.to.coda()` such that the MCMC can be analyzed by the facilities provided in the package `coda` (see Plummer et al., 2006, and Section 14.4.2).

The package `mcmc`

In the package `mcmc` a random walk Metropolis algorithm has been implemented (see Geyer and Johnson, 2015). The package is hosted on CRAN and considered a core package in the “Bayesian” Task View. Within the package, S3 classes and methods are implemented. The computationally burdensome routines are interfaced from functions written in the C language. The package is accompanied by four vignettes and two data sets containing simulated data from a logistics regression model.

With respect to conducting MCMC, the package contains the generic function `metrop()`. The random walk Metropolis algorithm is implemented via the method `metrop.metropolis()`. The generic function is specified with eight arguments. The first, `obj`, expects an R function that evaluates the logarithm of the unnormalized probability density. The function has to be specified such that its first argument is the state vector of the equilibrium distribution. Additional arguments—aside from the state vector—can be passed down to the R function provided for `obj` via the ellipsis argument of `metrop()`. The MCMC is started from an initial state according to the real vector provided as a second argument `initial` to `metrop()`. The shape of the MCMC is defined by the arguments `nbatch` for the number of batches, `blen` for the length of batches, and `nspac` for the spacing of the iterations (thinning). The latter two arguments have default values of 1. The proposal step size for the state vector—that is, how fast the chain will progress through the state space—can be controlled by the argument `scale` with a default of 1. Smaller values lead in general to a higher acceptance rate and vice versa. It is further possible to specify a vector and/or a matrix for `scale`, and the random draws from the standard normal distribution are multiplied by this object for calculating the newly proposed state. User-defined transformations of the MCMC are computed by providing a function to the argument `out.fun`. If this argument is left unspecified, then the mean values of the states are computed. Finally, the function closure of the generic is endowed with a logical switch `debug` that will return additional information if set to `TRUE`; its default value is `FALSE`. The MCMC calculations are conducted by interfacing the C routine `metrop` in a call to `.Call()` within the function body of `metrop.metropolis()`.

The function returns an S3 class `list` object with class attributes `mcmc` and `metropolis`. The two important elements of this list are `accept` and `batch` to store the acceptance rate and the result of the MCMC, respectively. This can be either an $(nbatch \times p)$ matrix, where p equals the dimension of the state vector, or the result of the user-provided function `out.fun`. The remaining `list` elements are

pertinent to the call of the function `metrop()` and the initial and final random seeds of the MCMC run.

The package **MCMCpack**

The package **MCMCpack** is primarily dedicated to Bayesian inference on posterior simulations of predefined statistical model classes (see Martin et al., 2011). The package is contained in the CRAN “Bayesian,” “Distributions,” “Multivariate,” “Psychometrics,” and “Survival” Task Views. It is considered a core package in the first of these. Akin to **mcmc**, the computationally intensive routines are interfaced from routines written in a higher programming language. In the case of **MCMCpack**, the Scythe Statistical Library is employed and the routines are written in C++ (see Pemstein et al., 2011).

With respect to the concept of probabilistic utility, the function `MCMCmetrop1R()` can be used for exploring the density. Similar to the previously described method `metrop.metropolis()` contained in **mcmc**, within `MCMCmetrop1R()` a random walk Metropolis algorithm has been implemented. The R function of the unnormalized density has to be provided as argument `fun`. Whether the returned value thereof should be treated as the logarithm of this density or not can be set by the logical argument `logfun`. Its default argument is `TRUE` and hence, by not specifying otherwise, the returned value of the function for argument `fun` is treated as the logarithm of the density. The initial values for the state space have to be supplied as argument `theta.init`. The length of the chain and how many values should be treated as burn-in samples, that is, the count of state space samples that should be discarded, can be controlled by the argument `mcmc` and `burnin`, respectively. To ameliorate autoregressive dependencies between the sampled states, a thinning parameter `thin` can be set. Its default value is 1, so that no thinning will take place unless otherwise specified. Additional information about the chain’s progression is provided by assigning an integer value greater than zero to the argument `verbose`; that is, for each multiple integer of `verbose` the values of the state space vector, the value of the density, and the Metropolis acceptance rate is printed on-screen. In addition, a random seed can be specified via the argument `seed`, and how progressively proposed steps of the state space are created is controlled by the argument `V`, which is a shorthand for the variance-covariance matrix for the Gaussian proposal distribution. Finally, the function’s closure of `MCMCmetrop1R()` includes arguments `optim.method`, `optim.lower`, `optim.upper`, and `optim.control` that are passed to `optim()` for an initial maximization stage. In case this optimization should fail, an MCMC can still be generated by setting the argument `force.samp` to `TRUE`; its default value is `FALSE`. The progression of the MCMC samples is computed by interfacing with the C++ routine `MCMCmetrop1R_cc` of the Scythe Statistical Library in the function’s body. The function `MCMCmetrop1R()` returns a `list` object of informal S3 class `mcmc`, such that it can be utilized by the methods provided in the package **coda** (see Plummer et al., 2006, and Section 14.4.2).

The package **rstan**

The **rstan** package is a user interface to the **Stan** library (see Stan Development Team, 2015). The latter is a programming language itself (see Stan: A C++ Library for Probability and Sampling, Version 2.7.0 2015). This language is targeted at full Bayesian statistical inference via MCMC techniques. The **R** package **stan** accesses the C++ header files of **Stan** which are contained in the **R** package **StanHeaders** (see Goodrich et al., 2016). Having said this, a caveat for the **R** user is the need to provide a model formulation in the **Stan** language. However, the **R** user is rewarded by employing a feature-rich language that is pretty powerful. The language is well documented in its user guide and reference manual (see Stan Modeling Language Users Guide and Reference Manual, Version 2.7.0 2015). The **R** package **rstan** is hosted on CRAN and is shipped with a vignette by which the user is guided on how to interface **Stan** from **R**. The package employs S4 classes and methods.

The cornerstone function of **rstan** is **stan()**. The purpose of this function is to fit a model in the **Stan** modelling language, and it returns an S4 object of class **stanfit**. The definition of a **Stan** model can be provided by different means. First, the code for the **Stan** model can reside in a separate file, passed to **stan()** by its argument **file**. Second, the model definition can also be passed to the function as a character string via the function's argument **model_code**. After **stan()** is invoked, the **Stan** model is compiled and executed. The returned object can also be reused for running an MCMC with a different MCMC specification by providing the **stanfit** object as argument **fit**. Data items that are used in the **Stan** model have to be passed as a named **list** object to **stan()** by its **data** argument. If one is interested in the results for a subset of the model's parameters only, one can specify the names of these as a character vector and pass it to the function as argument **pars**. For instance, if a location vector is named **mu** and a dispersion matrix **Sigma**, and if one is only interested on the MCMC result for the location vector, one could curtail the results by calling **stan()** with **pars = "mu"**, for instance. The specification on how the MCMC is populated is controlled by the arguments **algorithm**, **iter**, **warmup**, **chains**, **init**, **thin**, and **seed**. The user can choose between two algorithms for searching the state space, namely the "No-U-Turn" sampler ("NUTS") and Hybrid Monte Carlo ("HMC"). The length of the chain is set by the argument **iter**, and how many state simulations of these should be discarded as burn-in period is controlled by the argument **warmup**. The latter takes as default value half of the chain length. Furthermore, the user can specify how many chains should be populated independently from each other by the argument **chains**. The speed of running multiple MCMCs can be increased by utilizing the function's in-built parallelization facility through the argument **cores**. Parameter initialization is defined by the argument **init**. Here, the user can set initial values to be either zero (**init = 0**, or **init = "0"**), or randomly drawn (**init = "random"**), or defined as a function that returns a named **list** object, or, finally, a **list** object of named lists. For obtaining replicate results a random seed can be provided by the argument **seed**. Finally, for ameliorating the degree of autoregressive dependencies between consecutive state samples, a thinning parameter **thin** can be provided. Its default value is

1; that is, every sampled state will be returned and no thinning is accomplished. The remaining arguments of `stan()` are dedicated to fine-tuning the algorithm's behavior, to the provision of detailed information about the MCMC progression, and/or whether to use the alternative C++ libraries Boost and Eigen. By default, the ones shipped within the packages **BH** and **ReppEigen** will be utilized, respectively (see Eddelbuettel et al., 2016, Bates and Eddelbuettel, 2013).

The function `stan()` returns an object of S4 class `stanfit`. For this kind of object, methods `show()`, `summary()`, `pairs()`, `plot()`, `traceplot()`, and numerous extractor functions are available. For instance, the posterior means can be extracted by calling `get_posterior_mean()` and an `mcmc` object for further analysis within the facilities provided by the package **coda** can be coerced from a `stanfit` object by the routine `as.mcmc.list()`.

Finally, worth mentioning is the function `lookup()` which serves as a dictionary for looking up the equivalent of **R** functions in the context of the Stan modelling language. This function comes in handy for novice users of Stan who want to have a concise overview of how an **R** statement can be expressed in Stan. The function takes as argument a character string of an **R** function name and returns a `data.frame` if the second argument `ReturnType` is set to "ANY". Otherwise, the argument expects a valid Stan data type as character, for example, "matrix".

14.4.2 Packages for analyzing MCMC

The package **bmk**

The package **bmk** contains functions for diagnosing the convergence properties of Markov chains (see Krachey and Boone, 2012). The package is hosted on CRAN and is written purely in the **R** language. It does not exploit any class/method scheme and hence can be considered as a collection of **R** functions. It has dependencies on the packages **coda**, **plyr**, and **functional** (see Plummer et al., 2006, Wickham, 2011, Danenberg, 2014, respectively). The former package will be discussed in the next subsection. Furthermore, **bmk** is endowed with five data sets for elucidating the application of the diagnostic functions. The functions `bmkconverge()`, `bmk-sensitive()`, and `bmksummary()` are exported via directives in the package's NAMESPACE and will be described next.

The convergence of a Markov chain can be assessed by the function `bmkconverge()`, which exploits the concept of the Hellinger distance between two distributions (see Boone et al., 2014, Hellinger, 1909). For two discrete probability distributions $P = (p_1, \dots, p_N)$ and $Q = (q_1, \dots, q_N)$, the Hellinger distance $H(P, Q)$ is defined as

$$H(P, Q) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^N (\sqrt{p_i} - \sqrt{q_i})^2}. \quad (14.14)$$

The measure takes values in the interval $[0, 1]$ and the maximum distance is achieved if one distribution assigns zero probability to every set to which the other distribution assigns a positive probability. Hence, for evaluating the convergence in the context of Markov chains, a user should be interested in distance measures close to zero.

The function's arguments are `inputlist1` and `binsize`. The MCMC samples are passed to the former argument and the latter determines by how many samples (length of the bins) the chain should be split. The distances are then computed for consecutive bins and hence the length of the chain should equal an integer multiple of the bin size. In the function's body the Hellinger distance is computed within the non-exported function `HWconverg1()`, which can, however, be accessed by exploiting the triple colon operator, that is, `bmk:::HWconverg1()`. The distance measure itself is defined in the function `HDistNoSize()`, which is also not exported, but can be considered as the workhorse function in **bmk**.

The sensitivity of two Markov chains with respect to the specification of the assumed parameters' prior distributions and/or the hyper-parameters of the statistical model in question can be assessed by computing the Hellinger distance between the two by the function `bmksensitive()`. This function has two arguments, namely `inputlist1` and `inputlist2` for the two Markov chains with differing input settings.

Finally, the function `bmksummary()` returns descriptive statistics of the parameters' distributions, the Hellinger distance measures between two consecutive state samples, and Gelman and Rubin's convergence diagnostic (see Gelman and Rubin, 1992). The latter will be described in the next subsection, as **bmk** exploits the implementation in **coda**. The function `bmksummary()` takes one argument, namely `inputlist`, for providing the MCMC.

The package **coda**

The package **coda** is the most encompassing package on CRAN with respect to the assessment of MCMC (see Plummer et al., 2006). It is listed in the CRAN Task Views “Bayesian” and “gR,” and considered a core package in the former. The package is written solely in the **R** language and utilizes the **S3** class/methods framework. The package has no dependencies to other contributed packages, but imports the **lattice** package (see Sarkar, 2008) for certain graphical displays of MCMC. The functions and methods provided by **coda** can be grouped loosely into the categories plotting and diagnostic testing. In the following, key functions of each category will be presented.

Before we proceed, it is worth mentioning that objects of informal class `mcmc` can be created by calling `mcmc()`. This function is endowed with the arguments `data` for the `vector/matrix` object of an MCMC output, `start` and `end` for defining the iteration number of the first and last state samples, and `thin` for the thinning interval between the state space samples of the chain. Some of the available **R** packages and the ones presented in this subsection provide the option for coercing the MCMC output to an object of this class, and hence there is no need to call `mcmc()` explicitly. As an aside, **coda** offers a coercion method `as.mcmc()`, such that `vector/matrix` objects can be converted to `mcmc`-class objects. Whether an object belongs to this class can be checked by calling `is.mcmc()`. Similarly, for parallel runs of MCMC the **S3** class `mcmc.list` is made available and objects thereof can be created by calling `mcmc.list()`. This creator function has the ellipsis as sole argument and expects objects of informal class `mcmc`. Objects can be coerced to and/or

checked for this class by the methods `as.mcmc.list()` and `is.mcmc.list()`, respectively.

For the graphical display of Markov chain(s) the methods/functions `autocorr.plot()`, `crosscorr.plot()`, `cumplot()`, `densplot()`, `gelman.plot()`, `geweke.plot()`, `plot.mcmc()`, `traceplot()`, and `trellisplot()` are available (in alphabetical order). The first resort for summarizing MCMC output graphically is to call the `plot()` method for `mcmc` objects. This method will produce for each parameter a trace plot of the chain's progression and a density plot, by default. Either set of plots can be suppressed by setting the arguments `trace = FALSE` or `density = FALSE`, respectively. The trace and density plots are created within the `plot()` method by calling the `traceplot()` `densplot()` functions. The mixing properties of a Markov chain can be inspected by the auto- and cross-correlation of/between the parameters. These can be computed with the functions `autocoor()` and `crosscorr()`, respectively. The plots of these ACF and CCF can be produced with the corresponding `plot()` methods, that is, `autocorr.plot()` and `crosscorr.plot()`. All of these functions take as first argument `x` an object of class `mcmc`. As the complexity of the graphical output increases with the count of parameters, space-conserving trellis-plots are made available via the S3 methods `acfplot()` for `mcmc` and `mcmc.list` objects. In addition to the functions and methods that graphically depict the mixing properties of a Markov chain, the effective sample size—that is, the one implied by the autocorrelation adjustment—is returned by the function `effectiveSize()`. Its sole argument `x` expects an `mcmc` object. The progression for a recursive computation of the test statistics for Gelman and Rubin's convergence test and/or the approach chosen by Geweke is displayed by the functions `gelman.plot()` and `geweke.plot`, respectively. These two forms of assessing the convergence of an MCMC chain will be discussed in the next paragraph.

Diagnostic checking of the chain's stationarity condition and/or mixing behavior can be conducted with the functions `gelman.diag()`, `geweke.diag()`, `heidel.diag()`, `HPDinterval()`, and `raftery.diag()` (in alphabetical order).

Gelman and Rubin's convergence diagnostic (see Gelman and Rubin, 1992) is implemented as function `gelman.diag()`. This convergence test can only be applied to parallel runs of MCMC and hence the first function argument `x` must be of S3 class `mcmc.list`. Furthermore, the test is applicable to normally distributed targets and is similar to a classical analysis of variance (ANOVA) for single parameters. That is, the variances of a parameter per MCMC output is compared to the overall dispersion for all Markov chains. Under the assumption of a normal target distribution, the resultant test statistic is Student's *t* distributed, where the degrees of freedom are estimated by the method of moments. Values of the test statistic greater than one indicate a lack of convergence. A multivariate extension of this convergence measure has been introduced into the literature by Brooks and Gelman (1998), and its computation is conducted by default, but can be suppressed if the argument `multivariate` is set to `FALSE`. The function returns an object of S3 class `gelman.diag` which is endowed with a `print()` method.

The next diagnostic test to be presented is Geweke's convergence diagnostic (see Geweke, 1992). This test is made available as function `geweke.diag()`. The test compares the means of a parameter computed from data points at the beginning and at the end of a chain. Convergence to the stationary distribution is given, if these two estimates statistically coincide. By default, the first batch of values for computing the location is set to 10% of the chain length, but can be changed by the argument `frac1`. The count of state samples at the end of the chain is set to 50% of the Markov chain length by default and can be changed by providing a different fraction for the argument `frac2`. The test statistic itself is a Z-score for a test of equality of means between the first and last part of the Markov chain. This test is conducted for each parameter in the chain.

The convergence diagnostic proposed by Heidelberger and Welch is based on the Cramér–von Mises statistic to test the null hypothesis that the sampled state values are stemming from a stationary distribution (see Heidelberger and Welch, 1983). The test procedure is to compute the test statistic on the state samples for each parameter for the complete chain length and then subsequently reducing the chain length by 10%. If the null hypothesis has to be rejected after the half-life of the chain, non-convergence to the stationary distribution is implied. In this case, the authors suggest increasing the chain length by a factor of 1.5. This testing procedure is implemented as function `heidel.diag()`, which takes as first argument `x` an object of S3 class `mcmc`. The significance level is set by the argument `pvalue` to a default value of 0.05. The target value for the ratio between the half width and the sample mean is determined by the argument `eps` and takes a default value of 0.1.

The highest posterior density (HPD) interval for each parameter is returned by the function `HPDinterval()`. The function takes as input argument `obj` an object of either S3 class `mcmc` or `mcmc.list`. The nominal width of the interval is set by the argument `prob`, with a default value of 0.95. The HPD is based on the empirical cumulative distribution function and returns the lower and upper ends of the quantiles, such that the probability mass between these two points coincides with the nominal probability and the width of the interval is the narrowest, that is, the most concentrated.

The last diagnostic test to be presented is the one introduced into the literature by Raftery and Lewis (1992), Raftery and Lewis (1995). The purpose of this test is twofold. First, the test gives hindsight as to whether a Markov chain has converged to the stationary distribution. Second, the test provides bounds for the accuracy of the estimated quantiles for the functions of the parameters. This diagnostic test is implemented as function `raftery.diag()`. The function's closure takes five arguments, namely `data` for the `mcmc` object, `q` for the quantile of the distribution for a parameter with a default value of 0.025, `r` for the desired margin of error around the quantile, `s` for the confidence level of obtaining an estimate in the interval $(q \pm r)$, with a default probability of 0.95, and `converge.eps` for setting the required precision, with a default of 0.001. Assuming independence between the state space samples, a minimum number of runs can be established for a given quantile with lower and upper bounds for a given confidence level (that is, from the input values of the arguments `q`, `r`, and `s`). If the supplied MCMC output falls short of this

minimum length, the testing cannot be accomplished and an indicative error message is returned indicating how long the chain must be. In general, the length of the Markov chain must be greater than this minimum length due to the positive autocorrelations between the sampled states. A dependence factor, I , that takes the size of autocorrelation into account indicates by how much the length of the chain must be changed to meet the desired accuracy. Incidentally, for negatively autocorrelated states the dependence factor will be less than one and hence the required length of a chain is less than the minimum required length as deduced from independence. The authors suggest that values of $I >= 5$ should be considered as an indication of strong autocorrelation that might have been caused by either the starting values of the chain, or high correlations between the posterior distributions, or a too-narrow search in the state space. In any case, a rerun of the MCMC is required. The function returns an object of `S3` class `raftery.diag` which is endowed with a `print()` method.

Finally, it is worth mentioning the function `codamenu()`, which provides a simple user interface for choosing between the implemented plotting and diagnostic facilities contained in the package.

14.5 Empirical application

14.5.1 Exemplary utility function

In this subsection, a utility function is introduced that directly captures the downside risk of a portfolio allocation. This utility function was proposed by Consiglia et al. (2001) and used in Rossi et al. (2002) and Marschinski et al. (2007). This type of utility function can be used in the context of asset liability management, whereby the time horizon of the investment has an impact on the allocation. The function is defined as

$$\mathbb{E}[u(\alpha, L, \lambda, T)] = \sum_{n=1}^{N_T} \Delta t [\mathbb{E}(U(n\Delta t)) - \lambda \mathbb{E}(D(n\Delta t))], \quad (14.15)$$

where $U(n\Delta t)$ is a function for modelling the upside potential of an allocation, and $D(n\Delta t)$ is a function for mirroring the downside risk of this allocation for a portfolio return at time $n\Delta t$. This utility function has additional arguments L , λ , and T that represent the targeted return, the degree of risk aversion, and the investment horizon, respectively. As can be seen by the index of the summation, the total time period of the investment consists of N_T periods with time increments (frequency) of size Δt such that $T = N_T \Delta t$. This implies that the utility function depicts a “fix-me” approach, that is, the wealth is allocated at the beginning of T and the allocation is not changed in the interim. Thus, for a risk averse investor the desired return progression is to stay close to the targeted return. The time horizon of the investment horizon will matter in the sense that the longer it is, the more one is moved to allot funds to the riskier assets and vice versa. This statement becomes evident if one assumes a normally distributed portfolio return. Under this assumption, the utility function can be stated as (see Rossi

et al. 2002, Marschinski et al. 2007)

$$\mathbb{E}[u(\boldsymbol{\alpha}, L, \lambda, T)] = \sum_{n=1}^{N_T} \Delta t[n\Delta t M f_2(n\Delta t) - \sqrt{n\Delta t} S f_1(n\Delta t)], \quad (14.16)$$

where the variables/functions that appear on the right-hand side of the equation are defined as:

$$M = \boldsymbol{\omega}' \boldsymbol{\mu} - L, \quad (14.17a)$$

$$S^2 = \boldsymbol{\omega}' \boldsymbol{\Sigma} \boldsymbol{\omega}, \quad (14.17b)$$

$$\eta = \frac{M}{2S}, \quad (14.17c)$$

$$f_1(t) = (\lambda - 1) \frac{e^{-t\eta^2}}{\sqrt{2\pi}}, \quad (14.17d)$$

$$f_2(t) = \frac{1 + \lambda}{2} - \frac{\lambda - 1}{2} \operatorname{erfc}(\sqrt{t\eta}), \quad (14.17e)$$

with $\operatorname{erfc}()$ signifying the complementary error function, $\boldsymbol{\omega}$ the weight vector, $\boldsymbol{\mu}$ the expected returns of the assets, and $\boldsymbol{\Sigma}$ the variance-covariance matrix of the returns.

In Listing 14.4 the R code for implementing this utility function is displayed.

First, three auxiliary functions are defined, namely `erfc()` for the complementary error function, and `f1()` and `f2()` for computing the expected down- and upside utilities, respectively. The employed utility functions, `U()` and `PUL()`, are defined in the last part of the listing. The closure of `U()` is specified with arguments `w` for the weight vector, `mu` for the expected returns, `Sigma` for the variance-covariance matrix of the returns, `Lambda` for the risk aversion parameter, `L` for the targeted portfolio return, and `period` for the length of the investment horizon. This code design has been chosen for convenience purposes whereby the vectorization capabilities of the R language are exploited for computing the summation term on the right-hand side of the utility function. In the function's body of `PUL()` the unscaled log-density of the reinterpreted utility function is defined. Compared to `U()`, this function is endowed with an additional argument `nu` for specifying the convergence rate. In the function's body the utility level is computed first by calling `U()` and assigned to the object `uval`. The value of the unscaled log-density is then returned as the product of `uval` and `nu`. Because this and the following example take the perspective of a long-only investor, negative weights are excluded from the state space of feasible allocations. This is accomplished by returning a value of minus infinity for the log-density.

Because in the subsequent examples the facilities of the `rstan` package will be employed, the corresponding setup in the Stan language is exhibited in the stan code Listing 14.1.

Within the listing, four code blocks have been defined. First, the block `functions` contains the auxiliary functions `f1()` and `f2()` and the log-density defined as `pu_log()`. The function is specified with five arguments. The first is the weight vector `w` and the remaining four elements are objects for the expected returns, the variance-covariance matrix, the hyper-parameters `v` and `lambda`, and the argument for the

R code 14.4 Definition of the utility function.

```

## 1
## Auxilliary functions for utility 2
## 3
## complementary error function 4
erfc <- function(x) 2 * pnorm(x * sqrt(2), lower = FALSE) 5
## Downside 6
f1 <- function(x, Lambda, eta){ 7
  (Lambda - 1) * exp(-x * eta^2) / sqrt(2 * pi) 8
}
## Upside 9
f2 <- function(x, Lambda, eta){ 10
  (1 + Lambda) / 2 - (Lambda - 1) / 2 * erfc(sqrt(x) * eta) 11
}
## 12
## Utility 13
## 14
U <- function(w, mu, Sigma, Lambda, L, iperiod){ 15
  M <- drop(crossprod(w, mu) - L) 16
  S <- drop(sqrt(crossprod(w, Sigma) %*% w)) 17
  eta <- M / (2 * S) 18
  ti <- 1:iperiod 19
  dt <- 1 / iperiod 20
  f1val <- f1(ti * dt, Lambda, eta) 21
  f2val <- f2(ti * dt, Lambda, eta) 22
  uval <- sum(dt * (ti * dt * M * f2val - sqrt(ti * dt) * 23
                    S * f1val)) 24
  uval 25
}
## 26
## Probabilistic re-interpretation of utility 27
## (unscaled log-density) 28
## 29
PUL <- function(w, mu, Sigma, Lambda, L, iperiod, nu){ 30
  if(any(w < 0)){ 31
    return(-Inf) 32
  } else if(any(w >= 1)){ 33
    return(-Inf) 34
  } else { 35
    w <- w / sum(w) 36
    uval <- U(w, mu, Sigma, Lambda, L, iperiod) 37
    return(nu * uval) 38
  } 39
}

```

Stan code 14.1 Definition of the utility function in the **Stan** language.

```

functions{  

  // downside function  

  real f1(real x, real Lambda, real eta){  

    real valf1;  

    valf1 <- (Lambda - 1) * exp(-x * eta^2) / sqrt(2 * pi());  

    return(valf1);  

  }  

  // upside function  

  real f2(real x, real Lambda, real eta){  

    real valf2;  

    valf2 <- (1 + Lambda) / 2 - (Lambda - 1) / 2 *  

      erfc(sqrt(x) * eta);  

    return(valf2);  

  }  

  // utility function (log probabilistic specification)  

  real pu_log(vector w, vector mu, matrix Sigma, real Lambda,  

    real L, int iperiod, real nu){  

    real l1;  

    real M;  

    real S;  

    real eta;  

    real dt;  

    vector[iperiod] val;  

    M <- w' * mu - L;  

    S <- sqrt(w' * Sigma * w);  

    eta <- M / (2 * S);  

    dt <- 1 / (1.0 * iperiod);  

    for(i in 1:iperiod){  

      val[i] <- i * dt * M * f2(i * dt, Lambda, eta) - sqrt(i * dt)  

        * S * f1(i * dt, Lambda, eta);  

    }  

    return(nu * mean(val));  

  }  

}  

data{  

  int<lower=0> N; // number of assets  

  int<lower=0> iperiod; // investment period  

  real<lower=0> nu; // convergence rate  

  vector[N] mu; // vector of expected returns  

  matrix[N, N] Sigma; // variance-covariance matrix  

  real<lower=0> Lambda; // risk aversion  

  real<lower=0> L; // target portfolio return  

}  

parameters {  

  simplex[N] w; // weight vector  

}  

model{  

  increment_log_prob(pu_log(w, mu, Sigma, Lambda, L,  

    iperiod, nu));  

}

```

investment horizon N_t . The function returns the value of the log-density. Within the second block the required data objects are defined, that is, the data items needed to compute the value of the log-density. The `parameters` block just takes one object, namely `w`. The type of `w` is `simplex`, which is a special built-in type in `Stan`. Objects of this type have the property that all their elements are greater than zero and sum to one. Hence, this type is ideally suited for a long-only portfolio optimization with a budget constraint. The last block `model` defines the probability model for which the MCMC should be populated. Given that a user-defined log-density is defined, this task is accomplished with the `increment_log_prob Stan()` function.

14.5.2 Probabilistic versus maximized expected utility

By using the utility functions defined in the previous section, the concept of probabilistic utility is now applied to an equity portfolio consisting of the S&P 500, the Russell, the DAX, and the FTSE index series. It will be shown how the above-mentioned MCMC packages can be used.

In Listing 14.5, the required packages are brought into memory first. The equity indices are part of the `MultiAsset` data set, which is contained in the **FRAPO** package. The MCMC is conducted by alternative means, that is, the expected values of the allocations are determined by utilizing the MCMC-related functions in the packages **adaptMCMC**, **MCMCpack**, **mcmc**, and **rstan**. For computing the asset allocations by maximizing the utility function, the package **Rsolnp** will be used (see Ghalanos and Theussl, 2015).

Commencing in line 8 of the listing, the data set is loaded and the annualized, discrete, percentage returns of the equity indices are calculated and assigned to the object `Requity`. Next, the sample means and the variance-covariance matrix are assigned to the objects `muSample` and `SigmaSample`, respectively. The comparison between the maximum expected utility and the probabilistic utility allocations is cast in terms of a targeted return equal to 10% (object `targetR`), a risk aversion parameter set to 3 (object `riskA`), an investment horizon encompassing 60 periods (object `Nt`), and an asymptotic conversion rate equal to the square root of N_t , that is, object `convR`.

The maximization of the expected utility is conducted in lines 24–43 of Listing 14.5. First, the objective function to be minimized is defined as object `f0()`. Within the function's body the utility level is computed for a given set of parameters and the negative value is returned. In line 34, a function representing the budget constraint is defined as object `budget()`. This function is then used together with the right-hand side set equal to one (argument `eqB`) in the call to `solnp()`. The optimization algorithm is initialized with an equal-weighted solution (object `w0`) and the argument for the objective function, `fun`, is set to `f0` in the call to `solnp()`. The non-negativity constraints for the elements of the weight vector are swiftly accomplished by setting `rep(0, N)` for the lower bound argument `LB`. The remaining arguments in the call to `solnp()` are passed down to the utility function `U()` as defined in Listing 14.4. The optimization result is assigned to the object `optMEU`, and in the last lines the solution is stored in `wMEU` as percentages and the associated utility level as object `UMEU`.

R code 14.5 Probabilistic versus maximized expected utility, part one.

```

## Loading of packages
library(FRAPO)
library(adaptMCMC)
library(MCMCpack)
library(memc)
library(rstan)
library(Rsolnp)
## Computation of equity returns
data(MultiAsset)
Assets <- timeSeries(MultiAsset,
                      charvec = rownames(MultiAsset))
R <- 100 * ((1 + returns(Assets, method = "discrete",
                           percentage = FALSE))^12 - 1)
#R <- returns(Assets, method = "discrete", percentage = FALSE)
Requity <- R[, c("GSPC", "RUA", "GDAXI", "FTSE")]
## Parameter settings / initialization
muSample <- colMeans(Requity)
sigmaSample <- cov(Requity)
targetR <- 10
riskA <- 3
Nt <- 60
convR <- sqrt(nrow(Requity))
N <- ncol(Requity)
##
## Maximizing expected utility
##
## objective function to be minimized
f0 <- function(pars, mu, Sigma, Lambda, L, iperiod){
  uval <- U(pars, mu, Sigma, Lambda, L, iperiod)
  -1.0 * uval
}
## budget constraint for optimization
budget <- function(pars, mu, Sigma, Lambda, L, iperiod){
  sum(pars)
}
## initial point
w0 <- rep(1/N, N)
## Computing MEU allocation
optMEU <- solnp(pars = w0, fun = f0, eqfun = budget,
                 eqB = 1.0, LB = rep(0, N), mu = muSample,
                 Sigma = sigmaSample, Lambda = riskA,
                 L = targetR, iperiod = Nt)
wMEU <- optMEU$pars * 100
UMEU <- U(wMEU / 100, mu = muSample, Sigma = sigmaSample,
           Lambda = riskA, L = targetR, iperiod = Nt)

```

R code 14.6 Probabilistic versus maximized expected utility, part two.

```

McmcLength <- 5e4
## adaptMCMC
ans1 <- MCMC(p = PUL, n = McmcLength, init = w0,
               mu = muSample, Sigma = sigmaSample,
               Lambda = riskA, L = targetR, iperiod = Nt,
               nu = convR, acc.rate = 0.5)
ans1$acceptance.rate
w1 <- colMeans(ans1$samples)
w1 <- w1 / sum(w1) * 100
w1
## MCMCpack
ans2 <- MCMCmetrop1R(fun = PUL, theta.init = w0,
                       mcmc = McmcLength, mu = muSample,
                       V = 1e-3 * diag(N), Sigma = sigmaSample,
                       Lambda = riskA, L = targetR,
                       iperiod = Nt, nu = convR)
w2 <- colMeans(ans2)
w2 <- w2 / sum(w2) * 100
w2
## mcmc
ans3 <- metrop(obj = PUL, initial = w0, nbatch = McmcLength,
                blen = 1, scale = 0.025, mu = muSample,
                Sigma = sigmaSample, Lambda = riskA,
                L = targetR, iperiod = Nt, nu = convR)
ans3$accept
w3 <- colMeans(ans3$batch)
w3 <- w3 / sum(w3) * 100
w3
## rstan
pudat <- list(N = N, Lambda = riskA, mu = muSample,
                Sigma = sigmaSample, L = targetR,
                iperiod = Nt, nu = convR)
nchain <- 4
StanFile <- file.path(find.package("FRAPO"), "BookEx",
                       "C14S1.stan")
ans4 <- stan(file = StanFile, data = pudat, iter = McmcLength,
             chains = nchain)
w4 <- drop(get_posterior_mean(ans4, pars = "w")
           [, nchain + 1]) * 100
w4
## Summarizing results
Wall <- round(rbind(wMEU, w1, w2, w3, w4), 2)
rownames(Wall) <- c("MEU", "adaptMCMC", "MCMCpack", "mcmc",
                     "rstan")
CR <- apply(Wall, 1, function(x) cr(x, sigmaSample))
Wans <- cbind(Wall, CR)
colnames(Wans) <- c("GSPC", "RUA", "GDAXI", "FTSE",
                    "Concentration")
Wans

```

The allocations according to the concept of probabilistic utility are computed in Listing 14.6. In the first line, the length of the Markov chains is set to `5e4`. In the subsequent lines the weight vectors expressed as percentages are determined by calling the respective functions of the packages `adaptMCMC`, `MCMCpack`, `mcmc`, and `rstan` and assigned to the objects `w1` to `w4`, respectively. To avoid rounding issues, the allocations are rescaled to fulfill the budget constraint. Commencing in line 42, the portfolio allocations are combined in the object `Wall` and the implied concentration ratios for each allotment is computed in line 45 (object `CR`). Both objects are joined by column and provided in Table 14.1.

The maximization of the expected utility yields a concentrated portfolio solution with allotments only to the S&P 500 and FTSE 100 indices. The latter accounts for roughly 87% of the invested wealth. The allocations according to the concept of probabilistic utility are, first, more balanced across the assets, and secondly are pretty similar to each other with respect to the chosen `R` package. The allocations to the Russell and DAX indices are the smallest and the allotment to the S&P 500 remained almost unchanged. Hence, the high concentration in the FTSE 100 market implied by the MEU optimization has been ameliorated by utilizing the probabilistic utility approach. This observation is also mirrored by the concentration ratios.

14.5.3 Simulation of asset allocations

In this section a comparison between the portfolio optimization concepts of maximizing the expected utility and probabilistic utility is conducted. This comparison is cast in terms of a Michaud-type simulation (see Michaud, 1989, Michaud, 1998), where the observed data are treated not as a random sample, but are assumed to be the population adhering to a distribution with known parameters. Random samples are generated from this “known” data-generating process which can then be used in the two kinds of optimization, that is, maximum expected and probabilistic utility. The resultant outcome can then be compared to the “true” optimal allocation and its implied utility.

In this simulation the same data set and settings are reused as in Listings 14.4–14.6. The first part of the simulation is shown in Listing 14.7.

The listing consists of two sections. In the first, the function `PuMuSim()` is defined. For a given random draw `x` from the assumed “true” distribution, the

Table 14.1 Probabilistic utility: comparison of allocations.

Package	GSPC	RUA	GDAXI	FTSE	Concentration
MEU	14.6	0.0	0.0	85.4	0.7
adaptMCMC	18.8	11.6	4.2	65.4	0.4
MCMCpack	15.1	8.2	4.1	72.6	0.5
mcmc	14.8	9.1	4.2	71.9	0.5
rstan	11.3	6.7	3.7	78.2	0.6

R code 14.7 Utility simulation.

```

## Function for utility simulation
PuMuSim <- function(x){
  SmpL <- nrow(x)
  muS <- colMeans(x)
  sigmaS <- cov(x)
  PUans <- metrop(obj = PUL, initial = w0, nbatch = McmcLength,
                    blen = 1, scale = 1e-2, mu = muS, Sigma = sigmaS,
                    Lambda = riskA, L = targetR, iperiod = Nt,
                    nu = sqrt(SmpL))
  PUW <- colMeans(PUans$batch)
  PUW <- PUW / sum(PUW)
  PU <- U(w = PUW, mu = muSample, Sigma = sigmaSample,
           Lambda = riskA, L = targetR,
           iperiod = Nt)
  MUans <- solnp(pars = w0, fun = f0, eqfun = budget,
                  eqB = 1.0, LB = rep(0, N), mu = muS,
                  Sigma = sigmaS, Lambda = riskA,
                  L = targetR, iperiod = Nt)
  MUW <- MUans$pars
  MU <- U(w = MUW, mu = muSample, Sigma = sigmaSample,
           Lambda = riskA, L = targetR, iperiod = Nt)
  list(U = c(MU, PU), PUW = PUW, MUW = MUW)
}

## Initializing objects
Draws <- 100
Idx <- 1:Draws
Samples <- c(12, 18, 24, 30, 36, 42, 48, 54, 60)
SS <- length(Samples)
PU <- matrix(NA, ncol = length(Samples), nrow = Draws)
MU <- matrix(NA, ncol = length(Samples), nrow = Draws)
colnames(PU) <- colnames(MU) <- paste("S", Samples, sep = "")
PUW <- array(NA, dim = c(Draws, N, SS))
MUW <- array(NA, dim = c(Draws, N, SS))
set.seed(123456)
RDrawList <- lapply(Idx, function(x)
  mvtnorm(n = max(Samples),
           mu = muSample, Sigma = sigmaSample))

## Carrying out simulation
for(i in 1:length(Samples)){
  cat(paste("Computing for Sample Size", Samples[i], "\n"))
  SampleL <- Samples[i]
  ListData <- lapply(Idx, function(x) RDrawList[[x]]
    [1:Samples[i], ])
  SimResults <- lapply(ListData, PuMuSim)
  MU[, i] <- unlist(lapply(SimResults, function(x) x$U[1]))
  PU[, i] <- unlist(lapply(SimResults, function(x) x$U[2]))
  PUW[, , i] <- matrix(unlist(lapply(SimResults,
    function(x) x$PUW)),
    ncol = N, nrow = Draws,
    byrow = TRUE)
  MUW[, , i] <- matrix(unlist(lapply(SimResults,
    function(x) x$MUW)),
    ncol = N, nrow = Draws,
    byrow = TRUE)
}

}

```

allocations according to the concepts of maximum expected and probabilistic utility are computed and the implied utility levels of these two weight vectors within the function's body. These results are collected in a named `list` object with elements "`U`", "`PUW`", and "`MUW`". The first is a two-element vector with the utility levels given by the maximum expected and probabilistic utility optimizations; the corresponding allocations are stored in the elements "`PUW`" and "`MUW`" for concepts of probabilistic and maximum expected utility portfolio optimization, respectively.

In the second section of Listing 14.7, starting in line 24, objects used later in the simulation are initialized and the simulation is cast in a `for` loop commencing on line 38. With respect to the initialization of objects, the count of random draws is set by the object `Draws` and the sequence of draws is assigned to `Idx`. The latter object will be used later in the call to `lapply()` whence the randomly generated returns are passed to the function `PuMuSim()`. Next, a vector for the assumed sample lengths is created (object `Samples`) and its length is set to `SS`, short for "sample size." For the storage of the simulation results four array objects are created. First, in the `matrix` objects `PU` and `MU` the utility levels for each draw and each sample size of the probabilistic and maximum expected utility will be stored, respectively. Likewise, the `array` objects `PUW` and `MUW` are created for holding the allocations per draws and per assumed sample sizes. Before commencing the simulation a random seed is set.

With these settings, the simulation can be conducted in a pretty straightforward manner. First, a `list` object `RDrawList` of random draws is created, where the row dimension of each element is set to the maximum of `Samples`. These random numbers are drawn from a multivariate normal distribution with location parameter equal to `muSample` and variance-covariance matrix equal to `sigmaSample` by calling the function `mvrnorm()` contained in the `MASS` package. Within the `for` loop, a `list` object `ListData` is then created with a count of elements equal to `Draws` and each element is a `matrix` object containing the randomly generated returns of row size according to the *i*th element of `Samples`. The allocations and utility levels per random draw and per given sample size can then be swiftly computed by passing the elements of the `ListData` object in a `lapply()` call to the function `PuMuSim()`. The result thereof, `SimResults`, is a list of `list` objects. In the remainder of the `for` loop, the utility levels and allocations are extracted from `SimResults` and assigned to the relevant row/column entries of the objects `MU`, `PU`, `MUW`, and `PUW`.

In Listing 14.8, the results are analyzed graphically. First, the deviations from the "true" level of the achievable maximum expected utility are computed for the two optimization approaches (objects `PUD` and `MUD`). The descriptive statistics mean, minimum, and maximum for these deviations are computed next and assigned to the objects `PuDMean`, `MudMean`, `PuDmin`, `MuDMin`, `PuDMax`, and `MudMax`, with obvious meaning. Here, the `apply()` function is used for the column dimension of the objects `PUD` and `MUD`. The graphical presentation of these descriptive statistics is created as a bar chart in the subsequent lines. It consists of the mean deviations from the "true" utility with minimum and maximum values. The outcome is shown in Figure 14.7.

For smaller sample sizes the implied utility levels of the probabilistic approach are more centered around the "true" MEU allocation, as can be deduced from Figure 14.7.

R code 14.8 Graphical analysis of utility simulation.

```

## Analyzing results
## Computing distances
MUD <- PUD <- matrix(NA, nrow = Draws, ncol = SS)
for(i in 1:SS){
  MUD[, i] <- (UMEU - MU[, i]) / abs(UMEU)
  PUD[, i] <- (UMEU - PU[, i]) / abs(UMEU)
}
## Graph
PuDMean <- apply(PUD, 2, function(x) mean(x))
MuDMean <- apply(MUD, 2, function(x) mean(x))
PuDMin <- apply(PUD, 2, function(x) min(x))
MuDMin <- apply(MUD, 2, function(x) min(x))
PuDMax <- apply(PUD, 2, function(x) max(x))
MuDMax <- apply(MUD, 2, function(x) max(x))
ylims <- range(na.omit(c(PuDMax, MuDMax, PuDMin, MuDMin)))
plot(cbind(1:SS, PuDMean), type = "p", col = "blue", pch = 17,
     cex = 1.2, ylim = ylims, ylab = "Relative deviations from
     'true' utility", xlab = "Sample Sizes", axes = FALSE)
points(1:SS, PuDMax, type = "p", col = "blue", cex = 1.2,
       pch = 22)
points(1:SS, PuDMin, type = "p", col = "blue", cex = 1.2,
       pch = 22)
points(1:SS, MuDMean, type = "p", col = "darkred", cex = 1.2,
       pch = 19)
points(1:SS, MuDMax, type = "p", col = "darkred", cex = 1.2,
       pch = 22)
points(1:SS, MuDMin, type = "p", col = "darkred", cex = 1.2,
       pch = 22)
arrows(x0 = 1:SS, y0 = PuDMin, y1 = PuDMax, code = 3,
       col = "blue", length = 0.0, angle = 90, lwd = 2)
arrows(x0 = 1:SS, y0 = MuDMin, y1 = MuDMax, code = 3,
       col = "darkred", length = 0.0, angle = 90, lwd = 1.0)
axis(1, at = 1:SS, Samples)
axis(2, pretty(ylims), pretty(ylims), las = 2)
box()
legend("topright", legend = c("PU mean deviation",
                               "PU min/max deviation",
                               "MU mean deviation",
                               "MU min/max deviation"),
       pch = c(17, 22, 19, 22),
       col = c(rep("blue", 2), rep("darkred", 2)))
abline(h = 0, col = "gray")

```

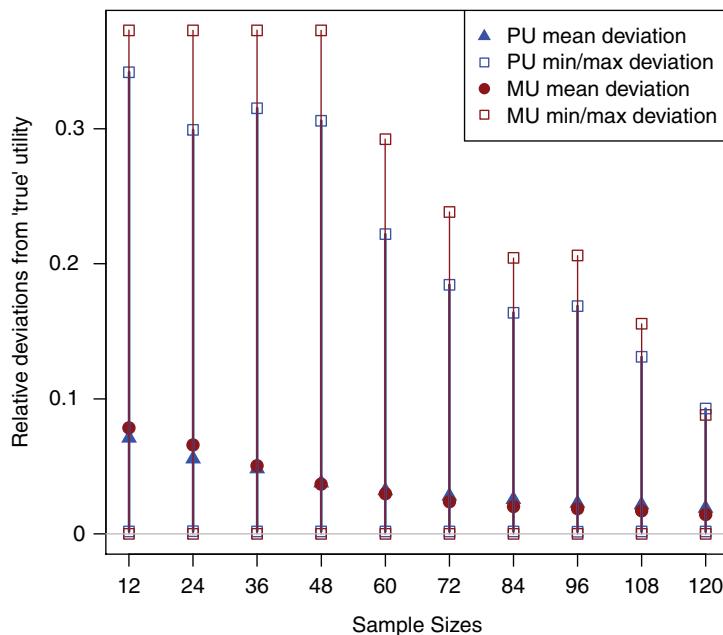


Figure 14.7 Graphical analysis of utility simulation.

Table 14.2 Probabilistic utility versus maximum expected utility: mean allocations.

Samples	GSPC		RUA		GDAXI		FTSE	
	MU	PU	MU	PU	MU	PU	MU	PU
Sample: 12	18.9	19.0	6.2	14.7	16.8	17.8	58.1	48.5
Sample: 24	14.0	16.5	6.7	12.2	16.0	16.2	63.3	55.1
Sample: 36	16.3	17.5	3.1	10.5	13.9	14.7	66.7	57.3
Sample: 48	17.2	17.1	2.4	10.2	10.2	11.9	70.2	60.8
Sample: 60	15.5	16.4	1.9	9.3	8.9	10.7	73.8	63.6
Sample: 72	16.3	16.4	1.5	9.2	7.2	9.5	74.9	65.0
Sample: 84	16.4	16.8	1.2	8.4	6.5	9.0	75.9	65.9
Sample: 96	15.1	15.8	1.3	7.8	5.9	8.2	77.7	68.2
Sample: 108	15.9	16.4	1.0	7.8	5.7	7.8	77.4	68.0
Sample: 120	15.8	16.1	1.2	7.7	4.7	7.0	78.3	69.2

R code 14.9 Allocation analysis of utility simulation.

```

## Mean allocations of PU and MU
1
PUWmean <- MUWmean <- matrix(NA, nrow = SS, ncol = N)
2
for(i in 1:SS){
3
  PUWmean[i, ] <- colMeans(PUW[, , i])
4
  MUWmean[i, ] <- colMeans(MUW[, , i])
5
}
6
## Min of PU and MU allocations
7
PUWmin <- MUWmin <- matrix(NA, nrow = SS, ncol = N)
8
for(i in 1:SS){
9
  PUWmin[i, ] <- apply(PUW[, , i], 2, min)
10
  MUWmin[i, ] <- apply(MUW[, , i], 2, min)
11
}
12
## Max of PU and MU allocations
13
PUWmax <- MUWmax <- matrix(NA, nrow = SS, ncol = N)
14
for(i in 1:SS){
15
  PUWmax[i, ] <- apply(PUW[, , i], 2, max)
16
  MUWmax[i, ] <- apply(MUW[, , i], 2, max)
17
}
18
## Range of PU and MU allocations
19
PUWrange <- PUWmax - PUWmin
20
MUWrange <- MUWmax - MUWmin
21
rownames(PUWmean) <- paste("Sample", Samples, sep = "-")
22
colnames(PUWmean) <- colnames(Requity)
23
rownames(MUWmean) <- rownames(PUWmean)
24
colnames(MUWmean) <- colnames(Requity)
25
rownames(PUWrange) <- paste("Sample", Samples, sep = "-")
26
colnames(PUWrange) <- colnames(Requity)
27
rownames(MUWrange) <- rownames(PUWrange)
28
colnames(MUWrange) <- colnames(Requity)
29

```

For sample sizes less than 60 observations the mean PU deviation is less than the corresponding value of the MEU allocations. As indicated in this figure, the MEU solutions are more sensitive to the random samples compared to the PU approach. For greater sample sizes, this difference becomes negligible, as the PU allocation approaches the MEU solution (e.g., for a sample size of 120 observations, which corresponds to a ten-year period for the monthly frequency).

Finally, in Listing 14.9 the mean allocations and their span are computed per sample size and optimization approach. Recall from Listing 14.7 that the optimal allocations are stored in three-dimensional arrays PUW and MUW for the PU and MU approaches, respectively. The mean allocations across the draws for each of these concepts are stored per sample size into the rows of the objects PUWmean and MUWmean for the PU and MU results, respectively. In a similar way, the minimum, maximum, and the span for each set are computed in the following `for` loops and

Table 14.3 Probabilistic utility versus maximum expected utility: span of weights.

Samples	GSPC		RUA		GDAXI		FTSE	
	MU	PU	MU	PU	MU	PU	MU	PU
Sample: 12	100.0	71.7	100.0	62.8	100.0	92.8	100.0	92.6
Sample: 24	100.0	63.6	100.0	67.1	100.0	83.9	100.0	90.6
Sample: 36	100.0	54.2	77.7	37.5	100.0	88.0	100.0	91.4
Sample: 48	100.0	71.3	62.4	33.7	100.0	86.7	100.0	87.3
Sample: 60	83.4	55.2	48.9	33.7	86.0	70.7	100.0	83.7
Sample: 72	88.2	49.6	61.6	35.7	76.0	63.5	98.4	76.7
Sample: 84	78.7	55.9	40.1	27.0	69.4	59.1	91.8	77.5
Sample: 96	75.6	51.9	45.6	30.6	69.7	58.7	77.9	68.5
Sample: 108	81.7	51.8	49.8	23.7	59.1	49.5	81.7	72.1
Sample: 120	80.7	56.7	43.5	28.1	42.5	41.9	80.7	72.4

the results are now stored into the objects `PUWrange` and `MUWrange` with obvious meaning. Finally, the four matrix objects are endowed with row and column names.

As can be seen from Table 14.2, the PU approach yields less concentrated portfolio solutions compared to the MEU optimizations. Furthermore, the PU allocations are less sensitive to sampling. As can be deduced from Table 14.3, the MEU solutions can yield allocations that span from one extreme to the other, that is, from 0% to 100% allotment in one single asset for sample sizes up to 60 observations. The span of weights and hence the sampling sensitivity is much more muted for the PU approach. An intuitive explanation of this result is that means of a random variable are less volatile than the random variable itself. As such, the concept of probabilistic utility can be fruitfully employed for avoiding concentrated portfolio solutions as encountered in empirical applications of direct maximization of expected utility.

References

- Bates D. and Eddelbuettel D. 2013 Fast and elegant numerical linear algebra using the `RcppEigen` package. *Journal of Statistical Software* **52**(5), 1–24.
- Boone E., Merrick J., and Krachev M. 2014 A Hellinger distance approach to MCMC diagnostics. *Journal of Statistical Computation and Simulation* **84**(4), 833–849.
- Brandimarte P. 2014 *Handbook in Monte Carlo Simulation, Applications in Financial Engineering, Risk Management and Economics* Wiley Handbooks in Financial Engineering and Econometrics. John Wiley & Sons, Hoboken, NJ.
- Brooks S. and Gelman A. 1998 General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics* **7**, 434–455.
- Brooks S., Gelman A., Jones G., and Meng X.-L. 2011 *Handbook of Markov Chain Monte Carlo*. Chapman & Hall / CRC, Boca Raton, FL.
- Consiglio A., Cocco F., and Zenios S. 2001 The value of integrative risk management for insurance products with guarantees. *Journal of Risk Finance* **2**, 6–16.

- Danenberg P. 2014 *functional: Curry, Compose, and other higher-order functions*. R package version 0.6.
- Duane S., Kennedy A., Pendleton B., and Roweth D. 1987 Hybrid Monte Carlo. *Physical Letters* **B195**, 216–222.
- Eddelbuettel D., Emerson J., and Kane M. 2016 *BH: Boost C++ Header Files*. R package version 1.60.0-2.
- Gelman A. and Rubin D. 1992 Inference from iterative simulation using multiple sequences. *Statistical Science* **7**, 457–511.
- Geweke J. 1992 Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments In *Bayesian Statistics 4* (ed. Bernardo J. M., Berger J. O., Dawid A. P., and Smith A. F. M.), pp. 169–193. Oxford Science Publications, Oxford.
- Geyer C. and Johnson L. 2015 *mcmc: Markov Chain Monte Carlo*. R package version 0.9-4.
- Ghahalo A. and Theussl S. 2015 *Rsolnp: General Non-linear Optimization Using Augmented Lagrange Multiplier Method*. R package version 1.16.
- Gilks W., Richardson S., and Spiegelhalter D. 1995 *Markov Chain Monte Carlo in Practice Interdisciplinary Statistics*. Chapman & Hall / CRC, Boca Raton, FL.
- Goodrich B., Gelman A., Carpenter B., Hoffman M., Lee D., Betancourt M., Brubaker M., Guo J., Li P., Riddell A., Inacio M., Morris M., Arnold J., Goedman R., Lau B., Trangucci R., Gabry J., Kucukelbir A., Grant R., Tran D., Malecki M., and Gao Y. 2016 *StanHeaders: C++ Header Files for Stan*. R package version 2.9.0.
- Hastings W. 1970 Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* **57**, 97–109.
- Heidelberger P. and Welch P. 1983 Simulation run length control in the presence of an initial transient. *Operations Research* **31**, 1109–1144.
- Hellinger E. 1909 Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. *Journal für die reine und angewandte Mathematik (Crelle's Journal)* **136**, 210–271.
- Hoffman M. and Gelman A. 2014 The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research* **15**, 1593–1623.
- Krachey M. and Boone E. 2012 *bmk: MCMC diagnostics package*. R package version 1.0.
- Marschinski R., Rossi P., Tavoni M., and Cocco F. 2007 Portfolio selection with probabilistic utility. *Annals of Operations Research* **151**, 223–239.
- Martin A., Quinn K., and Park J. 2011 MCMCpack: Markov chain Monte Carlo in R. *Journal of Statistical Software* **42**(9), 1–21.
- Metropolis N., Rosenbluth A., Rosenbluth M., Teller A., and Teller E. 1953 Equations of state calculations by fast computing machines. *Journal of Chemical Physics* **21**, 1087–1091.
- Meyn S. and Tweedie R. 1993 *Markov Chains and Stochastic Stability*. Springer-Verlag, New York.
- Michaud R. 1989 The Markowitz optimization enigma: Is optimized optimal. *Financial Analyst Journal* **45**, 31–42.
- Michaud R. 1998 *Efficient Asset Management: A Practical Guide to Stock Portfolio Optimization and Asset Allocation*. Oxford University Press, New York.
- Neal R. 2011 MCMC using Hamiltonian dynamics In *Handbook of Markov Chain Monte Carlo* (ed. Brooks S., Gelman A., Jones G., and Meng X.-L.) Handbooks of Modern Statistical Methods Chapman & Hall / CRC Boca Raton, FL pp. 113–162.

- Novomestky F. and Nadarajah S. 2012 *truncdist: Truncated Random Variables*. R package version 1.0-1.
- Pemstein D., Quinn K., and Martin A. 2011 The Scythe statistical library: An open source C++ library for statistical computation. *Journal of Statistical Software* **42**(12), 1–26.
- Plummer M., Best N., Cowles K., and Vines K. 2006 CODA: Convergence diagnosis and output analysis for MCMC. *R News* **6**(1), 7–11.
- Raftery A. and Lewis S. 1992 One long run with diagnostics: Implementation strategies for Markov chain Monte Carlo. *Statistical Science* **7**, 493–497.
- Raftery A. and Lewis S. 1995 Implementing MCMC In *Markov Chain Monte Carlo in Practice* (ed. Gilks W., Richardson S., and Spiegelhalter D.) Chapman and Hall / CRC Boca Raton, FL pp. 115–130.
- Robert C. and Casella G. 2004 *Monte Carlo Statistical Methods* 2nd edn. Springer-Verlag, New York.
- Robert C. and Casella G. 2010 *Introducing Monte Carlo Methods with R*. Springer-Verlag, New York.
- Rossi P., Tavoni M., Cocco F., and Marschinski R. 2002 Portfolio selection with probabilistic utility, Bayesian statistics and Markov chain Monte Carlo. arXiv:cond-mat/0211480.
- Sarkar D. 2008 *Lattice: Multivariate Data Visualization with R*. Springer, New York.
- Scheidegger A. 2012 *adaptMCMC: Implementation of a generic adaptive Monte Carlo Markov Chain sampler*. R package version 1.1.
- Stan Development Team 2015 *Stan Modeling Language User's Guide and Reference Manual, Version 2.8.0*. <http://mc-stan.org/>.
- Vihola M. 2012 Robust adaptive Metropolis algorithm with coerced acceptance rate. *Statistics and Computing* **22**(5), 997–1008.
- von Neumann J. 1951 Various techniques used in connection with random digits, notes by G.E. Forsythe. *National Bureau of Standards Applied Math Series* **12**, 36–38.
- Wickham H. 2011 The split-apply-combine strategy for data analysis. *Journal of Statistical Software* **40**(1), 1–29.

Appendix A

Package overview

The R packages cited and/or used in this book are listed in the tables below. First, the packages have been ordered alphabetically and the package's descriptions are provided. In the next table, the packages are sorted by topic. Within this table, the package's version numbers, dates, and sources are provided. All of these package have been updated prior to publishing and the example code contained in this book has been processed with these packages.

A.1 Packages in alphabetical order

Name	Title
adaptMCMC	Implementation of a generic adaptive Monte Carlo Markov Chain sampler
AER	Applied econometrics with R
bayesGARCH	Bayesian estimation of the GARCH(1,1) model with Student- <i>t</i> innovations
BH	Boost C++ header files
BLCOP	Black–Litterman and copula opinion pooling frameworks
bmk	MCMC diagnostics package
cccp	Cone-constrained convex problems
ccgarch	Conditional correlation GARCH models
chron	Chronological objects which can handle dates and times
coda	Output analysis and diagnostics for MCMC

Name	Title
copula	Multivariate dependence with copulas
covRobust	Robust covariance estimation via nearest neighbor cleaning
ctv	CRAN Task Views
date	Functions for handling dates
Davies	The Davies quantile function
DEoptim	Global optimization by differential evolution
DEoptimR	Differential evolution optimization in pure R
dse	Dynamic systems estimation (time series package)
EvalEst	Dynamic systems estimation – extensions
evd	Functions for extreme value distributions
evdbayes	Bayesian analysis in extreme value theory
evir	Extreme values in R
extRemes	Extreme value analysis
fArma	ARMA time series modelling
fBasics	Rmetrics – Markets and basic statistics
fCopulae	Rmetrics – Bivariate dependence structures with copulae
fExtremes	Rmetrics – Extreme financial market data
fGarch	Rmetrics – Autoregressive conditional heteroskedastic modelling
fMultivar	Rmetrics – Analyzing and modelling multivariate financial return distributions
forecast	Forecasting functions for time series and linear models
fPortfolio	Rmetrics – Portfolio selection and optimization
fPortfolioBacktest	Rmetrics – Portfolio backtesting
FRAPO	Financial risk modelling and portfolio optimization with R
fTrading	Technical trading analysis
fts	R interface to tslib (a time series library in C++)
functional	Curry, compose, and other higher-order functions
fUnitRoots	Trends and unit roots
GeneralizedHyperbolic	The generalized hyperbolic distribution
GEVStableGarch	ARMA–GARCH/APARCH models with GEV and stable distributions
ghyp	A package on the generalized hyperbolic distribution and its special cases
gld	Estimation and use of the generalized (Tukey) lambda distribution

Name	Title
glpkAPI	R Interface to C API of GLPK
gogarch	Generalized orthogonal GARCH (GO-GARCH) models
gumbel	The Gumbel–Hougaard copula
Hmisc	Harrell miscellaneous
in2extRemes	Into the extRemes package
ismev	An introduction to statistical modelling of extreme values
its	Irregular time series
lattice	Trellis graphics for R
lgarch	Simulation and estimation of log-GARCH models
limSolve	Solving linear inverse models
linprog	Linear programming / optimization
lmomco	L-moments, censored L-moments, trimmed L-moments, L-comoments, and many distributions
lpSolve	Interface to lp_solve v. 5.5 to solve linear/integer programs
lpSolveAPI	R interface for lp_solve version 5.5.2.0
MASS	Support functions and data sets for Venables and Ripley's MASS
mcmc	Markov chain Monte Carlo
MCMCpack	Markov chain Monte Carlo (MCMC) package
MSBVAR	Markov switching, Bayesian, vector autoregression models
nloptr	R interface to NLOpt
parma	Portfolio allocation and risk management applications
PerformanceAnalytics	Econometric tools for performance and risk analysis
plyr	Tools for splitting, applying, and combining data
PortfolioAnalytics	Portfolio analysis, including numerical methods for optimization of portfolios
QRM	Provides R code to examine quantitative risk management concepts
quadprog	Functions to solve quadratic programming problems
quantmod	Quantitative financial modelling framework
Rcpp	Seamless R and C++ integration
RcppArmadillo	Rcpp integration for the Armadillo templated linear algebra library
RcppDE	Global optimization by differential evolution in C++

Name	Title
RcppEigen	Rcpp integration for the Eigen templated linear algebra library
Renext	Renewal method for extreme values extrapolation
RenextGUI	GUI for Renext
Rglpk	R/GNU Linear Programming Kit interface
rmgarch	Multivariate GARCH models
rneos	XML-RPC interface to NEOS
robust	Robust library
robustbase	Basic robust statistics
rportfolios	Random portfolio generation
rrcov	Scalable robust estimators with high breakdown point
Rsocp	An R extension library to use SOCP from R
Rsolnp	General Non-linear Optimization Using Augmented Lagrange Multiplier Method
rstan	R interface to Stan
Rsymphony	Symphony in R
rugarch	Univariate GARCH models
RUnit	R unit test framework
shinystan	Interactive visual and numerical diagnostics and posterior analysis for Bayesian models
SkewHyperbolic	The skew hyperbolic Student <i>t</i> distribution
sn	The skew-normal and skew- <i>t</i> distributions
sos	Search contributed R packages, sort by package
StanHeaders	C++ header files for Stan
systemfit	Estimating systems of simultaneous equations
tawny	Provides various portfolio optimization strategies including random matrix theory and shrinkage estimators
tframe	Time frame coding kernel
tframePlus	Time frame coding kernel extensions
timeDate	Rmetrics – Chronological and calendar objects
timeSeries	Rmetrics – Financial time series objects
tis	Time indexes and time indexed series
truncdist	Truncated random variables
tseries	Time series analysis and computational finance
TTR	Technical trading rules
urca	Unit root and cointegration tests for time series data
VarianceGamma	The variance gamma distribution
vars	VAR modelling
xts	Extensible time series
zoo	S3 infrastructure for regular and irregular time series (Z's ordered observations)

A.2 Packages ordered by topic

Name	Version	Date	Source
Copula			
BLCop	0.3.1	02-04-2015	Gochez et al. (2015)
copula	0.999-14	10-26-2015	Hofert et al. (2015)
fCopulae	3011.81	09-17-2014	Würtz and Setz (2014a)
gumbel	1.10-1	07-20-2015	Dutang (2015)
QRM	0.4-13	03-18-2016	Pfaff and McNeil (2016)
EVT			
evd	2.3-2	12-25-2015	Stephenson (2002)
evdbayes	1.1-1	03-31-2014	Stephenson and Ribatet (2014)
evir	1.7-3	07-26-2012	Pfaff and McNeil (2012)
extRemes	2.0-7	12-18-2015	Gilleland and Katz (2011)
fExtremes	3010.81	12-17-2013	Würtz (2013b)
in2extRemes	1.0-2	07-24-2015	Gilleland and Katz (2011)
ismev	1.41	04-28-2016	Heffernan and Stephenson (2016)
QRM	0.4-13	03-18-2016	Pfaff and McNeil (2016)
Renext	3.0-0	06-23-2015	Deville (2015a)
RenextGUI	1.3-0	07-02-2015	Deville (2015b)
GARCH			
bayesGARCH	2.0.2	12-15-2015	Ardia (2015)
ccgarch	0.2.3	03-24-2014	Nakatani (2014)
fGarch	3010.82	05-01-2013	Würtz and Chalabi (2013)
GEVStableGarch	1.1	08-20-2015	do Rego Sousa et al. (2015)
gogarch	0.7-2	07-28-2012	Pfaff (2012)
lgarch	0.6-2	09-15-2015	Sucarrat (2015)
rmgarch	1.3-0	12-28-2015	Ghalanos (2015a)
rugarch	1.3-6	08-16-2015	Ghalanos (2015b)
tseries	0.10-35	02-20-2015	Trapletti and Hornik (2016)

Name	Version	Date	Source
GHD			
fBasics	3011.87	10-29-2014	Würtz et al. (2014a)
GeneralizedHyperbolic	0.8-1	01-05-2015	Scott (2015)
ghyp	1.5.6	02-05-2013	Luethi and Breymann (2013)
QRM	0.4-13	03-18-2016	Pfaff and McNeil (2016)
SkewHyperbolic	0.3-2	01-05-2015	Scott and Grimson (2015)
VarianceGamma	0.3-1	01-05-2015	Scott and Dong (2015)
GLD			
Davies	1.1-8	11-07-2012	Hankin and Lee (2006)
fBasics	3011.87	10-29-2014	Würtz et al. (2014a)
gld	2.3.3	04-20-2016	King et al. (2016)
lmomco	2.2.2	03-23-2016	Asquith (2016)
MCMC			
adaptMCMC	1.1	06-22-2012	Scheidegger (2012)
bmk	1.0	10-17-2012	Krachey and Boone (2012)
coda	0.18-1	10-16-2015	Plummer et al. (2006)
mcmc	0.9-4	07-17-2015	Geyer and Johnson (2015)
MCMCpack	1.3-6	04-15-2016	Martin et al. (2011)
rstan	2.9.0-3	02-12-2016	Stan Development Team (2015)
shinystan	2.1.0	01-06-2016	Gabry (2016)
Misc			
BH	1.60.0-2	05-07-2016	Eddelbuettel et al. (2016)
ctv	0.8-1	01-10-2015	Zeileis (2005)
functional	0.6	07-16-2014	Danenberg (2014)
Hmisc	3.17-4	05-02-2016	Harrell and Dupont (2016)
lattice	0.20-33	07-14-2015	Sarkar (2008)
plyr	1.8.3	06-12-2015	Wickham (2011)
Rcpp	0.12.4	03-26-2016	Eddelbüttel and François (2011)
RcppArmadillo	0.6.700.6.0	05-06-2016	Eddelbüttel and Sanderson (2014)
RcppEigen	0.3.2.8.1	03-01-2016	Bates and Eddelbuettel (2013)

Name	Version	Date	Source
RUnit	0.4.31	11-06-2016	Burger et al. (2015)
sos	1.3-8	09-25-2013	Graves et al. (2013)
StanHeaders	2.9.0	01-03-2016	Goodrich et al. (2016)
truncdist	1.0-1	03-16-2012	Novomestky and Nadarajah (2012)
Optimization			
cccp	0.2-4	02-10-2015	Pfaff (2015)
DEoptim	2.2-3	01-09-2015	Ardia et al. (2015)
DEoptimR	1.0-4	10-23-2015	Conceicao and Mächler (2015)
glpkAPI	1.3.0	01-07-2015	Gelius-Dietrich (2015)
limSolve	1.5.5.1	11-03-2014	Soetaert et al. (2009)
linprog	0.9-2	10-17-2012	Henningsen (2012)
lpSolve	5.6.13	09-19-2015	Berkelaar (2015)
lpSolveAPI	5.5.2.0-17	01-13-2016	Konis (2016)
nloptr	1.0.4	08-04-2014	Ypma et al. (2014)
quadprog	1.5-5	04-17-2013	Turlach and Weingessel (2013)
RcppDE	0.1.5	01-22-2016	Eddelbüttel (2016)
Rglpk	0.6-1	07-02-2015	Theussl and Hornik (2015)
rneos	0.3-1	03-06-2016	Pfaff (2016b)
Rsocp	271.1	11-11-2014	Chalabi and Würtz (2014)
Rsolnp	1.16	12-28-2015	Ghalanos and Theussl (2015)
Rsymphony	0.1-22	04-13-2016	Harter et al. (2016)
Portfolio			
fPortfolio	3011.81	10-30-2014	Würtz et al. (2014b)
fPortfolioBacktest	2110.4		Würtz et al. (2010)
parma	1.5-2	07-03-2015	Ghalanos and Pfaff (2015)
PerformanceAnalytics	1.4.3541	09-16-2014	Peterson and Carl (2014)
PortfolioAnalytics	1.0.3636	04-19-2015	Peterson and Carl (2015)
rportfolios	1.0	01-06-2012	Novomestky (2012)
tawny	2.1.2	05-07-2014	Lee and Rowe (2014)
Robust			
covRobust	1.1-0	05-10-2013	Wang et al. (2013)
fPortfolio	3011.81	10-30-2014	Würtz et al. (2014b)
MASS	7.3-45	04-21-2016	Venables and Ripley (2002)

Name	Version	Date	Source
robust	0.4-16	05-18-2014	Wang et al. (2014)
robustbase	0.92-5	07-22-2015	Rousseeuw et al. (2015)
rrcov	1.3-11	02-15-2016	Todorov and Filzmoser (2009)
Series			
AER	1.2-4	06-06-2015	Kleiber and Zeileis (2008)
chron	2.3-47	06-24-2015	James and Hornik (2015)
date	1.2-34	03-17-2014	Therneau et al. (2014)
fBasics	3011.87	10-29-2014	Würtz et al. (2014a)
fts	0.9.9	04-04-2014	Armstrong (2014)
its	1.1.8	09-06-2009	Armstrong (2009)
tframe	2015.12-1	12-16-2015	Gilbert (2015b)
tframePlus	2015.1-2	04-28-2015	Gilbert (2015c)
timeDate	3012.100	01-23-2015	Würtz et al. (2015a)
timeSeries	3022.101.2	12-14-2015	Würtz et al. (2015b)
tis	1.30	06-09-2015	Hallman (2015)
tseries	0.10-35	05-02-2016	Trapletti and Hornik (2016)
xts	0.9-7	01-02-2014	Ryan and Ulrich (2014)
zoo	1.7-13	05-03-2016	Zeileis and Grothendieck (2005)
TAA			
BLCOP	0.3.1	02-04-2015	Gochez et al. (2015)
dse	2015.12-1	12-16-2015	Gilbert (2009)
EvalEst	2015.4-2	05-02-2015	Gilbert (2015a)
fArma	3010.79	06-24-2013	Würtz (2013a)
fMultivar	3011.78	09-06-2014	Würtz and Setz (2014b)
forecast	7.1	04-14-2016	Hyndman (2016)
FRAPO	0.4-0		Pfaff (2016a)
fTrading	3010.78	12-10-2013	Würtz (2013c)
fUnitRoots	3010.78	06-24-2013	Würtz (2013d)
MSBVAR	0.9-2	02-10-2015	Brandt (2015)
quantmod	0.4-5	07-24-2015	Ryan (2015)
sn	1.3-0	11-11-2015	Azzalini (2015)
systemfit	1.1-18	08-27-2015	Henningsen and Hamann (2007)
TTR	0.23-1	03-21-2016	Ulrich (2016)
urca	1.2-9	01-11-2016	Pfaff (2008a)
vars	1.5-2	07-22-2013	Pfaff (2008b)

References

- Ardia D. 2015 *bayesGARCH: Bayesian Estimation of the GARCH(1,1) Model with Student-t Innovations in R*. Version 2.0.2.
- Ardia D., Mullen K., Peterson B., and Ulrich J. 2015 *DEoptim: Differential Evolution in R*. version 2.2-3.
- Armstrong W. 2009 *its: Irregular Time Series*. R package version 1.1.8.
- Armstrong W. 2014 *fts: R interface to tslib (a time series library in C++)*. R package version 0.9.9.
- Asquith W. 2016 *lmomco – L-moments, censored L-moments, trimmed L-moments, L-comoments, and many distributions*. R package version 2.2.2.
- Azzalini A. 2015 *The R package sn: The skew-normal and skew-t distributions (version 1.3-0)*. Università di Padova, Italia.
- Bates D. and Eddelbuettel D. 2013 Fast and elegant numerical linear algebra using the RcppEigen package. *Journal of Statistical Software* **52**(5), 1–24.
- Berkelaar M. 2015 *lpSolve: Interface to “lp_solve” v. 5.5 to Solve Linear/Integer Programs*. R package version 5.6.13.
- Brandt P. 2015 *MSBVAR: Markov-Switching, Bayesian, Vector Autoregression Models*. R package version 0.9-2.
- Burger M., Jünemann K., and König T. 2015 *RUnit: R Unit Test Framework*. R package version 0.4.31.
- Chalabi Y. and Würtz D. 2014 *Rsocp: An R extension library to use SOCP from R*. R package version 271.1.
- Conceicao E. and Mächler M. 2015 *DEoptimR: Differential Evolution Optimization in pure R*. R package version 1.0-4.
- Danenberg P. 2014 *functional: Curry, Compose, and other higher-order functions*. R package version 0.6.
- Deville Y. 2015a *Renext: Renewal Method for Extreme Values Extrapolation*. R package version 3.0-0.
- Deville Y. 2015b *RenextGUI: GUI for Renext*. R package version 1.3-0.
- do Rego Sousa T., Otiniano C., and Lopes S. 2015 *GEVStableGarch*. R package version 1.1.
- Dutang C. 2015 *gumbel: package for Gumbel copula*. R package version 1.10-1.
- Eddelbuettel D., Emerson J., and Kane M. 2016 *BH: Boost C++ Header Files*. R package version 1.60.0-2.
- Eddelbüttel D. 2016 *RcppDE: Global optimization by Differential Evolution in C++*. R package version 0.1.5.
- Eddelbüttel D. and François R. 2011 *Rcpp: Seamless R and C++ integration*. *Journal of Statistical Software* **40**(8), 1–18.
- Eddelbüttel D. and Sanderson C. 2014 *RcppArmadillo: Accelerating R with high-performance C++ linear algebra*. *Computational Statistics and Data Analysis* **71**, 1054–1063.
- Gabry J. 2016 *shinystan: Interactive Visual and Numerical Diagnostics and Posterior Analysis for Bayesian Models*. R package version 2.1.0.
- Gelius-Dietrich G. 2015 *glpkAPI: R Interface to C API of GLPK*. R package version 1.3.0.
- Geyer C. and Johnson L. 2015 *mcmc: Markov Chain Monte Carlo*. R package version 0.9-4.

- Ghalanos A. 2015a *rmgarch: Multivariate GARCH models*. R package version 1.3-0.
- Ghalanos A. 2015b *rugarch: Univariate GARCH models*. R package version 1.3-6.
- Ghalanos A. and Pfaff B. 2015 *parma: Portfolio allocation and risk management applications*. R package version 1.5-2.
- Ghalanos A. and Theussl S. 2015 *Rsolnp: General non-linear optimization using augmented Lagrange Multiplier Method*. R package version 1.16.
- Gilbert P. 2009 *Brief User's Guide: Dynamic Systems Estimation*. CRAN.
- Gilbert P. 2015a *EvalEst: Dynamic Systems Estimation – Extensions*. R package version 2015.4-2.
- Gilbert P. 2015b *tframe: Time Frame Coding Kernel*. R package version 2015.12-1.
- Gilbert P. 2015c *tframePlus: Time Frame Coding Kernel Extensions*. R package version 2015.1-2.
- Gilleland E. and Katz R. 2011 New software to analyze how extremes change over time. *Eos* **92**(2), 13–14.
- Gochez F., Chandler-Mant R., Jin S., and Xie J. 2015 *BLCOP: Black–Litterman and Copula Opinion Pooling Frameworks*. R package version 0.3.1.
- Goodrich B., Gelman A., Carpenter B., Hoffman M., Lee D., Betancourt M., Brubaker M., Guo J., Li P., Riddell A., Inacio M., Morris M., Arnold J., Goedman R., Lau B., Trangucci R., Gabry J., Kucukelbir A., Grant R., Tran D., Malecki M., and Gao Y. 2016 *StanHeaders: C++ Header Files for Stan*. R package version 2.9.0.
- Graves S., Dorai-Raj S., and Francois R. 2013 *sos: Search contributed R packages, sort by package*. R package version 1.3-8.
- Hallman J. 2015 *tis: Time Indexes and Time Indexed Series*. R package version 1.30.
- Hankin R. and Lee A. 2006 A new family of non-negative distributions. *Australia and New Zealand Journal of Statistics* **48**, 67–78.
- Harrell F. and Dupont C. 2016 *Hmisc: Harrell Miscellaneous*. R package version 3.17-4.
- Harter R., Hornik K., and Theussl S. 2016 *Rsymphony: SYMPHONY in R*. R package version 0.1-22.
- Heffernan E. and Stephenson A. 2016 *ismev: An Introduction to Statistical Modeling of Extreme Values*. R package version 1.41.
- Henningsen A. 2012 *linprog: Linear Programming / Optimization*. R package version 0.9-2.
- Henningsen A. and Hamann J. 2007 systemfit: A package for estimating systems of simultaneous equations in R. *Journal of Statistical Software* **23**(4), 1–40.
- Hofert M., Kojadinovic I., Mächler M., and Yan J. 2015 *copula: Multivariate Dependence with Copulas*. R package version 0.999-14.
- Hyndman R. 2016 *forecast: Forecasting functions for time series and linear models*. R package version 7.1.
- James D. and Hornik K. 2015 *chron: Chronological Objects which Can Handle Dates and Times*. R package version 2.3-47. S original by David James, R port by Kurt Hornik.
- King R., Dean B., and Klinke S. 2016 *gld: Estimation and use of the generalised (Tukey) lambda distribution*. R package version 2.3.3.
- Kleiber C. and Zeileis A. 2008 *Applied Econometrics with R*. Springer-Verlag, New York.
- Konis K. 2016 *lpSolveAPI: R Interface for lp_solve version 5.5.2.0*. R package version 5.5.2.0-17.

- Krachey M. and Boone E. 2012 *bmk: MCMC diagnostics package*. R package version 1.0.
- Lee B. and Rowe Y. 2014 *tawny: Provides various portfolio optimization strategies including random matrix theory and shrinkage estimators*. R package version 2.1.2.
- Luethi D. and Breymann W. 2013 *ghyp: A package on the generalized hyperbolic distribution and its special cases*. R package version 1.5.6.
- Martin A., Quinn K., and Park J. 2011 MCMCpack: Markov chain Monte Carlo in R. *Journal of Statistical Software* **42**(9), 1–21.
- Nakatani T. 2014 *ccgarch: An R Package for Modelling Multivariate GARCH Models with Conditional Correlations*. R package version 0.2.3.
- Novomestky F. 2012 *rportfolios: Random portfolio generation*. R package version 1.0.
- Novomestky F. and Nadarajah S. 2012 *truncdist: Truncated Random Variables*. R package version 1.0-1.
- Peterson B. and Carl P. 2014 *PerformanceAnalytics: Econometric tools for performance and risk analysis*. R package version 1.4.3541.
- Peterson B. and Carl P. 2015 *PortfolioAnalytics: Portfolio Analysis, Including Numerical Methods for Optimization of Portfolios*. R package version 1.0.3636.
- Pfaff B. 2008a *Analysis of Integrated and Cointegrated Time Series with R* 2nd edn. Springer, New York.
- Pfaff B. 2008b VAR, SVAR and SVEC models: Implementation within R package vars. *Journal of Statistical Software* **27**(4), 1.
- Pfaff B. 2012 *gogarch: Generalized Orthogonal GARCH (GO-GARCH) models*. R package version 0.7-2.
- Pfaff B. 2016a *Financial Risk Modelling and Portfolio Optimisation with R* 2nd edn. John Wiley & Sons, Ltd, London.
- Pfaff B. 2016b *rneos: XML-RPC Interface to NEOS*. R package version 0.3-1.
- Pfaff B. 2015 *cccp: Cone Constrained Convex Problems*. R package version 0.2-4.
- Pfaff B. and McNeil A. 2012 *evir: Extreme Values in R*. R package version 1.7-3.
- Pfaff B. and McNeil A. 2016 *QRM: Provides R-language Code to Examine Quantitative Risk Management Concepts*. R package version 0.4-13.
- Plummer M., Best N., Cowles K., and Vines K. 2006 CODA: Convergence diagnosis and output analysis for MCMC. *R News* **6**(1), 7–11.
- Rousseeuw P., Croux C., Todorov V., Ruckstuhl A., Salibian-Barrera M., Verbeke T., Koller M., and Mächler M. 2015 *robustbase: Basic Robust Statistics*. R package version 0.92-5.
- Ryan J. 2015 *quantmod: Quantitative Financial Modelling Framework*. R package version 0.4-5.
- Ryan J. and Ulrich J. 2014 *xts: eXtensible Time Series*. R package version 0.9-7.
- Sarkar D. 2008 *Lattice: Multivariate Data Visualization with R*. Springer, New York.
- Scheidegger A. 2012 *adaptMCMC: Implementation of a generic adaptive Monte Carlo Markov Chain sampler*. R package version 1.1.
- Scott D. 2015 *GeneralizedHyperbolic: The Generalized Hyperbolic Distribution*. R package version 0.8-1.
- Scott D. and Dong C. Y. 2015 *VarianceGamma: The Variance Gamma Distribution*. R package version 0.3-1.

- Scott D. and Grimson F. 2015 *SkewHyperbolic: The Skew Hyperbolic Student t-Distribution*. R package version 0.3-2.
- Soetaert K., Van den Meersche K., and van Oevelen D. 2009 *limSolve: Solving Linear Inverse Models*. R package 1.5.1.
- Stan Development Team 2015 *Stan Modeling Language User's Guide and Reference Manual, Version 2.9.0*. <http://mc-stan.org/>.
- Stephenson A. 2002 evd: Extreme value distributions. *R News* **2**(2), 0.
- Stephenson A. and Ribatet M. 2014 *evdbayes: Bayesian Analysis in Extreme Value Theory*. R package version 1.1-1.
- Sucarrat G. 2015 *lgarch: Simulation and estimation of log-GARCH models*. R package version 0.6-2.
- Therneau T., Lumley T., Halvorsen K., and Hornik K. 2014 *date: Functions for handling dates*. R package version 1.2-34. S original by Terry Therneau, R port by Thomas Lumley, Kjetil Halvorsen, and Kurt Hornik.
- Theussl S. and Hornik K. 2015 *Rglpk: R/GNU Linear Programming Kit Interface*. R package version 0.6-1.
- Todorov V. and Filzmoser P. 2009 An object-oriented framework for robust multivariate analysis. *Journal of Statistical Software* **32**(3), 1–47.
- Trapletti A. and Hornik K. 2016 *tseries: Time Series Analysis and Computational Finance*. R package version 0.10-35.
- Turlach B. A. and Weingessel A. 2013 *quadprog: Functions to solve Quadratic Programming Problems*. R package version 1.5-5.
- Ulrich J. 2016 *TTR: Technical Trading Rules*. R package version 0.23-1.
- Venables W. N. and Ripley B. D. 2002 *Modern Applied Statistics with S* 4th edn. Springer, New York.
- Wang J., Zamar R., Marazzi A., Yohai V., Salibian-Barrera M., Maronna R., Zivot E., Rocke D., Martin D., Maechler M., and Konis K. 2014 *robust: Robust Library*. R package version 0.4-16.
- Wang N., Raftery A., and Fraley C. 2013 *covRobust: Robust Covariance Estimation via Nearest Neighbor Cleaning*. R package version 1.1-0.
- Wickham H. 2011 The split-apply-combine strategy for data analysis. *Journal of Statistical Software* **40**(1), 1–29.
- Würtz D. 2013a *fArma: ARMA Time Series Modelling*. R package version 3010.79.
- Würtz D. 2013b *fExtremes: Rmetrics – Extreme Financial Market Data*. R package version 3010.81.
- Würtz D. 2013c *fTrading: Technical Trading Analysis*. R package version 3010.78.
- Würtz D. 2013d *fUnitRoots: Trends and Unit Roots*. R package version 3010.78.
- Würtz D. and Chalabi Y. 2013 *fGarch: Rmetrics – Autoregressive Conditional Heteroskedastic Modelling*. R package version 3010.82.
- Würtz D., Chalabi Y., Chen W., and Ellis A. 2010 *Portfolio Optimization with R/Rmetrics*. Rmetrics Association & Finance Online, www.rmetrics.org. R package version 2110.4.
- Würtz D. and Setz T. 2014a *fCopulae: Rmetrics – Bivariate Dependence Structures with Copulae*. R package version 3011.81.

- Würtz D. and Setz T. 2014b *fMultivar: Rmetrics – Analysing and Modeling Multivariate Financial Return Distributions*. R package version 3011.78.
- Würtz D., Setz T., and Chalabi Y. 2014a *fBasics: Rmetrics – Markets and Basic Statistics*. R package version 3011.87.
- Würtz D., Setz T., and Chalabi Y. 2014b *fPortfolio: Rmetrics – Portfolio Selection and Optimization*. R package version 3011.81.
- Würtz D., Chalabi Y., Maechler M., and Byers J. 2015a *timeDate: Rmetrics – Chronological and Calendar Objects*. R package version 3012.100.
- Würtz D., Setz T., and Chalabi Y. 2015b *timeSeries: Rmetrics – Financial Time Series Objects*. R package version 3012.99.
- Ypma J., Borchers H., and Eddelbüttel D. 2014 *nloptr: R interface to NLOpt*. R package version 1.0.4.
- Zeileis A. 2005 CRAN task views. *R News* **5**(1), 39–40.
- Zeileis A. and Grothendieck G. 2005 *zoo: S3 infrastructure for regular and irregular time series*. *Journal of Statistical Software* **14**(6), 1–27.

Appendix B

Time series data

There are several possibilities for dealing with time series data in R. In this appendix a concise account is provided for the date/time classes and the available implementations for creating and handling time series data objects. This exposition is based on Ripley and Hornik (2001), Grothendieck and Petzoldt (2004), Zeileis and Grothendieck (2005), Wüertz et al. (2009), and Chalabi et al. (2011). In the next section, the classes and available methods for handling dates and times are presented. In the remaining part, the various possibilities for creating a time series object and its manipulations are described.

B.1 Date/time classes

Classes and methods for defining and manipulating dates and times are provided in the base R distribution, as well as in contributed packages. In the following the classes and methods are presented in increasing order of complexity.

The class `Date` was introduced to base R in version R-1.9.0. This class supports only dates without a time stamp. Internally, dates are represented as the day count since 1 January 1970. The printing of these objects can be adjusted according to the percentage abbreviations listed in `?strptime` with the `format()` method defined for these objects. A coerce method (`as.Date()`) and utility functions for extracting the day of week (`weekdays()`) and the month (`months()`) are available, too. Furthermore, methods for labeling of axes in plots and histograms are made available, as well as facilities for computing/manipulating objects of this class.

A second class for handling dates only has been implemented in the CRAN package `date` (see Therneau et al. 2014). In this package an S3 class `date` is defined in which dates are stored as the count of days since 1 January 1960. This origin is identical to the chosen reference date in SAS. The package offers methods for converting

Julian dates to calendar dates, so that the latter can be expressed in various formats, and vice versa. Furthermore, a coerce method (`as.date()`) and a logical test to check whether an object is of class `date` (`is.date()`) are also made available. Simple \pm operations can be computed with objects of this class.

The following classes enable the user to handle date/time objects. The least complex of this classes is `chron`, provided in the CRAN package of the same name (see James and Hornik 2015). Objects of class `chron` do not support time zones or daylight saving time rules. Objects are created by calling the generator function `chron()`, where the dates are provided as vector argument `dates`. and the time stamps as argument `times`. The input and output format can be controlled by the arguments `format` and `out.format`, respectively. Here, the abbreviations as detailed in `?strptime` can be employed. A time origin different from the default of 1 January 1970 can be provided by the argument `origin`. Alternatively, the `chron` origin can also be set via an `options()` expression. If only a `dates`. argument is provided, the function returns an object of S3 class `dates`; if only a time stamp has been passed to `chron()`, an object with class attribute `times` will be returned, which has the characteristic of a super class and hence `chron` and `dates` objects inherit methods defined for `times` objects. Objects that refer only to either `dates` or `times` can also be created by the generator functions `dates()` and `times()`, respectively. Internally, `times` are represented as fractions of a day, so that 1.5 will signify noon of the day in question. Numerous methods have been specified for these kinds of objects. Arithmetic operations are defined in the form of differences between `chron` objects and the addition of a constant. All logical comparisons are supported, too. The conversion between Julian and calendar dates is implemented as functions `julian()`, `month.day.year()`, and `day.of.week()`. Whether or not a year is a leap year can be assessed by calling `leap.year()` with the year as argument. A sequence of dates is swiftly returned by the function `seq.dates()`, and for sub-setting a `chron` or `dates` object a `cut()` method has been defined. Certain periods from either a `chron` or `dates` object can be returned by the functions `days()`, `weekdays()`, `months()`, `quarters()`, and `years()`, with obvious meaning. Similarly, with respect to `times` objects, the functions `hours()`, `minutes()`, and `second()` can be utilized for extracting the elements of a time stamp. Whether a certain date adheres to a weekend or a holiday can be checked with the functions `is.weekend()` and `is.holiday()`, respectively. By default, only U.S. holidays during the year 1992 are supplied as object `.Holidays`, but this object can be amended/modified by holiday dates supplied by the user. Finally, coercing methods from objects of classes `yearmon()`, `yearqtr()`, and `ts` are provided, too. The first two classes are defined in the package `zoo`, to be described in Section B.5, and the time series class `ts`, which will be discussed in Section B.2.

In the base distribution of R, POSIX classes are defined for the handling of dates associated with a time stamp. The classes are `POSIXct` and `POSIXlt`, with a super/parent class `POSIXt`. The former class is intended for handling calendar times (hence the suffix “ct”) and the latter is intended for handling local times (hence the suffix “lt”). A motivation and concise description of these classes is given in Ripley and Hornik (2001). Objects of class `POSIXct` are represented by the count of

seconds since 1 January 1970 (GMT). In contrast, dates and times in the form of `POSIXlt` objects are represented as a list of nine elements: `sec` for the seconds, `min` for the minutes, `hour` for the hour, `mday` for the day in a month, `mon` for the month in a year, `year` for the year, `wday` for the day of the week (from 0 for Sunday to 6 for Saturday), `yday` for the day count in a year, and `isdst` as a logical flag whether a daylight saving scheme was in existence at the particular date and time. `POSIXlt` objects can further contain an optional time zone attribute (`tzone`), and objects of class `POSIXct` can also be amended by such an attribute (the default will be GMT). The POSIX class in the wider sense therefore provides a means of handling different time zone dates/times as well as periods of daylight saving time. For these S3 classes, methods for conversion to/from character representation and date/time manipulations are specified. The former group consists of the methods `format()`, `as.character()`, `strftime()`, and `strptime()`. Furthermore, objects of one child class can be coerced into the other child class by utilizing the `as.POSIXct()` and `as.POSIXlt()` methods, respectively. Arithmetic operations are defined for adding/subtracting a constant (in seconds) from these objects. Any logical operator can be applied to two sets of POSIX objects for comparison. The cutting and sequencing of POSIX objects can be achieved by the methods `cut()` and `seq()`, and the rounding and truncation of POSIX objects is provided by the methods `round()` and `trunc()`. These four methods are defined in the super class `POSIXt`. Similarly, methods for extracting elements pertinent to a date are `weekdays()`, `months()`, `quarters()`, and `julian()` for objects inheriting from this super class, with obvious meaning. Finally, time intervals created by either the subtraction of two POSIX objects or by the function `difftime()` will produce an object of class `difftime`. The printing of the time difference can be tailored by a `format()` method and/or by choosing the units of the time interval. By default, the units will be equal to the lowest frequency implied by the time interval. A coercing method for character and numeric values is also defined: (`as.difftime()`). Finally, mathematical operations for rounding, extracting the significant digits, finding floor and ceiling values, truncating, and determining the absolute value of a time interval are defined as methods.

The most sophisticated representation of date/time objects has been accomplished in the class `timeDate`. This class is defined in the contributed package **timeDate** (see Würtz et al. 2015a), and a concise description of the class and its associated methods and functions is given in Chalabi et al. (2011). Objects of S4 class `timeDate` consist of three slots: the slot `Data` of class `POSIXct` and the slots `format` and `FinCenter`, both characters. The `format` slot contains information on how `Data` will be displayed, and `FinCenter` is the time domain of the financial center. This latter slot is of importance when data from different time zones and/or financial centers are aggregated. By default, GMT is assumed for both: the time zone of the data and the time zone of the financial center. Aside from the time offset with respect to GMT of a financial center, further information can be queried about it, for example, the day-count convention and the opening hours of a stock exchange. This information can be queried and modified with the `setRmetricsOptions()` function, as the package **timeDate** is part of the Rmetrics bundle. Objects of formal S4 class

`timeDate` can be generated by calling `timeDate()`, or `timeSequence()`, or `timeCalendar()`. The first function takes a character vector of dates/times as input (the format of this character vector can be provided by a `format` argument), the time zone by the argument `zone`, and the financial center by the argument `Fin-Center`. If the latter two arguments are not specified, GMT will be assumed. A set of equally spaced dates/times can be created with second function. It takes as arguments the starting date/time (`from`) and either the ending date/time (`to`) or the length of the `timeDate` object to be returned. The frequency has to be specified via its `by` argument. This character string defines the increments as one of `sec`, `min`, `hour`, `day`, `week`, `month`, `year`, with obvious meaning. Finally, if the calendar date information is available as elements, an object of class `timeDate` is returned by the last function, `timeCalendar()`, where by default the current year will be used as the year stamp and the user can supply vectors for the months, days, hours, minutes, and seconds, as well as information on the time zone and the financial center. If the latter two are omitted from the call, then GMT will be assumed for both. Strengths of the `timeDate` package are the testing for week days, business days, and holidays on the one hand, and the creation of so-called special dates. Within the former group the functions `isWeekday()`, `isWeekend()`, `isBizday()`, `isHoliday()`, and `dayOfWeek()` are at hand. The holiday calendar is specific for a financial center, but user-defined holiday calendars can be set up rather swiftly by either amending an existent holiday calendar or by specifying a new calendar consisting of a `timeDate` object referring to the dates of the holidays. Special date sequences, such as the last day in a month, quarter, or year or the `n`th weekday in a month or quarter are created by calling one of the functions listed below:

- `timeFirstDayInMonth()`
- `timeLastDayInMonth()`
- `timeFirstDayInQuarter()`
- `timeLastDayInQuarter()`
- `timeNthNdayInMonth()`
- `timeLastNdayInMonth()`
- `timeNdayOnOrAfter()`
- `timeNdayOnOrBefore()`

Finally, the start and end dates of a `timeDate` object can be queried with the functions `start` and `end`, respectively. Incidentally, `min()`, `max()`, and `range()` methods are made available, too. A window of date times can be extracted from a `timeDate` object by means of the `window()` function, and sub-setting possible by the `[]` operator.

B.2 The `ts` class in the base package `stats`

First, the S3 class `ts` in the base package `stats` is well suited for handling regularly spaced time series data. The generator function is `ts()`, which returns a time series object with class attribute `ts`, in the case of a univariate series, that is, if a vector has been supplied, or an object with class attributes `c("mts", "ts")`, if a multivariate series has been provided, that is, a matrix or a data frame. Objects can be coerced to `ts` objects by the `as()` method and tested if they belong to this class with the function `is.ts()`. The start and end dates are set by the arguments `start` and `end`, respectively, and the frequency of the regularly spaced observations by the argument `frequency`. Instead of the frequency, its reciprocal can be provided for the argument `deltat`. The dates for the first and last observations can be either specified as a single number or as two-element integer vectors that represent the natural time unit. Objects of this class own a time series property that can be shown by the function `tsp()`. The time series property is an attribute of a `ts` object and is a three-element vector consisting of the start and end periods and the frequency. Checking whether an arbitrary object is endowed with this attribute can be conducted with the function `hasTsp()`. Furthermore, a time series property can be modified/amended to an object `x` by `tsp(x) <- value`. The time component of an object of class `ts` can be retrieved with the function `time()`. Sub-setting a time series object to a narrower sample range is accomplished by using the `window()` function. The dates of the first and last observations are returned by the functions `start()` and `end()`, respectively. Finally, `plot()` and `print()` methods are defined for objects of this class.

B.3 Irregularly spaced time series

The class `ts` is confined to the representation of regularly spaced series. The creation and manipulation of irregularly spaced time series is covered by the two CRAN packages `its` and `tseries`, respectively (see Armstrong 2009; Trapletti and Hornik 2016). The former is solely dedicated to irregularly spaced time series; the latter package contains a class definition for these special time series with associated functions and methods.

An S3 class `irts` is defined in the package `tseries`. Objects with this class attribute are defined as a list object with elements `time` and `value` for holding the date/time stamps and the values of the time series, respectively. Akin to the class design in the package `its`, the element `time` is of class `POSIXct` and the object of the second list element can be either a `vector` or a `matrix` object. Methods for coercion and logical testing for this kind of S3 object are implemented as `as.irts()` and `is.irts()`, respectively. For objects with class attribute `irts`, methods for plotting, printing, and the extraction of date/time stamps and/or the values of the series have been defined. The S3 methods for plotting are: `lines()` for superimposing on an existing plotting device the progression of the irregular time series, `plot()` for plotting the time series itself as a time series chart, and `points()` for

superimposing points on an existing graphic so that the points of the irregular time series are joined by line segments. The `print()` method has a `format` argument, by which the representation of the date/time stamps can be controlled. The date/time stamps can be recovered from an `irts` object with the function `time()` and the values of the series itself with the function `value()`. Sub-setting of an `irts` object can be accomplished by the `[]` operator. Aside from these methods, functions are provided for inspection and analysis of the date/time stamps and the handling of `irts` objects. In the former group, the count of seconds after midnight is obtained by calling `daysecond()` and the day of the week by calling the function `weekday()`. The weekdays are coded as integers starting with 0 for Sunday and ending with 6 for Saturday. It is further possible to assess whether an entry falls on a business day or on a weekend by utilizing the functions `is.businessday()` or `is.weekend()`. Intermittent values of an irregularly spaced time series can be approximated by interpolation for a prespecified sequence of date/time stamps with the function `approx.irts()`. Finally, reading/writing from/to a file connection can be accomplished with the functions `read.irts` and `write.irts`. Both functions have a `format` argument for specifying the representation of the date/time stamps.

In the CRAN package `its` an S4 class of the same name is provided for handling irregular time series objects. Objects of this class are endowed with the two slots `dates` and `.Data` for holding the date/time information as a `POSIXt` object and the series data as a `matrix` object, respectively. Objects of this class are created by the generator function `its()`. Methods for coercing a matrix to an `its` object and testing for such an object are provided by `as.its()` and `is.its()`, respectively. Because the slot `.Data` is of class `matrix`, all arithmetic and logical operations defined for these objects are available for `its` objects, too. The former methods assume that the first column contains the count of seconds since 1 January 1970 as date/time stamps. Further, methods for coercion of `its` objects to lists and data frames are available via the methods `as.list()` and `as.data.frame()`, respectively. The dates/times and the data can be extracted from `its` objects with the function `dates()` and `core()`. Both functions can also be used for assigning dates/times – in the form of `POSIXt` objects – and time series data – in the form of `matrix` objects – to an existing `its` object. With respect to the date/time stamps of an irregular series, the functions `daysecondIts()` and `weekdayIts()` for determining the count of seconds since midnight and/or whether the dates/times are pertinent to weekdays are also available. A last observation carry forward function, `locf()`, is available for filling `NA` values with the entries of previous non-`NA` values. Two objects of formal class `its` can be joined by alignment, by appending, or by collapsing with the functions `alignedIts()`, `appendIts()`, or `collapseIts`, respectively. There are two means of lagging a time series object available: `lagIts()` and `lagdistIts()`. The former function lags the time series by `k` periods, if `k` is a positive integer, and leads a time series if `k` is negative. The latter function applies the former function to a sequence of lags defined by the arguments `kmin` and `kmax`. The returned object contains the sequenced lagged as additional columns of the original series. The sub-setting of an `its` object can be achieved in two ways. First, the function `rangeIts()` works

similarly to the previously described `window()` function. Second, the function `extractIts()` returns a semi-regular `its` object, whereby the user can provide the periodicity (week, month, and year) and whether all data points, or only the first or the last observation for a period, are returned (`all`, `last`, `first`). The function's arguments are named `period` and `find` for the two sets of possibilities. The accrued annual interest rate as a per-period return is returned by the function `accrueIts()` in the form of a decimal return. The day count convention to be used in this computation is controlled by the argument `daysperyear`. Finally, for downloading data from Internet services, the function `priceIts()` has been defined and the reading/writing from/to CSV files can be accomplished with the functions `readcsvIts()` and `writetcsvIts()`, respectively.

B.4 The package `timeSeries`

An S4 class definition `timeSeries` is provided in the package of the same name (see Würtz et al. 2015b). This class has a total of eight slots, including the slots of the date/time presentation as given by the class `timeDate`. Objects of this class can be created by either the generator function `timeSeries()`, or by reading from a file connection (`readSeries()`), or by means of coercion. The two most important arguments for the generator function are `data` and `charvec`. As input for `data` a matrix object or an object that can be coerced to it must be provided; `charvec` can basically be any kind of character vector containing the date/time stamps of `data`. The only requirement for `charvec` is that it can be coerced to an object of class `timeDate`. The format of the vector supplied for `charvec` can be specified as argument `format`. In addition to these arguments, information on the time zone (argument `zone`), on the financial center (argument `FinCenter`), new column names (`units`), a description of the time series (argument `description`), a title for the returned object (argument `title`), and elaborative identifiers for the records (argument `recordIDs`) can be provided, too, but are left unset by default. For the latter, coercing methods for objects of classes `numeric`, `data.frame`, `matrix`, `ts`, `character`, and `zoo` are provided. Whether an object `x` is of formal class `timeSeries` can be tested by calling `is.timeSeries(x)`. The emptiness of the slot `positions`, which carries the date/time information, can be assessed with the function `is.signalSeries()`; if empty, the time series object would only consist of the time series data itself. The characteristics of a `timeSeries` object with respect to its dimension and regularity/periodicity can be tested with the functions `isMultivariate()` and `isUnivariate()`, for the former, and `isRegular()`, `isDaily()`, `isMonthly()`, `isQuarterly()`, and `frequency()` for the latter tests, with obvious meanings. The series and date/time elements of a `timeSeries` object can be queried and assigned by the functions `seriesData()` and `time()`, respectively. The start and end dates/times of a series are returned by the functions `start()` and `end()`. Because it is not required that the date/time information in the `charvec` vector is ordered per se, an object of class `timeSeries` can be ordered with respect to its positions with the `sort()` method (increasing as default) or can be reverted with respect to time by the `rev()` method. Resampling of a time

series can be accomplished with the `sample()` method. For displaying the contents of a `timeSeries` object, `print()`, `head()`, and `tail()` methods are defined. Incidentally, the `show()` method for `timeSeries` objects only returns the count of entries as defined in `getRmetricsOption("max.print")`, which is limited to 100 rows by default. Hence, the complete time series is returned on screen by calling `print()` or by increasing `max.print` accordingly if the time series consists of more entries. The sub-setting of a time series can be conducted either by employing the `[]` operator or by means of the `window()` method. The latter requires `start` and `end` as arguments for the data window. Akin to `window()` is the `cut()` method, whereby the returned data ranges are defined by the arguments `from` and `to`. Methods for displaying a time series graphically are `plot()`, `lines()`, and `points()`. The first method has argument `plot.type` for determining whether the time series should be plotted separately (`single`) or together (`multiple`, the default). The latter two methods can be used for superimposing a time series on an existing device.

The basic methods and functions for creating and handling `timeSeries` objects have been presented. In this paragraph, methods and functions for manipulating/computing these objects are discussed. Key descriptive statistics can be calculated per column with the function `colStats()`; the function to be applied to each column is provided as argument `FUN`. Wrapper functions for the most frequently encountered descriptive statistical measures are defined as `colFoo()`, where `Foo` is a placeholder for `Sds`, `Vars`, `Skewness`, `Kurtosis`, `Maxs`, `Mins`, `Prods`, `Quantiles`, `Means`, `Stdevs`, and `Avg`s. The functions `colFoo()` are all specified with an ellipsis argument, which is passed down to the function in question. For applying a function only over certain date/time ranges the functions `fapply()` and `applySeries()` have been implemented. Aside from the argument for the function (`FUN`), both take as further arguments vectors which determine the start and end points of the data windows that are each supplied to `FUN`. The difference between these two functions is that `applySeries()` has a `by` argument that refers to a coarser periodicity and will be used if `from` and `to` are left `NULL`. A `lag()` method for lagging/leading a time series by `k` periods has been implemented. Similarly, a `diff()` method is available. Here, a positive integer value indicates the count of periods by which a series will be lagged. With respect to financial time series data a method for computing returns of a series is available as `returns()`. The user can determine whether discrete or continuous returns are computed, whether these are expressed as decimal or percentage figures, whether `NA` values should be removed prior to calculating the returns, and whether the series should be trimmed for `NA` values (first row) after the computation. Moving averages (weighted) can be computed in terms of applying a linear `filter()` to a time series. Also related to financial time series applications are the functions `drawdowns()` for determining the draw-downs of a return series, `spreads()` for determining the bid–ask spread, and `midquotes()` for the implied mid-quote between a bid and an ask series. The length of runs, as an indication of the randomness of a series, is returned by the function `runlengths()`. With respect to the creation of pseudo-uniform variables of a series, the order statistics are returned by the function `orderStatistics()`.

and the associated ranks by the function `rank()`. Cumulated column and row sums are returned by the functions `colCumsums()` and `rowCumsums()`, respectively. The latter is handy for calculating portfolio performance based measures on a per period basis, if the time series contains its constituents. With respect to the column dimension, functions for the cumulated maxima, minima, products, and returns are also available as functions `colCummaxs()`, `colCummins()`, `colCumprods()`, and `colCumreturns()`, respectively. In the last function, the user can specify whether the cumulated returns are computed for discrete or continuous returns. Finally, two `timeSeries` objects can be merged together by the `merge` and `bind` methods, and a single object can be aligned to a finer grid of date times with the functions `align()` and `alignDailySeries()`. With respect to the latter, the function `dummyDailySeries()` is also useful for creating a dummy series that can then be employed in a merge with another `timeSeries` object.

B.5 The package `zoo`

An S3-based infrastructure for handling regular and irregular time series has been implemented in the CRAN package `zoo` (see Zeileis and Grothendieck 2005). The package's name is an acronym for "Zeileis's ordered observations." A feature of this implementation is the class independence of the date/time object. The class definitions in the packages `ts`, `tseries`, and `timeSeries` required explicitly or implicitly through coercion that the date/time stamps belong to a certain class, that is, either `POSIXt` or `POSIXct` or `timeDate`. This is not the case for the S3 class `zoo` where the index can be of almost any class. Furthermore, the package bridges the gap between the most simple time series representation in the form of objects with class attribute `ts` and the more advanced features and manipulations of time series data, by defining the `zooreg` class which inherits from the `zoo` class. The former is particularly specified for regularly spaced time series. The usage of the package's facilities is detailed in a few vignettes that are shipped with it. In the following, the major functions and methods offered are presented and the reader is referred to the above sources for further details.

The generator function for creating `zoo` objects is `zoo()`. The function has three arguments: `x` for the values of the time series, `order.by` for the index vector, and the optional argument `frequency`. The important requirements for the index vector are that it can be ordered and that its entries can be matched against another index vector, where applicable. If the latter argument is left `NULL` (the default), the function will return an object with class attribute `zoo`; if specified in accordance with the provided index, an object with class attributes `c("zooreg", "zoo")` will be returned. The child class is similar in nature to the `ts` class provided in the `stats` package of the base R distribution. This kind of object can also be created by calling the function `zooreg()` with basically the same set of arguments as in `ts()`. Objects of class `zoo` consist of the object holding the time series values, that is, a vector or a matrix or a factor, and has a further attribute, namely the `index`. Objects of class `zooreg` have in addition a `frequency` attribute. Whether an object is a regular time series can be determined with the method `is.regular()`. This function has an additional

argument, `strict`, to test whether the series is regular, but has NA values or not, that is, the time stamps of the non-NA values are non-consecutive (irregular). In the following we will only refer to `zoo` objects, as all of the methods and functions presented can also be applied to `zooreg` objects.

As well as these generator functions, `zoo` objects can be created by coercion from `ts`, `irts`, `its`, `fts`, `mcmc`, `tis`, or `xts` objects.¹ Furthermore, coercion methods from `zoo` objects to either `data.frame`, `list`, `matrix`, or `vector` objects are also available.

When a `zoo` object has been created its content can be further inspected by the available `print()`, `head()`, `tail()`, `str()`, and `summary()` methods. Graphical displays of a `zoo` object can be created by calling either the `plot()` method or by superimposing a time series on an existing graphics device with the methods `lines()` or `points()`. Lattice plots of `zoo` objects can be produced by the provided `xyplot()` method. Similar to the `plot()` method, lines and points can be superimposed on an existing lattice plot with the methods `llines()` and `lpoints`, respectively. In addition, functions for tailoring the panel shape of lattice plots are also made available.

Numerous S3 methods and functions are made available for objects with these class attributes. First, the series data can be recovered by the `coredata()` method, which removes any `zoo`-related attributes from the object. The time index can be queried by employing either the `index()` or the `time()` methods. These three functions can also be used in the assignment of values to a `zoo` object. For sub-setting `zoo` objects, a `window()` method is available that can also be utilized in assigning new values to an object. The same can be achieved with the `[]` method. The start and end time points are returned by calling `start()` and `end()`, respectively.

In the package `zoo`, the handling of NA values can be accomplished by (in alphabetical order):

- `na.aggregate`: NA values are replaced by an aggregated value derived from a lower frequency grid of the series.
- `na.approx()`: NA values are replaced by means of linear interpolation. Leading NA values will be discarded.
- `na.contiguous()`: The longest sequence of non-NA values is recovered from the `zoo` object.
- `na.fill()`: NA values are replaced by user-specified values/rules.
- `na.locf()`: The last non-NA value will be carried forward as in the package `its`. Leading NA values will be discarded.
- `na.omit()`: All NA values will be removed from the object.

¹ The class `fts` is defined in the package `fts` (see Armstrong, 2014), the class `mcmc` is provided by the package `coda` (see Plummer et al. 2006), and the class `tis` is shipped with the package `tis` (see Hallman, 2015). The class `xts` will be described in Section B.6.

- `na.spline()`: NA values will be replaced by values derived from a cubic spline function.
- `na.StructTS`: NA values will be replaced by values derived from a structural time series model by means of Kalman filtering.
- `na.trim()`: Leading and trailing NA values will be removed from the object.

For aggregating a time series from a higher-frequency (say daily) to a lower-frequency series (say monthly), an `aggregate()` method is at hand, whereby the user has to provide the index for the coarser time grid of equal length to the index itself and a function that can be applied as the aggregation rule. Two `zoo` objects can be combined/joined by the provided `rbind()`, `cbind()`, and/or `merge()` methods. For lagging and leading a time series, a `lag()` method has been specified. Noteworthy here is that the argument `k` for the lag order must be specified as a negative for lagging and positive for leading the series entries by `k` periods. In addition, differenced series are returned by applying the `diff()` method. This method has as a logical argument the switch `arithmetic` for controlling whether arithmetic (the default) or geometric differences should be computed.

For applying a function on a rolling window, the method `rollapply()` is provided. The length of the moving window is set by the argument `width`; whether the resultant values of the function to be applied shall pertain to the left end, center, or right end is controlled by the argument `align`. Resulting NA values can be either filled (argument `fill`) or padded (argument `na.pad`). Further, the logical switch `by.column` determines whether the function `FUN` should be applied per column (the default) or on the whole set contained in a multivariate time series. Hence, by setting this argument to `FALSE` a moving grand mean can be computed, for instance. A partial application of the function taking NA entries into account can be achieved with the `partial` argument. Because quite often it is desired to right-align the returned values of the function, a method `rollapplyr()` has been defined, which is a wrapper of `rollapply()` whereby the `align` argument is preset to "right". In a similar vein, the methods `rollmean()`, `rollmeanr()`, `rollmax()`, `rollmaxr()`, `rollmedian()`, and `rollmedianr()` are made available with obvious meanings.

Incidentally, user-defined index classes are supported and in the package `zoo` pre-defined classes for supporting regular monthly and quarterly time indexes are already implemented as classes `yearmon` and `yearqtr`, respectively. Finally, similar to previous classes, directly reading/writing from/to a file connection can be accomplished with the functions `read.zoo()` and `write.zoo()`, which are based on the functions `read.table()` and `write.table()`, respectively.

B.6 The packages `tframe` and `xts`

So far, many possibilities for dealing with time series data in R have been presented and it is up to the user to choose the class that will suit his or her task best. However,

matters are different for package developers. They are faced with the problem that their functions and methods have to interact with these quite numerous ways time series can be created and the different classes for date/time stamps. Providing a solution to this problem is the motivation for the packages **xts** (see Ryan and Ulrich 2014) and **tframe** (see Gilbert 2015a), namely, to provide a unifying class and method infrastructure such that a developer has only to cope with these time series classes. Of course, employing the facilities of **xts** or **tframe** is not confined to this group of R users.

The package **tframe** is hosted on CRAN and R-Forge. As well as providing functions for handling time series irrespective of the representation of dates and times, further facilities for plotting and data manipulation have been implemented. The package **tframePlus** is an extension of **tframe**, in particular with respect to the second group of provided methods (see Gilbert 2015b). A classed time frame attribute **tframe** is set or extracted to an object by the function **tframe()**. Alternatively, a time frame can be superimposed on an object by either of the functions **tfSet()** or **tfraimed()**, or by means of coercion: as **.tframe()**. The existence of a **tframe** object is assessed by calling either **tframe()** or **tfraimed()**. The equality of two objects can be determined by the function **testEqual()**, and whether the time frames of two objects are the same with the function **testEqualframes()**. The classed **tframe** attribute of an object **bar** can also be transferred to another object **foo** by using **frame()** as extractor function on the former object and the same function in the assignment of the latter, that is, **tfframe(foo) <- tfframe(bar)**. The start and end periods are returned by either calling **start()** or **tfstart()** for the beginning period, or **end()** or **tfend()** for the ending period of a time frame. For multivariate time series objects, the functions **latestStart()** and **latestStartIndex()** are available. These functions return either the start date of the series where observations start at the latest time and the latter returns the column index pertinent to the series in question. Similarly, the functions **earliestEnd()**, **earliestEndIndex()**, and **latestEnd()**, **latestEndIndex()** are made available, with obvious meanings. The sample size is recovered by the function **tfSpan()** and sub-setting of a time series is accomplished by **tfwindow()**, whereby the relevant new **start** and **end** periods are specified. A **tframe** object can be expanded horizontally by means of the function **tfExpand()** and vertically, that is, by right-appending a time series to an existing **tframe** object, by the **tbind()** method. The analogue to this function with respect to the time dimension is **splice()**. Dropping of NA entries is accomplished by calling the function **trimNA()**. The names of the series can either be extracted or assigned to a **frame** object by the function **seriesNames()**. A certain series of a multiple time series object is extracted by the function **selectSeries()**. Utility functions are **tfL()** for lagging a time series, **diff()** and **difflog()** for computing (logarithmic) differences, **tfplot()** for producing time series plots, and **tfprint()** for producing a formatted print of a **tframe** object. Reading and writing a **tframe** object from/to a file connection is accomplished by the functions **tsScan()** and **tsWrite**, respectively. In the complementary package **tframePlus**, facilities for the aggregation of time series (**as.quarterly()**, **as.annually()**, and **as.weekly()**) are

made available. The aggregation method has to be provided as argument `FUN`; the default is set to compute sums of the data belonging to the lower-frequency periods. The date/time representation can be replaced by a new date/time class with the function `changeTSrepresentation()`. To apply a function to consecutive periods of a coarser time grid, the function `rollAggregate()` is made available. Cumulated sums of a time series are returned by calling `tfI()`. As well as producing time series plots with the function `tfplot()`, a routine for generating perspective plots of a multivariate time series is `tfpersp()`. Finally, `tf`rame objects can be written to a spreadsheet file (format either `xls` or `csv`) by the functions `TSwriteXLS()` or `TSwriteCSV()`, respectively.

Objects of S3 class `xts` are amended and modified versions of `zoo` objects and can be created by either calling its generator function `xts()` or by means of coercion (methods `as.xts` and `try.xts()`). The modifications implemented are that (i) a time-based index is required for `xts` objects, whereas for `zoo` objects any ordered index can be used; (ii) `xts` objects are amended by internal attributes which store the class of the original series data (`.CLASS`) and a row names attribute (`.rownames`), which takes care of the date/time information of the series; and (iii) the possibility of adding user-specified attributes. Because the `xts` class is derived from `zoo`, basically all the methods and functions presented in the previous sub-section can also be applied to `xts` objects. Hence, in the following only the facilities particular to `xts` objects will be presented.

The first group of functions to described will refer to the date/time handling of `xts` objects. The function `endpoints()` returns the index entries of a time series that adhere to the last “date/time” of a period. The periodicities supported are: microseconds, milliseconds, seconds, minutes, hours, days, weeks, months, quarters, and years. Next, to recover the first or last `n` observations of a time series the methods `first()` and `last()` are made available. A feature of these methods is that aside from a numeric value for the count of periods to return, this can also be expressed as the count of a periodicity, for example, `first(x, "1 month")` would return the time series data pertinent to the first calendar month. The index values only could then be recovered by either of the methods `index()` or `time`. Date/time stamps as `POSIXct` objects for the first or last index entry for a specified period are returned by the functions `firstof()` and `lastof()`. The index class of an `xts` object can be queried with the function `indexClass()`, and can also be employed in its assignment. Furthermore, the index can be converted to a different class through coercion by means of the function `convertIndex()`. The supported index classes are: `Date`, `POSIXct`, `chron`, `yearmon`, `yearqtr`, or `timeDate`. Testing whether the index class is supported or not can be accomplished with either `is.timeBased()` or `timeBased()`. The time zone of the index can be queried or assigned with the function `indexTZ()`. Similarly, the format of an index representation can be retrieved or modified with the function `indexFormat()`. Assessing whether a time series index is ordered can be accomplished with the function `isOrdered` (increasing/decreasing and with/without duplicate index entries). The index entries of a time series can be made unique with either of the functions `make.index.unique()` or `make.time.unique()`. This is achieved either by dropping duplicate entries or

by adding fractions of a second to a time stamp. The default increment is set to a hundredth of a second, which might be applicable for high-frequency data. For producing a sequence of date/time vectors the functions `timeBasedSeq()` or `timeBase-dRange()` can be utilized (the latter will only return a two-element vector in which the elapsed seconds since 1 January 1970 for the start and end dates/times are returned). The specification of the start, end, and periodicity for these two functions is pretty user friendly. This information is supplied to the function in the form of a character string with format equal to either `from/to/by` or `from::to::by`. Incidentally, the latter can be omitted and then the date/time increments do depend on the granularity of the provided date/time specifications for `from` and/or `to`. The class of the returned object can be controlled by means of the `retclass` argument, and the length of the returned object can be controlled by the argument `length.out`, which takes precedence if in conflict with the stated range/periodicity. An estimate for the periodicity of an `xts` object is returned by the function `periodicity()`, which is accomplished by computing the median time between observations. The count of a certain periodicity of an `xts` object is returned by the function `nfoo()`, where `foo` is a placeholder for either `seconds`, `minutes`, `hours`, `days`, `weeks`, `months`, `quarters`, or `years`.

In the second group, functions will be presented for manipulating, aggregating, and sub-setting `xts` objects. Akin to the `aggregate()` method in the package `zoo`, two sets of facilities for applying a function to date/frequency ranges have been implemented. The first are the `apply.foo()` functions, where `foo` is a placeholder for `daily`, `weekly`, `monthly`, `quarterly`, or `yearly`. Here, the function `FUN` will be applied to the time series data contained in the non-overlapping periods as defined by `foo`. The second set of functions consist of `period.apply()` and wrapper functions `period.foo()`, where `foo` is a shortcut for the functions `max`, `min`, `prod`, or `sum`. These routines return the function value pertinent to the time intervals implied by the argument `index`, and the sub-sequences are defined as `INDEX[k]` to `INDEX[k + 1]` for `k = 1 : (length(INDEX) - 1)`. Finally, the method `split.xts()` enables the user to split an `xts` object according to a certain periodicity. This method returns a `list` object of `xts` objects in which the elements are the time series data of the requested non-overlapping periods.

References

- Armstrong W. 2009 *its: Irregular Time Series*. R package version 1.1.8.
- Armstrong W. 2014 *fts: R interface to tslib (a time series library in C++)*. R package version 0.9.9.
- Chalabi Y., Mächler M., and Würtz D. 2011 Rmetrics – timeDate Package. *The R Journal* 3(1), 19–24.
- Gilbert P. 2015a *tframe: Time Frame Coding Kernel*. R package version 2015.12-1.
- Gilbert P. 2015b *tframePlus: Time Frame Coding Kernel Extensions*. R package version 2015.1-2.
- Grothendieck G. and Petzoldt T. 2004 R Help Desk: Date and time classes in R. *R News* 4(1), 29–32.

- Hallman J. 2015 *tis: Time Indexes and Time Indexed Series*. R package version 1.30.
- James D. and Hornik K. 2015 *chron: Chronological Objects which Can Handle Dates and Times*. R package version 2.3-47. S original by David James, R port by Kurt Hornik.
- Plummer M., Best N., Cowles K., and Vines K. 2006 CODA: Convergence diagnosis and output analysis for MCMC. *R News* **6**(1), 7–11.
- Ripley B. and Hornik K. 2001 Date-time classes. *R News* **1**(2), 8–11.
- Ryan J. and Ulrich J. 2014 *xts: eXtensible Time Series*. R package version 0.9-7.
- Therneau T., Lumley T., Halvorsen K., and Hornik K. 2014 *date: Functions for handling dates*. R package version 1.2-34. S original by Terry Therneau, R port by Thomas Lumley, Kjetil Halvorsen, and Kurt Hornik.
- Trapletti A. and Hornik K. 2016 *tseries: Time Series Analysis and Computational Finance*. R package version 0.10-35.
- Würtz D., Chalabi Y., and Ellis A. 2009 *A Discussion of Time Series Objects for R in Finance*. Rmetrics Ebook Series. Finance Online GmbH, Zürich.
- Würtz D., Chalabi Y., Maechler M., and Byers J. 2015a *timeDate: Rmetrics – Chronological and Calendar Objects*. R package version 3012.100.
- Würtz D., Setz T., and Chalabi Y. 2015b *timeSeries: Rmetrics – Financial Time Series Objects*. R package version 3022.101.2.
- Zeileis A. and Grothendieck G. 2005 *zoo: S3 infrastructure for regular and irregular time series*. *Journal of Statistical Software* **14**(6), 1–27.

Appendix C

Back-testing and reporting of portfolio strategies

C.1 R packages for back-testing

The following packages provide either functions or class/method definitions for conducting portfolio back-tests and analyzing the results thereof. It should be noted that the returned results of these back-tests might fall short of the user's requirements and therefore might be amended by further statistics and figures, for instance, by employing the facilities offered in the **PerformanceAnalytics** package (see Peterson and Carl 2014). Possibilities for generating tailor-made reports will be introduced in the next section.

The **stockPortfolio** package (see Diez and Christou 2012) is solely dedicated to stock portfolios and provides routines for data retrieval from Yahoo, portfolio optimization, and carrying out a back-test for a given strategy. The latter is implemented as function `testPort()`. The package is hosted on CRAN.

In the package **backtest** (see Enos et al. 2015) an S4 class `backtest` is defined with associated methods for exploring portfolio-based conjectures about the included assets. This class definition is not confined to assets belonging to a certain class, but the constituent financial instruments can be stocks, bonds, swaps, options, or currencies. Objects of this class can be generated with the function `backtest()`. The package is hosted on CRAN.

Last, but not least, since the publication of the first edition, the package **fPortfolioBacktest** (see Würtz et al. 2010) has been merged into the package **fPortfolio**. Similar to **backtest**, an S4 class `fPFOLIOBACKTEST` is defined, which contains

all of the necessary back-test information. Objects of this class are created with the function `portfolioBacktest()`; the user has control over the rebalancing periodicity, the kind of portfolio optimization, and whether a smoothing algorithm for the portfolio weights should be employed.

C.2 R facilities for reporting

Quite often the need arises to prepare summary reports of a portfolio strategy/optimization and/or evaluate the performance of a certain indicator or market risk model. Below, R packages and functions are listed that enable the user to either produce output in the form of a certain file type or to pass R objects in a format that can be digested by other applications for further processing. The reader is referred to the Omegahat project (<http://www.omegahat.net>) for additional packages that might be useful in this respect.

- Generic:
 - PDF: `Sweave()` in **utils** (see Leisch 2002); **Hmisc** (see Harrell and Dupont 2016); **r2lh** (see Genolini et al. 2011); **xtable** (see Dahl 2016).
 - HTML: **batade** (see Daisuke 2012); **HTMLUtils** (see Löcher 2015); **hwriter** (see Pau 2014); **R2HTML** (see Lecoutre 2003); **r2lh** (see Genolini et al. 2011); **shiny** (see Chang et al. 2016); **SortableHTMLTables** (see White 2012); **xtable** (see Dahl 2016).
 - XML: **Runiversal** (see Satman 2012); **XML** (see Lang 2016); **XMLSchema** (see Lang 2012).
- Platform- or application-specific:
 - MS Windows: **RDCOMClient** (see Lang 2007); **RDCOMServer** (see Lang 2005).
 - MS Office: **R2PPT** (see Jones 2012); **excel.link** (see Demin 2016); **xlsReadWrite** (see Suter 2011); **XLConnect** (see Mirai Solutions GmbH 2015); **xlsx** (see Dragulescu 2014).
 - OpenOffice: **ODB** (see Mareschal 2012); **odfWeave** (see Kuhn et al. 2014).

C.3 Interfacing with databases

There are quite a few R packages available that allow the user to import and export data from a (R)DBMS [(relational) database management system]. A typical work flow would then consist of importing the data sample from a database, executing the risk and/or portfolio optimization computations, and exporting the results back into the database for further processing. The following list describes the R packages hosted on CRAN that assist with this process. In addition to the packages' documentation, the reader is referred to the R manual *R Data Import/Export* for

further information. A special interest group (SIG) email list, **R-sig-DB**, dedicated to interfacing with databases from **R** is also available. Subscriptions to this list can be established via <https://stat.ethz.ch/mailman/listinfo/r-sig-db>.

- DBMS-specific (in alphabetical order):
 - Berkeley: **RBerkeley** (see Ryan and Rudis 2015).
 - H2: **RH2** (see Grothendieck and Mueller 2014).
 - Mongo: **RMongo** (see Chheng 2013); **rmongodb** (see MongoDB and Schmidberger 2014).
 - Oracle: **ROracle** (see Mukhin et al. 2016).
 - PostgreSQL: **RPostgreSQL** (see Conway et al. 2016); **rpg** (see Keitt 2015).
 - SQL: **RMySQL** (see Ooms et al. 2016); **RSQLite** (see Wickham et al. 2014); **dbConnect** (see Kurkiewicz et al. 2012).
- Generic (in alphabetical order):
 - Generic database interfaces: **DBI** (see R Special Interest Group on Databases 2016).
 - Java API: **RJDBC** (see Urbanek 2014).
 - ODBC: **RODBC** (see Ripley and Lapsley 2016).

References

- Chang W., Cheng J., Allaire J., Xie Y., and McPherson J. 2016 *shiny: Web Application Framework for R*. R package version 0.13.2.
- Chheng T. 2013 *RMongo: MongoDB Client for R*. R package version 0.0.25.
- Conway J., Eddelbuettel D., Nishiyama T., Prayaga S., and Tiffin N. 2016 *RPostgreSQL: R interface to the PostgreSQL database system*. R package version 0.4-1.
- Dahl D. 2016 *xtable: Export tables to LaTeX or HTML*. R package version 1.8-2.
- Daisuke I. 2012 *batade: HTML reports and so on*. R package version 0.1.
- Demin G. 2016 *excel.link: Convenient Data Exchange with Microsoft Excel*. R package version 0.9.4.
- Diez D. and Christou N. 2012 *stockPortfolio: Build stock models and analyze stock portfolios*. R package version 1.2.
- Dragulescu A. 2014 *xlsx: Read, write, format Excel 2007 and Excel 97/2000/XP/2003 files*. R package version 0.5.7.
- Enos J., Kane D., Campbell K., Gerlanc D., Schwartz A., Suo D., Colin A., , and Zhao L. 2015 *backtest: Exploring portfolio-based conjectures about financial instruments*. R package version 0.3-4.

- Genolini C., Desgraupes B., and Franca L. 2011 *r2lh: R to LaTeX and HTML*. R package version 0.7.
- Grothendieck G. and Mueller T. 2014 *RH2: DBI/RJDBC interface to H2 Database*. R package version 0.2.3.
- Harrell F. and Dupont C. 2016 *Hmisc: Harrell Miscellaneous*. R package version 3.17-4.
- Jones W. 2012 *R2PPT: Simple R Interface to Microsoft PowerPoint using rcom or RDCOM-Client*. R package version 2.1.
- Keitt T. H. 2015 *rpg: Easy Interface to Advanced PostgreSQL Features*. R package version 1.4.
- Kuhn M., Weston S., Coulter N., Lenon P., and Otles Z. 2014 *odfWeave: SWeave processing of Open Document Format (ODF) files*. R package version 0.8.4.
- Kurkiewicz D., Hofmann H., and Genschel U. 2012 *dbConnect: Provides a graphical user interface to connect with databases that use MySQL*. R package version 1.0.
- Lang D. 2005 *R-DCOM object server*. R package version 0.6-1.
- Lang D. 2007 *RDCOMClient: R-DCOM client*. R package version 0.93-0.
- Lang D. 2012 *XMLSchema: R facilities to read XML schema*. R package version 0.7-0.
- Lang D. 2016 *XML: Tools for parsing and generating XML within R and S-Plus*. R package version 3.98-1.4.
- Lecoutre E. 2003 The R2HTML package. *R News* 3(3), 33–36.
- Leisch F. 2002 Dynamic generation of statistical reports using literate data analysis. In *Compstat 2002 - Proceedings in Computational Statistics* (ed. Härdle W. and Rönz B.), pp. 575–580. Physika Verlag, Heidelberg.
- Löcher M. 2015 *HTMLUtils: Facilitates Automated HTML Report Creation*. R package version 0.1.7.
- Mareschal S. 2012 *ODB: Open Document Databases (.odb) management*. R package version 1.1.1.
- Mirai Solutions GmbH 2015 *XLConnect: Excel Connector for R*. R package version 0.2-11.
- MongoDB and Schmidberger M. 2014 *rmongodb: R-MongoDB driver*. R package version 1.8.0.
- Mukhin D., James D., and Luciani J. 2016 *ROracle: OCI based Oracle database interface for R*. R package version 1.2-2.
- Ooms J., James D., DebRoy S., Wickham H., and Horner J. 2016 *RMySQL: Database Interface and 'MySQL' Driver for R*. R package version 0.10.9.
- Pau G. 2014 *hwriter: HTML Writer – Outputs R objects in HTML format*. R package version 1.3.2.
- Peterson B. and Carl P. 2014 *PerformanceAnalytics: Econometric tools for performance and risk analysis*. R package version 1.4.3541.
- R Special Interest Group on Databases 2016 *DBI: R Database Interface*. R package version 0.4-1.
- Ripley B. and Lapsley M. 2016 *RODBC: ODBC Database Access*. R package version 1.3-13.
- Ryan J. and Rudis B. 2015 *RBerkeley: Oracle 'BerkeleyDB' Interface for R*. R package version 0.7-5.
- Satman M. 2012 *Runiversal: Runiversal – Package for converting R objects to Java variables and XML*. R package version 1.0.2.
- Suter H.-P. 2011 *xlsReadWrite: Read and write Excel files (.xls)*. R package version 1.5.4.

- Urbanek S. 2014 *RJDBC*: Provides access to databases through the JDBC interface. R package version 0.2-5.
- White J. 2012 *SortableHTMLTables*: Turns a data frame into an HTML file containing a sortable table. R package version 0.1-3.
- Wickham H., James D., and Falcon S. 2014 *RSQLite*: SQLite Interface for R. R package version 1.0.0.
- Würtz D., Chalabi Y., Chen W., and Ellis A. 2010 *Portfolio Optimization with R/Rmetrics*. Rmetrics Association & Finance Online, www.rmetrics.org. R package version 2110.4.

Appendix D

Technicalities

This book was typeset in **LaTeX**. In addition to the publisher's style file, the following **LaTeX** packages have been used (in alphabetical order): amsfonts, amsmath, amssymb, booktabs, float, listings, longtable, natbib, rotfloat, tikz, url, and xspace. The subject index was generated with the program `makeindex` and the bibliography with **BibTeX**. The program `aspell` was employed for spell-checking. `Emacs` was used as the text editor, with the LISP modules `ESS` and `AUCTeX`. The processing of all the files to create the book was accomplished by the `make` program, and `Subversion` was employed as a source control management system.

All **R** code examples have been processed as **Sweave** files. Therefore, the proper working of the **R** commands is guaranteed. In the `.Rprofile` file the seed for generating random numbers has been set to `set.seed = 123456` and as a random number generator **R**'s default setting was employed, so random numbers have been generated according to the Mersenne Twister algorithm. Where possible, the results are exhibited as tables by making use of the function `latex()` contained in the contributed package **Hmisc** (see Harrell and Dupont 2016). The examples have been processed under **R** version 3.3.0 (2016-05-03) on an x86-64 PC with Linux as the operating system and kernel 7x64. Linux is a registered trademark of Linus Torvalds (Helsinki, Finland), the original author of the Linux kernel.

Reference

Harrell F. and Dupont C. 2016 *Hmisc: Harrell Miscellaneous*. R package version 3.17-4.

Index

Page numbers in *italics* indicate figures; page numbers in **bold** indicate tables and listings

- accept–reject sampling, 344, 346–347, 346, 351
- ACF *see* autocorrelation function
- adaptMCMC** package, 354–355, 366–369
- Akaike information criterion (AIC), 69, 76–77, 279, 302–303
- APARCH model, 119–120, 122–123
- AR *see* autoregressive
- ARCH *see* autocorrelated conditional heteroscedastic
- Archimedean copulae, 140–142, 146–148, 155
- ARFIMA time series process, 301–302
- ARIMA time series process, 301–302, 328
- ARMA time series process, 122–123, 279–281, 297–302
- asymmetric power ARCH model, 119–120, 122–123
- autocorrelated conditional heteroscedastic (ARCH) models, 116–120, 118
- autocorrelation function (ACF) dependence, 154–155, 155 extreme value theory, 111–114
- financial market data, 31
- probabilistic utility, 348, 352, 360, 362
- tactical asset allocation, 279, 301–302
- autoregressive (AR) time series process, 275–277, 300–302
- average draw-down (AvDD), 239–241, 255–260
- bayesGARCH** package, 120–121
- Bayesian estimation dependence, 138 extreme value theory, 95–96
- tactical asset allocation, 289–292, 303–304
- volatility, 120–121
- BCC *see* budgeted CVaR concentration
- Beveridge–Nelson representation, 288–289
- Black–Litterman (BL) model, 142–144, 274, 289–292, 295–297, 307–314
- BLCop** package, 142–144, 295–297, 310–311, 314

- block maxima method, 89, 90–92, 91, 103–109, **104**, **105**
- bmk** package, 358–359
- box-plots, 186–187, **187**, 313, **316**
- Box–Jenkins model, 279, 302
- budgeted CVaR concentration (BCC), 204, 224–225
- capital asset pricing model (CAPM), 290
- capital market line (CML), 50, 233
- CAPM *see* capital asset pricing model
- Cauchy copulae, 147
- ccp** package, 183–186, 190–195, 207
- ccgarch** package, 121–122
- CDaR *see* conditional draw-down at risk
- central limit theorem (CLT), 93
- CI *see* confidence intervals
- Clayton copulae, 141–142, 155–157, **156**
- CLI *see* command line interface
- CLT *see* central limit theorem
- clustering, **113**, **114**, 117–118, 268–271
- CML *see* capital market line
- coda** package, 359–362
- coherent measures of risk, 43–44
- co-integration analysis, 307–308, **308**
- command line interface (CLI), 10–11
- co-monotonicity, 135, 140
- complexity, 14–15
- concordance, 137–138
- conditional draw-down at risk (CDaR), 239–241, 243–244, 250–251, 255–265, **261**
- conditional value at risk (CVaR)
- diversification of risks, 201–204, 211, 222–225
- risk-optimal portfolios, 234–238, 241, 250–255, **252**, **254**, 255
- conditional variance, 117–119
- confidence intervals (CI), 171
- confidence levels
- risk-optimal portfolios, 230–233, 236–238, 250, 255
- tactical asset allocation, 291–292
- constant proportion portfolio insurance (CPPI), 326–327
- convergence diagnostics, 359–361
- COP *see* copula opinion pooling
- copulae
- classification, 139–142, **143**
- concepts and definitions, 136
- correlations and dependence, 137–138
- dependence, 136–157
- diversification of risks, 205–207, 217
- empirical applications, 148–157
- GARCH–copula model, 148–155, **152**, **153**
- mixed copula approaches, 155–157, **156**
- R language packages, 142–148, 152–153, 156–157
- robust portfolio optimization, 180–181
- tactical asset allocation, 292–297
- copula opinion pooling (COP) model, 292–297, 313–318, **314**, **315**, **317–318**
- copula** package, 144–146, 180–181
- Cornish–Fisher extension, 40–41, 203, 250
- correlation coefficients
- dependence, 133–135, 137–138, 145–148
- diversification of risks, 204, 206
- counter-monotonicity, 135, 140
- covariance-stationary processes, 275–281
- covRobust** package, 174
- CPPI *see* constant proportion portfolio insurance
- cross-correlation, 32–35, **34**
- CVaR *see* conditional value at risk

- daily earnings at risk, 39
- data-generating processes (DGP), 180–186
- DCC *see* dynamic conditional correlation
- DE *see* differential evolution
- density functions
- dependence, 139, 144–145, 151
 - extreme value theory, 94–97
 - return distributions, 58–62, 59, 70–71, 76
 - tactical asset allocation, 276–277, 315, 318
- DEoptim** package, 207–209
- DEoptimR** package, 207–209
- dependence, 133–159
- concepts and definitions, 133
 - concordance, 137–138
 - copulae, 136–157
 - correlation and distributions, 133–135
 - discordance, 139–140
 - diversification of risks, 199, 204–207, 210–221
 - empirical applications of copulae, 148–157
 - GARCH–copula model, 148–155, **152, 153**
 - independence, 139–141, 145–146, 361–362
 - mixed copula approaches, 155–157, **156**
 - probabilistic utility, 361–362
 - R language packages, 142–148, 152–153, 156–157
 - tail dependencies, 138–139, 141, 145, 199, 204–207, 210–221, **218**
- dependograms, 145–146
- DGP *see* data-generating processes
- differential evolution (DE) algorithm, 207–210
- discordance, 139–140
- discrete loss distribution, 235–236
- distribution functions
- dependence, 133–135, 138–141, 144–151
- extreme value theory, 90–96
- probabilistic utility, 344–351
- robust portfolio optimization, 187
- tactical asset allocation, 293–295, 313–318
- volatility, 117, 122–123
- see also* return distributions
- diversification of risks, 198–227
- concepts and definitions, 198–199
 - diversification ratio, 199–200
 - empirical applications, 212–225, **213**
- equal-risk contribution portfolios, 201–204, 212–216
- limiting contributions to expected shortfall, 221–225
- marginal risk contribution portfolios, 203, 207, 212–215, **214–215**
- most-diversified portfolio, 199–201
- risk contribution constrained portfolios, 201–204
- risk-optimal portfolios, 260, 268
- R language packages, 207–212
- tail-dependent portfolios, 199, 204–207, 210–221
- diversification ratio (DR), 199–200
- DR *see* diversification ratio
- draw-down portfolios, 238–241, 243–244, 250–251, 254–265, **256, 257, 264**
- dse** package, 297–300
- dynamic conditional correlation (DCC), 127
- Eclipse IDE, 11
- EDA *see* exploratory data analysis
- Edgeworth expansion, 41
- efficient frontier of mean-variance, 169, 172, 190–195
- EGARCH model, 119
- elliptical copulae, 146–147

- elliptical uncertainty, 190–195, **190, 192**
- Emacs IDE, 11
- empirical mean-variance portfolios, 50–52
- entropy pooling (EP) model, 292, 293–295, 304, 318–324, **320–322**
- equal-risk contribution (ERC) portfolio, 201–204, 212–216, 214–215, 223–225, **223**, 268–269
- equity indexes, 307–308
- ERC *see* equal-risk contribution
- ES *see* expected shortfall
- ESS IDE, 11
- estimation error, 51–52
- evdbayes** package, 95–96
- evd** package, 94–95
- evir** package, 30–31, 96–98, 104, 106
- EVT *see* extreme value theory
- excess returns, 189–190, **189**, 290
- expected shortfall (ES), 39–44
- diversification of risks, 221–225, **223**
- extreme value theory, 93–94, 101–102, 111
- return distributions, 63, 77–80, **78, 80**
- risk-optimal portfolios, 228, 234, 237–238, 250, 258–260, 265
- tactical asset allocation, 331
- volatility, 130
- exploratory data analysis (EDA), 96, 99–100, 103
- exponential GARCH model, 119
- extRemes** package, 98–99
- extreme value theory (EVT), 89–115
- block maxima method, 89, 90–92, 91, 103–109, **104, 105**
- concepts and definitions, 89–90
- copulae, 146–147
- diversification of risks, 206–207
- empirical applications, 103–114
- methods and models, 89, 90–94
- peaks-over-threshold method, 89, 92–95, 97, 110–114, **110**
- Poisson processes, 89, 97–98
- R language packages, 89, 94–103
- robust portfolio optimization, 163–164, 174
- r*th largest order models, 91–92, 107–109, **108, 109**
- fArma** package, 300–301
- fBasics** package, 30–31, 66–67, 74–77, 84–86, 110–114
- fCopulae** package, 146–147
- feasible portfolios, 48–49
- fExtremes** package, 99–101, 110–114
- fGarch** package, 122, 152–153
- FIML *see* full-information maximum likelihood
- financial crises, 3
- financial market data, 29–36
- extreme value theory, 92–94
- implications for risk models, 35–36
- multivariate series, 32–35
- quantile–quantile plots, 31–32
- return distributions, 80–86
- rolling correlations, 33–35, 35
- stylized facts for financial market returns, 29–35, 80–82
- univariate series, 29–32
- forecasting
- diversification of risks, 204
- tactical asset allocation, 274, 280–281, 284, 300–302, 308–310, 313–322, 327–329, **329**
- volatility, 126, 129–130
- forecast** package, 301–302
- fPortfolio** package, 174–175, 186
- risk-optimal portfolios, 241–243, 251, 260–261, 265–268
- tactical asset allocation, 296–297, 310–311

- FRAPO** package, 4, **23**
 data sets, 23–24
 dependence, 156–157
 diversification of risks, 210–212, 217, 222
 installation, 22–23
 portfolio optimization, 24–28
 probabilistic utility, 366
 return distributions, 82–86
 risk-optimal portfolios, 243–245, 251, 255, 260–261, 266–268
 R language, 22–28
 robust portfolio optimization, 186
 tactical asset allocation, 319, **327**, 328
 Fréchet distribution, 90–91, 109
 Fréchet–Hoeffding bounds, 139
 full-information maximum likelihood (FIML), 284
- GARCH models, 118–130, **128**
 GARCH–copula model, 148–155, **152**, **153**
 Gauss copulae, 140–141
 Gauss–Seidel algorithm, 284
 Gelman and Rubin’s convergence diagnostic, 360
 generalized extreme value (GEV) distribution, 91–93, 96, 98, 100–109
 generalized hyperbolic distribution (GHD), 57–60, 59, 66–71, 74–82, **75**, **76**, **78**
GeneralizedHyperbolic package, 67–69
 generalized inverse Gaussian (GIG) distribution, 58, 67–68, 70
 generalized lambda distribution (GLD), 57, 60–66, **61**, **61**, 63, 71–74, 82–86, **83**
 generalized Pareto distribution (GPD), 92–93, 95–102, 111–114, 112
 GEV *see* generalized extreme value
- GEVStableGarch** package, 122–123
 Geweke’s convergence diagnostic, 361
GHD *see* generalized hyperbolic distribution
ghd package, 80–82
ghyp package, 69–70
GIG *see* generalized inverse Gaussian
GLD *see* generalized lambda distribution
 global minimum variance (GMV) portfolio
 diversification of risks, 199, 201, 211–216, 223–225, **223**
 modern portfolio theory, 48–49, 48
 risk-optimal portfolios, 253–265, **261**
 R language, 25–27
GLPK *see* GNU Linear Programming Kit
glpkAPI package, 245–247
GMV *see* global minimum variance
 GNU, 7, 11
 GNU Linear Programming Kit (GLPK), 241, 243–247
gogarch package, 123–124, **124**
 goodness-of-fit, 65–66, 68, 79, 145–146
GPD *see* generalized Pareto distribution
 graphical user interfaces (GUI), 10–12, 102–103
 Gumbel copulae, 141–142, 147–148, 155–157, **156**
 Gumbel distribution, 90–91, 100
gumbel package, 147–148
- Hannan–Quinn information criteria, 279, 303
 hedge-fund type strategies, 52
 Heidelberger and Welch’s convergence diagnostic, 361
 Herfindahl–Hirschmann index, 200
 highest posterior density (HPD), 361
 histogram-based estimation, 65

- HPD *see* highest posterior density
- hyperbolic (HYP) distribution, 57–59, 67–68, 70, 74–82, **78, 81**
- IDE *see* integrated development environments
- IFM *see* inference for margins
- iid *see* independent and identically distributed
- Imomco** package, 82–84
- impulse response functions (IRF), 286
- in2extRemes** package, 98–99
- independence, 139–141, 145–146, 361–362
- independence sampler, 351–353
- independent and identically distributed (iid) processes
- dependence, 149
- financial market data, 29, 35
- risk measures, 39
- inference for margins (IFM), 142, 150–151
- integrated development environments (IDE), 10–12
- integration analysis, 307–308, **308**
- IRF *see* impulse response functions
- ismev** package, 101, 104–106
- JGR IDE, 11
- Joe–Clayton copulae, 155
- Kendall’s tau rank correlation, 135, 137–138, 142, 145–148, 154
- kurtosis
- return distributions, 62, 69, 80, 86
 - risk measures, 40–41
 - robust portfolio optimization, 187
- Lagrangian multiplier, 171
- least squares (LS) method, 165–166, 284
- leveraged portfolio, 15–17
- lgarch** package, 123–125
- likelihood-ratio test, 279–280
- linear programming (LP), 237, 241, 243–249, 330–331, **331**
- linprog** package, 247–248
- log-GARCH models, 123–125
- log-likelihood
- extreme value theory, 93, 95, 101, 106, 114
 - return distributions, 67–72, 77
 - tactical asset allocation, 276–277
 - volatility, 127–128
- long-only portfolio
- diversification of risks, 202–203
 - modern portfolio theory, 52
 - R language, 15–17
- loss distributions, 38–40, 40
- lpSolveAPI** package, 248–249
- lpSolve** package, 248–249
- LS *see* least squares
- MA *see* moving average
- MAD *see* median absolute deviation
- Mahalanobis distances, 166, 177–179
- marginal risk contributions (MRC), 203, 207, 212–215
- market price risk model, 329–330, **330**
- Markov chain Monte Carlo (MCMC)
- accept–reject sampling, 344, 346–347, **346, 351**
 - concepts and definitions, 342–343
 - convergence diagnostics, 359–361
 - empirical application, 362–375
 - exemplary utility function, 362–366, **364**
 - extreme value theory, 95–96
 - integration methods for probabilistic utility function, 343–345, **344, 345**
 - Markov chains, 347–349, **349, 350**
 - maximum expected utility, 339, 366–369, **367–368**
 - Metropolis–Hastings algorithm, 349–354, 352, **353**
 - Monte Carlo approaches, 343–347

- probabilistic utility, 339, 342–369
 progression of estimates, 344
 R language packages, 354–362
 simulation of Markov chains, 349
 tactical asset allocation, 303
 transition kernels, 347–348
 volatility, 121
- Markowitz, Harry, 3–5, 46–50
- MASS** package, 175–176, 371
- Mathematical Programming System (MPS), 246–249
- maximum draw-down (MaxDD), 239–241, 243–244, 251, 255–260
- maximum expected utility (MEU), 339, 366–375, **367–368**
- maximum likelihood (ML) method
 dependence, 145
 extreme value theory, 93, 95, 101, 104
 return distributions, 60, 66–67
 robust portfolio optimization, 164–166, 168, 170
 tactical asset allocation, 276–277
 volatility, 120
- maximum product spacing, 66
- maximum Sharpe ratio (MSR)
 portfolio, 48–50, 48, 203, 311–312, **311–312**, 316–317, 323–326
- MCC** *see* minimum CVaR concentration
- MCD** *see* minimum covariance determinant
- MCMC** *see* Markov chain Monte Carlo
- mcmc** package, 355–356, 366–369
- MCMCpack** package, 356, 366–369
- MDP** *see* most-diversified portfolio
- mean residual life (MRL) plot, 93, 101, 111–112
- mean-variance portfolio
 modern portfolio theory, 48–52
 risk-optimal portfolios, 228–229, 232–235, 237–238
- robust portfolio optimization, 168–174, 192–195, **192–194**, 195
- mean-VaR portfolio, 228, 229–234, 231, 234
- measuring risks *see* risk measures
- median absolute deviation (MAD), 175
- mES *see* modified expected shortfall
- M-estimators, 165–166, 176–178, 180–190
- Metropolis–Hastings (MH) algorithm, 349–354, 352, **353**
- MEU *see* maximum expected utility
- MH *see* Metropolis–Hastings
- Michaud-type simulation, 369–375
- MILP *see* mixed integer linear programming
- minimum covariance determinant (MCD) estimator, 166–167, 177–178, 180–190
- minimum CVaR concentration (MCC), 204, 222–225
- minimum tail-dependent (MTD) portfolio, 210–216, 220
- minimum-variance portfolio (MVP), 181–188, **184**, 232–234, 238, 251–255, **252**, 255, 268–269
- minimum-VaR portfolios, 232
- minimum volume ellipsoid (MVE) estimator, 166–167, 178, 180–190
- min–max approach, 170
- mixed integer linear programming (MILP), 245, 248–249
- ML *see* maximum likelihood
- MM-estimators, 165–166, 178, 180–190
- modified expected shortfall (mES), 41, 43
- modified value at risk (mVaR), 40–43, 41
- moment matching, 64–65
- monotonicity, 43, 135, 139–140

- Monte Carlo analysis
 dependence, 144
 modern portfolio theory, 51
 risk measures, 43
 tactical asset allocation, 293–294, 303
see also Markov chain Monte Carlo simulation
 most-diversified portfolio (MDP), 199–201, 204, 212–216
 moving average (MA) time series process, 277–278, 300–302
 MPS *see* Mathematical Programming System
 MRC *see* marginal risk contributions
 MRL *see* mean residual life
MSBVAR package, 302–304
 MSR *see* maximum Sharpe ratio
 MTD *see* minimum tail-dependent multivariate distribution
 dependence, 134–135, 144, 149
 diversification of risks, 203–205
 extreme value theory, 94–95
 financial market data, 32–35
 return distributions, 69
 risk measures, 43
 risk-optimal portfolios, 229, 235–236, 250–251
 robust portfolio optimization, 171, 175–176, 180–181
 tactical asset allocation, 281–289
 time series models, 281–289
 volatility, 121–127
 mVaR *see* modified value at risk
 MVE *see* minimum volume ellipsoid
 MVP *see* minimum-variance portfolio
 nearest neighbor (NN) procedure, 174
 NIG *see* normal inverse Gaussian
nloptr package, 304
 NN *see* nearest neighbor
 normal inverse Gaussian (NIG)
 distribution, 57, 59, 66–67, 70, 74–80, **78**
 object-based programming, 13–14
 object-oriented programming, 12–13, 19–22
OBPI *see* option-based portfolio insurance
OGK *see* orthogonalized Gnanadesikan–Kettering
OLS *see* ordinary least squares
 one-step-ahead forecasts, 308–310, 318–319
 opinion forming framework, 143
 option-based portfolio insurance (OBPI), 326–327
 ordinary least squares (OLS) method, 276, 284, 304
 orthogonalized Gnanadesikan–Kettering (OGK) estimator, 167–168, 176–178, 180–190
 outliers *see* extreme value theory
PACF *see* partial autocorrelation function
 parametric inference for margins, 151
 partial autocorrelation function (PACF), 31, 111–114, 279, 302
 peaks-over-threshold (POT) method, 89, 92–95, 97, 100–101, 110–114, **110**
 Pearson’s correlation
 dependence, 133–135, 137–138, 145–148
 diversification of risks, 204, 206
 percentile-based estimation, 65
PerformanceAnalytics package, 249–251, 255
 Poisson processes, 89, 97–98
PortfolioAnalytics package, 211–212, 216, 222, 304
 portfolio back-testing
 return distributions, 84, 84
 risk-optimal portfolios, 251–253, 260–265, **261–262, 265**
 R language, 27

- robust portfolio optimization, 186–195, **186–189**
- tactical asset allocation, 312, 320, 322–324, **324**
- volatility, 126, 128–130
- portfolio optimization
 - concepts and definitions, 3–5
 - dependence, 148–157
 - diversification of risks, 198–227
 - empirical mean-variance portfolios, 50–52
 - estimation error, 51–52
 - Markowitz, Harry, 3–5, 46–50
 - modern portfolio theory, 46–53
 - probabilistic utility, 339–377
 - risk measures, 42–44
 - risk-optimal portfolios, 228–273
 - R language, 24–28
 - robust portfolio optimization, 163–197
 - tactical asset allocation, 274–338
 - utility, 49–50
- portfolio weight class, 15–22
- PortSol class, 25–26
- positive homogeneity, 43–44
- posterior distribution, 293–295, 311–312, 315–319, **319**
- POT *see* peaks-over-threshold
- PP *see* probability–probability
- prior distribution, 311–312, 315–319, **319**
- probabilistic utility (PU), 339–377
 - accept–reject sampling, 344, 346–347, **346**, 351
 - concepts and definitions, 339–342, **341**, **342**
 - convergence diagnostics, 359–361
 - empirical application, 362–375
 - exemplary utility function, 362–366, **364**
- Markov chain Monte Carlo
 - simulation, 339, 342–369
- Markov chains, 347–349, **349**, 350
- maximum expected utility, 339, 366–375, **367–368**
- Metropolis–Hastings algorithm, 349–354, 352, **353**
- Monte Carlo approaches, 343–347
- R language packages, 354–362
- simulation of asset allocations, 369–375, **370**, **372–374**, 373
- simulation of Markov chains, **349**
 - transition kernels, 347–348
- probability–probability (PP) plots, 67–69, 104–106, 108–109, 111
- protection strategies
 - analysis of results, 333–334, **333**
 - concepts and definitions, 324–327
 - data preparation, 328
 - forecasting model, 328–329, **329**
 - formulation of linear program, 330–331, **331**
 - market price risk model, 329–330, **330**
 - portfolio simulation, 332–333, **332**
 - tactical asset allocation, 324–334
- QQ *see* quantile–quantile
- QRM package, 70, 101–102, 148, 152–153, 156–157
- quadprog package, 296–297
- quantile functions, 94–96
- quantile–quantile (QQ) plots
 - dependence, 153–154, **154**
 - extreme value theory, 98–100, 104–106, 108–109, 111
- financial market data, 31–32
- return distributions, 67–69, 74–76, **76**
- quasi-Newton optimizer, 127
- Raftery and Lewis test, 361–362
- random walk sampler, 351–353
- RC *see* reference class
- RccpDE package, 209–210
- reference class (RC), 14, 19–22
- Renext/RenextGUI package, 102–103

- return distributions, 57–88
 concepts and definitions, 57
 density functions, 58–62, 70–71, 76
 financial market data, 80–86
 generalized hyperbolic distribution, 57–60, 59, 66–71, 74–82, **75, 76, 78**
 generalized lambda distribution, 57, 60–66, 61, **61**, 63, 71–74, 82–86, **83**
 goodness-of-fit, 65–66, 68, 79
 histogram-based estimation, 65
 hyperbolic distribution, 57–59, 67–68, 70, 74–82
 maximum likelihood method, 60, 66–67
 maximum product spacing, 66
 moment matching, 64–65
 normal inverse Gaussian distribution, 57, 59, 66–67, 70, 74–80
 percentile-based estimation, 65
 risk measures, 39–40, 77–80, **78**
 risk modelling, 74–86
 R packages for GHD, 66–71
 R packages for GLD, 71–74, 82–84
 shape plots, 63, 80–82, **81**, 82, 84–86, **85**, 86
 stylized facts, 80–82
 valid parameter constellations in GLD, 61–62
 VaR for a single stock, 82–84, **83**
 reverse optimization, 290
Rglpk package, 243–247, 331
 risk aversion
 modern portfolio theory, 49–50
 probabilistic utility, 340
 robust portfolio optimization, 172
see also diversification of risks
 risk diversification *see* diversification of risks
 risk-free assets, 232–233
 risk measures, 37–45
coherent measures of risk, 43–44
 concept and definitions, 37
 dependence, 149, 152
 diversification of risks, 201–204, 211, 221–225
 expected shortfall, 39–44
 extreme value theory, 93–94, 101–102, 111
 loss/returns distributions, 38–40
 portfolio optimization, 42–44
 return distributions, 77–80, **78**
 synopsis of risk measures, 37–41
 tactical asset allocation, 331
 value at risk, 38–44
 volatility, 126, 130
see also risk-optimal portfolios
 risk modelling
 concepts and definitions, 3–5
 dependence, 133–159
 extreme value theory, 89–115
 financial market data, 35–36
 return distributions, 74–86
 volatility, 116–132
 risk-optimal portfolios, 228–273
 back-test comparison for stock portfolio, 260–265
 boundaries of mean-VaR portfolios, 230–232
 clustering, 268–271
 concepts and definitions, 228–229
 confidence levels, 230–233, 236–238, 250, 255
 discrete loss distribution, 235–236
 diversification of risks, 260, 268
 empirical applications, 251–271
 linear programming, 237, 241, 243–249
 mean-variance portfolios, 228–229, 232–235, 237–238
 mean-VaR portfolios, 228, 229–234, 231, 234
 minimum-variance portfolios, 232–234, 238, 251–255, **252**, 255, 268–269
 optimal CVaR portfolios, 234–238, 241, 251–255, **252**, 254, 255

- optimal draw-down portfolios, 238–241, 243–244, 250–251, 254–265, **256**, 257
- portfolio back-testing, 251–253, 260–265, **261–262**, **265**
- risk-free assets, 232–233
- risk surface plots, 242–243, 265–271, **267**, **269**, **270**, **271**
- R language packages, 241–251
- tangency mean-VaR portfolio, 233–234
- risk/return points, 48–49
- risk surface plots, 242–243, 265–271, **267**, **269**, **270**, **271**
- R language, 6–28
- classes, methods, and functions, 12–22
 - concepts and definitions, 4–5
 - conferences and workshops, 10
 - dependence, 142–148, 152–153, 156–157
 - diversification of risks, 207–212
 - extreme value theory, 89, 94–103
 - FRAPO package, 22–28
 - getting help, 7–10
 - help facilities within R, 8–10
 - mailing lists, 9–10
 - manuals and articles, 7–8
 - Markov chain Monte Carlo simulation, 354–362
 - origin and development, 6–7
 - packages for GHD, 66–71
 - packages for GLD, 71–74, 82–84
 - portfolio optimization, 24–28
 - probabilistic utility, 354–362
 - risk-optimal portfolios, 241–251
 - robust portfolio optimization, 174–195
 - tactical asset allocation, 295–307
 - volatility, 120–128
 - working with R, 10–12
- rmgarch** package, 125–127
- robustbase** package, 176
- robust** package, 176–178
- robust portfolio optimization, 163–197
- classical statistics, 180–190
 - concepts and definitions, 163
 - data-generating processes, 180–186, **181–182**
 - efficient frontier of mean-variance, 169, 172, 190–195
 - empirical applications, 180–195
 - extreme value theory, 163–164, 174
 - M- and MM- estimators, 165–166, 176–178, 180–190
 - OGK estimator, 167–168, 176–178, 180–190
 - portfolio back-testing, 186–195, **186–189**
 - probabilistic utility, 340
 - R language packages, 174–195
 - robust optimization, 168–174, 190–195
 - robust scaling-derived estimators, 166–167, 177–178, 180–190
 - robust statistics, 164–168, 180–190
 - second-order cone program, 173, 179–180, 190–195
 - selected robust estimators, 165–168
 - Stahel–Donoho estimator, 167, 177–178, 180–190
 - trimming, 164
 - uncertainty sets and problem formulation, 168–174, 190–195
 - winsorizing, 164
- rolling correlations, 33–35, 35
- rolling window optimization, **188**
- rrcov** package, 178–179, 181–183
- Rsoep** package, 179–180
- rstan** package, 357–358, 363–369, **365**
- RStudio** package, 11
- Rsymphony** package, 249
- rth* largest order models, 91–92, 107–109, **108**, **109**
- rugarch** package, 125–127

- S3 framework, 14–15, 22
 S4 framework, 14–19, 21–22, 25
 Schwarz information criterion, 279, 302–303
SDE *see* Stahel–Donoho estimator
 second-order cone program (SOCP), 173, 179–180, 190–195
 seemingly unrelated regression (SUR), 303
 selectMethods function, 27
 S-estimator, 166–167, 178, 180–190
 setAs function, 14, 19
 setClass function, 14–16, 19
 setGeneric function, 14, 18–19
 setMethods function, 14–15, 17–19, 26
 setRefClass function, 20–22
 setValidity function, 14–16
 shape parameter, 93
 shape plots, 63, 63, 80–82, 81, 82, 84–86, 85, 86
 Sharpe ratios, 48–50, 203, 311–312, 316–317, 323–326
 showMethods function, 26–27
 skewed distribution
 return distributions, 62, 69–71, 80–82, 84–86
 risk measures, 40–41, 41
 robust portfolio optimization, 187
 volatility, 117, 122–123
SkewHyperbolic package, 70–71
 skew Laplace distribution, 69
slam package, 246
 S language, 6–7
SMEM *see* structural multiple equation models
SOCP *see* second-order cone program
 Stahel–Donoho estimator (SDE), 167, 177–178, 180–190
 standardized residuals, 151, 154–155
 structural multiple equation models (SMEM), 281–284
 structural vector autoregressive (SVAR) models, 284–287, 303–304
 structural vector error correction (SVEC) models, 285, 288–289, 306–307
 Student's *t* distribution
 dependence, 140, 142, 148, 153–154
 return distributions, 70–71
 risk measures, 40–41, 41
 robust portfolio optimization, 176
 volatility, 117, 122–123
 stylized facts
 dependence, 148–149
 multivariate series, 32–35, 33, 34, 35
 return distributions, 80–82
 univariate series, 29–32, 30, 31, 32
 sub-additivity, 43–44
 SUR *see* seemingly unrelated regression
 SVAR *see* structural vector autoregressive
 SVEC *see* structural vector error correction
 tactical asset allocation (TAA), 274–338
 ARMA time series process, 279–281, 297–302
 autoregressive time series process, 275–277, 300–302
 Black–Litterman model, 274, 289–292, 295–297, 307–313
 concepts and definitions, 274
 copula opinion pooling model, 292–297, 313–318, 314, 315, 317–318
 empirical applications, 307–334
 entropy pooling model, 292, 293–295, 304, 318–324, 320–322
 moving average time series process, 277–278, 300–302
 multivariate time series models, 281–289

- partial market model, 282–283, 283
 protection strategies, 324–334
 R language packages, 295–307
 structural multiple equation models, 281–284
 structural vector autoregressive models, 284–287, 303–304
 structural vector error correction models, 285, 288–289, 306–307
 time series models, 274, 275–289, 297–302, 304–307
 univariate time series models, 275–281
 vector autoregressive models 284–286, 299–300, 302–307, **309**
 vector error correction models, 285, 287–288, 305–310
 tail dependencies, 138–139, 141, 145, 199, 204–207, 210–221, **218**
 three-stage least squares (3SLS) method, 284
 time series models
 ARMA time series process, 279–281, 297–302
 autoregressive time series process, 275–277, 300–302
 moving average time series process, 277–278, 300–302
 multivariate time series models, 281–289
 structural multiple equation models, 281–284
 structural vector autoregressive models, 284–287, 303–304
 structural vector error correction models, 285, 288–289, 306–307
 tactical asset allocation, 274, 275–289, 297–302, 304–307
- univariate time series models, 275–281
 vector autoregressive models 284–286, 299–300, 302–307, **309**
 vector error correction models, 285, 287–288, 305–310
timeSeries package, 30–33, 253
 Tinbergen arrow diagram, 282
 transition kernels, 347–348
 translation invariance, 43
 trimming, 164
truncdist package, 347
tseries package, 127–128
 two-stage least squares (2SLS) method, 284
- uncertainty sets, 168–174, 190–195
 unconditional variance, 117–119
 unit root tests, 307–308, **308**
 univariate time series models, 29–32, **30**, 31, 32, 275–281
urca package, 304–307
 utility function, 49–50
- value at risk (VaR), 38–44, 40, 41
 diversification of risks, 201–204, 211, 222–225
 extreme value theory, 93–94, 101–102, 111
 return distributions, 63, 77–80, **78**, 79, 82–84, **83**
 risk-optimal portfolios, 228, 229–238, 241, 250–255, 265
 volatility, 126
VAR *see* vector autoregressive
 variance-covariance matrix
 dependence, 140, 149
 diversification of risks, 198–199, 202, 210
 modern portfolio theory, 47, 50–51
 return distributions, 68–69
 risk measures, 42
 risk-optimal portfolios, 250

- variance-covariance matrix
 (*continued*)
 robust portfolio optimization, 171
 volatility, 127–128
- VarianceGamma** package, 71
- vars** package, 304–307, **306**
- VECM** *see* vector error correction models
- vector autoregressive (VAR) models, 284–286, 299–300, 302–307, **309**
- vector error correction models
 (VECM), 285, 287–288, 305–310, **309**
- Vim** package, 11–12
- volatility, 116–132
 class of ARCH models, 116–120, 118
- clustering, 117–118
- concepts and definitions, 116
- diversification of risks, 200–201
- empirical application of volatility models, 128–130, **128**, 129
- R language packages, 120–128
- volatility-weighted average
 correlation, 200–201
- volatility-weighted concentration ratio, 200
- wealth distribution, 49–50
- Weibull distribution, 90–91
- winsorizing, 164
- Wold representation, 280–281, 286
- zoo** package, 32

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley's ebook EULA.