Portfolio Optimization with R/Rmetrics

Henry Clausen
Diethelm Würtz

Rmetrics Association & Finance Online Publishing

R/AMPL API 2.0

RMETRICS CORE TEAM

MARCH 2015

PREFACE

ABOUT THIS HANDBOOK

This handbook is a User and Reference Guide for the Rmetrics R/AMPL API 2.0. It brings the whole world of professional solvers to your fingertips!

ABOUT PART I

The sections in Part I

- R/AMPL Solver Interface
- Coin-or Infrastructure
- Kestrel/Neos Solver Interface
- Ampl/Rneos Solver Interface

describe the R/AMPL API 1.0 solver environment taht is about 4 years old. Note all four sections have to be upgraded to Version 2.0! This is not yet done.

GETTING STARTED

What you need to run the examples in the handbook ...

- Install the appropriate AMPL Demo Version for Windows, Max OSX, or Linux. Be sure that the executibles are in the PATH.
- Load Rmetrics Package *fPortfolio*
- Source the functions listed in Appendix I.

Try and enjoy it!

Zurich, March 2015

CONTENTS

PART I

SOLVERS

CHAPTER 1

R/AMPL SOLVER INTERFACE

```
> library(fPortfolio)
```

This chapter presents the Rmetrics interface for portfolio optimization together with AMPL. AMPL is an algebraic modeling language for solving linear, quadratic, and nonlinear optimization problems in discrete or continuous variables.
We report how to download, to install and how to use AMPL together with Rmetrics. We introduce how to invoke AMPL from the console. Furthermore we explain the idea and concept behind the Rmetrics/AMPL interface. Finally we introduce the Rmetrics AMPL library which allows to set up the most often used mathematical programming and portfolio models.

1.1 AMPL: A MODEL PROGRAMMING LANGUAGE

AMPL was developed at Bell Laboratories. The strength of AMPL is, that this programming language uses common notation and familiar concepts to formulate optimization models and examine solutions, while the computer manages communication with an appropriate commercial or open source solver. The home of AMPL[1] is the definite source for all kind of information about AMPL, including the AMPL book.

```
http://www.ampl.com/BOOK/index.html
AMPL: A Modeling Language for Mathematical Programming
Robert Fourer, David M. Gay, and Brian W. Kernighan
Brooks Cole Publishing Company, Cengage Learning, 2002
517 + xxi pp., ISBN 0-534-38809-4
```

[1] http://www.ampl.com

How to get AMPL

In the following we briefly describe how to buy, how to get a trial version, or how to download the student edition of the AMPL software.

30-day AMPL trial version

AMPL offers a full-featured, unrestricted copy of AMPL with a trial license good for 30 days. Note, the AMPL trial does not have solver built ins. More information you can get from the AMPL trial page[2] where also links to solver trials are given.

Student edition

For the student editions[3] of AMPL and solvers detailed information can also be found on the AMPL server.

Quick Start:
"To start learning about AMPL [■] download chapter 1 of the AMPL book, then proceed in any of the following ways to access the AMPL software and solvers:"

1. Try AMPL Page: "Go to the Try AMPL! page, where you can send models, data, and commands to a remote version of the AMPL Student Edition and a selection of solvers. A simple web interface lets you choose examples from the book and then experiment with changes in a series of runs.
2. Windows: "Download the AMPL Student Edition zip archive file, amplcml.zip. Then follow the instructions below to unpack and run the AMPL program, the CPLEX, Gurobi, and MINOS solvers, and the Kestrel client for free access to additional solvers over the Internet. Once AMPL is running, you can type commands just as they are shown in the AMPL book."
3. Unix/Linux: "The AMPL Student Edition and compatible solvers are available for all of the popular Unix workstations, as well as for Linux and other Unix-based operating systems on PCs. Consult the table in the instructions below to download and set up the AMPL binary that is appropriate for your platform. Then visit AMPL's solver download instructions to obtain one or more solvers for AMPL; Linux and Solaris users may alternatively download the Kestrel client to get access to solvers over the Internet. Once AMPL is running, you can type commands just as they are shown in the AMPL book.

Download and installation instructions

AMPL gives instructions for downloading its interface and related software. Here comes a short summary of the information provided by AMPL.

- All downloadable versions of AMPL are Student Editions, which are limited to problems of 300 variables and 300 constraints and objectives, but are full-featured in other respects. They can be used for small-scale evaluation and testing.

[2] http://www.ampl.com/trial.html
[3] http://www.ampl.com/DOWNLOADS/index.html

- Evaluation copies and updates for AMPL Professional Editions are available from AMPL vendors.
- Several solvers can be downloaded separately in binary executable form. Problem size limitations vary, but are mostly similar to the limitations on the AMPL Student Edition. Information on unrestricted versions can be found in the AMPL solver listing.

Solvers provided by AMPL

Here comes a summary with the commercial solvers provided by AMPL.

- CPLEX: Solves linear and convex quadratic programs by simplex or interior-point methods, and linear and convex quadratic integer programs by a branch-and-bound procedure.
- DONLP2: Solves nonlinear optimization problems using a sequential quadratic programming algorithm and dense-matrix linear algebra
- Gurobi: Solves linear programs by primal or dual simplex methods, and linear mixed-integer programs by a branch-and-bound procedure.
- KNITRO: Solves linear and nonlinear optimization problems in continuous variables, using a choice of interior-point and active-set methods. General constraints are allowed, and functions may be convex or nonconvex. KNITRO uses full 2nd-derivative information from AMPL, and supports AMPL's complementarity constraints.
- LOQO: Solves linear and nonlinear optimization problems in continuous variables, using interior-point methods.
- lp_solve: Solves linear and linear integer programs of moderate scale.
- MINOS: Solves linear programs by the primal simplex method, and nonlinear optimization problems in continuous variables by use of a reduced-gradient approach.
- SNOPT: Solves linear programs by the primal simplex method, and nonlinear optimization problems in continuous variables by use of a sequential quadratic programming approach that employs limited-memory approximations to 2nd-derivative information.

1.2 WORKING WITH AMPL ON THE CONSOLE

Download and install AMPL, with the appropriate license for you, either the full, a trial or the student edition. Start to read Chapter 1 of the AMPL book or inspect a tutorial from the web to learn the language.

Example: Solve the MV Portfolio interactively with AMPL

Let us optimize the mean-variance Markowitz portfolio problem with long only constraints. Here are the necessary three steps.

Step 1:
Write the model file and save it under the name `ampl.mod`:

```
# Quadratic Programming
# Mean-Variance Markowitz Problem with Long Only Constraints
```

```
param N;                                    # Number of assets
param mu{1..N};                             # Asset means
param Sigma{1..N, 1..N};                    # Covariance matrix
param targetReturn;                         # Target return

var w{1..N} >= 0;                           # Variable definition

minimize Risk:                              # Objective function
  sum {i in 1..N} sum{j in 1..N} w[i]*Sigma[i,j]*w[j];

subject to Return:                          # Return Constraints
  sum{i in 1..N} mu[i]*w[i] = targetReturn;

subject to Budget:                          # Budget Constraints
  sum{i in 1..N} w[i] = 1;
```

Step 2:
Write the data file and save it under the name `ampl.dat`:

```
# Data File for the Swiss Pension Fund Benchmark

# The Number of Assets:
param N := 6;

# The Return Vector:
param mu :=
1  4.066340e-05
2  0.08417544
3  0.02389356
4  0.005531533
5  0.05905151
6  0.08576789;

# The Covariance Matrix:
param Sigma :
   1           2          3            4            5           6       :=
1  0.01589955  -0.01274142 0.001799383  0.009803865 -0.01588837 -0.01323794
2 -0.01274142   0.5846121  0.03033648  -0.01407469   0.4115984   0.2983962
3  0.001799383  0.03033648 0.08513795   0.0009250538 0.02481619  0.01554892
4  0.009803865 -0.01407469 0.0009250538 0.01495111  -0.02332223 -0.01724687
5 -0.01588837   0.4115984  0.02481619  -0.02332223   0.5350326   0.3648052
6 -0.01323794   0.2983962  0.01554892  -0.01724687   0.3648052   0.3231242;
;

# The Target Return:
param targetReturn := 0.04307677 ;
```

Step 3:
Write the run file and save it under the name `ampl.run`:

```
option solver cplex;

model ampl.mod;
data ampl.dat;
```

```
solve;
display w > ampl.txt;

exit;
```

Now start a console window and run the optimization. Be sure that the `ampl` and binary executibles, here `cplex`, are in your binary search path.

```
> ampl ampl.run
CPLEX 11.2.0: optimal solution; objective 0,06006761091
14 QP barrier iterations
No basis.

> cat ampl.txt
w[*] :=
1  3.31387e-08
2  0.00864064
3  0.254318
4  0.335778
5  4.89336e-09
6  0.401263
```

These are the weights of the six assets optimized for the long only mean-variance portfolio with a target return 0.04307677 and target Risk (variance) 0.06006761091.

1.3 THE RMETRICS INTERFACE CONCEPT

Writing a simple R Interface for AMPL is quite easy following the concept:

1. Assume all the information on the AMPL mode, data, and run specofocations is available.
2. Write user supplied functions `amplModel*()`, `amplData*()`, `amplRun*()` which create the `foo.mod`, `foo.dat`, and `foo.run` AMPL files. Here `foo` is considered as project name.
3. Run AMPL through the system command `system("ampl foo.run")`. The result will be stored in AMPLs output file `foo.txt`.

The Rmetrics R/AMPL interface provides several functions to write and print AMPL model files, data files and run files. Here is a list of these utility functions.

LISTING 1.1: AMPL R UTILITY FUNCTIONS

```
Function:              Description:
  amplModelOpen         Opens a writes to an AMPL model file
  amplModelAdd          Adds model specs to an existing AMPL model file
  amplModelShow         Shows the content of an AMPL .mod file
Function:              Description:
  amplDataOpen          Opens and writes the header to an AMPL data file
```

```
   amplDataAddValue      Adds a numeric value to an AMPL data file
   amplDataAddVector     Adds a numeric vector to an AMPL data file
   amplDataAddMatrix     Adds a numeric matrix to an AMPL data file
   amplDataSemicolon     Adds a semicolon on the end of a data input line
  amplDataShow           Shows the content of an AMPL data file
Function:              Description:
  amplRunOpen           Opens a run file
  amplRunAdd            Adds run specs to an existing AMPL run file
  amplRunShow           Shows the content of an AMPL run file
```

To run the example from the previous section under R we proceed in the following way.

Step 1:
Write the model file from R and save it under the name `ampl.mod`, i.e. for the project name we have choosen `"ampl"` and the file extension is `"mod"`.

```
> amplModelOpen("ampl")
> model = c(
      "# Quadratic Programming",
      "# Mean-Variance Markowitz Problem with Long Only Constraints",
      "param N;",
      "param mu{1..N};",
      "param Sigma{1..N, 1..N};",
      "param targetReturn;",
      "var w{1..N} >= 0;",
      "minimize Risk: sum {i in 1..N} sum{j in 1..N} w[i]*Sigma[i,j]*w[j];",
      "subject to Return: sum{i in 1..N} mu[i]*w[i] = targetReturn;",
      "subject to Budget: sum{i in 1..N} w[i] = 1;")
> amplModelAdd(model, "ampl")
> amplModelShow("ampl")

# Quadratic Programming
# Mean-Variance Markowitz Problem with Long Only Constraints
param N;
param mu{1..N};
param Sigma{1..N, 1..N};
param targetReturn;
var w{1..N} >= 0;
minimize Risk: sum {i in 1..N} sum{j in 1..N} w[i]*Sigma[i,j]*w[j];
subject to Return: sum{i in 1..N} mu[i]*w[i] = targetReturn;
subject to Budget: sum{i in 1..N} w[i] = 1;
```

Step 2:
Write the data file and save it under the name `ampl.dat`:

```
> N = 6
> data = 100 * LPP2005REC[, 1:N]
> amplDataOpen("ampl")
> amplDataAddValue("N", N, "ampl")
> amplDataAddVector("mu", colMeans(data), "ampl")
> amplDataAddMatrix("Sigma", cov(data), "ampl")
> amplDataAddValue("targetReturn", mean(data), "ampl")
```

Step 3:
Write the run file and save it under the name `ampl.run`:

```
> amplRunOpen("ampl")
> run = c(
     "option solver cplex;",
     "model ampl.mod;",
     "data ampl.dat;",
     "solve;",
     "display w > ampl.txt;",
     "exit;")
> amplRunAdd(run, "ampl")
> amplRunShow("ampl")
option solver cplex;
model ampl.mod;
data ampl.dat;
solve;
display w > ampl.txt;
exit;
```

Final Step:
Run the portfolio Optimization.

```
> system("ampl ampl.run")
> as.data.frame(readLines("ampl.txt"))
  readLines("ampl.txt")
1             w [*] :=
2        1  3.31445e-08
3         2  0.00862809
4           3  0.254323
5           4  0.335773
6        5  4.89408e-09
7           6  0.401276
8                    ;
9
```

This approach can now easily be customized for solving portfolios in a very general and powerful environment, the R/AMPL interface.

CHAPTER 2

COIN-OR INFRASTRUCTURE

```
> library(fPortfolio)
```

This chapter explains how to use the Coin-Or infrastructure for portfolio optimization together with the Rmetrics to AMPL interface. "The Computational Infrastructure for Operations Research, COIN-OR[1], or simply COIN, project is an initiative to spur the development of open-source software for the operations research community". More about the implementation of the interior-point filter line-search algorithm for large-scale nonlinear programming can be found in Andreas Waechter and Lorenz T. Bieglery, [2004], and in the introducton to COIN-OR tools for optimization by Ralphs, [2009].
COIN-OR includes more than 30 projects. The projects include solvers for linear programming nonlinear programming mixed integer linear programming, mixed integer nonlinear programming (convex and nonconvex) stochastic linear programming, and semidefinite programming. Most of the solvers are licensed under the EPL.

2.1 BINARY DISTRIBUTION PROJECT

Binary Files[2] can also be obtained from the COIN-OR web site. The goal of the COIN-OR binary distribution project is to provide sets of libraries and executables precompiled and tested on the most popular platforms for those users who do not need to look at or modify the source code of the COIN projects themselves. Currently supported platforms are Windows, Linux and MacOS.

[1] http://www.coin-or.org
[2] https://projects.coin-or.org/CoinBinary

Under windows we have installed COIN-OR in the directory
`mathrmC:`

`AMPL`
`coin-or` which holds the subdirectories for the `bin`, the include `lib` and the `share` directories from the COIN-OR distribution. Do not forget to add the path variable to the search environment. That is all what we have to do. To get the software, download it[3] and follow the installation instractions[4].

The "CoinAll" distribution

"This part of the *CoinBinary* project is an effort to develop a distribution consisting of a set of consistent, interoperable binaries built from the source code of as large a subset of COIN-OR projects as possible. The idea is to allow a user who wants binaries for a large number of COIN projects and wants to ensure that they will all interoperate to be able to download them all at once in a single distribution." Projects currently included in *CoinAll* are

LISTING 2.1: PROJECTS INCLUDED IN COINALL

```
Solver:             Description:
bonmin              NL Mixed Integer Programming
cbc                 LP branch-and-cut solver
clp                 LP simplex solver
couenne             branch-and-bound NL MI Programming
ipopt               IP general large-scale NL optimization
symphony            Linear Mixed Integer Programming
```

To get information from inside R on your downloaded version, for example type

```
> system("ipopt -v")
> system("bonmin -v")
> system("couenne -v")
```

2.2 SOLVER FOR QUADRATIC PROGRAMMING PROBLEMS

The COIN-OR infrastructure offers us for the quadratic programming problems the solver `ipopt`.

[3] http://www.coin-or.org/download/binary
[4] https://projects.coin-or.org/CoinBinary/wiki/InstallationInstructions

Example: Global minimum Markowitz portfolio - ipopt

`ipopt` is the default solver for continuous nonlinear programs. To solve the long only mean-variance portfolio optimization problem we can use this solver.
Load the data

```
> # Load Data:
> n = 6 # number of assets
> lppData = 100 * LPP2005REC[, 1:6]
> Cov = cov(lppData)
```

write the AMPL model file

```
> # Write Model File:
> amplModelOpen("globmin")
> model <- c(
     "param Cov{1..6,1..6} ;",
     "var weights{1..6} >= 0;",
     "minimize Risk: sum{i in 1..6} sum{j in 1..6} weights[i] * Cov[i,j] * weights[j] ;",
     "s.t. Budget: sum{i in 1..6} weights[i] = 1 ;" )
> amplModelAdd(model, "globmin")
> amplModelShow("globmin")
```

```
param Cov{1..6,1..6} ;
var weights{1..6} >= 0;
minimize Risk: sum{i in 1..6} sum{j in 1..6} weights[i] * Cov[i,j] * weights[j] ;
s.t. Budget: sum{i in 1..6} weights[i] = 1 ;
```

copy the covariance matrix to the AMPL data file

```
> # Write Data File:
> amplDataOpen("globmin")
> amplDataAddMatrix(data="Cov", matrix=Cov, "globmin")
```

and compose the AMPL run file

```
> # Write Run File:
> amplRunOpen("globmin")
> run <- c(
     "option solver ipopt ;",
     "model globmin.mod ;",
     "data globmin.dat ;",
     "solve ;",
     "display weights > globmin.txt ;",
     "exit ;")
> amplRunAdd(run, "globmin")
> amplRunShow("globmin")
```

```
option solver ipopt ;
model globmin.mod ;
data globmin.dat ;
solve ;
display weights > globmin.txt ;
exit ;
```

Now we arr ready to optimize the portfolio and print the optimal weights

```
> command = "ampl globmin.run"
> system(command, show.output.on.console = TRUE)
> read.csv("globmin.txt")
  weights.......
1           1  0
2           2  0
3           3  0
4           4  0
5           5  0
6           6  0
7              ;
```

Exercise: Critical line algorithm

Use the COIN `ipopt` to solve a portfolio with an objective function specified by the critical line algorithm.

2.3 SOLVER FOR MIXED INTEGER PROGRAMMING

To solve a mixed-integer nonlinear program then the `bonmin` solver is the proper one.
The `couenne` solver can be used used alternatively for mixed-integer nonlinear programs.

CHAPTER 3

KESTREL/NEOS SOLVER INTERFACE

```
> library(fPortfolio)
```

This chapter presents the AMPL based Kestrel/Neos interface for portfolio optimization within Rmetrics. The NEOS Server provides access to a variety of optimization resources and solvers via the Internet. Dolan, Fourer, Goux, Munson, and Sarich [2006] have created an interface named Kestrel which allows to access easily the Server. "The NEOS Server enables local modeling environments to request optimization services and retrieve the results for local visualization and analysis, so that users have the same convenient access to remote NEOS solvers as to those installed locally." We use the Kestrel agent implemented for the AMPL modeling environments, amother is availalble for modeling with GAMS. These agents have been designed in such a way that subproblems can be queued for execution and later retrieval of results. The authers claim that this makes a rudimentary form of parallel processing avialable.

3.1 KESTREL: INTERFACE

The Kestrel client[1] can be downloaded from the NEOS Web Server
To run Kestrel together with AMPL you need to have the AMPL interactive environment installed on your local computer. This we have already done in chpater ■. For the Kestrel installation proceed in the following way.

Kestrel Client Installation

1. For Windows: Download the file `kestrel.zip`, and unzip the file into the same directory as the AMPL executable.

[1] http://www.neos-server.org/neos/kestrel.html

2. Unix: Note, requires python to be installed, Download the file `kestrel.tar.gz`, and unzip the file into a directory in the path, we recommend the same directory as the AMP binary files.

Using Kestrel from within the AMPL Environment

1. Design your model as you normally would. When choosing options, everything should remain as per usual with the following exceptions. Choose option solver kestrel; instead of the usual solver name. Choose the solver you want with

```
option kestrel\_options 'solver=<solverName>';
```

   Specify NEOS Server URL

```
option neos\_server 'www.neos-server.org:3332';
```

   If you do not know what solvers are available via Kestrel, submitting a job (see below) with a nonexistent solver will return a list of enabled solvers.

2. When your `kestrel_options` are set, submit the job to the NEOS Server by typing

```
solve;
```

   If you are somehow disconnected from the Kestrel server during your job execution, you can specify

```
option kestrel\_options 'job=<jobNumber> password=<password>';
```

   and ask kestrel to

```
solve;
```

3. If your job is still in progress, your AMPL session will resume waiting. Otherwise, your results will be retrieved. Jobs are removed from the NEOS Server after some length of time (usually two days), so you will not be able to retrieve your job this way after that time. To resume normal kestrel solver operation type

```
option kestrel\_options 'solver=<solverName>';
```

Note: On Windows you will get an error message that the client requires python24.dll to run. Please ignore this message, since the client can execute without this library.

3.2 RUNNING A KESTREL JOB

First we prepare model, data and run files, then we submit the job and retrieve the results.

Preparing Model, Data and Run Files

Step 1:
Write the model file and save it under the name `ampl.mod`:

```
> amplModelOpen(project = "kestrel")
> model <- c("param nAssets;", "param mu{1..nAssets};", "param Cov{1..nAssets, 1..nAssets};",
     "param targetReturn;", "var x{1..nAssets} >= 0;", "minimize Risk:",
     "  sum {i in 1..nAssets} sum{j in 1..nAssets} x[i]*Cov[i,j]*x[j];",
     "subject to Return:", "  sum{i in 1..nAssets} mu[i]*x[i] = targetReturn;",
     "subject to fullInvestment:", "  sum{i in 1..nAssets} x[i] = 1;")
> amplModelAdd(model, project = "kestrel")
```

Step 2:
Write the data file and save it under the name `ampl.dat`:

```
> amplDataOpen(project = "kestrel")
> amplDataAddValue("nAssets", value = 6, "kestrel")
> R <- LPP2005.RET[, 1:6]
> amplDataAddVector("mu", colMeans(R), "kestrel")
> amplDataAddMatrix("Cov", cov(R), "kestrel")
> targetReturn <- mean(R)
> amplDataAddValue("targetReturn", value = targetReturn, "kestrel")
```

Step 3:
Write the run file and save it under the name `kestrel.run`:

```
> # AMPL Run File:
> amplRunOpen("kestrel")
> run <- c(
     "reset;",
     "option solver kestrel;",
     "option kestrel_options 'solver=loqo';",
     "model kestrel.mod;",
     "data kestrel.dat;",
     "solve;",
     "display x > kestrel.txt;",
     "exit;")
> amplRunAdd(run, "kestrel")
```

Submitting the Optimization Problem

Now start a console window and run the optimization. Be sure that the `ampl` and `kestrel` binary executibles are in the binary search path.

```
> system("ampl kestrel.run")
```

Retrieving the Results from the Optimization Problem

```
> cat(readLines("kestrel.txt"), sep = "\n")
```

```
x [*] :=
1  0
2  0
3  0
4  0
5  0
6  0
;
```

These are the weights of the six assets optimized for the long only mean-variance portfolio with a target return $R_{target} = 0.04307677$.

CHAPTER 4

AMPL/RNEOS SOLVER INTERFACE

```
> library(fPortfolio)
> library(rneos)
```

This chapter presents the AMPL based R/Neos interface for portfolio optimization within Rmetrics. Like in the case of the Kestrel client the contributed `rneos` package provides similar access functionality to a variety of optimization resources and solvers via the NEOS server. Bernhard Pfaff [2006] has created an `R` based interface and `R` package named `rneos` which allows easily to access and to communicate with the NEOS server.

4.1 RNEOS: NEOS SERVER INTERFACE

In his `R` package `rneos` Bernhard Pfaff has implemented the XML-RPC based API of the *Network-Enabled Optimization System*, NEOS. General information[1] and especially information about the NEOS API[2] is available fom the NEOS server.

For a description of XML-RPC[3] we refer to the XMLRPC server, and to wikipedia[4].

The package `rneos` utilizes S4-classes and methods for the communication with the NEOS server. The package depends on the packages `bitops`, `XMLRPC`, `RCurl`, and `XML`. A first idea what these packages are good for, can be obtained by inspecting their package descriptions. You can use for this the function `packageDescription()`.

[1] http://www.neos-server.org/neos
[2] http://www.neos-server.org/neos/NEOS-API.html
[3] http://www,xmlrpc.com
[4] http://en.wikipedia.org/wiki/XML-RPC

The package bitops has functions for bitwise operations. The original S functions were written by Steve Dutky, and the initial R port and extensions go back to Martin Maechler.
The package XML provides many approaches for both reading and creating XML (and HTML) documents (including DTDs), both local and accessible via HTTP or FTP. The package suggest bitops for download.
The package RCurl provides a general network HTTP and FTP client interface for R. The package is authored by Duncan Temple Lang. It depends on bitops.
The package XMLRPC comes with functions for remote procedure calls, RPC, via XML in R. The author of the package is again Duncan Temple Lang. The package imports RCurl and XML, the software is availalble on the Omegahat server.

Installation on Windows

The packages bitops, RCurl and XML are available from CRAN, XMLRPC from Omegahat. bitops and XMLRPC can be installed as Windows binaries directly from the CRAN server. RCurl and XML are much more tricky to get them installed under Windows. Precompiled packages for RCurl are available from Brian Ripley's web site

```
http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/2.13/RCurl_1.6-10.1.zip
http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/2.14/RCurl_1.7-0.1.zip
http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/2.15/RCurl_1.7-0.1.zip
```

Precompiled versions for XML can be downloaded also from his site

```
http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/2.13/XML_3.4-2.2.zip
http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/2.14/XML_3.4-2.2.zip
http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/2.15/XML_3.4-2.2.zip
```

For XMLRCP we refer in case of the source code to the Omegahat web server

```
http://www.omegahat.org/XMLRPC/
```

and for the Windows binaries the reference is again Brian Ripley's web server.

```
http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/2.13/XMLRPC_0.2-4.zip
http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/2.14/XMLRPC_0.2-4.zip
http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/2.15/XMLRPC_0.2-4.zip
```

If you have successfully installed all these packages, then you should get the following messages

```
> library(rneos)
Loading required package: XMLRPC
Loading required package: RCurl
Loading required package: bitops
Loading required package: XML
```

when you have loaded the `rneos` library.

4.2 USING RNEOS TOGETHER WITH AMPL

The `Rneos` Interface can be used in several ways,

1. Through the Internet: upload of model and datafiles
2. Through Email: Upload of model and data files
3. AMPL/GAMS via Kestrel
4. NEOS API (XML-RPC): Clients

Clients are availalble in several languages, the one which is of interest for us is the `R` client. We think one of the most comfortable ways is to use the `R` client together with `AMPL`. That is the approach we present and discuss in the following.

4.3 PREPARING MODEL, DATA AND RUN FILES

Our `AMPL` Mathematical Programming library (unpublished) contains AMPL model scripts for the most common portfolio optimization problems. Here we show how to write `model`, `data`, and `run` files and how they work together with `Rneos`.

Preparing the Model File

Start with the `model` file for the standard long-only Markowitz mean–variance portfolio model

```
> amplModelOpen(project="rneos")
> model <- c(
     "param nAssets;",
     "param mu{1..nAssets};",
     "param Cov{1..nAssets, 1..nAssets};",
     "param targetReturn;",
     "var x{1..nAssets} >= 0;",
     "minimize Risk: sum {i in 1..nAssets} sum{j in 1..nAssets} x[i]*Cov[i,j]*x[j];",
     "subject to Return: sum{i in 1..nAssets} mu[i]*x[i] = targetReturn;",
     "subject to fullInvestment: sum{i in 1..nAssets} x[i] = 1;")
> amplModelAdd(model, project="rneos")
```

Preparing the Data File

Add the AMPL data file

```
> R <- LPP2005.RET[, 1:6]
> amplDataOpen(project="rneos")
> amplDataAddValue("nAssets", 6, "rneos")
> amplDataAddVector("mu", colMeans(R), "rneos")
> amplDataAddMatrix("Cov", cov(R), "rneos")
> amplDataAddValue("targetReturn", mean(R), "rneos")
```

Preparing the Run File

And finally compose the AMPL run file

```
> amplRunOpen("rneos")
> run <- c(
      "solve ;",
      "display x;",
      "exit ;")
> amplRunAdd(run, "rneos")
```

Here we have the used the AMPL R constructior functions which we have introduced in the chapter about the R/AMPL interface. Whereas the AMPL model and data files are the same as when used together with a local software installation or together with the Kestrel interface, note the run file is different. The name of the solver has not to be specified in the file.

Submitting the Optimization Problem

To submit the problem to the server we have to compose model, data, and run strings of length one, where the entries are separated by new line characters.

```
> model <- paste(readLines("rneos.mod"), sep = " ", collapse = "\n")
> data <- paste(readLines("rneos.dat"), sep = " ", collapse = "\n")
> run <- paste(readLines("rneos.run"), sep = " ", collapse = "\n")
```

We have also to name the solver, and to which category the solver belongs to on the NEOS server. Let us use the Coin-OR "ipopt" solver which is part of the category "nco", *non-linear constraint optimization.*

```
> solver <- "ipopt"
> category <- "nco"
```

A list of solvers and categories[5] can be found on the NEOS server.
The next step is to compose and submit the job. We use the following four functions from Berhard Pfaff's rneos package

[5] http://www.neos-server.org/neos/solvers/index.html

```
NgetSolverTemplate
CreateXmlString
NsubmitJob
NgetFinalResults
```

```
> amplSpec <- list(
     model = model, data = data, commands = run,
     comments = "NEOS")
> solverTemplate <- NgetSolverTemplate(
     category = category, solvername = solver, inputMethod = "AMPL")
> xmls <- CreateXmlString(
     neosxml = solverTemplate, cdatalist = amplSpec)
> submittedJob <- NsubmitJob(
     xmlstring = xmls, user = "rneos", interface = "", id = 0)
> ans <- NgetFinalResults(
     obj = submittedJob, convert = TRUE)
```

Exercise: submitRneos Function

It is left to the reader to write a function `submitRneos(model, data, run)()` with input arguments for the AMPL files and which gives back the results returned from the NEOS Server.

Extracting Information Returned from NEOS

All results from the NEOS server have to be extracted from the returned value, `ans`. This requires some string manipulations on the text file. As an example we show how to retrieve the optimized weights. The returned variable `ans` is an object class S4 and contains the following slots

```
> slotNames(ans)
[1] "ans"    "method" "call"   "nc"
```

The entry @ans provides the desired results.

```
> nAssets <- 6
> out <- strsplit(ans@ans, split = "\n")[[1]]
> Index <- (grep("x .*. :=", out) + 1):(grep("^;$", out) - 1)
> Out <- out[Index]
> splits <- na.omit(as.numeric(strsplit(paste(Out, collapse = " "), " ")[[1]]))
> weights <- splits[seq(2, 2 * nAssets, by = 2)]
> names(weights) <- colnames(R)
> weights
       SBI        SPI        SII        LMI        MPI        ALT
5.0113e-04 1.4284e-02 2.5336e-01 3.3584e-01 7.5345e-05 3.9594e-01
```

Note, to retrieve information from the returned text file may be sometimes very ugly, when we do it in this way. As an alternative one can use (i) formatted output, or (ii) directly access the underlying XML protocol.

PART II

MEAN-VARIANCE DESIGNS

CHAPTER 5

MEAN-VARIANCE PORTFOLIOS

5.1 INTRODUCTION

In this chapter of the handbook we present the optimization of the mean-variance Markowitz portfolio problem and related problems in a broad sense. We will specifically talk about:

- Global Minimum Variance Portfolio
- Efficient Min-Risk Markowitz Portfolio
- Efficient Max-Return Markowitz Portfolio
- Efficient Frontier Calculation
- Maximum Sharpe Ratio Portfolio
- Feasible Set Calculation

Section 2 introduces the standard Markowitz portfolio that minimizes the sample covariance matrix as risk measure subject to the constraints of a desired return, a fully invested budget, and of long-only positions. This yields a risk optimimized portfolio on the efficient frontier. The objective function is quadratic and the constraints are linear. In section 3 we consider the dual case: Maximizing the return and minimizing the risk. The objective function is now linear and the constraints are quadratic. In Section 4 we combine the view of minimizing risk and maximizing the return. This yields the critical line algorithm (CLA) with *risk aversion* as a control parameter. Varying the risk aversion we can go along the whole efficient frontier. The endpoints of the efficient frontier are represented by the global minimum risk portfolio and the global maximum return portfolio are discussed in Section 5 Note, the standard CLA algorithm for the mean-variance portfolio has a quadratic objective and linear constraints.

Section 6 shows how to optimize he *Max Sharpe Ratio* Portfolio.

Throughout this chapter we use as an example the Swiss pension fund benchmark portfolio. The data are part of the Rmetrics package `timeSeries` and are loaded together with the `fPortfolio` package. As the benchmark portfolio we use the equal weights portfolio which is characterized by the following settings.
The `targetReturn` for the equal weights portfolio is defined by the *grand mean* of the portfolio scenarios, and the `targetRisk` is defined by the *grand variance* of the portfolio. `mu` is the vector of the sample means of the assets, and `Sigma` the sample covariance matrix.

5.2 FEASIBLE SET

A *portfolio* is a collection of assets investments. The amount of investment in asset i is denoted by w_i. The portfolio can then be characterized by the vector w of all asset weights w_i.
Each portfolio has its own properties such as its *expected return*

$$R := w'\mu, \tag{5.1}$$

where μ is the vector of expected returns of each asset, or its *variance*

$$\sigma^2(w) := w'\Sigma w, \tag{5.2}$$

where Σ is the *covariance* matrix between the individual daily asset returns. Normally, a requirement of modern portfolio theory is that the portfolio is fully invested. If we normalize the total investment volume to 1, we get the constraint

$$w'1 = \sum_{i=1}^{N} w_i = 1 \tag{5.3}$$

Furthermore, investments in portfolio theory are normally assumed to be long-only, i.e. short selling is not allowed. This leads to the additional constraint

$$w_i \geq 0 \forall\, i \in \{1, ..., N\} \tag{5.4}$$

For models that allow short selling see chapter ...
These constraints create a finite volume inside the N-dimensional vector space that is created by all possible w-vectors. This volume is called the

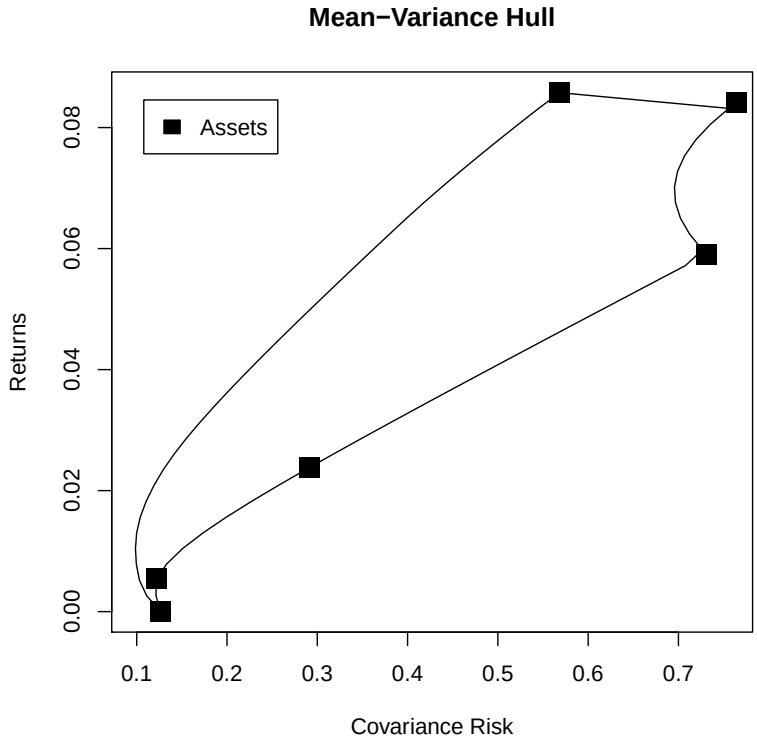


FIGURE 5.1: Feasible set and the positions of individual assets inside the set. The assets are members of the LPP2005 index.

feasible set of portfolios. The frontier of the feasible set is called the *hull*. The feasible set and the hull are often projected onto the 2-dimensional plane spanned by the portfolio variance and the expected return. The feasible set of all allowed portfolios of the Swiss pension fund index *LPP2005* is depicted in figure 5.1.

5.3 GLOBAL MINIMUM VARIANCE PORFOLIO

Harry Markowitz proposed 1952 in his article *Portfolio selection** `reference` to identify the riskiness of an asset with the variance of its returns. It is the simplest established risk measure and for that reason the basis for the traditional portfolio theory.
We already saw the variance of the portfolio returns in equation 5.2. The simplest optimized portfolio within our investment boundaries is the *Global Minimum Variance Portfolio* (in short *MVGLOB*). This is the portfolio with the lowest variance achievable. Since the variance is quadratic in portfolio weights, we can use a quadratic optimization model:

$$\min_{w} w'\Sigma w \tag{5.5}$$

$$s.t.$$

$$1'w = 1$$

$$w_i \geq 0$$

To estimate the covariance matrix Σ, we will use the sample covariance estimate:

$$\Sigma_{i,j} := \frac{1}{n-1}\sum_{s=1}^{S}(X_{i,s} - \bar{X}_i)(X_{j,s} - \bar{X}_j) \tag{5.6}$$

For other covariance estimation methods, see chapter ....
The R/AMPL Model File is then described by introducing the parameters N and Σ and variables w_i, the objective function and the constraints of equation 5.5:

```
> modelMVGLOB <- c(
   "param N ;",
   "param Sigma{1..N,1..N} ;",
   "var w{1..N} >= 0, default 1/N ;",
   "minimize Objective: sum{i in 1..N} sum{j in 1..N} w[i] * Sigma[i,j] * w[j] ;",
   "subject to Budget: sum{i in 1..N} w[i] >= 1 ;")
> amplModelFile(model=modelMVGLOB, project="myPortfolio")
```

Note that the $w_i \geq 0$-constraint is formulated directly in the introduction of w. We also specified the initial value of w to be the equal weights portfolio since this portfolio is alreay inside the feasible set. This initialization reduces the solving time , especially for very large portfolios.

The R/AMPL Run File must then specifiy the solver, optimize the objective, and print the weights into a text-file. As mentioned, our objective is a

quadratic problem, therefore we choose to use the solver cplex that is able to solve linear and quadratic problems efficiently. Our run-file now consists of the specification of the solver, the model file and the data file, and the solve and the print command.

```
> runMVGLOB <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runMVGLOB, project="myPortfolio")
```

In fact, this R/AMPL Run File is only dependent on the specific model in terms of the general objective form, i.e. it can be used for every other linear/quadratic portfolio problem. Therefore, we will continue to use this run file for every other problem of that form instead of always formulating a new run file. In section , a run file for non-linear problems will be introduced in the same manner.
For the AMPL Data File, we have to specify the parameters (N and Σ) from the model file.

```
> Scenarios <- 100*LPP2005.RET[, 1:6]
> requiredData(modelMVGLOB)
[1] "N"     "Sigma"

>   N <- ncol(Scenarios)
>   Sigma <- cov(Scenarios)
> dataMVGLOB <- dataAUTO(modelMVGLOB)
> amplDataFile(data=dataMVGLOB, project="myPortfolio")
```

The R-function *cov* calculates the sample covariance matrix using formula 5.6.
We then optimize the `MVGLOB` Portfolio

```
> system("ampl myPortfolio.run > myPortfolio.out")
```

and extract the weights:

```
> weightsMVGLOB <- as.numeric(scan("myPortfolio.txt"))
> names(weightsMVGLOB) <- colnames(Scenarios)
> weightsMVGLOB
      SBI      SPI      SII      LMI      MPI      ALT
0.355437 0.000000 0.089050 0.489347 0.002576 0.063590
```

Finally, we summarize our results:

```
> mu <- colMeans(Scenarios)
> SummaryMVGLOB <- c(
   TargetReturn = mu %*% weightsMVGLOB,
   CovarianceRisk = sqrt ( weightsMVGLOB %*% Sigma %*% weightsMVGLOB ),
   HerfindahlIndex = 1 - weightsMVGLOB %*% weightsMVGLOB)
> SummaryMVGLOB
```

```
TargetReturn  CovarianceRisk HerfindahlIndex
    0.010455        0.098623        0.622224
```

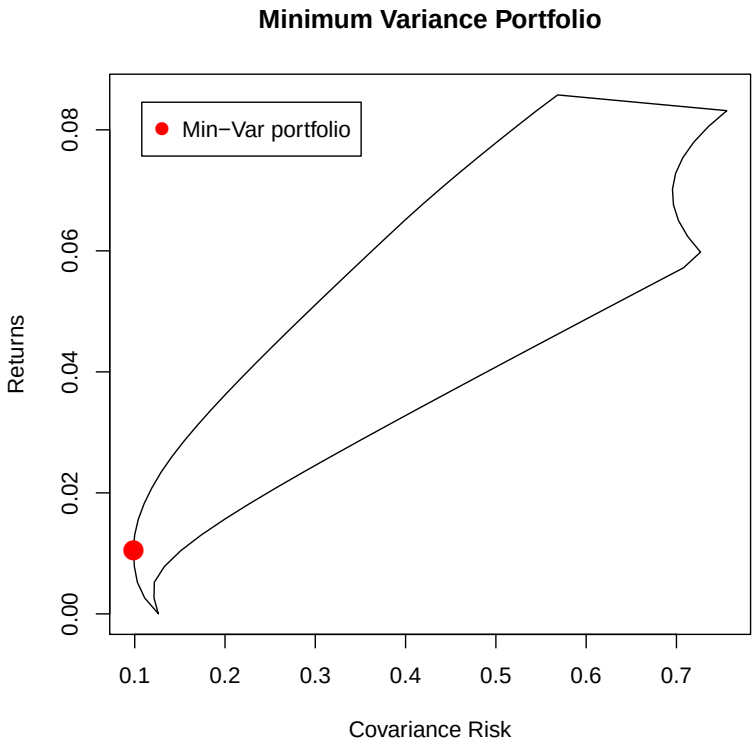


FIGURE 5.2: Feasible set and the position of minimum variance portfolio.

5.4 THE EFFICIENT MINIMUM VARIANCE PORTFOLIO

The standard model in portfolio design is the *Efficient Minimum Variance Portfolio* (MV1),also called *Mean-Variance Portfolio*. The model minimizes the sample covariance risk Σ for a desired predefined target return $\overline{r}$:

$$\min_{w} w'\Sigma w \tag{5.7}$$

$$s.t.$$

$$\mu' w \geq \overline{r}$$
$$1' w = 1$$
$$w_i \geq 0$$

The R/AMPL model file is similar to the `MVGLOB`-model except that the parameters μ and $\overline{r}$ as well as an additional constraint have to be added:

```
> modelMV1 <- c(
  "param N ;",
  "param mu{1..N} ;",
  "param Sigma{1..N,1..N} ;",
  "param targetReturn ;",
  "var w{1..N} >= 0, default 1/N ;",
  "minimize Objective: sum{i in 1..N} sum{j in 1..N} w[i] * Sigma[i,j] * w[j] ;",
  "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
  "subject to Return: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;")
> amplModelFile(model=modelMV1, project="myPortfolio")
```

The AMPL run file stays exactly the same as for the `MVGLOB`-portfolio:

```
> runMV1 <-  c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runMV1, project="myPortfolio")
```

From the portfolio settings we construct the R/AMPL data file where we only have to add the values for μ and $\overline{r}$ since N and Σ are already specified:

```
> Scenarios <- 100*LPP2005.RET[, 1:6]
> requiredData(modelMV1)
[1] "N"            "mu"           "Sigma"        "targetReturn"
>   N <- ncol(Scenarios)
>   Sigma <- cov(Scenarios)
>   mu <- colMeans(Scenarios)
>   targetReturn <- mean(mu)
> dataMV1 <- dataAUTO(modelMV1)
> amplDataFile(data=dataMV1, project="myPortfolio")
```

Optimize the Portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMV1 <- as.numeric(scan("myPortfolio.txt"))
> names(weightsMV1) <- colnames(Scenarios)
```

Summarize the results:

```
> SummaryMV1 <- c(
   TargetReturn = mu %*% weightsMV1,
   CovarianceRisk = sqrt ( weightsMV1 %*% Sigma %*% weightsMV1 ),
   HerfindahlIndex = 1 - weightsMV1 %*% weightsMV1)
> SummaryMV1
   TargetReturn  CovarianceRisk HerfindahlIndex
       0.043077        0.245088        0.661479
```

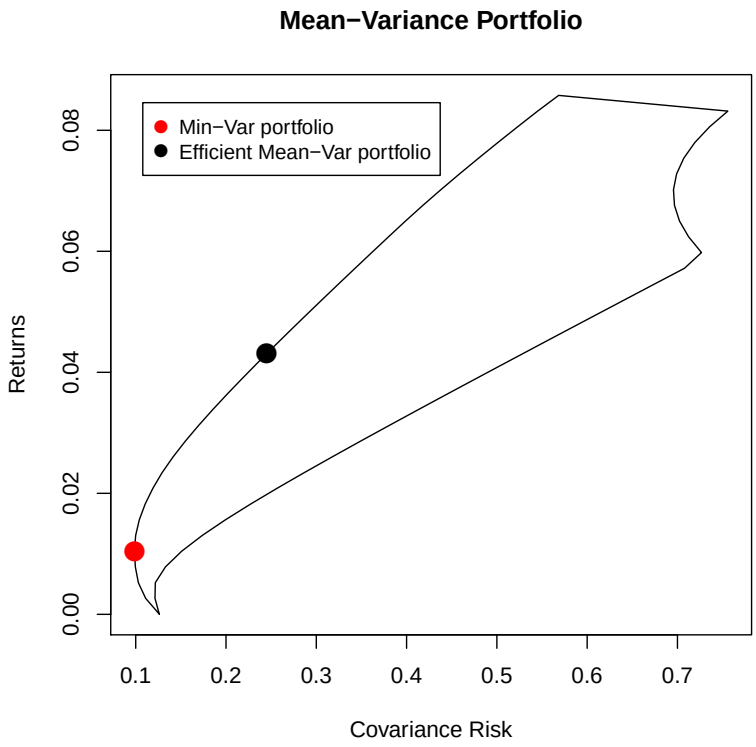


FIGURE 5.3: Position of a Mean-Variance portfolio inside the feasible set.

5.5 EFFICIENT MAXIMUM RETURN PORTFOLIO

Let us consider the portfolio optimization from the other way around: Fix the risk and maximize the return. The problem in equation 5.7 is then transformed into the following optimization problem:

$$\begin{aligned} & \max_{w} \mu' w && (5.8) \\ s.t. & \\ & w'\Sigma w \leq \overline{\sigma}^2 \\ & 1'w = 1 \\ & w_i \geq 0 \end{aligned}$$

The R/AMPL model file for a single maximum return mean-variance portfolio is now given by:

```
> modelMV2 <- c(
  "param N ;",
  "param targetRisk ;",
  "param mu{1..N} ;",
  "param Sigma{1..N,1..N} ;",
  "var w{1..N} >= 0, default 1/N ;",
  "maximize Objective: sum{i in 1..N} mu[i] * w[i] ;",
  "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
  "subject to Risk: sum{i in 1..N} sum{j in 1..N} w[i] * Sigma[i,j] * w[j] <= targetRisk ;")
> amplModelFile(model=modelMV2, project="myPortfolio")
```

The R/AMPL run file is unchanged

```
> runMV2 <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(runMV2, "myPortfolio")
```

Instead of the target return, the R/AMPL data file now needs the target risk. In this example, we set the target risk to be the variance of the portfolio calculated from the MV1 model:

```
> Scenarios <- 100*LPP2005.RET[, 1:6]
> requiredData(modelMV2)
[1] "N"          "targetRisk" "mu"         "Sigma"

>  N <- ncol(Scenarios)
>   Sigma <- cov(Scenarios)
>   mu <- colMeans(Scenarios)
>   targetRisk <- (weightsMV1 %*% Sigma %*% weightsMV1)[1]
```

```
> dataMV2 <- dataAUTO(modelMV2)
> amplDataFile(data=dataMV2, project="myPortfolio")
```

Optimize the Portfolio:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMV2 <- as.numeric(scan("myPortfolio.txt"))
> names(weightsMV2) <- colnames(Scenarios)
```

The weights for this portfolio should be the same as those for the mean-variance portfolio:

```
> rbind(MV1 = weightsMV1, MV2 = weightsMV2)
    SBI      SPI     SII     LMI MPI     ALT
MV1   0 0.008628 0.25432 0.33577   0 0.40128
MV2   0 0.008627 0.25432 0.33577   0 0.40128
```

and as exptected, this is true.

5.6 EQUI-DISTANT RETURN FRONTIER PORTFOLIO

There are obviously many efficient portfolios, depending on the value one puts on the expected return. The whole set of optimal portfolios is called the efficient frontier. The way to compute the efficient frontier is to loop in equi-distant return steps from the global minimum variance portfolio up to the single asset portfolio with the asset that shows the highest expected return. We call these portfolios the equi-distant return, short EDR, portfolios.
In order execute our desired loop in AMPL, we have to add three new parameters to the MV1-model file: The maximal target return (which is just the maximum of μ), the minimal target return (which is the return of the MVGLOB portfolio) and the number of frontier portfolios that should be calculated.

```
> modelMVEDR <- c(
   "param minReturn ;",
   "param maxReturn ;",
   "param nReturn ;",
   "param targetReturn ;",
   "param N ;",
   "param mu{1..N} ;",
   "param Sigma{1..N,1..N} ;",
   "var w{1..N} >= 0, default 1/N ;",
   "minimize Objective: sum{i in 1..N} sum{j in 1..N} w[i] * Sigma[i,j] * w[j] ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to Reward: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;")
> amplModelFile(model=modelMVEDR, project="myPortfolio")
```

or in short:

```
> modelMVEDR <- c(
   "param minReturn ;",
   "param maxReturn ;",
   "param nReturn ;",
   modelMV1)
>   amplModelFile(model=modelMVEDR, project="myPortfolio")
```

The R/AMPL Run File has to be modified for this algorithm since now we are looping over several target returns. In order to get a value for the `minReturn`-parameter, we solve the problem first for a target return set to $-\infty$ in order to calculate the expected return of the minimum variance portfolio. We then define the values for the `minReturn`- and `maxReturn`-parameter. Afterwards, we introduce a loop and replace the value of $\overline{r}$ for every iteration:

```
> runMVEDR <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex;",
   "let targetReturn := -999;",
```

```
    "solve;",
    "let minReturn := sum{i in 1..N} w[i]*mu[i];",
    "let maxReturn := max{i in 1..N} mu[i];",
    "for {i in 0..nReturn} {",
    "  let targetReturn := minReturn + i*(maxReturn-minReturn)/nReturn ;",
    "  solve ;",
    "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
    "};",
    "exit ;")
> amplRunFile(run=runMVEDR, project="myPortfolio")
```

Since we specify the values inside the R/AMPL run file, we do not have to set any values for the `targetReturn`-,`minReturn`- oder `maxReturn`-parameter. If we want for instance 33 points on our efficient frontier, the R/AMPL data file reads:

```
> Scenarios <- 100*LPP2005REC[,1:6]
> requiredData(modelMVEDR)
[1] "minReturn"    "maxReturn"    "nReturn"      "N"            "mu"
[6] "Sigma"        "targetReturn"

>   N <- ncol(Scenarios)
>   Sigma <- cov(Scenarios)
>   mu <- colMeans(Scenarios)
>   targetReturn <- NA
>   minReturn <- NA
>   maxReturn <- NA
>   nReturn <- 33
> dataMVEDR <- dataAUTO(modelMVEDR)
> amplDataFile(data=dataMVEDR, project="myPortfolio")
```

Optimize the portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMVEDR <- matrix(as.numeric(scan("myPortfolio.txt")), byrow=TRUE, ncol=N)
> colnames(weightsMVEDR) <- colnames(Scenarios)
> rownames(weightsMVEDR) <- paste0("MVEDR-", 0:nReturn)
```

Next compute Returns and Risks for all portfolios along the frontier

```
> Returns <- weightsMVEDR%*%mu
> Risks <- sqrt(diag(weightsMVEDR %*% Sigma %*% t(weightsMVEDR )))
```

Plot Risk and Returns:

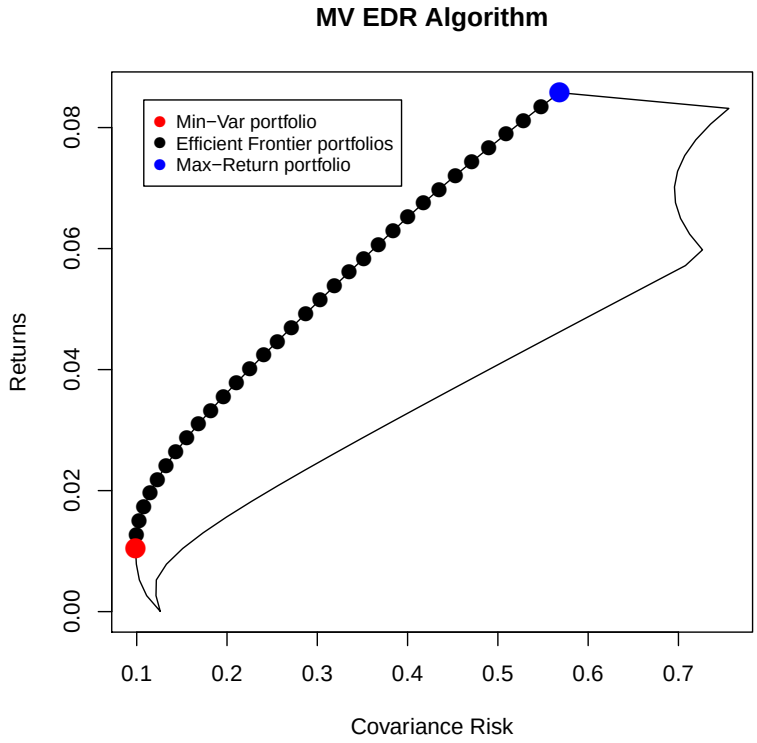


FIGURE 5.4: Equi-distant return frontier for the mean-variance Markowitz Portfolio

5.7 CRITICAL LINE ALGORITHM

The *Critical Line Algorithm* introduces multi-objective optimization methods. Its implementation can be found in the section ....

5.8 MAXIMUM SHARPE RATIO PORTFOLIO

The *Maximum Sharpe Ratio* portfolio is the portfolio on the efficient frontier with the highest reward/risk ratio. It can be found by solving the following optimization problem:

$$\max_{w} \frac{\mu' w}{\sqrt{w'\Sigma w}} \tag{5.9}$$

$$s.t.$$

$$1'w = 1$$

$$w_i \geq 0$$

We can straightforward implement this problem in AMPL.
R/AMPL Model File:

```
> modelSHARPE <- c(
  "param N ;",
  "param mu{1..N};",
  "param Sigma{1..N, 1..N} ;",
  "var w{1..N} >= 0, default 1/N ;",
  "maximize Objective: sum{k in 1..N} mu[k] * w[k] /",
     "sqrt ( sum{i in 1..N} sum{j in 1..N} w[i] * Sigma[i,j] * w[j] ) ;",
  "subject to Budget: sum{i in 1..N} w[i] = 1 ;")
> amplModelFile(model=modelSHARPE, project="myPortfolio")
```

Since the portfolio program has become a non-linear optimization program, we use the *MINOS* solver. Thus the specified solver in the AMPL run file has to be replaced:

```
> runSHARPE <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver minos ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runSHARPE, project="myPortfolio")
```

Since the Sharpe Ratio model file does not require additional parameters, we can generate and save the R/AMPL data file:

```
> Scenarios <- 100*LPP2005.RET[, 1:6]
> requiredData(modelSHARPE)
[1] "N"     "mu"    "Sigma"
>   N <- ncol(Scenarios)
>   Sigma <- cov(Scenarios)
>   mu <- colMeans(Scenarios)
> dataSHARPE <- dataAUTO(modelSHARPE)
> amplDataFile(data=dataAUTO(modelSHARPE), project="myPortfolio")
```

Solve the Portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsSHARPE <- as.numeric(scan("myPortfolio.txt"))
> names(weightsSHARPE) <- colnames(Scenarios)
```

The *Sharpe-Ratio* takes the value

```
> sharpeReturn <- (mu %*% weightsSHARPE)[1]
> sharpeRisk <- sqrt(weightsSHARPE %*% Sigma %*% weightsSHARPE)[1]
> sharpeRatio <- sharpeReturn / sharpeRisk
> sharpeRatio
```

```
[1] 0.18471
```

Summarize the result:

```
> SummarySHARPE <- c(
   TargetReturn = mu %*% weightsSHARPE,
   CovarianceRisk = sqrt ( weightsSHARPE %*% Sigma %*% weightsSHARPE ),
   HerfindahlIndex = 1 - weightsSHARPE %*% weightsSHARPE)
> SummarySHARPE
```

```
   TargetReturn  CovarianceRisk HerfindahlIndex
       0.028323        0.153339        0.577287
```

Note that the Sharpe ratio portfolio lies on the efficient frontier:

Plot the Risk/Reward Diagram:

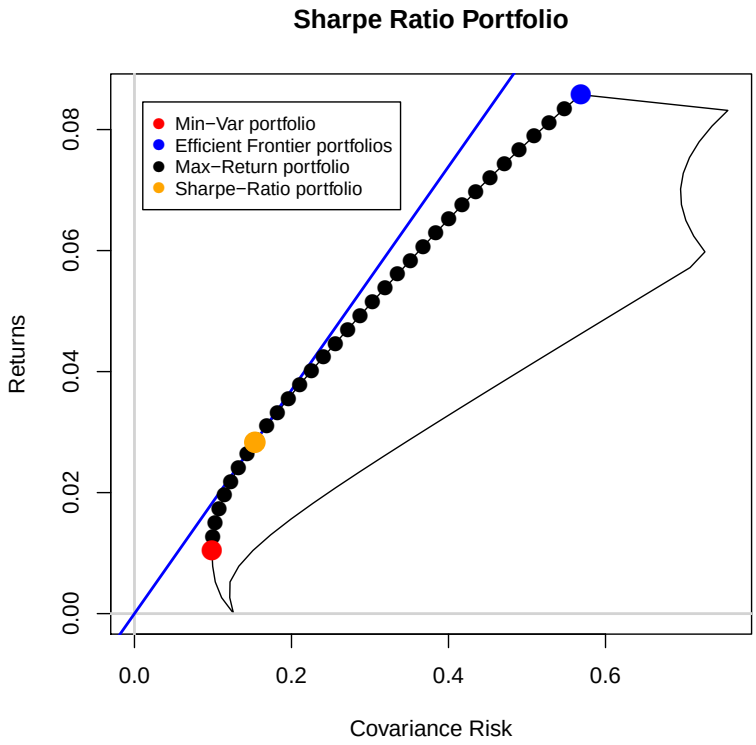
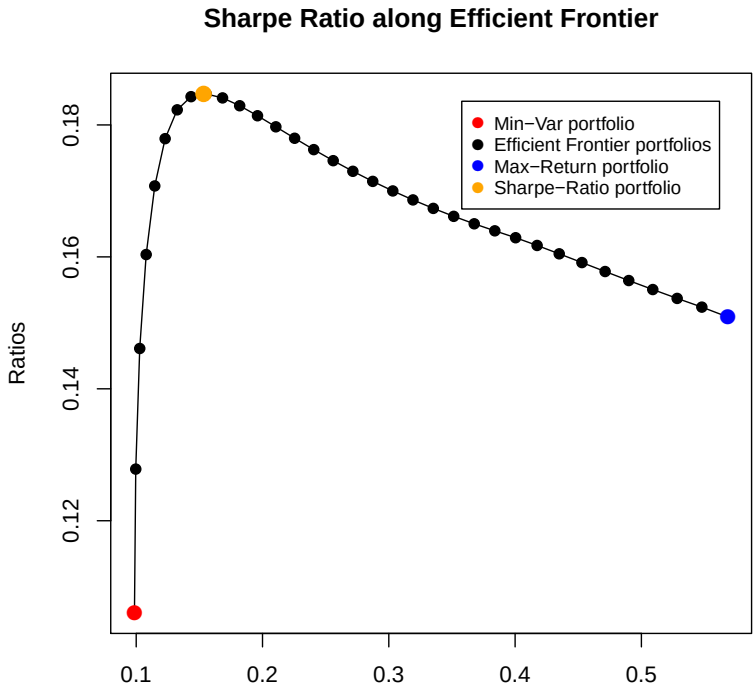


FIGURE 5.5: Position of the Sharpe-Ratio portfolio



5.9 QUADRATIC SHARPE-RATIO PORTFOLIO

You may have noticed that the problem in equation 5.9 is convex, but *non-linear*. In order to solve it, `minos` uses less efficient solving methods than achievable for linear or quadratic problems. Stoyanev, Rachev and Fabozzi showed that the optimization problem in equation 5.9 can easily be transformed into the following quadratic problem in order to reduce calculation time:

$$\min_{w,t} w'\Sigma w \tag{5.10}$$

$$s.t.$$

$$1'w = t$$
$$\mu' w = 1$$
$$w_i \geq 0$$
$$t \geq 0$$

The Sharpe-Ratio portfolio is then given by w/t.
Again, we can go straightforward to implementing this problem in AMPL:
R/AMPL model file:

```
> modelSHARPE2 <- c(
   "param N ;",
   "param mu{1..(N+1)};",
   "param Sigma{1..(N+1), 1..(N+1)} ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var t >= 0 ;",
   "minimize Objective:  sum{i in 1..N} sum{j in 1..N} w[i] * Sigma[i,j] * w[j];",
   "subject to Reward : sum{k in 1..N} mu[k] * w[k] = 1 ;",
   "subject to Budget: sum{i in 1..N} w[i] = t ;")
> amplModelFile(model=modelSHARPE2, project="myPortfolio")
```

R/AMPL run file:

```
> runSHARPE2 <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m]/t > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runSHARPE2, project="myPortfolio")
```

R/AMPL data file:

```
> Scenarios <- 100*LPP2005.RET[, 1:6]
> requiredData(modelSHARPE2)
[1] "N"     "mu"    "Sigma"
```

```
>   N <- ncol(Scenarios)
>   Sigma <- cov(Scenarios)
>   mu <- colMeans(Scenarios)
> dataSHARPE2 <- dataAUTO(modelSHARPE2)
> amplDataFile(data=dataAUTO(modelSHARPE2), project="myPortfolio")
```

Optimize the portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsSHARPE2 <- as.numeric(scan("myPortfolio.txt"))
> names(weightsSHARPE2) <- colnames(Scenarios)
> rbind(weightsSHARPE,weightsSHARPE2)
                SBI      SPI    SII     LMI MPI     ALT
weightsSHARPE     0 0.000469 0.1824 0.57529   0 0.24185
weightsSHARPE2    0 0.000469 0.1824 0.57529   0 0.24185
```

As expected, we get the same weights for the quadratic as for the non-linear problem.

5.10 MEAN-VARIANCE HULL

The Hull surrounds the feasible set. It consists of three parts:

- The efficient frontier
- The minimum variance locus
- The maximum variance locus

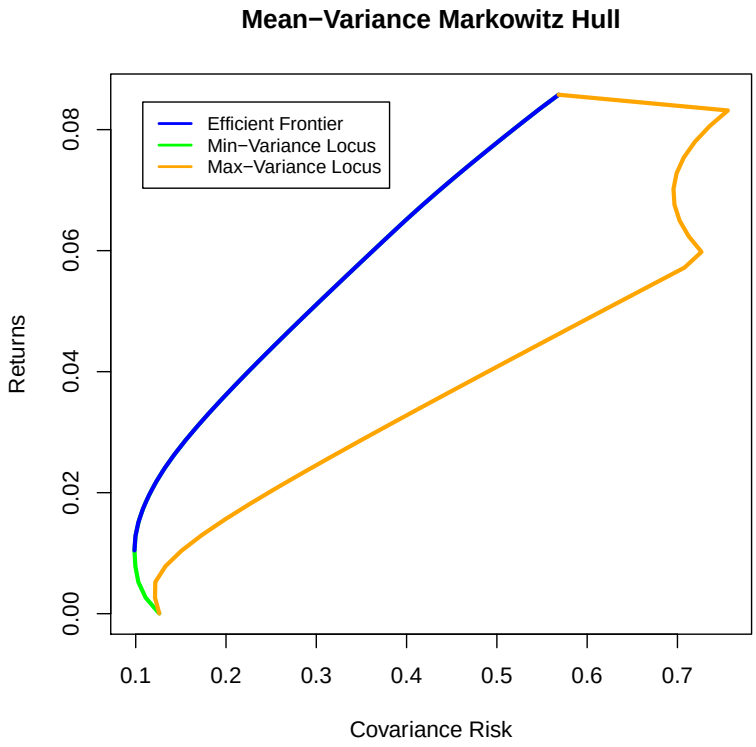


FIGURE 5.7: Max Sharpe Ratio or Tangency Mean-Variance Portfolio

The efficient frontier and the minimum variance locus can be easily calculated in one step. We optimize EDR portfolios, but now we start from the minimum return portfolio, and go up to the maximum return portfolio. Both portfolios consist of a single asset.
The calculation of the maximum variance locus is a bit more complicated:

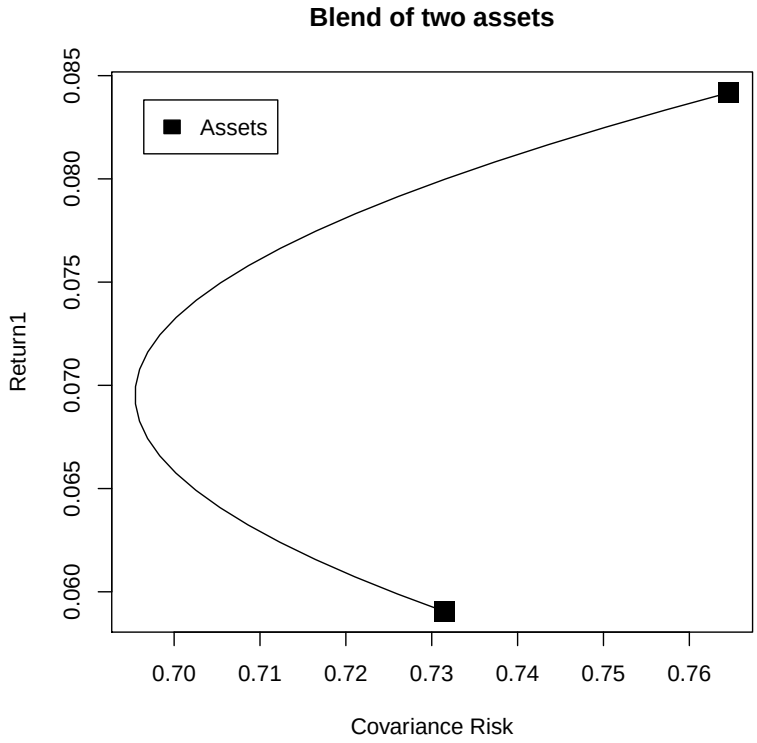


FIGURE 5.8: Blend between two individual assets.

Blends

Blends are portfolios that are a combination of two assets.
Since the variance is a convex risk measure, every combination of two blends cannot have a higher variance than both of the blends alone. This means that the maximum variance locus will only consist of blends. The union of the pairwise solutions then yields the rhs of the hull. This is also evident in figure 5.9:

Implementation of the Hull-calculation algorithm

Text....
The R/AMPL Model File and R/AMPL Run File are exactly like the ones from the equidistant return frontier:

```
> modelMVMINHULL <- c(
   "param minReturn ;",
   "param maxReturn ;",
```

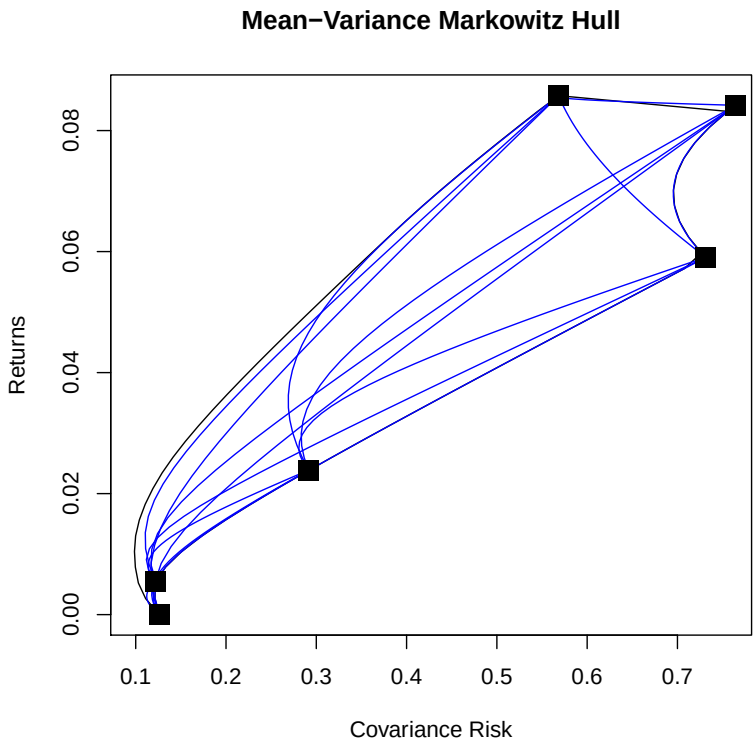


FIGURE 5.9: All blends inside the feasible set.

```
    "param nReturn ;",
    modelMV1)
> amplModelFile(model=modelMVMINHULL, project="myPortfolio")
```

The R/AMPL Run File:

```
> runMVMINHULL <- c(
    "model myPortfolio.mod ;",
    "data myPortfolio.dat ;",
    "option solver cplex;",
    "for {i in 0..nReturn} {",
    "  let targetReturn := minReturn + i*(maxReturn-minReturn)/nReturn ;",
    "  solve ;",
    "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
    "};",
    "exit ;")
> amplRunFile(run=runMVMINHULL, project="myPortfolio")
```

The difference between both calculations is that this time we are calculating the locus and the frontier at once. We therefore set minReturn to be

equal to min(mu). The R/AMPL Data File looks like this:

```
> Scenarios <- 100*LPP2005REC[, 1:6]
> requiredData(modelMVMINHULL)
[1] "minReturn"    "maxReturn"    "nReturn"      "N"            "mu"
[6] "Sigma"        "targetReturn"

>   N <- ncol(Scenarios)
>   mu <- colMeans(Scenarios)
>   Sigma <- cov(Scenarios)
>   targetReturn <- NA
>   minReturn <- min(mu)
>   maxReturn <- max(mu)
>   nReturn <- 33
> dataMVMINHULL <- dataAUTO(modelMVMINHULL)
> amplDataFile(data=dataMVMINHULL, project="myPortfolio")
```

Optimize the Portfolio and extract the Weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMVMINHULL <- matrix(as.numeric(scan("myPortfolio.txt")), byrow=TRUE, ncol=6)
> colnames(weightsMVMINHULL) <- colnames(Scenarios)
> rownames(weightsMVMINHULL) <- paste0("MINHULL-", 1:dim(weightsMVMINHULL)[1])
```

Next we compute the maximum variance locus of the Markowitz portfolio by the maximum risk intersection of all pairwise portfolios

```
> markowitzHull <- function(mu, Sigma, Return, Risk) {
   # Minimum Risks:
   minRisks <- Risk
   Returns <- risks <- Return
   # Maximum Risks:
   maxRisks <- rep(-Inf, length(Returns))
   nAssets <- ncol(Sigma)
   for (i in 1:(nAssets - 1)) {
     for (j in (i + 1):nAssets) {
       mu2 <- mu[c(i, j)]
       Sigma2 <- Sigma[c(i, j), c(i, j)]
       Index <- which(Returns >= min(mu2) & Returns <= max(mu2))
       if (length(Index) > 0) {
         Index <- (1:length(Returns))[Index]
         for (k in Index) {
           weights <- (Returns[k] - mu2[2])/(mu2[1] - mu2[2])
           weights <- c(weights, 1 - weights)
           Risk <- sqrt(weights %*% Sigma2 %*% weights)[[1]]
           maxRisks[k] <- max(maxRisks[k], Risk) }
       }
     }
   }
   # Hull:
   risk <- c(minRisks, rev(maxRisks[-1])[-1], minRisks[1])
   return <- c(Returns, rev(Returns[-1])[-1], Returns[1])
   hull <- cbind(Risks = risk, Returns = return)
   # Return Value:
   hull
 }
```

The input for the `markowitzHull()` are the column means (mu) oa fthe assets, the covariance matrix `Sigma`, and the `Return` and `Risk` values along the minimum variance locus and the efficient frontier.
Compute measures:

```
> Return <- Risk <- NULL
> for (i in 0:nReturn) {
   Return <- c(Return, mu %*% weightsMVMINHULL[i+1,])
   Risk <- c(Risk, sqrt(weightsMVMINHULL[i+1,] %*% Sigma %*% weightsMVMINHULL[i+1,]) ) }
> hull <- markowitzHull(mu, Sigma, Return, Risk)
```

CHAPTER 6

LOWER PARTIAL MOMENTS

6.1 INTRODUCTION

In this chapter we introduce portfolios with *Lower Partial Moments* risk measures. Topics include:

- LPMNONLIN - Nonlinear Lower Partial Moments Portfolio
- LPMSHORTFLL - Mean-Shortfall Portfolio
- LPMSEMIVAR - Mean-SemiVariance Portfolio
- LPMQUAD - Generalized Quadratic Lower Partial Moments Portfolio

Lower Partial Moments are risk measures that penalize returns below a certain threshold value. The use of such risk measure was already mentioned by Markowitz [1952] in a reference to semi-standard deviation portfolios. This was later formalized into a very general class of measures by Stone [1973], and Petersen and Satchel [19xx].

The lower partial moments framework, as we use here in the following was introduced by Fishburn [1977]. He defined the risk measure in the continuous case as

$$LPM_{a,\tau}(f) = E[\max(\tau - x, 0)^a] = \int_{-\infty}^{\tau} (\tau - x)^a f(x) dx$$

a is a positive parameter which represents the rate at which the deviations below the threshold τ are penalized and $f(x)$ is the distribution function of the returns. In the discrete case we get

$$LMP_{a,\tau}(x) = \frac{1}{m}\sum_{i=1}^{m}\max(\tau - x, 0)^a$$

Usually one uses this risk measure for portfolio optimization in a standardized form, i.e. the risk is raised to the power of $1/a$. The risk measure for portfolio problem can now be posed as follows:

$$LPM_{a,\tau}(x, w) = \Big(\frac{1}{S}\sum_{s=1}^{S}\max\Big(0, \tau - \sum_{i=1}^{N} r_{s,i}\, w_i\Big)^a\Big)^{1/a} \tag{6.1}$$

Special cases are $a = 0$ representing the shortfall probability or Safety-First model of Roy [1952]. $a = 1$ corresponds to the below target shortfall risk, and $a = 2$ to the shortfall variance which is equivalent to the semi-variance when $\tau = E(x)$. Note, when $a = 1$, a linear programming formulation exists. Note furthermore that the LPM is a convex risk measure only for $a \geq 1$.
This formulation of the LPM portfolio is nonlinear and thus inefficiently solvable. Nawrocki and Staples [1989] devised measures which transform the LPM function to allow for a quadratic programming implementation of the LPM portfolio. We will first present the nonlinear implementation of the LPM portfolio in chapter 2, and afterwards compare it to the quadratic implementation of Nawrocki and Staples.
Throughout this chapter we use as an example the Swiss pension fund benchmark portfolio. The data are part of the Rmetrics package `timeSeries` and are loaded together with the `fPortfolio` package.

6.2 NONLINEAR LOWER PARTIAL MOMENTS PORTFOLIO

Using equation 6.1, the Mean-LPM portfolio problem can be formulated as following:

$$\min_{w}\Big(\frac{1}{S}\sum_{s=1}^{S}\max\Big(0,\tau-\sum_{i=1}^{N}r_{s,i}\,w_i\Big)^{a}\Big)^{1/a}$$

$$s.t.$$

$$1'w=1$$
$$\mu' w\geq \overline{r}$$
$$w_i\geq 0$$

We now construct the R/AMPL model file straightforward:

```
> modelLPMNONLIN <- c(
  "# LPMNONLIN - Single Min-Risk Nonlinear LPM Portfolio:",
  "param N ;",
  "param S ;",
  "param Scenarios{1..S,1..N} ;",
  "param mu{1..N} ;",
  "param a ;",
  "param tau ;",
  "param targetReturn ;",
  "var w{1..N} >= 0, default 1/N ;",
  "minimize Objective: ( 1/S * sum{s in 1..S} (max ( tau -",
  "(sum{i in 1..N} Scenarios[s,i] * w[i]) , 0))^a )^(1/a) ;",
  "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
  "subject to Return: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;")
> amplModelFile(model=modelLPMNONLIN, project="myPortfolio")
```

Since the obective is non-linear, we need to include a non-linear solver such as minos in our R/AMPL run file (i.e. we could just use the run file from the Sharpe portfolio implementation):

```
> runLPMNONLIN <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver minos ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runLPMNONLIN, project="myPortfolio")
```

From the portfolio settings we construct the R/AMPL data file. In this first example we set $\tau=0$ and $a=1$.

```
> Scenarios <- 100*LPP2005.RET[117:377, 1:6]
> requiredData(modelLPMNONLIN)
```

```
[1] "N"            "S"            "Scenarios"    "mu"           "a"
[6] "tau"          "targetReturn"
```

```
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   a <- 1
>   tau <- 0
>   targetReturn <- mean(mu)
> dataLPMNONLIN <- dataAUTO(modelLPMNONLIN)
> amplDataFile(data=dataLPMNONLIN, project="myPortfolio")
```

Optimize the portfolio and extract the weights for $a = 1$:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsLPMNONLIN <- as.numeric(scan("myPortfolio.txt"))
> names(weightsLPMNONLIN) <- colnames(Scenarios)
> weightsLPMNONLIN
     SBI      SPI      SII      LMI      MPI      ALT
0.000000 0.005578 0.133504 0.472620 0.000000 0.388298
```

As expected, if we set $a = 2$, we get different weights:

```
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   a <- 2
>   tau <- 0
>   targetReturn <- mean(mu)
> dataLPMNONLIN <- dataAUTO(modelLPMNONLIN)
> amplDataFile(data=dataLPMNONLIN, project="myPortfolio")
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsLPMNONLIN <- as.numeric(scan("myPortfolio.txt"))
> names(weightsLPMNONLIN) <- colnames(Scenarios)
> weightsLPMNONLIN
    SBI     SPI     SII     LMI     MPI     ALT
0.00000 0.00000 0.29799 0.34011 0.00000 0.36190
```

6.3 LINEAR MEAN-SHORTFALL PORTFOLIO

If $a = 1$, equation 6.1 represents the below target shortfall portfolio. This portfolio allows for a linear implementation:

$$\min_{w} \frac{1}{S} \sum_{s=1}^{S} d_s$$

$$s.t.$$

$$\tau - \sum_{i=1}^{N} w_i r_{i,s} \leq d_s$$

$$1' w = 1$$

$$\mu' w \geq \overline{r}$$

$$w_i \geq 0$$

$$d_s \geq 0$$

R/AMPL model file:

```
> modelLPMSHORTFALL <- c(
  "# LPMSHORTFALL - Single Min-Risk Shortfall LPM Portfolio:",
  "param N ;",
  "param S ;",
  "param Scenarios{1..S,1..N} ;",
  "param mu{1..N} ;",
  "param tau ;",
  "param targetReturn ;",
  "var w{1..N} >= 0, default 1/N ;",
  "var d{1..S} >= 0 ;",
  "minimize Objective:  1/S * sum{s in 1..S} d[s] ;",
  "subject to Shortfall{s in 1..S}: tau- sum{i in 1..N} w[i] *Scenarios[s,i]<= d[s] ;",
  "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
  "subject to Return: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;")
> amplModelFile(model=modelLPMSHORTFALL, project="myPortfolio")
```

Since we now have a linear implementation, the R/AMPL run file is just our standard linear run file:

```
> runLPMSHORTFALL <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runLPMSHORTFALL, project="myPortfolio")
```

Since all parameters are already specified, we can just create our R/AMPL data file with $\tau = 0$:

```
> requiredData(modelLPMSHORTFALL)
[1] "N"            "S"            "Scenarios"    "mu"           "tau"
[6] "targetReturn"
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   tau <- 0
>   targetReturn <- mean(mu)
> dataLPMSHORTFALL <- dataAUTO(modelLPMSHORTFALL)
> amplDataFile(data=dataLPMSHORTFALL, project="myPortfolio")
```

Optimize the portfolio and extract the weights

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsLPMSHORTFALL <- as.numeric(scan("myPortfolio.txt"))
> names(weightsLPMSHORTFALL) <- colnames(Scenarios)
> weightsLPMSHORTFALL
     SBI      SPI      SII      LMI      MPI      ALT
0.000000 0.005578 0.133504 0.472620 0.000000 0.388298
```

As expected, the weights are the same as in the nonlinear approach with $a = 1$.

6.4 MEAN-SEMIVARIANCE PORTFOLIO

If $a = 2$, equation 6.1 represents the one-sided semivariance portfolio.
This portfolio allows for a quadratic implementation:

$$\min_w \frac{1}{S}\sum_{s=1}^{S} d_s^2$$

$$s.t.$$

$$\tau - \sum_{i=1}^{N} w_i r_{i,s} \leq d_s$$

$$1'w = 1$$

$$\mu' w \geq \overline{r}$$

$$w_i \geq 0$$

$$d_s \geq 0$$

R/AMPL model file:

```
> modelLPMSEMIVAR <- c(
  "# LPMSEMIVAR - Single Min-Risk SEMIVAR LPM Portfolio:",
  "param N ;",
  "param S ;",
  "param Scenarios{1..S,1..N} ;",
  "param mu{1..N} ;",
  "param tau ;",
  "param targetReturn ;",
  "var w{1..N} >= 0, default 1/N ;",
  "var d{1..S} >= 0 ;",
  "minimize Objective:  1/S * sum{s in 1..S} d[s]*d[s] ;",
  "subject to Semivar{s in 1..S}: tau- sum{i in 1..N} w[i] *Scenarios[s,i]<= d[s] ;",
  "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
  "subject to Return: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;")
> amplModelFile(model=modelLPMSEMIVAR, project="myPortfolio")
```

Again, we can just use our usual linear R/AMPL run file:

```
> runLPMSEMIVAR <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runLPMSEMIVAR, project="myPortfolio")
```

From the portfolio settings we construct the R/AMPL data file with $\tau = 0$:

```
>   requiredData(modelLPMSEMIVAR)
```

```
[1] "N"            "S"            "Scenarios"    "mu"           "tau"
[6] "targetReturn"
```

```
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   tau <- 0
>   targetReturn <- mean(mu)
> dataLPMSEMIVAR <- dataAUTO(modelLPMSEMIVAR)
> amplDataFile(data=dataLPMSEMIVAR, project="myPortfolio")
```

Optimize the portfolio and extract the weights

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsLPMSEMIVAR <- as.numeric(scan("myPortfolio.txt"))
> names(weightsLPMSEMIVAR) <- colnames(Scenarios)
> weightsLPMSEMIVAR
    SBI     SPI     SII     LMI     MPI     ALT
0.00000 0.00000 0.29799 0.34012 0.00000 0.36190
```

As expected, the weights are the same as in the nonlinear approach with $a = 2$.

6.5 QUADRATIC LOWER PARTIAL MOMENTS PORTFOLIO

Taking inspiration from the quadratic Markowitz portfolio, Nawrocki (1991,1992) proposed a new approach to calculating the LPM of a portfolio that includes the co-lower partial moments of the individual assets.
The *co-lower partial moment* (*CLPM*) between two assets is defined as

$$\begin{aligned} & CLPM_{a,\tau}(f(x_i,x_j)) \\ &= E[\max(\tau - x_i, 0)^{a-1}(\tau - x_j)] \\ = \int_{-\infty}^{\tau} \int_{-\infty}^{\infty} & (\tau - x_i)^{a-1}(\tau - x_j) f(x_i) f(x_i, x_j) d\, x_i d\, x_j \end{aligned}$$

In the discrete case we get

$$CLMP_{a,\tau}(x_i,x_j) = \frac{1}{m}\sum_{i=1}^{m} \max(\tau - x, 0)^{a-1}(\tau - x)$$

The LPM of the portfolio is now defined as

$$\begin{aligned} & LPM_{a,\tau} \\ = \sum_{i=1}^{N}\sum_{j=1}^{N} & w_i w_j CLMP_{a,\tau}(x_i,x_j) \\ & = w^T \cdot CLMP_{a,\tau} \cdot w \end{aligned}$$

where $CLMP_{a,\tau}$ is the matrix of all $CLMP_{a,\tau}(x_i,x_j)$. For $a = 2$, this approach is equivalent to the nonlinear formulation of equation 6.1.
We can now directly use the Mean-Variance portfolio model and replace the Σ-matrix with the $CLMP_{a,\tau}$-matrix. The R/AMPL model file looks as following:

```
> modelLPMQUAD <- c(
   "param N ;",
   "param mu{1..N} ;",
   "param CLMP{1..N,1..N} ;",
   "param targetReturn ;",
   "var w{1..N} >= 0, default 1/N ;",
   "minimize Objective: sum{i in 1..N} sum{j in 1..N} w[i] * CLMP[i,j] * w[j] ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to Return: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;")
> amplModelFile(model=modelLPMQUAD, project="myPortfolio")
```

Note that this is exactly the same model file as for the mean-variance portfolio, we just renamed the Σ-matrix to $CLMP$.
Consequently we can again just use the linear R/AMPL run file from the Markowitz approach:

```
> runLPMQUAD <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runLPMQUAD, project="myPortfolio")
```

From the portfolio settings we construct the R/AMPL data file. The $CLMP_{a,\tau}$-matrix can be calculated with the ...function....
Note that it might be reasonable to normalize the $CLMP$-matrix for numerical stability, since for high values for a, its entries become very small:

```
> requiredData(modelLPMQUAD)
[1] "N"            "mu"           "CLMP"         "targetReturn"
>   N <- ncol(Scenarios)
>   mu <- colMeans(Scenarios)
>   tau <- 0
>   a <- 2
>   targetReturn <- mean(mu)
>   CLMP <- assetsLPM(Scenarios,tau,a)$Sigma
>   #NORMALIZE#
>   CLMP <- CLMP/(det(CLMP))^(1/N)
> dataLPMQUAD <- dataAUTO(modelLPMQUAD)
> amplDataFile(data=dataLPMQUAD, project="myPortfolio")
```

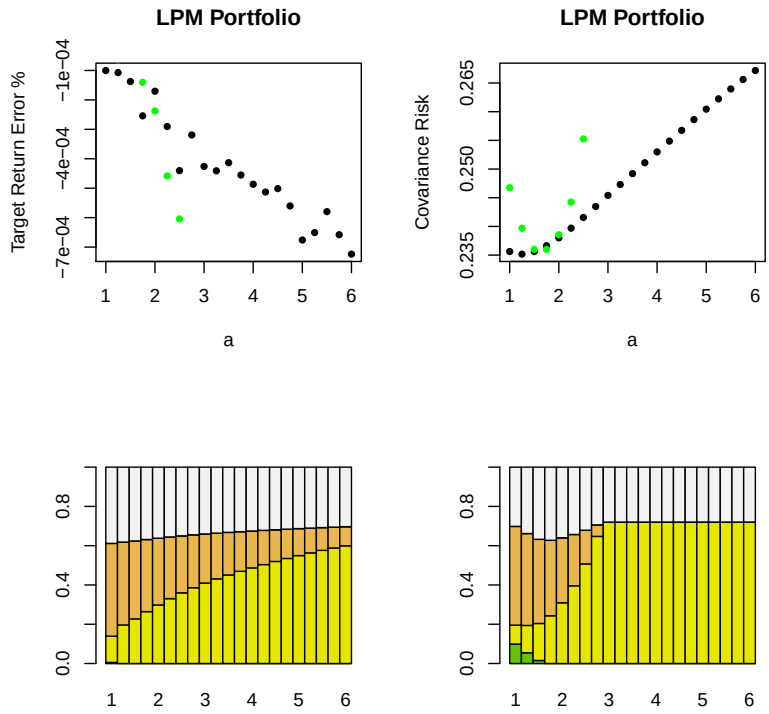
Optimize the portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsLPMQUAD <- as.numeric(scan("myPortfolio.txt"))
> names(weightsLPMQUAD) <- colnames(Scenarios)
> weightsLPMQUAD
    SBI     SPI     SII     LMI     MPI     ALT
0.00000 0.00000 0.30934 0.33097 0.00000 0.35969
```

As expected, the weights are the same as for the semi-variance approach (up to the second digit):

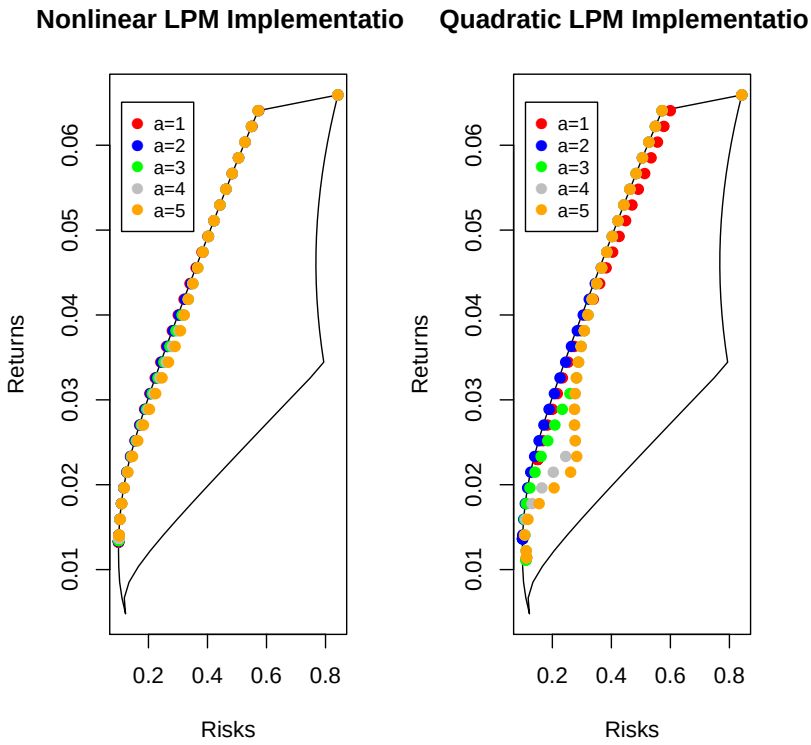
```
> weightsLPMSEMIVAR
    SBI     SPI     SII     LMI     MPI     ALT
0.00000 0.00000 0.29799 0.34012 0.00000 0.36190
```

IMPORTANT: If a gets to big, the CLMP-matrix will become degenerate, and cplex is not able to solve the problem anymore.



6.6 DEPENDENCE ON a

We will now compare the different approaches for different a.
To do.....
Plot the results as a function of a:


Nonlinear LPM Implementatio
Quadratic LPM Implementatio
a=1
a=2
a=3
a=4
a=5
Returns
Risks
0.01
0.02
0.03
0.04
0.05
0.06
0.2
0.4
0.6
0.8

PART III

ROBUST PORTFOLIO ESTIMATIONS

CHAPTER 7

COVARIANCE ROBUSTIFICATION

Mean-variance portfolios constructed using the sample mean and covariance matrix of asset returns often perform poorly out-of-sample due to estimation errors in the covariance matrix. As a consequence, minimum-variance portfolios may yield unstable weights that fluctuate substantially over time. This loss of stability may also lead to extreme portfolio weights and dramatic swings in weights with only minor changes in expected returns or the covariance matrix. Consequentially, we observe frequent re-balancing and excessive transaction costs.
Robust statistics provides an alternative approach to classical statistical methods. The idea behind robust statistics is to produce estimators that are not unduly affected by small departures from model assumptions. In this chapter we will introduce several robust covariance estimators that may reduce estimation errors in the covariance matrix.

7.1 INTRODUCTION

As discussed above, the efficient mean-variance portfolio model is defined as following:

$$\begin{aligned} & \min_{w} w'\Sigma w \\ s.t. & \\ & 1'w = 1 \\ & \mu' w = \overline{r} \\ & w_i \geq 0 \end{aligned} \tag{7.1}$$

where Σ is the covariance matrix. When estimating the covariance matrix, we calculated the sample covariance matrix that is defined as

$$\Sigma_{i,j} = \frac{1}{n-1} \sum_{s=1}^{S} (X_{i,s} - \bar{X}_i)(X_{j,s} - \bar{X}_j) \tag{7.2}$$

The calculation is implemented in the function `cov` that we used. This estimator is optimal when the returns come from a multivariate normal distributions. For other distributions and in the presence of outliers, this estimator is known to be notoriously error-prone. In addition, the mean-variance portfolio model is very sensitive to the covariance estimates. Therefore, a better approach is to apply robust covariance estimation methods.
Our AMPL model file calculating the mean-variance portfolio looks like this:

```
> modelMV1 <- c(
   "param N ;",
   "param mu{1..N} ;",
   "param Sigma{1..N,1..N} ;",
   "param targetReturn ;",
   "var w{1..N} >= 0, default 1/N ;",
   "minimize Objective: sum{i in 1..N} sum{j in 1..N} w[i] * Sigma[i,j] * w[j] ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to Reward: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;")
```

Since the estimation of the covariance matrix plays no role in our model file, we obviously just have to replace the covariance matrix in our data file in order to work with a more robust estimator of the covariance matrix. For example, if we want to use Spearman's rank estimator, we just use the implemented R function and replace the covariance matrix in the data file:

```
> #Scenarios <- 100*LPP2005REC[,1:6]
>
> data <- read.csv(file="assetReturns.csv",header=T,sep=",",row.names=)
> row.names(data)=data[,1]
> Scenarios <- as.timeSeries(data[1509:1809,2:dim(data)[2]])
> requiredData(modelMV1)
[1] "N"            "mu"           "Sigma"        "targetReturn"
>   N <- ncol(Scenarios)
>   mu <- colMeans(Scenarios)
>   targetReturn <- mean(mu)
>   Sigma <- cov(Scenarios, method="spearman")
> dataMV1 <- dataAUTO(modelMV1)
> amplDataFile(data=dataMV1, project="myPortfolio")
```

In the following, we will present several ways to calculate a robust covariance matrix. The robust matrix can always be inserted into the data file as presented in this example. The following approaches are discusssed:

- Rank Correlation Estimators
- High Breakdown Point Estimators
- Shrinkage Estimators
- Bayesian Change-Point Estimator

7.2 RANK CORRELATION ESTIMATORS

The sample covariance relies on the normality of the sample points, but asset returns are in general not normally distributed. Instead of using the measured returns, rank correlation estimators use the rank of the returns to compute the correlation and are thus less dependent on the underlying distribution of the returns.
The covariance can between two assets can be written as a function of the correlation function $\rho(X_i, X_j)$:

$$\sigma_{i,j} = \frac{\rho(X_i, X_j)}{\sigma_i, \sigma_j} \tag{7.3}$$

Pearson's correlation estimator relies on an assumed underlying normal distribution. Rank correlation estimators are more conservative and make no assumptions about the underlying distribution. The most prominent ones are Spearman's and Kendall's correlation estimators.
The *Spearman correlation* is computed from:

$$\rho_S(X_i, X_j) = \frac{1/(N-1)\sum_{s=1}^{S}(\mathrm{rg}(X_{i,s}) - (N+1)/2)(\mathrm{rg}(X_{j,s}) - (N+1)/2)}{\sigma_{\mathrm{rg}_i}, \sigma_{\mathrm{rg}_j}} \tag{7.4}$$

The *Kendall correlation* is computed from:

$$\tau_S(X_i, X_j) = \frac{1}{N(N-1)/2} \sum_{1 \le s < t \le N} \mathrm{sgn}(X_{i,s} - X_{i,t}) \cdot \mathrm{sgn}(X_{j,s} - X_{j,t}) \tag{7.5}$$

The `cov`-function from the `stats`-package includes methods to compute Spearman's r resp. Kendall's τ. It is quite easy to compute the corresponding covariance method from these quantities:

```
> SpearmanR <- cov(Scenarios, method="spearman")
> rhoSpearman <- t(SpearmanR/sqrt(diag(SpearmanR)))/sqrt(diag(SpearmanR))
> SigmaSpearman <- t(rhoSpearman*sqrt(diag(cov(Scenarios))))*sqrt(diag(cov(Scenarios)))
> Kendalltau <- cov(Scenarios, method="kendall")
> rhoKendall <- t(Kendalltau/sqrt(diag(Kendalltau)))/sqrt(diag(Kendalltau))
> SigmaKendall <- t(rhoKendall*sqrt(diag(cov(Scenarios))))*sqrt(diag(cov(Scenarios)))
```

Risk/Reward Plot:

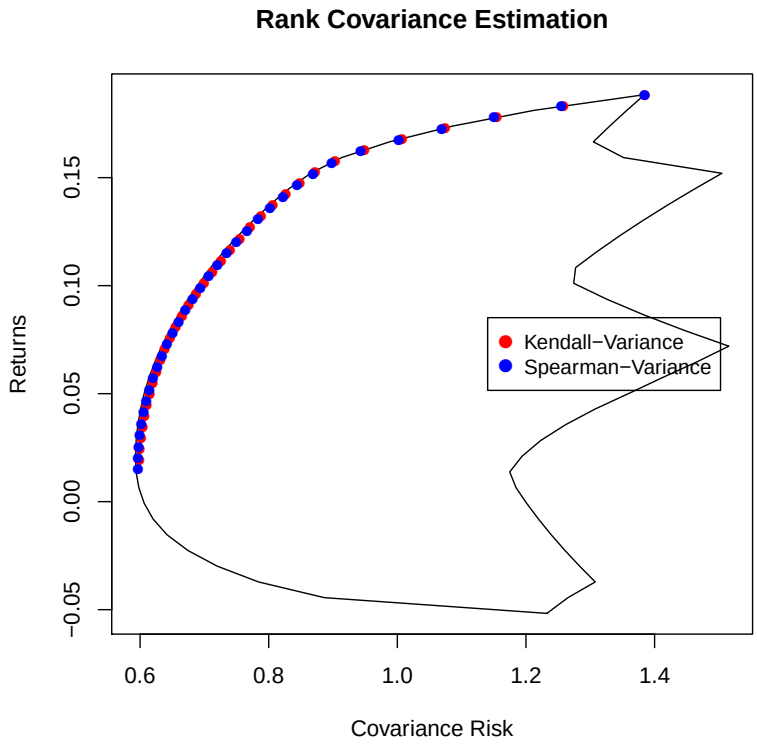


FIGURE 7.1: Efficient Mean-Variance portfolios that are optimized with Spearman- and Kendall-covariance matrices.

7.3 HIGH BREAKDOWN POINTS ESTIMATORS

A common source for estimation errors in statistics are outliers. It is well documented that even a single observation that deviates from the assumed normal distribution could deteriorate the sample covariance estimator, i.e. the sample covariance is not robust against outliers. High breakdown points estimators are designed to withstand a certain amount of defective sample points.
Rousseeuw (1984) The *Minimum Covariance Determinant Estimator* (MCD) for example finds the subset of data points that minimizes the determinant of the sample covariance matrix. For a specific number h between $(S+N+1)/2$ and S, the estimator finds the subset H_0 with h data points such that

$$H_0 = \arg\min_H \ \det\big(\operatorname{cov}(\mathbf{X}_i | i \in H)\big) \tag{7.6}$$

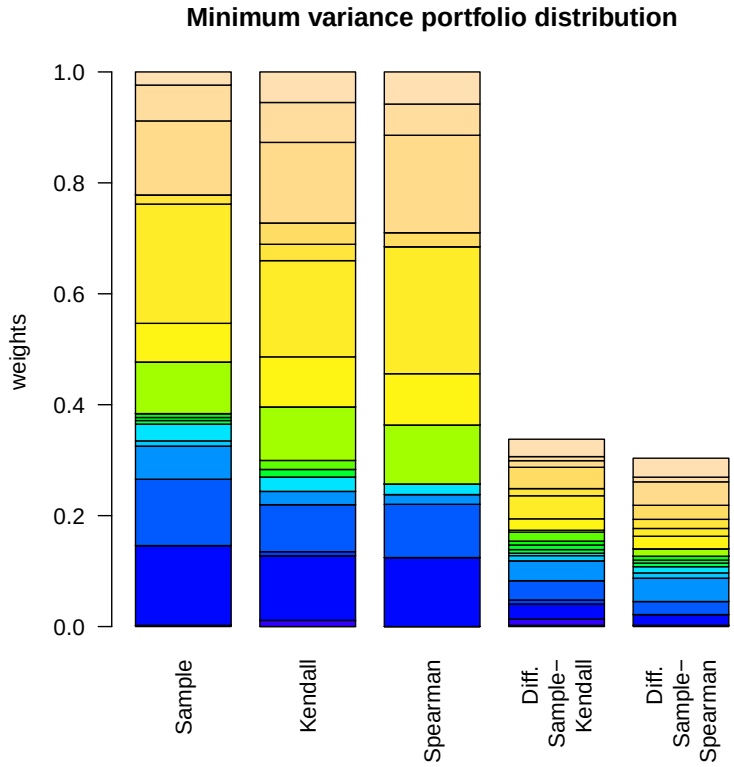


FIGURE 7.2: Comparison of the weight distribution of the minimum-variance portfolio optimized with the sample covariance estimator, the Spearman estimator and the Kendall estimator. Also the weight deviations of the Spearman and Kendall portfolio with respect to the sample covariance portfolio are displayed.

where cov is the sample covariance estimator. Obviously, this estimator does not work if we have more assets than data points.
Rousseeuw (1984) The *Minimum Volume Ellipsoid Estimator* (MVE) is similar to the MCD estimator: It also minimizes the determinant of its covariance matrix estimation $\mathbf{C}$, but the $\mathbf{C}$ is subject to the following constraint:

$$\max_{t\in\mathbb{R}^p} \sum_{i=1}^{S} \Theta\Big(c^2 - (X_S - t)^t \mathbf{C}^{-1} (X_S - t)\Big) \geq h \tag{7.7}$$

where $\Theta(\cdot)$ is the Heaviside function, c is a constant depending on N, and h a number between $S/2$ and S determining the number of data points that have to be included inside the ellipsoid.
Both estimators are included in the R-package `MASS`:

```
> SigmaMCD <- MASS::cov.rob(x = Scenarios, method = "mcd", quantile.used = )$cov
> SigmaMVE <- MASS::cov.rob(x=Scenarios, method="mve")$cov
```

Another high breakpoint estimator is the *Orthogonalized Gnanadesikan-Kettenring Estimator* (*OGK*). It uses eigenvalue decomposition and scaling in order lessen the influence of outliers. The `rrcov` contains a function to calculate the OGK estimate:

```
> SigmaOGK <- robustbase::covOGK(X = Scenarios, sigmamu = robustbase::scaleTau2)$cov
```

The *Nearest Neighbor Variance Estimator* (*NNVE*) was introduced by Wang & Raftery in 2002. It measures the "outlyingness" of a data point by the distance between the point and its Kth nearest neighbor in order to avoid defective data points. An implementation of the estimator can be found in the R-package `fAssets`:

```
> require(fAssets)
> SigmaNNVE <- assetsMeanCov(x=Scenarios, method="nnve")$cov
```

Risk/Reward Plot:

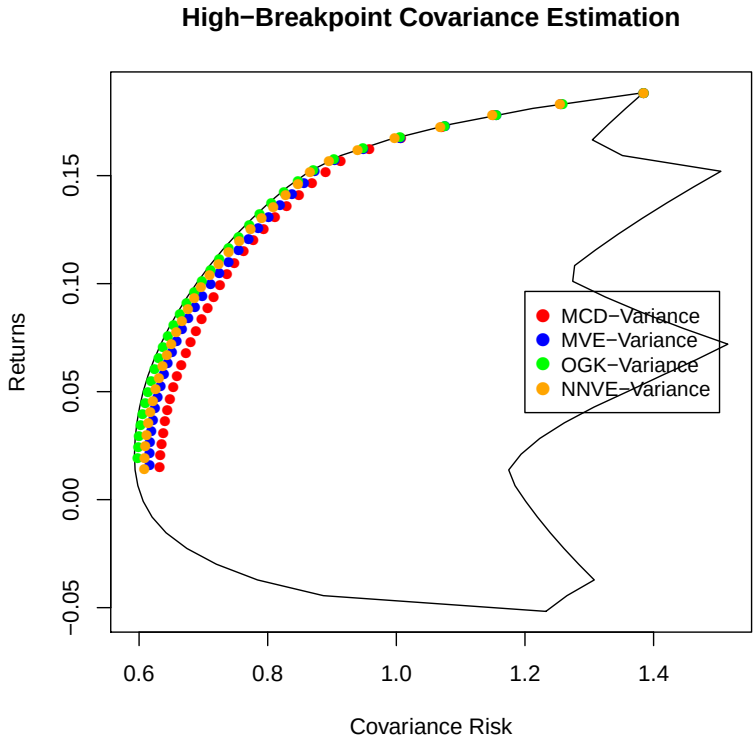


FIGURE 7.3: Efficient Mean-Variance portfolios that are optimized with Spearman- and Kendall-covariance matrices.

7.4 SHRINKAGE ESTIMATORS

If the number of time series records is small and the number of considered assets increases, then the sample estimator of covariance becomes more and more unstable. Specifically, it is possible to provide estimators that improve considerably upon the maximum likelihood estimate in terms of mean-squared error. Moreover, when the number of records is smaller than the number of assets, the sample estimate of the covariance matrix becomes singular.

A simple version of a shrinkage estimator of the covariance matrix is constructed as follows. We consider a convex combination of the empirical estimator with some suitable chosen target, e.g. the diagonal matrix. Subsequently, the mixing parameter is selected to maximize the expected accuracy of the shrinked estimator. This can be done by cross-validation, or by using an analytic estimate of the shrinkage intensity. Apart from increased efficiency, the shrinkage estimate has the additional advantage

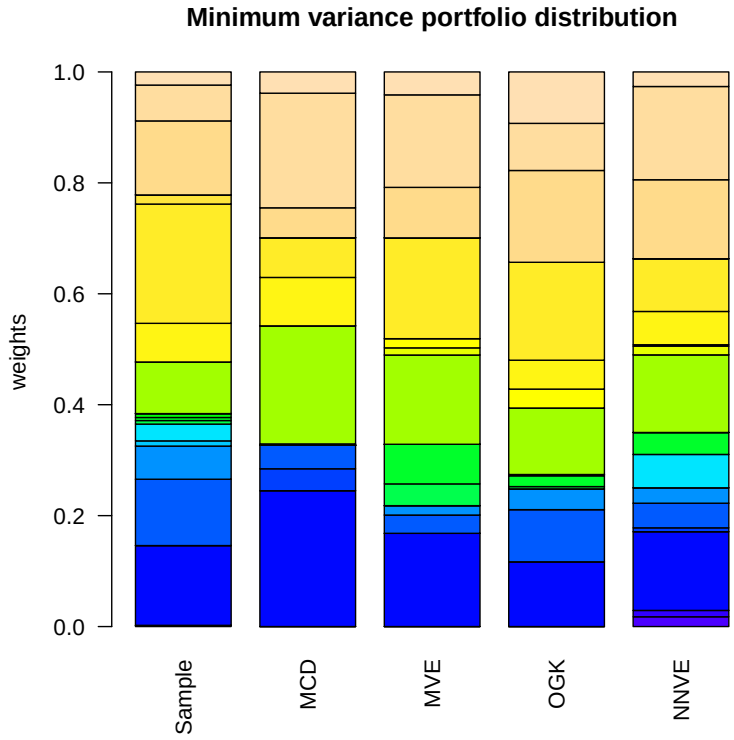


FIGURE 7.4: Comparison of the weight distribution of the minimum-variance portfolio optimized with the sample covariance estimator, the MCD, the MVE, the OGK and the NNVE covariance estimator.

that it is always positive definite and well conditioned.
More sophisticated shrinkage estimators include the shrinkage estimator by Schaefer and Strimmer or the bagged covariance estimator (Breiman 1996 *Bagging Predictors*). Both are implemented in the R-package `fAssets`:

```
> require(fAssets)
> SigmaShrink <- assetsMeanCov(x=Scenarios, method="shrink")$cov


> SigmaBagged <- assetsMeanCov(x=Scenarios, method="bagged", baggedR=100)$cov
```

Risk/Reward Plot:

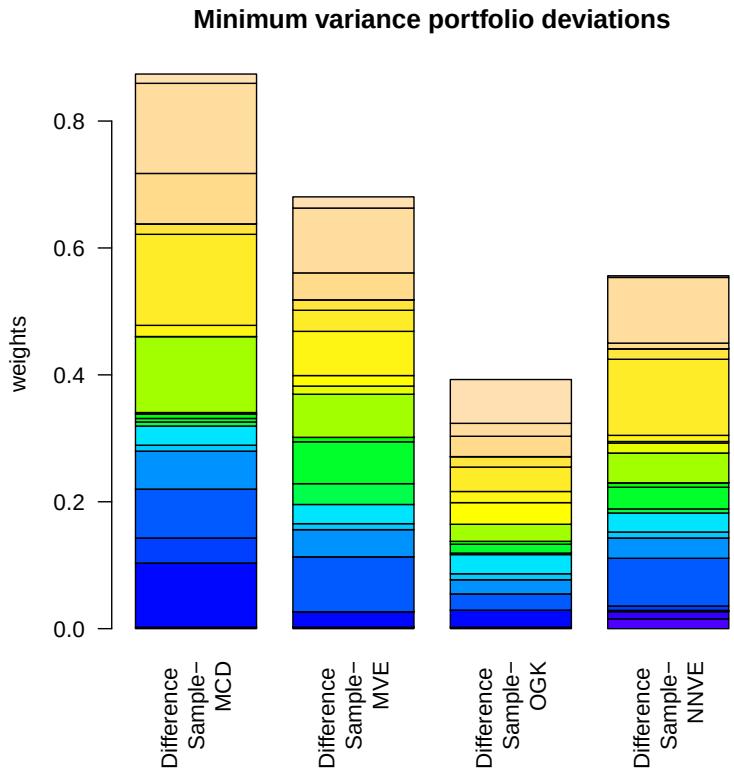


FIGURE 7.5: Comparison of the weight deviations of the minimum-variance portfolios that were optimized with the MCD, the MVE, the OGK and the NNVE covariance estimator with respect to the sample covariance estimator.

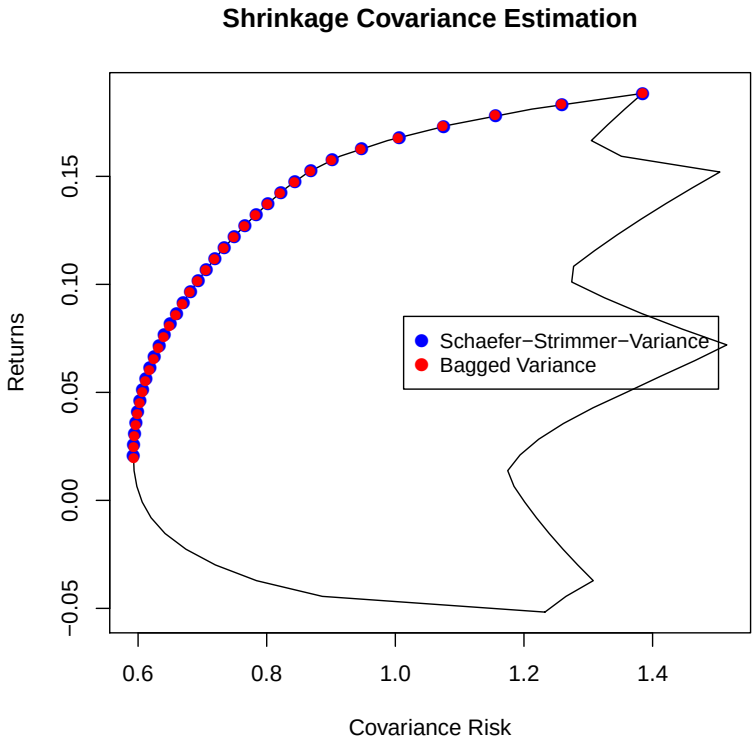


FIGURE 7.6: Efficient Mean-Variance portfolios that are optimized using the shrinkage estimator by Schaefer and Strimmer, and the bootstrap estimator by Breiman.

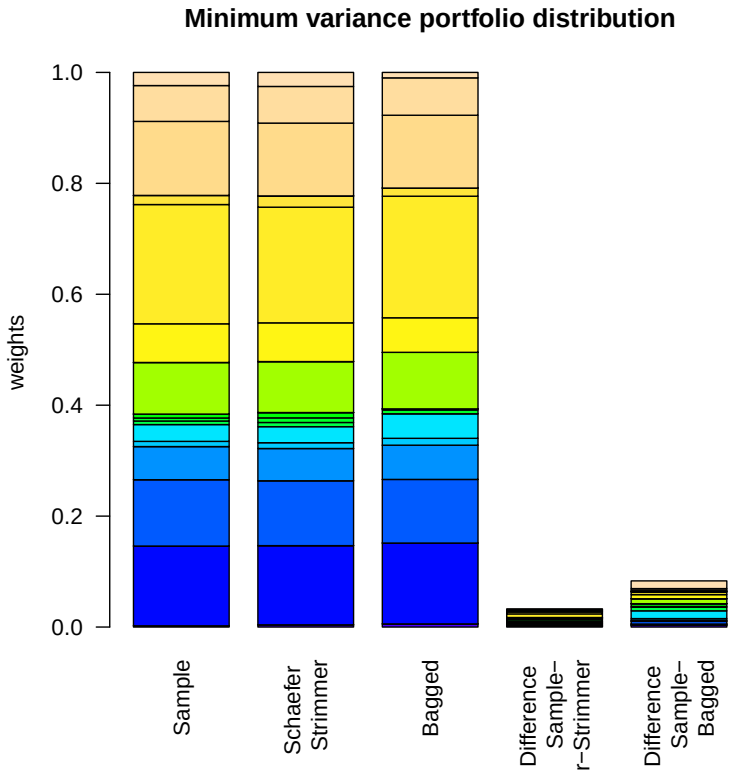


FIGURE 7.7: Comparison of the weight distribution of the minimum-variance portfolio optimized with the sample covariance estimator, the MCD, MVE and OGK covariance estimator. Also the weight deviations of the MCD, MVE and OGK portfolio with respect to the sample covariance portfolio are displayed.

CHAPTER 8

M AND S ESTIMATORS

8.1 INTRODUCTION

Motivation for this work, see abstract from:

Portfolio Selection with Robust Estimation
Victor DeMiguel, Francisco J. Nogales
Submitted to Operations Research Manuscript OPRE-2007-02-106

As meantioned before, mean-variance portfolios constructed using the sample mean and covariance matrix of asset returns perform poorly out-of-sample due to estimation error. Moreover, it is commonly accepted that estimation error in the sample mean is much larger than in the sample covariance matrix. For this reason, practitioners and researchers have recently focused on the minimum-variance portfolio, which relies solely on estimates of the covariance matrix, and thus, usually performs better out-of-sample. But even the minimum-variance portfolios are quite sensitive to estimation error and have unstable weights that fluctuate substantially over time. In this paper, we propose a class of portfolios that have better stability properties than the traditional minimum-variance portfolios. The proposed portfolios are constructed using certain robust estimators and can be computed by solving a single nonlinear program, where robust estimation and portfolio optimization are performed in a single step. We show analytically that the resulting portfolio weights are less sensitive to changes in the asset-return distribution than those of the traditional minimum-variance portfolios. Moreover, our numerical results on simulated and empirical data confirm that the proposed portfolios are more stable than the traditional minimum-variance portfolios, while preserving (or slightly improving) their relatively good out-of-sample performance.

8.2 M PORTFOLIOS

The first class of portfolios proposed by DeMiguel and Nogales is based on the robust M-estimators. For a given portfolio the M-estimator for the portfolio's risk is

$$\begin{aligned} \min_{w,m} \frac{1}{S}\sum_{i=1}^{S}\rho(w' r_s - m) \qquad (8.1)\\ s.t. \\ 1'w = 1 \\ \mu' w = \overline{r} \\ w \geq 0 \end{aligned}$$

where the loss function ρ is a convex symmetric function with an unique minimum at zero, and m is the M-estimator of the portfolio return

$$m = \arg\min_{m} \frac{1}{S}\sum_{i=1}^{S}\rho(w' r_s - m) \qquad (8.2)$$

Note, particular cases of M-estimators are the sample mean and variance, which are obtained for $\rho(r) = r^2/2$, and the median and MAD portfolio, for $\rho(r) = |r|$).
Loss functions that can be used to compute M-estimators include: L_p, L_1, L_2, Huber, Cauchy, Welsch, see DeMiguel and Nogales.

8.3 HUBER LOSS

As an example, we present the implementation of the M-estimator portfolio under Huber loss because of its good out-of-sample performance. The Huber loss function looks as following:

$$\rho_\delta(x) = \begin{cases} \frac{1}{2}x^2 & \text{for} |x| \leq \delta, \\ \delta(|x| - \frac{1}{2}\delta) & \text{otherwise.} \end{cases} \qquad (8.3)$$

To receive our optimization model, we can go straightforward and plug the loss function into equation 8.5. In order to implement this model in AMPL, we have to rely on piecewise linear programming in order to model the Huber loss function. The implementation of piecewise linear functions in AMPL can be done quite easily, but we need a little trick to account for the quadratic part of the Huber loss:
AMPL allows to multiply a defined function with different factors, separated through breakpoints. Since the Huber loss contains a linear part

and a quadratic part, i.e. we have two different types of functions, we can separate these two parts into a sum and write it as following:

$$\rho_\delta(x) = \begin{cases} 1 \cdot \frac{1}{2}x^2 & \text{for} |x| \leq \delta, \\ 0 \cdot \frac{1}{2}x^2 & \text{otherwise,} \end{cases} + \begin{cases} 0 \cdot \delta(|x| - \frac{1}{2}\delta) & \text{for} |x| \leq \delta, \\ 1 \cdot \delta(|x| - \frac{1}{2}\delta) & \text{otherwise,} \end{cases} \tag{8.4}$$

i.e. we now have two different piecewise defined functions with a breakpoint at the same point δ. To implement this in AMPL is quite easy with the syntax for piecewise linear functions. Such a function is defined in AMPL as following:

```
>         "<<Break;  slope1, slope2 >> x"
```

This expression describes a piecewise linear function such as in figure 8.1. Inside the braces, we have three parameters: The slope of the first linear segment, the slope of the second linear segment, and the breakpoint, where the first segment ends and the second starts. The expression inside the brackets is then multiplied with the variable.
If we also linearize $|w'r_s - m|$, we then get the following optimization problem:

$$\begin{aligned} \min_{w,m,x} \frac{1}{S}\sum_{i=1}^{S} \rho(x_s) & \qquad (8.5) \\ s.t. & \\ 1'w &= 1 \\ \mu' w &= \overline{r} \\ w &\geq 0 \\ w'r_s - m &\leq x_s \\ -w'r_s + m &\leq x_s \end{aligned}$$

The AMPL model file then looks as following:

```
> modelHuber <- c(
   "param N ;",
   "param S ;",
   "param delta ;",
   "param mu{1..N} ;",
   "param Scenarios{1..S, 1..N} ;",
   "param targetReturn ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var x{1..S} >= 0 ;",
   "var m ;",
   "minimize Objective: sum{s in 1..S} (",
             " ( <<delta;  1, 0 >> x[s])^2 *0.5",
```

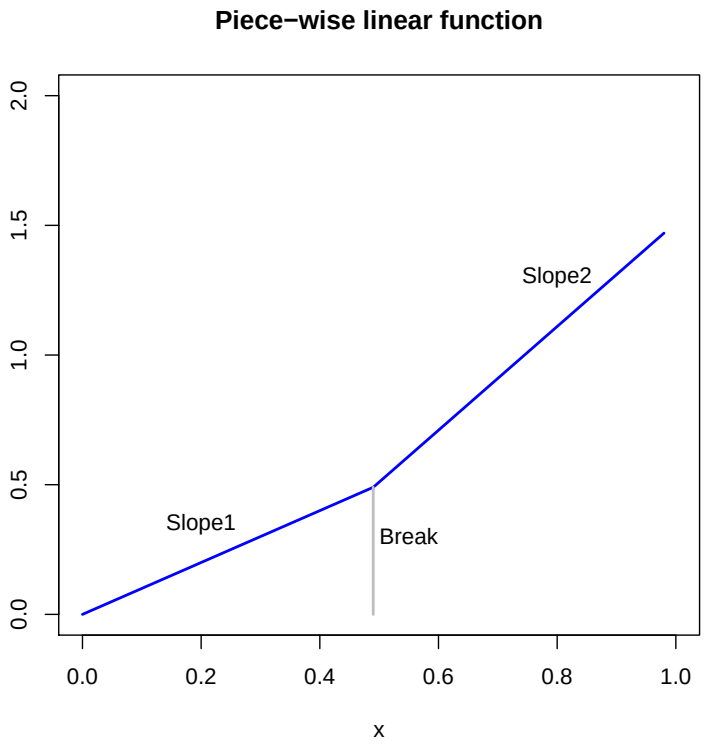


FIGURE 8.1: An example of a piecewise linear function-

```
          "+  (<<delta;  0, 1 >> x[s])*delta)/S ;",
   "subject to Budget: sum{n in 1..N} w[n] = 1 ;",
   "subject to Reward: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;",
   "subject to Minus {k in 1..S}: sum{n in 1..N} Scenarios[k,n] * w[n] - m - x[k] <= 0 ;",
   "subject to Plus {k in 1..S}: sum{n in 1..N}  Scenarios[k,n] * w[n] - m + x[k] >= 0 ;")
> amplModelFile(model=modelHuber, project="myPortfolio")
```

Since the function is non-linear, we use the solver minos. R/AMPL run file:

```
> runHuber <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver minos ;",
   "solve ;",
   "for {o in 1..N} printf \"%16.6f\", w[o] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runHuber, project="myPortfolio")
```

From the portfolio settings we construct the R/AMPL data file. The addi-

tional parameter we need to specify is `delta`:

```
> requiredData(modelHuber)
[1] "N"            "S"            "delta"        "mu"           "Scenarios"
[6] "targetReturn"

>   #Scenarios <- 100*LPP2005.RET[117:272, 1:6]
>   Scenarios <- 100*LPP2005.RET[117:217, 1:6]
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   targetReturn <- mean(mu)
>   delta <- 0.3*mean(abs(Scenarios))
> dataHuber <- dataAUTO(modelHuber)
> amplDataFile(data=dataHuber, project="myPortfolio")
```

Optimize the portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsHuber <- as.numeric(scan("myPortfolio.txt"))
> names(weightsHuber) <- colnames(Scenarios)
```

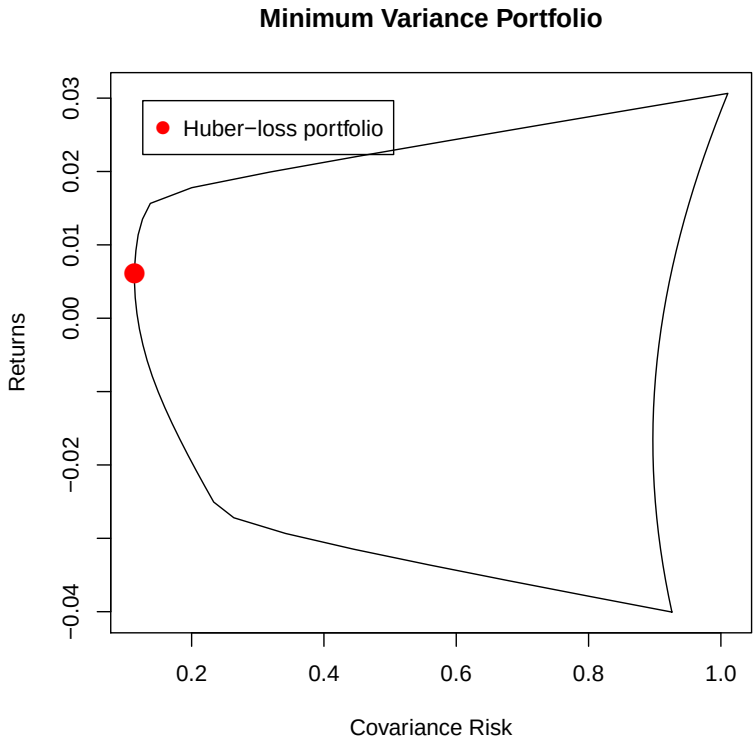


FIGURE 8.2: Feasible set and the position of minimum variance portfolio.

CALCULATE FULL TIME SERIES!

8.4 S PORTFOLIOS

The second class of portfolio policies proposed by DeMiguel and Nogales is based on the robust S-estimators. The main advantage of S-estimators is that they are equivariant with respect to scale; that is, multiplying the whole dataset by a constant does not change the value of the S-estimator. This is not the case for the M-estimators. The S-estimators of portfolio return and risk are defined as the values of m and s that solve the following optimization problem:

$$\begin{aligned} &\min_{m,s} s \\ s.t.& \\ &\frac{1}{S}\sum_{i=1}^{S}\rho\left(\frac{w' r_s - m}{\mathrm{s}}\right) = K \end{aligned}$$

where s is the loss function and K is the expectation of this loss function evaluated at a standard normal variable z, that is, $K = E(\rho(z))$. Note that the portfolio eturn deviations, $w\,r_s - m$, are scaled by the S-estimator for risk s. Intuitively, this is what makes the S-estimators scale invariant. DeMiguel and Nogales define the S-portfolio as the policy that minimizes the S-estimate of risk, namely, the portfolio that solves the following optimization problem:

$$\begin{aligned} &\min_{w,m,\mathrm{s}} \mathrm{s} \\ s.t.& \\ &\frac{1}{S}\sum_{i=1}^{S}\rho\left(\frac{w' r_s - m}{\mathrm{s}}\right) = K \\ &\mu' w = \overline{r} \\ &w \geq 0 \end{aligned}$$

Exercise: Tukey biweight loss

It is left to the reader to implement the Tukey biweight function as the loss function of an S-estimator in AMPL. The procedure is similar to the implementation of the Huber loss function in AMPL. For more information, see DeMiguel and Nogales, `http://faculty.london.edu/avmiguel/DeMiguel-Nogales-OR.pdf`.

CHAPTER 9

MAD-PORTFOLIOS

9.1 INTRODUCTION

In this chapter we introduce the mean absolute deviation (MAD) Portfolio. It was proposed by Konno and Yamazaki in 1992. Unlike the Markowitz model, the MAD-Portfolio model does not assume normality of stock returns. Instead, it measures the risk of a portfolio by the average of the absolute deviations:

$$\sum_{s=1}^{S} \frac{1}{S} \left| \sum_{i=1}^{N} (r_{i,s} - \mu_i) w_i \right| \tag{9.1}$$

In the case of normally distributed returns, the reduction of the mean absolute deviations is equivalent to minimizing the variance. For more fat-tailed distributed returns, the MAD-Portfolio is more robust and stable. Furthermore the MAD-Portfolio is easier to compute than Markowitz because it can be linearized, and it eliminates the need for a covariance matrix.
The topics presented in this chapter are:

- MADNONLIN - Nonlinear Minimum Risk Portfolio
- MAD1 - Minimum Risk Portfolio
- MADGLOB - Global Minimum Risk Portfolio
- MAD2 - Maximum Return Portfolio
- MADEDR - Equi-Distant Return Frontier
- MAD3 - Critical Line Algorithm
- MADRATIO - Reward/Risk Ratio Portfolio

- MADDIV - Herfindahl Risk Diversification
- MADHULL - Hull of the Mean-Variance Portfolio
- MADSET - Feasible Set of the Mean-Variance Portfolio

Throughout this chapter we use as an example the Swiss pension fund benchmark portfolio. The data are part of the Rmetrics package `timeSeries` and are loaded together with the `fPortfolio` package. As the benchmark portfolio we use the equal weights portfolio which is characterized by the following settings.

9.2 NONLINEAR MAD-PORTFOLIO

The aim of the MAD-portfolio approach is to minimize the mean absolute deviation of the returns. The function to calculate these deviations is given in equation 9.1. When we add the usual portfolio constraints, the minimum risk MAD-portfolio problem looks as following:

$$\begin{aligned} \min_x \quad & \sum_{s=1}^{S} \frac{1}{S} \left| \sum_{i=1}^{N} (r_{i,s} - \mu_i) w_i \right| \\ s.t. \quad & \\ & \sum_{i=1}^{N} w_i = 1 \\ & w_i \geq 0 \end{aligned}$$

Here N is the number of assets, S is the number of scenarios, r are the financial returns, w are the portfolio weights, and μ is the vector of the average asset returns.

Compose the R/AMPL model file:

```
> modelMADNONLIN <- c(
  "param N ;",
  "param S ;",
  "param mu{1..N} ;",
  "param Scenarios{1..S, 1..N} ;",
  "var w{1..N} >= 0, default 1/N ;",
  "minimize Objective: sum{s in 1..S} 1/S * abs ( sum{i in 1..N} (Scenarios[s, i] - mu[i] ) * w[i] ) ;",
  "subject to Budget: sum{i in 1..N} w[i] = 1 ;")
> amplModelFile(model=modelMADNONLIN, project="myPortfolio")
```

Since the model in nonlinear, we use the solver minos. THe R/AMPL run file looks as following:

```
> runMADNONLIN <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver minos ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runMADNONLIN, project="myPortfolio")
```

From the portfolio settings we construct the R/AMPL data file:

```
> Scenarios <- 100*LPP2005.RET[117:272, 1:6]
> requiredData(modelMADNONLIN)
[1] "N"         "S"         "mu"        "Scenarios"

>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
```

```
>   mu <- colMeans(Scenarios)
> dataMADNONLIN <- dataAUTO(modelMADNONLIN)
> amplDataFile(data=dataMADNONLIN, project="myPortfolio")
```

Optimize the Portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMADNONLIN <- as.numeric(scan("myPortfolio.txt"))
> names(weightsMADNONLIN) <- colnames(Scenarios)
> weightsMADNONLIN
     SBI      SPI      SII      LMI      MPI      ALT
0.51102 0.00000 0.10054 0.33332 0.00000 0.05512
```

Summarize the results:

```
> SummaryMADNONLIN <- c(
   TargetReturn = mu %*% weightsMADNONLIN,
   Risk = 1/S* sum(abs(t(t(Scenarios)-mu)%*%weightsMADNONLIN)),
   HerfindahlIndex = 1 - weightsMADNONLIN %*% weightsMADNONLIN)
> SummaryMADNONLIN
   TargetReturn            Risk HerfindahlIndex
       0.014446        0.085887        0.614610
```

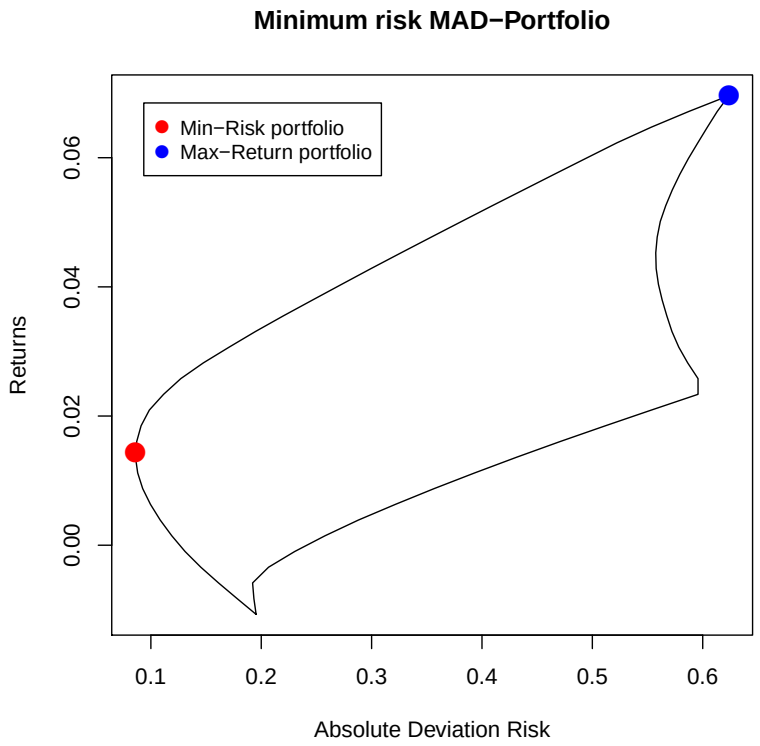


FIGURE 9.1: Minimum Risk MAD-Portfolio on the Mean-MAD hull

9.3 LINEAR MIN-RISK MAD-PORTFOLIO

Non-linear objective functions are much harder to optimize than linear ones. Fortunately, the MAD-Portfolio problem can be linearized very easily, as shown by Konno and Yamazaki (1991) or Scherer and Martin (2005). To linearize the problem, we have to remove the absolute value function in the objective. This can be done by adding the additional variables $x[s]$, and the constraints

$$\sum_{i=1}^{N}(r_{i,s}-\mu_i)w_i - x_s \leq 0 \tag{9.2}$$

and

$$\sum_{i=1}^{N}(r_{i,s}-\mu_i)w_i + x_s \geq 0 \tag{9.3}$$

The problem can then be written as

$$\min_{x,w} \frac{1}{S}\sum_{s=1}^{S} x_s \tag{9.4}$$

$$\begin{aligned} s.t. & \\ \sum_{i=1}^{N}(r_{i,s}-\mu_i)w_i - x_s &\leq 0 \\ \sum_{i=1}^{N}(r_{i,s}-\mu_i)w_i + x_s &\geq 0 \\ \sum_{i=1}^{N} w_i &= 1 \\ x_s &\geq 0 \\ w_i &\geq 0 \end{aligned}$$

It is straightforward to implement this problem into an R/AMPL model file:

```
> modelMADGLOB <- c(
   "param N ;",
   "param S ;",
   "param mu{1..N} ;",
   "param Scenarios{1..S, 1..N} ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var x{1..S} >= 0 ;",
   "minimize Objective: sum{s in 1..S} x[s] / S ;",
   "subject to Budget: sum{n in 1..N} w[n] = 1 ;",
   "subject to Minus {k in 1..S}: sum{n in 1..N} (Scenarios[k,n] - mu[n]) * w[n] - x[k] <= 0 ;",
   "subject to Plus {k in 1..S}: sum{n in 1..N}  (Scenarios[k,n] - mu[n]) * w[n] + x[k] >= 0 ;")
> amplModelFile(model=modelMADGLOB, project="myPortfolio")
```

The R/AMPL run file looks as following:

```
> runMADGLOB <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runMADGLOB, project="myPortfolio")
```

From the portfolio settings we construct the R/AMPL data file:

```
> requiredData(modelMADGLOB)
[1] "N"         "S"         "mu"        "Scenarios"
>   Scenarios <- 100*LPP2005.RET[117:217, 1:6]
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
```

```
>   mu <- colMeans(Scenarios)
> dataMADGLOB <- dataAUTO(modelMADGLOB)
> amplDataFile(data=dataMADGLOB, project="myPortfolio")
```

Optimize the portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMADGLOB <- as.numeric(scan("myPortfolio.txt"))
> names(weightsMADGLOB) <- colnames(Scenarios)
> weightsMADGLOB
      SBI      SPI      SII      LMI      MPI      ALT
0.651032 0.000000 0.133612 0.185117 0.000000 0.030239
```

And of course, if we compare the weights to the non-linear optimization problem we get the same numbers for the weights, at least to a precision of 6 digits.

9.4 LINEAR EFFICIENT MAD-PORTFOLIO

To compute the efficient MAD-portfolio, i.e. the portfolio that minimizes the mean absolute deviation for a desired predefined target return $\overline{r}$, we can just add the target return as a constraint to our model:

$$\min_{x,w} \frac{1}{S}\sum_{s=1}^{S} x_s$$

$$\begin{aligned} s.t. & \\ \sum_{i=1}^{N}(r_{i,s}-\mu_i)w_i - x_s &\leq 0 \\ \sum_{i=1}^{N}(r_{i,s}-\mu_i)w_i + x_s &\geq 0 \\ \sum_{i=1}^{N} w_i &= 1 \\ x_s &\geq 0 \\ w_i &\geq 0 \\ \sum_{i=1}^{N}\mu_i w_i &\geq \overline{r} \end{aligned}$$

The R/AMPL model file looks as following::

```
> modelMAD1 <- c(
   "param N ;",
   "param S ;",
   "param mu{1..N} ;",
   "param Scenarios{1..S, 1..N} ;",
   "param targetReturn ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var x{1..S} >= 0 ;",
   "minimize Objective: sum{s in 1..S} x[s] / S ;",
   "subject to Budget: sum{n in 1..N} w[n] = 1 ;",
   "subject to Minus {k in 1..S}: sum{n in 1..N} (Scenarios[k,n] - mu[n]) * w[n] - x[k] <= 0 ;",
   "subject to Plus {k in 1..S}: sum{n in 1..N}  (Scenarios[k,n] - mu[n]) * w[n] + x[k] >= 0 ;",
   "subject to Reward: sum{n in 1..N} mu[n] * w[n] >= targetReturn ;")
> amplModelFile(model=modelMAD1, project="myPortfolio")
```

R/AMPL run file:

```
> runMAD1 <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runMAD1, project="myPortfolio")
```

We have to add the target return to the R/AMPL data file. As the value, we take the expected return of the equal-weights-portfolio:

```
> requiredData(modelMAD1)
[1] "N"            "S"            "mu"           "Scenarios"    "targetReturn"
>   Scenarios <- 100*LPP2005.RET[117:272, 1:6]
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   targetReturn <- mean(mu)
> dataMAD1 <- dataAUTO(modelMAD1)
> amplDataFile(data=dataMAD1, project="myPortfolio")
```

Optimize the portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMAD1 <- as.numeric(scan("myPortfolio.txt"))
> names(weightsMAD1) <- colnames(Scenarios)
```

Summarize the results:

```
> SummaryMAD1 <- c(
   TargetReturn = mu %*% weightsMAD1,
   Risk = 1/S* sum(abs(t(t(Scenarios)-mu)%*%weightsMAD1)),
   HerfindahlIndex = 1 - weightsMAD1 %*% weightsMAD1)
> SummaryMAD1
   TargetReturn            Risk HerfindahlIndex
       0.026626        0.133678        0.451166
```

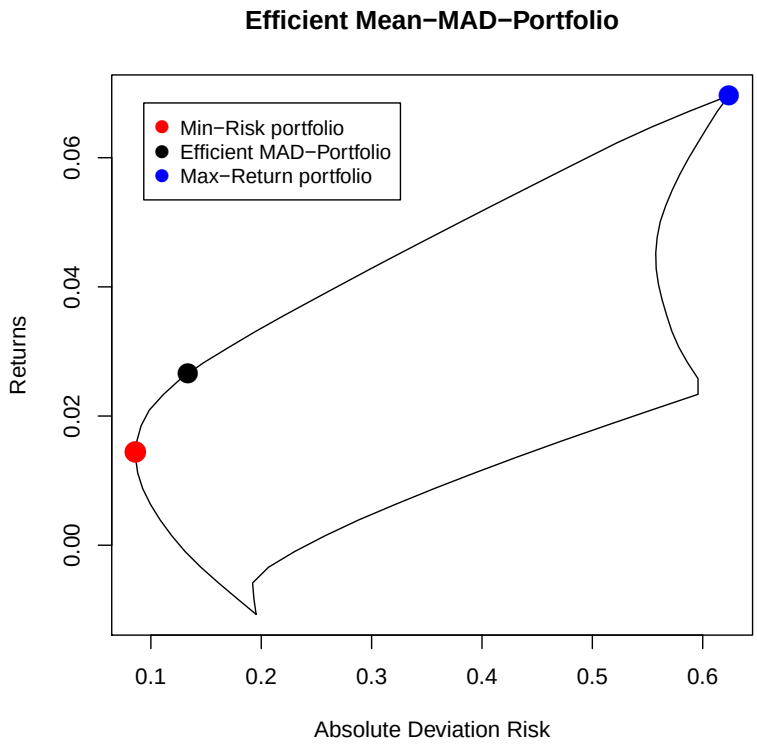


FIGURE 9.2: Efficient Frontier Portfolio on the Mean-MAD hull

9.5 MAX-RETURN MAD-PORTFOLIO

Similarly to the Maximum Return Markowitz Portfolio, we can exchange the target return with a target risk as a constraint, and proceed to maximize the expected return. The problem is then formulated as following:

$$\max_{x,w} \sum_{i=1}^{N} \mu_i w_i$$

$$s.t.$$

$$\sum_{i=1}^{N} (r_{i,s} - \mu_i) w_i - x_s \leq 0$$

$$\sum_{i=1}^{N} (r_{i,s} - \mu_i) w_i + x_s \geq 0$$

$$\sum_{i=1}^{N} w_i = 1$$

$$\frac{1}{S} \sum_{s=1}^{S} x_s \leq \overline{R}$$

$$x_s \geq 0$$

$$w_i \geq 0$$

where $\overline{R}$ is the target risk.
The R/AMPL model file can then be written as following:

```
> modelMAD2 <- c(
   "param N ;",
   "param S ;",
   "param targetRisk ;",
   "param mu{1..N} ;",
   "param Scenarios{1..S, 1..N} ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var x{1..S} >= 0 ;",
   "maximize Objective: sum{n in 1..N} mu[n] * w[n] ;",
   "subject to Budget: sum{n in 1..N} w[n] = 1 ;",
   "subject to Risk: sum{s in 1..S} x[s] / S <= targetRisk;",
   "subject to Minus {k in 1..S}: sum{n in 1..N} (Scenarios[k,n] - mu[n]) * w[n] - x[k] <= 0 ;",
   "subject to Plus {k in 1..S}: sum{n in 1..N} (Scenarios[k,n] - mu[n]) * w[n] + x[k] >= 0 ;")
> amplModelFile(model=modelMAD2, project="myPortfolio")
```

and the R/AMPL run file is unchanged

```
> runMAD2 <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runMAD2, project="myPortfolio")
```

From the portfolio settings we construct the R/AMPL data file. We again take the calculated risk from the MAD1 approach as the target risk.

```
> requiredData(modelMAD2)
[1] "N"          "S"          "targetRisk" "mu"         "Scenarios"

>   Scenarios <- 100*LPP2005.RET[117:272, 1:6]
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   targetRisk <- 1/S* sum(abs(t(t(Scenarios)-mu)%*%weightsMAD1))
> dataMAD2 <- dataAUTO(modelMAD2)
> amplDataFile(data=dataMAD2, project="myPortfolio")
```

Optimize the portfolio and extract the weights

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMAD2 <- as.numeric(scan("myPortfolio.txt"))
> names(weightsMAD2) <- colnames(Scenarios)
> weightsMAD2
      SBI       SPI       SII       LMI       MPI       ALT
0.032349 0.124594 0.000000 0.718923 0.000000 0.124134
```

As expected, the weights are exactly the same as for the risk-minimization approach.

9.6 EQUI-DISTANT RETURN FRONTIER

Again, we just add the additional parameters for computing the frontier to the existing MAD1 model file:

```
> modelMADEDR <- c(
   "param minReturn ;",
   "param maxReturn ;",
   "param nReturn ;",
   "param N ;",
   "param S ;",
   "param mu{1..N} ;",
   "param Scenarios{1..S, 1..N} ;",
   "param targetReturn ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var x{1..S} >= 0 ;",
   "minimize Objective: sum{s in 1..S} x[s] / S ;",
   "subject to Budget: sum{n in 1..N} w[n] = 1 ;",
   "subject to Minus {k in 1..S}: sum{n in 1..N} (Scenarios[k,n] - mu[n]) * w[n] - x[k] <= 0 ;",
   "subject to Plus {k in 1..S}: sum{n in 1..N}  (Scenarios[k,n] - mu[n]) * w[n] + x[k] >= 0 ;",
   "subject to Reward: sum{n in 1..N} mu[n] * w[n] >= targetReturn ;")
> amplModelFile(model=modelMADEDR, project="myPortfolio")
```

or in short:

```
> modelMADEDR <- c(
   "param minReturn ;",
   "param maxReturn ;",
   "param nReturn ;",
   modelMAD1)
> amplModelFile(model=modelMADEDR, project="myPortfolio")
```

As in the Markowitz approach, the R/AMPL Run File has to be modified for this algorithm since now we are looping over several target returns. In order to get a value for the `minReturn`-parameter, we solve the problem first for a target return set to $-\infty$ in order to calculate the expected return of the minimum risk portfolio. We then define the values for the `minReturn`- and `maxReturn`-parameter. Afterwards, we introduce a loop and replace the value of $\overline{r}$ for every iteration:

```
> runMVEDR <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex;",
   "let targetReturn := -999;",
   "solve;",
   "let minReturn := sum{i in 1..N} w[i]*mu[i];",
   "let maxReturn := max{i in 1..N} mu[i];",
   "for {i in 0..nReturn} {",
   "  let targetReturn := minReturn + i*(maxReturn-minReturn)/nReturn ;",
   "  solve ;",
   "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
   "};",
   "exit ;")
```

```
> amplRunFile(run=runMVEDR, project="myPortfolio")
```

From the portfolio settings we construct the R/AMPL data file. We do not need to specify the targetReturn, the minReturn or the maxReturn as this is done in the run file.

```
> requiredData(modelMADEDR)
[1] "minReturn"    "maxReturn"    "nReturn"      "N"            "S"
[6] "mu"           "Scenarios"    "targetReturn"
>   Scenarios <- 100*LPP2005.RET[117:272, 1:6]
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   targetReturn <- NA
>   minReturn <- NA
>   maxReturn <- NA
>   nReturn <- 33
> dataMADEDR <- dataAUTO(modelMADEDR)
> amplDataFile(data=dataMADEDR, project="myPortfolio")
```

Optimize the portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMADEDR <- matrix(as.numeric(scan("myPortfolio.txt")), byrow=TRUE, ncol=N)
> colnames(weightsMADEDR) <- colnames(Scenarios)
> rownames(weightsMADEDR) <- paste0("MADEDR-", 0:nReturn)
>
```

Summarize the results:

```
> Returns <- Risks <- NULL
> for (i in 0:nReturn) {
   Returns <- c(Returns, mu %*% weightsMADEDR[i+1,])
   Risks <- c(Risks, 1/S* sum(abs(t(t(Scenarios)-mu)%*%weightsMADEDR[i+1,]))) }
> SummaryMADEDR <- cbind(
   targetReturn=Returns,
   Risk = Risks,
   HerfindahlIndex = 1 - diag(weightsMADEDR %*% t(weightsMADEDR) ) )
```

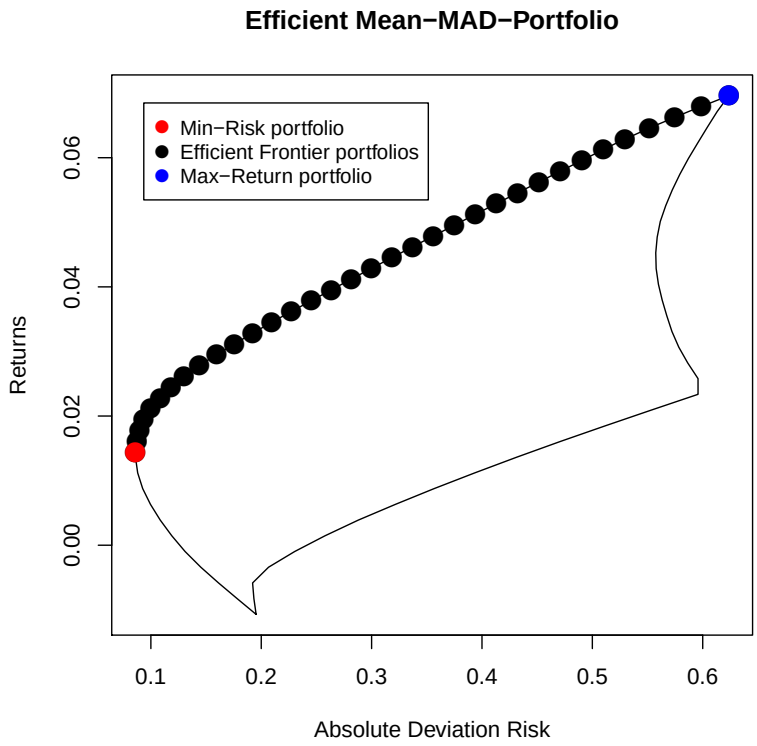


FIGURE 9.3: Efficient Mean-MAD-Portfolios on the Mean-MAD hull

9.7 CRITICAL LINE ALGORITHM MAD-PORTFOLIO

This algorithm will be presented in section ...

9.8 REWARD/RISK RATIO PORTFOLIO

Similarly to the Sharpe Ratio, we can define a Reward/Risk ratio for the MAD-Portfolio and try to maximize it. We can use the same framework we applied to find the Sharpe ratio with a linear optimization problem:

$$\min_{x,w,t} \sum_{s=1}^{S} x_s \tag{9.5}$$

$$\begin{aligned}
s.t. & \\
\sum_{i=1}^{N}(r_{i,s}-\mu_i)w_i - x_s &\leq 0 \\
\sum_{i=1}^{N}(r_{i,s}-\mu_i)w_i + x_s &\geq 0 \\
1'w &= t \\
\mu' w &= 1 \\
x_s &\geq 0 \\
t &\geq 0 \\
w_i &\geq 0
\end{aligned}$$

The R/AMPL model file then looks as following:

```
> modelMADRATIO <- c(
   "param N ;",
   "param S ;",
   "param mu{1..N} ;",
   "param Scenarios{1..S, 1..N} ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var x{1..S} >= 0 ;",
   "var t >= 0 ;",
   "minimize Objective: sum{s in 1..S} x[s] / S ;",
   "subject to Budget: sum{n in 1..N} w[n] = t ;",
   "subject to Reward: sum{n in 1..N} mu[n]*w[n] = 1 ;",
   "subject to Minus {k in 1..S}: sum{n in 1..N} (Scenarios[k,n] - mu[n]) * w[n] - x[k] <= 0 ;",
   "subject to Plus {k in 1..S}: sum{n in 1..N} (Scenarios[k,n] - mu[n]) * w[n] + x[k] >= 0 ;")
> amplModelFile(model=modelMADRATIO, project="myPortfolio")
```

Again we are now dealing with a non-linear function and therefore have to replace the cplex solver with minos:

```
> runMADRATIO <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m]/t > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runMADRATIO, project="myPortfolio")
```

From the portfolio settings we construct the R/AMPL data file

```
> requiredData(modelMADRATIO)
```

```
[1] "N"         "S"         "mu"        "Scenarios"
```

```
>   Scenarios <- 100*LPP2005.RET[117:272, 1:6]
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
> dataMADRATIO <- dataAUTO(modelMADRATIO)
> amplDataFile(data=dataMADRATIO, project="myPortfolio")
```

Optimize the portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMADRATIO <- as.numeric(scan("myPortfolio.txt"))
> names(weightsMADRATIO) <- colnames(Scenarios)
```

The Ratio takes the value

Summarize the results:

```
> SummaryRATIO <- c(
   TargetReturn = mu %*% weightsMADRATIO,
   MADRisk = 1/S* sum(abs(t(t(Scenarios)-mu)%*%weightsMADRATIO)),
   HerfindahlIndex = 1 - weightsMADRATIO %*% weightsMADRATIO)
> SummaryRATIO
```

```
   TargetReturn         MADRisk HerfindahlIndex
       0.020883        0.098404        0.553634
```

Note that the reward/risk ratio portfolio lies on the efficient frontier:

Plot the Risk/Reward Diagram:

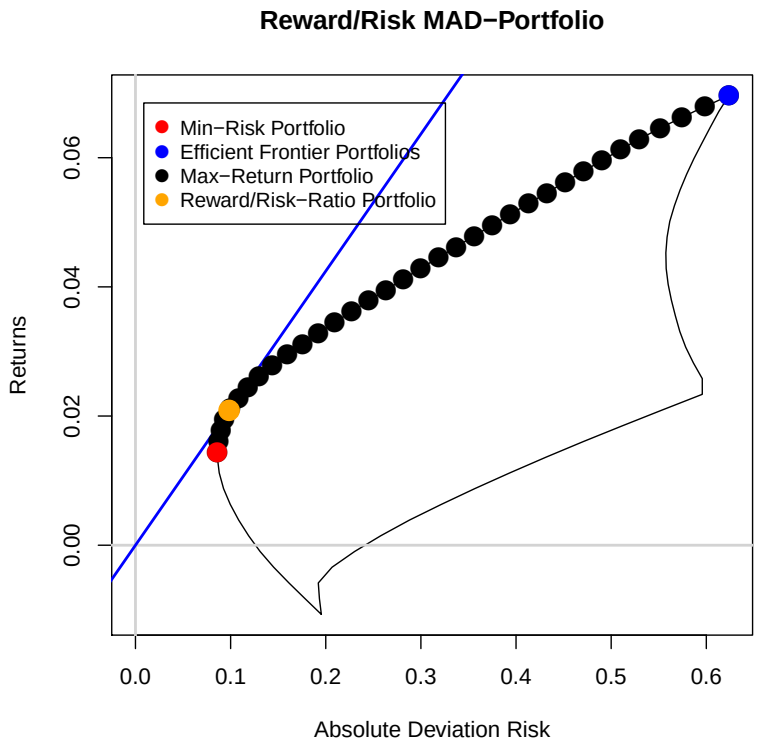
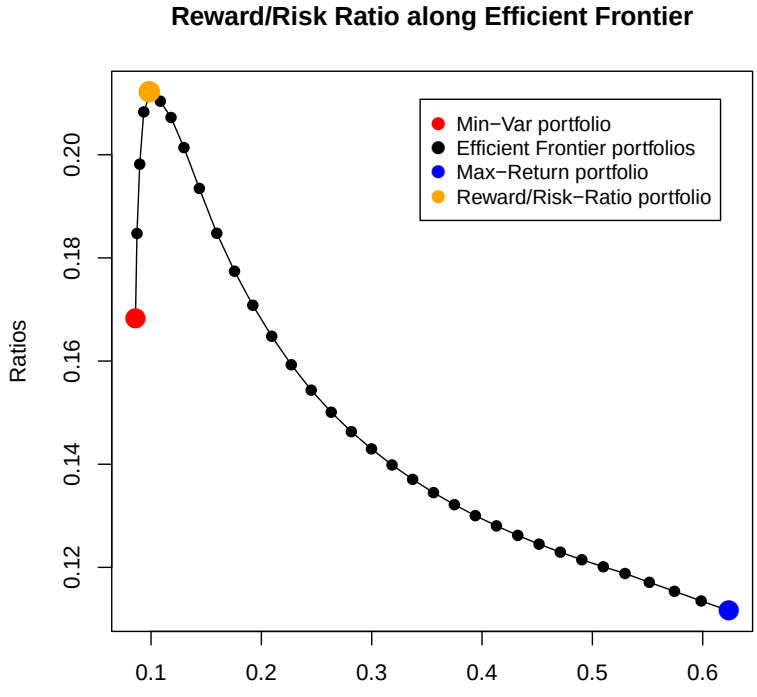


FIGURE 9.4: Position of the Reward/Risk-Ratio MAD-Portfolio



9.9 HULL OF THE MAD-PORTFOLIO

We use exactly the same steps, but replace the Markowitz-EDR model file with the MAD-EDR model file, and additionally replace the risk expression in the calculation of the Maximum Risk locus:

```
> modelMADMINHULL <- c(
   # MADMINHULL - Single Min-Risk MAD-Portfolio:",
   "param N ;",
   "param S ;",
   "param mu{1..N} ;",
   "param Scenarios{1..S, 1..N} ;",
   "param Return ;",
   "param minReturn ;",
   "param maxReturn ;",
   "param nReturn ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var x{1..S} >= 0 ;",
   "minimize Objective: sum{s in 1..S} x[s] / S ;",
   "subject to Budget: sum{n in 1..N} w[n] = 1 ;",
   "subject to Reward: sum{n in 1..N} mu[n] * w[n] = Return ;",
   "subject to Minus {k in 1..S}: sum{n in 1..N} (Scenarios[k,n] - mu[n]) * w[n] - x[k] <= 0 ;",
   "subject to Plus {k in 1..S}: sum{n in 1..N}  (Scenarios[k,n] - mu[n]) * w[n] + x[k] >= 0 ;")
> amplModelFile(model=modelMADMINHULL, project="myPortfolio")
```

R/AMPL run file

```
> runMADMINHULL <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex;",
   "for {i in 0..nReturn} {",
   "  let Return := minReturn + i*(maxReturn-minReturn)/nReturn ;",
   "  solve ;",
   "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
   "};",
   "exit ;")
> amplRunFile(run=runMADMINHULL, project="myPortfolio")
```

From the portfolio settings we construct the R/AMPL data file

```
> Scenarios <- 100*LPP2005.RET[117:272, 1:6]
> requiredData(modelMADMINHULL)
[1] "N"         "S"         "mu"        "Scenarios" "Return"    "minReturn"
[7] "maxReturn" "nReturn"

>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   Return <- mean(mu)
>   minReturn <- min(mu)
>   maxReturn <- max(mu)
>   nReturn <- 33
> dataMADMINHULL <- dataAUTO(modelMADMINHULL)
> amplDataFile(data=dataMADMINHULL, project="myPortfolio")
```

Optimize the portfolio and extract the weights

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMADMINHULL <- matrix(as.numeric(scan("myPortfolio.txt")), byrow=TRUE, ncol=N)
> colnames(weightsMADMINHULL) <- colnames(Scenarios)
> rownames(weightsMADMINHULL) <- paste0("MADMINHULL-", 0:nReturn)
```

Next we compute the maximum variance locus of the MAD-portfolio by the maximum risk intersection of all pairwise portfolios

```
> MADHull <- function(mu, Return, Risk) {
   # Minimum Risks:
   minRisks <- Risk
   Returns <- risks <- Return
   # Maximum Risks:
   maxRisks <- rep(-Inf, length(Returns))
   nAssets <- length(mu)
   for (i in 1:(nAssets - 1)) {
     for (j in (i + 1):nAssets) {
       mu2 <- mu[c(i, j)]
       Scenarios2 <- Scenarios[,c(i,j)]
       S <- nrow(Scenarios)
       Index <- which(Returns >= min(mu2) & Returns <= max(mu2))
       if (length(Index) > 0) {
         Index <- (1:length(Returns))[Index]
         for (k in Index) {
           weights <- (Returns[k] - mu2[2])/(mu2[1] - mu2[2])
           weights <- c(weights, 1 - weights)
           Risk <- 1/S* sum(abs(t(t(Scenarios2)-mu2)%*%weights))
           maxRisks[k] <- max(maxRisks[k], Risk) }
       }
     }
   }
   # Hull:
   risk <- c(minRisks, rev(maxRisks[-1])[-1], minRisks[1])
   return <- c(Returns, rev(Returns[-1])[-1], Returns[1])
   hull <- cbind(Risks = risk, Returns = return)
   # Return Value:
   hull
 }
```

The input for the `MADHull()` are the column means (mu) of the assets, the covariance matrix `Sigma`, and the `Return` and `Risk` values along the minimum variance locus and the efficient frontier.
Compute measures:

```
> Return <- Risk <- NULL
> for (i in 0:nReturn) {
   Return <- c(Return, mu %*% weightsMADMINHULL[i+1,])
   Risk <- c(Risk, 1/S* sum(abs(t(t(Scenarios)-mu)%*%weightsMADMINHULL[i+1,])) ) }
> hull <- MADHull(mu, Return, Risk)
```

Part IV

Mean-CVaR Designs

CHAPTER 10

MEAN-CVAR PORTFOLIOS

10.1 INTRODUCTION

In this chapter we introduce the Mean-CVaR Portfolio and show how to solve the following types of portfolios

- CVAR1 Minimum Risk Mean-CVaR Efficient Portfolio
- CVARGLOB Global Minimum Risk Efficient Portfolio
- CVAR2 Maximum Return Mean-CVaR Efficient Portfolio
- CVAREDR Equi-distant Return Mean-CVaR Frontier
- CVAR3 Mean-CVaR Critical Line Algorithm
- CVARSORTINO Reward/CVaR Ratio Portfolio
- CVARDIV Herfindahl Risk Diversification
- CVARHULL Mean-CVaR Hull
- CVARSET Mean-CVaR Feasible Set

Throughout this chapter we use as an example the Swiss pension fund benchmark portfolio. The data are part of the Rmetrics package `timeSeries` and are loaded together with the `fPortfolio` package. As the benchmark portfolio we use the equal weights portfolio which is characterized by the following settings.
The `targetReturn` for the equal weights portfolio is defined by the *grand mean* of the portfolio scenarios, and the `targetRisk` is defined by the *grand variance* of the portfolio. `mu` is the vector of the sample means of the assets, and `Sigma` the sample covariance matrix. Our default value for the VaR quantile, or conficence level, `alpha` is 5%.

Please note that although the theoretical definition of the CVaR is well defined, there exist different techniques to estimate the CVaR, which do not necessarily yield the exact same value. During this chapter, we will use the following function to estimate the CVaR from existing portfolio weights:

```
> pfolioCVaR <- function (x, weights = NULL, alpha = 0.05)
 {
     data <- as.matrix(x)
     if (is.null(weights))
         weights = rep(1/dim(data)[[2]], dim(data)[[2]])
     n <- dim(data)[1]
     Rp <- apply(t(t(data) * weights), 1, sum)
     sorted <- sort(Rp)
     n.alpha <- floor(n * alpha)
     VaR <- sorted[n.alpha]
     n.alpha <- max(1, floor(n * alpha) - 1)
     CVaRplus <- mean(sorted[1:n.alpha])
     lambda <- 1 - floor(n * alpha)/(n * alpha)
     ans <- as.vector(lambda * VaR + (1 - lambda) * CVaRplus)
     names(ans) <- "CVaR"
     attr(ans, "control") = c(CVaRplus = CVaRplus, lambda = lambda)
     ans
 }
```

10.2 GLOBAL MINIMUM RISK CVAR PORTFOLIO

For standard *global mimimun-risk portfolios* we optimize the weights by maximizing the *Conditional Value-at-Risk* for a long-only CVaR portfolio. Here, maximizing the conditional value-at-risk means minimizing the total risk. The linearized CVaR model can be written as:

$$\min_{w,VaR,x} \quad \frac{1}{\alpha S}\sum_{s=1}^{S} x_s - VaR \tag{10.1}$$

$$\begin{aligned} s.t. \\ 1'w &= 1 \\ w_i &\geq 0 \\ \sum_{i=1}^{N} r_{s,i}\, w_i - VaR + x_s &\geq 0 \\ x_s &\geq 0 \end{aligned}$$

The R/AMPL model file can be written as following:

```
> modelCVARGLOB <- c(
   "param N ;",
   "param S ;",
   "param Scenarios{1..S,1..N} ;",
   "param alpha ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var VaR ;",
   "var x{1..S} >= 0 ;",
   "minimize Objective: ( sum{s in 1..S} x[s] ) / ( alpha*S )-VaR ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to CVaR{s in 1..S}: sum{i in 1..N} Scenarios[s,i] * w[i] - VaR + x[s] >= 0 ;")
> amplModelFile(model=modelCVARGLOB, project="myPortfolio")
```

The R/AMPL run file looks as usual:

```
> runCVARGLOB <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "printf \"%16.6f\", Objective > myPortfolio.par ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runCVARGLOB, project="myPortfolio")
```

We have to specify the asset returns, the number of assets, the number of returns, and the quantile for the data file:

```
> requiredData(modelCVARGLOB)
[1] "N"         "S"         "Scenarios" "alpha"
```

```
> Scenarios <- 100*LPP2005REC[1:250,1:6]
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   alpha <- 0.05
> dataCVARGLOB <- dataAUTO(modelCVARGLOB)
> amplDataFile(data=dataCVARGLOB, project="myPortfolio")
```

Optimize the portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsCVARGLOB <- as.numeric(scan("myPortfolio.txt"))
> names(weightsCVARGLOB) <- colnames(Scenarios)
> weightsCVARGLOB
     SBI      SPI      SII      LMI      MPI      ALT
0.132539 0.000000 0.132857 0.666020 0.000000 0.068584
```

Summarize the results:

```
> SummaryCVARGLOB <- c(
 Return = mu %*% weightsCVARGLOB,
 CVaR = -pfolioCVaR(Scenarios,weightsCVARGLOB,alpha=alpha),
 CovarianceRisk = sqrt ( weightsCVARGLOB %*% Sigma %*% weightsCVARGLOB ),
 HerfindahlIndex = 1 - weightsCVARGLOB %*% weightsCVARGLOB)
> SummaryCVARGLOB
         Return       CVaR.CVaR  CovarianceRisk HerfindahlIndex
      0.0090221       0.2172455       0.1078056       0.5164960
```

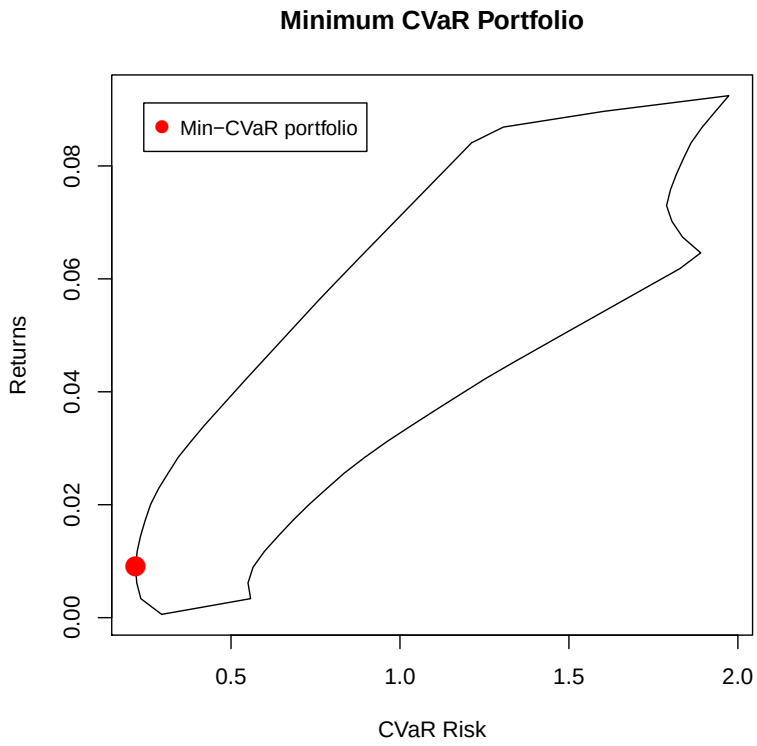


FIGURE 10.1: Feasible set and the position of the minimum CVaR portfolio.

10.3 EFFICIENT MIN-RISK PORTFOLIOS

For standard *efficient mimimun-risk portfolios* we optimize the weights by maximizing the Conditional Value-at-Risk for a long-only mean-CVaR portfolio. Here, maximizing the condititional value-at-risk means minimizing the total risk. The linearized model can be written as:

$$\min_{w,VaR,x} \quad \frac{1}{\alpha S}\sum_{s=1}^{S} x_s - VaR \tag{10.2}$$

$$\begin{aligned} s.t. & \\ 1'w &= 1 \\ \mu' w &\geq \overline{r} \\ w_i &\geq 0 \\ \sum_{i=1}^{N} r_{s,i} w_i - VaR + x_s &\geq 0 \\ x_s &\geq 0 \end{aligned}$$

The mean-CVaR portfolio R/AMPL model file looks as following:

```
> modelCVAR1 <- c(
   "param N ;",
   "param S ;",
   "param Scenarios{1..S,1..N} ;",
   "param mu{1..N} ;",
   "param alpha ;",
   "param targetReturn ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var VaR ;",
   "var x{1..S} >= 0 ;",
   "minimize Objective: ( sum{s in 1..S} x[s] ) / ( alpha*S )-VaR ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to Reward: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;",
   "subject to CVaR{s in 1..S}: sum{i in 1..N} Scenarios[s,i] * w[i] - VaR + x[s] >= 0 ;")
> amplModelFile(model=modelCVAR1, project="myPortfolio")
```

The R/AMPL run file looks as usual:

```
> runCVAR1 <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(runCVAR1, "myPortfolio")
```

From the portfolio settings we construct the R/AMPL data file. We now also have to specify the value of the target return which we set to be the grand mean of all assets:

```
> requiredData(modelCVAR1)
[1] "N"            "S"            "Scenarios"    "mu"           "alpha"
[6] "targetReturn"

> Scenarios <- 100*LPP2005REC[1:250,1:6]
>   N <- ncol(Scenarios)
```

```
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   alpha <- 0.05
>   targetReturn <- mean(mu)
> dataCVAR1 <- dataAUTO(modelCVAR1)
> amplDataFile(data=dataCVAR1, project="myPortfolio")
```

Optimize the CVAR1 model and extract the optimal weights

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsCVAR1 <- as.numeric(scan("myPortfolio.txt"))
> names(weightsCVAR1) <- colnames(Scenarios)
> weightsCVAR1
    SBI     SPI     SII     LMI     MPI     ALT
0.00000 0.00000 0.00000 0.54153 0.00000 0.45847
```

Summarize the results:

```
> SummaryCVaR1 <- c(
 Return = mu %*% weightsCVAR1,
 CVaR = -pfolioCVaR(Scenarios, weightsCVAR1,0.05),
 CovarianceRisk = sqrt ( weightsCVAR1 %*% Sigma %*% weightsCVAR1 ),
 HerfindahlIndex = 1 - weightsCVAR1 %*% weightsCVAR1)
> SummaryCVaR1
        Return       CVaR.CVaR  CovarianceRisk HerfindahlIndex
      0.041504        0.533257        0.245179        0.496551
```

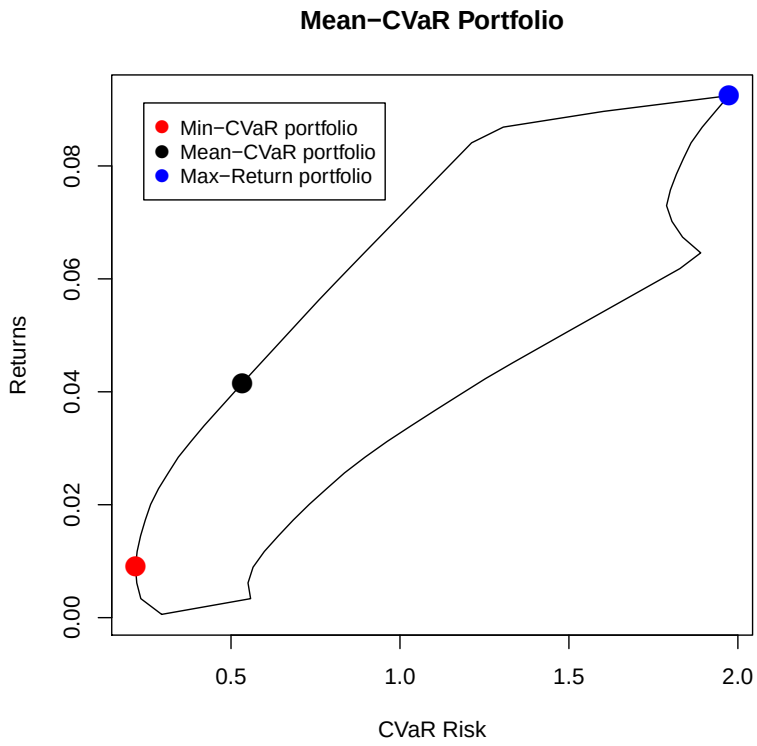


FIGURE 10.2: Feasible set and the position of an efficient Mean-CVaR portfolio.

10.4 EFFICIENT MAX-RETURN PORTFOLIOS

Krokhmal, Palmquist and Uryasev showed that the linearized mean-CVAR portfolio can also be calculated with the CVAR as the target constraint while minimizing the expected return, very similar to the MAD portfolio. The problem can then be written as following:

$$\max_{w,VaR,x} \quad \mu' w \tag{10.3}$$

$$s.t.$$

$$\begin{aligned} 1'w &= 1 \\ w_i &\geq 0 \\ \frac{1}{\alpha S}\sum_{s=1}^{S} x_s - VaR &\leq \overline{R} \\ \sum_{i=1}^{N} r_{s,i} w_i - VaR + x_s &\geq 0 \\ x_s &\geq 0 \end{aligned}$$

where $\overline{R}$ is the target risk.
In the R/AMPL model file, the risk and return are now interchanged:

```
> modelCVAR2 <- c(
   "param N ;",
   "param S ;",
   "param Scenarios{1..S,1..N} ;",
   "param mu{1..N} ;",
   "param alpha ;",
   "param targetRisk ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var VaR ;",
   "var x{1..S} >= 0 ;",
   "maximize Objective: sum{i in 1..N} mu[i] * w[i] ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to cvarRisk: ( sum{s in 1..S} x[s] ) / ( alpha*S )-VaR <= targetRisk;",
   "subject to CVaR{s in 1..S}: sum{i in 1..N} Scenarios[s,i] * w[i] - VaR + x[s] >= 0 ;")
> amplModelFile(model=modelCVAR2, project="myPortfolio")
```

The R/AMPL run file is the same as in the previous example

```
> runCVAR2 <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(runCVAR2, "myPortfolio")
```

The R/AMPL data file is also unchanged except the need of the target risk.
We will use the CVaR value of the CVaR1 result we already calculated:

```
> requiredData(modelCVAR2)
[1] "N"          "S"          "Scenarios"  "mu"         "alpha"
[6] "targetRisk"
```

```
>   Scenarios <- 100*LPP2005REC[1:250,1:6]
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   alpha <- 0.05
>   targetRisk <- SummaryCVaR1[2] # Grand mean CVaR Risk
> dataCVAR2 <- dataAUTO(modelCVAR2)
> amplDataFile(data=dataCVAR2, project="myPortfolio")
```

Optimize the CVaR2 model for the grand mean and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsCVAR2 <- as.numeric(scan("myPortfolio.txt"))
> names(weightsCVAR2) <- colnames(Scenarios)
> weightsCVAR2
    SBI     SPI     SII     LMI     MPI     ALT
0.00000 0.00000 0.00000 0.53301 0.00000 0.46699
```

We only get approximately the same weights as for the CVaR1 approach:

```
> weightsCVAR1
    SBI     SPI     SII     LMI     MPI     ALT
0.00000 0.00000 0.00000 0.54153 0.00000 0.45847
```

This is because of the differences in estimating the CVaR of a portfolio. To feed the CVaR constraint into the AMPL data file, we use our above defined pfolioCVaR function. In the AMPL model file, the CVaR constraint might correspond to a slightly different portfolio since equation 10.3 estimates the CVaR-value slightly lower than our function. If we would use the same CVaR-estimator as in the model, we would receive exactly the same weights.

10.5 EQUI-DISTANT RETURN FRONTIER

To optimize the mean-CVaR portfolio along the efficient frontier we span the efficient frontier and the minimum variance locus in equi-distant parts.
For the R/AMPL model file we can use that one from the CVAR1 model and add the parameters for the minimum and maximum returns, respectively. `minReturn` and `maxReturn` take on the values from the worst and best performing single assets. It looks as following;

```
> modelCVAREDR <- c(
   "param minReturn ;",
   "param maxReturn ;",
   "param nReturn ;",

   "param N ;",
   "param S ;",
   "param Scenarios{1..S,1..N} ;",
   "param mu{1..N} ;",
   "param alpha ;",
   "param targetReturn ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var VaR ;",
   "var x{1..S} >= 0 ;",
   "minimize Objective: ( sum{s in 1..S} x[s] ) / ( alpha*S )-VaR ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to Reward: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;",
   "subject to CVaR{s in 1..S}: sum{i in 1..N} Scenarios[s,i] * w[i] - VaR + x[s] >= 0 ;")
> amplModelFile(model=modelCVAREDR, project="myPortfolio")
```

or in short:

```
> modelCVAREDR <- c(
   "param minReturn ;",
   "param maxReturn ;",
   "param nReturn ;",
   modelCVAR1)
> amplModelFile(model=modelCVAREDR, project="myPortfolio")
```

As in the Markowitz approach, the R/AMPL Run File has to be modified for this algorithm since now we are looping over several target returns. In order to get a value for the `minReturn`-parameter, we solve the problem first for a target return set to $-\infty$ in order to calculate the expected return of the minimum risk portfolio. We then define the values for the `minReturn`- and `maxReturn`-parameter. Afterwards, we introduce a loop and replace the value of $\overline{r}$ for every iteration:

```
> runCVAREDR <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex;",
   "let targetReturn := -999;",
```

```
    "solve;",
    "let minReturn := sum{i in 1..N} w[i]*mu[i];",
    "let maxReturn := max{i in 1..N} mu[i];",
    "for {i in 0..nReturn} {",
    "  let targetReturn := minReturn + i*(maxReturn-minReturn)/nReturn ;",
    "  solve ;",
    "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
    "};",
    "exit ;")
> amplRunFile(run=runCVAREDR, project="myPortfolio")
```

From the portfolio settings we construct the R/AMPL data file. We do not need to specify the targetReturn, the minReturn or the maxReturn as this is done in the run file.

```
> requiredData(modelCVAREDR)
[1] "minReturn"    "maxReturn"    "nReturn"      "N"            "S"
[6] "Scenarios"    "mu"           "alpha"        "targetReturn"

>   Scenarios <- 100*LPP2005REC[1:250,1:6]
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   alpha <- 0.05
>   targetReturn <- NA
>   minReturn <- NA
>   maxReturn <- NA
>   nReturn <- 33
> dataCVAREDR <- dataAUTO(modelCVAREDR)
> amplDataFile(data=dataCVAREDR, project="myPortfolio")
```

Optimize the portfolio and extract weights

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsCVAREDR <- matrix(as.numeric(scan("myPortfolio.txt")), byrow=TRUE, ncol=N)
> colnames(weightsCVAREDR) <- colnames(Scenarios)
> rownames(weightsCVAREDR) <- 1:(nReturn+1)
```

To summarize, we create a data frame that holds the results for the return, the CVaR and covariance risk, and the Herfindahl index for the portfolio's diversification.

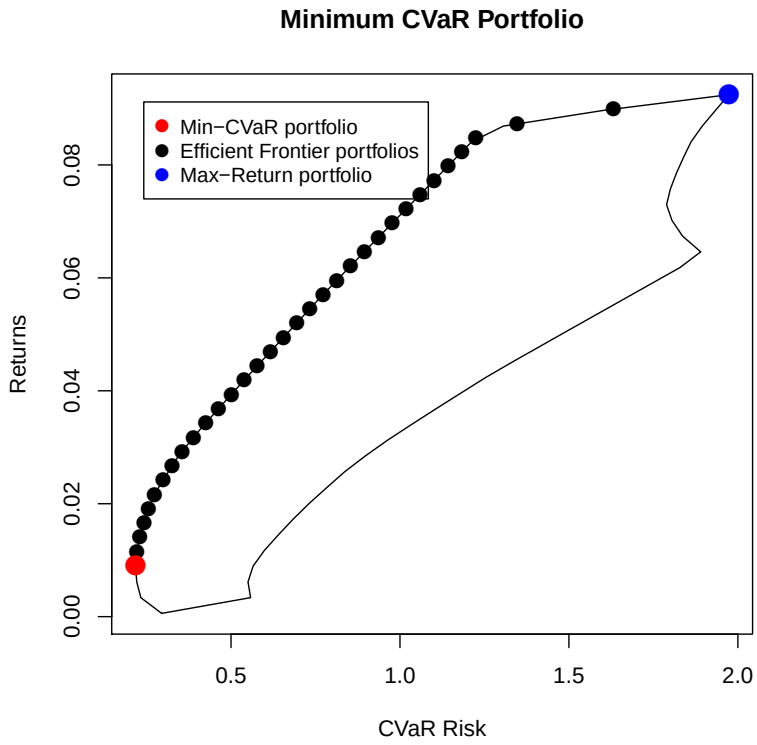


FIGURE 10.3: Feasible set and the position of the minimum CVaR portfolio.

10.6 CRITICAL LINE ALGORITHM

See chapter ...

10.7 STARR RATIO PORTFOLIO

The risk/reward ratio for the expected shortfall is called the *STARR ratio*. It is defined as

$$\frac{\mu' w}{\text{CVAR}_\alpha(w)} \tag{10.4}$$

The STARR ratio portfolio is the portfolio that maximizes the STARR ratio. Similar to the Sharpe ratio, it can be found by the following linear optimization problem:

$$\min_{w,\text{VaR},x,t} \frac{1}{\alpha S}\sum_{s=1}^{S} x_s - \text{VaR} \tag{10.5}$$

$$\begin{aligned} s.t. \\ 1' w &= t \\ \mu' w &= 1 \\ w_i &\geq 0 \\ t &\geq 0 \\ \sum_{i=1}^{N} r_{s,i} w_i - VaR + x_s &\geq 0 \\ x_s &\geq 0 \end{aligned} \tag{10.6}$$

R/AMPL model file

```
> modelCVARSTARR <- c(
   "param N ;",
   "param S ;",
   "param Scenarios{1..S,1..N} ;",
   "param mu{1..N} ;",
   "param alpha ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var VaR ;",
   "var x{1..S} >= 0 ;",
   "var t >= 0 ;",
   "minimize Objective:  ( sum{s in 1..S} x[s])/ ( alpha*S ) - VaR ;",
   "subject to Reward : sum{k in 1..N} mu[k] * w[k] = 1 ;",
   "subject to Budget: sum{i in 1..N} w[i] = t ;",
   "subject to CVaR{s in 1..S}: sum{i in 1..N} Scenarios[s,i] * w[i] - VaR + x[s] >= 0 ;")
> amplModelFile(model=modelCVARSTARR, project="myPortfolio")
```

R/AMPL run file:

```
> runCVARSTARR <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
```

```
    "solve ;",
    "for {m in 1..N} printf \"%16.6f\", w[m]/t > myPortfolio.txt ;",
    "exit ;")
> amplRunFile(runCVARSTARR, "myPortfolio")
```

R/AMPL data file:

```
> requiredData(modelCVARSTARR)
```

```
[1] "N"         "S"         "Scenarios" "mu"        "alpha"
```

```
>   Scenarios <- 100*LPP2005REC[1:250,1:6]
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   alpha <- 0.05
> dataCVARSTARR <- dataAUTO(modelCVARSTARR)
> amplDataFile(data=dataCVARSTARR, project="myPortfolio")
```

Optimize and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsCVARSTARR <- as.numeric(scan("myPortfolio.txt"))
> names(weightsCVARSTARR) <- colnames(Scenarios)
```

Summarize

```
> Sigma <- cov(Scenarios)
> SummaryCVARSTARR <- c(
 Return = mu %*% weightsCVARSTARR,
 CVaR = -as.numeric(pfolioCVaR(weights=weightsCVARSTARR,x=Scenarios,alpha=0.05)[1]),
 CovarianceRisk = sqrt ( weightsCVARSTARR %*% Sigma %*% weightsCVARSTARR ),
 HerfindahlIndex = 1 - weightsCVARSTARR %*% weightsCVARSTARR,
 STARR_Ratio = mu %*% weightsCVARSTARR /(-as.numeric(pfolioCVaR(weights=weightsCVARSTARR,x=Scenarios,alpha=0.
> SummaryCVARSTARR
```

```
         Return            CVaR  CovarianceRisk HerfindahlIndex     STARR_Ratio
       0.027878        0.337112        0.166287        0.412585        0.082696
```

Plot the Risk/Reward Diagram:

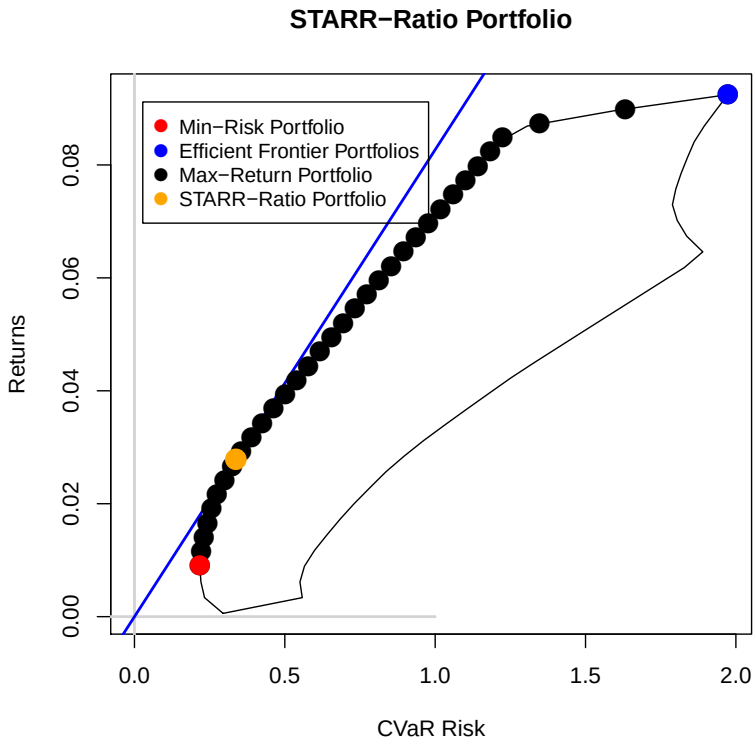
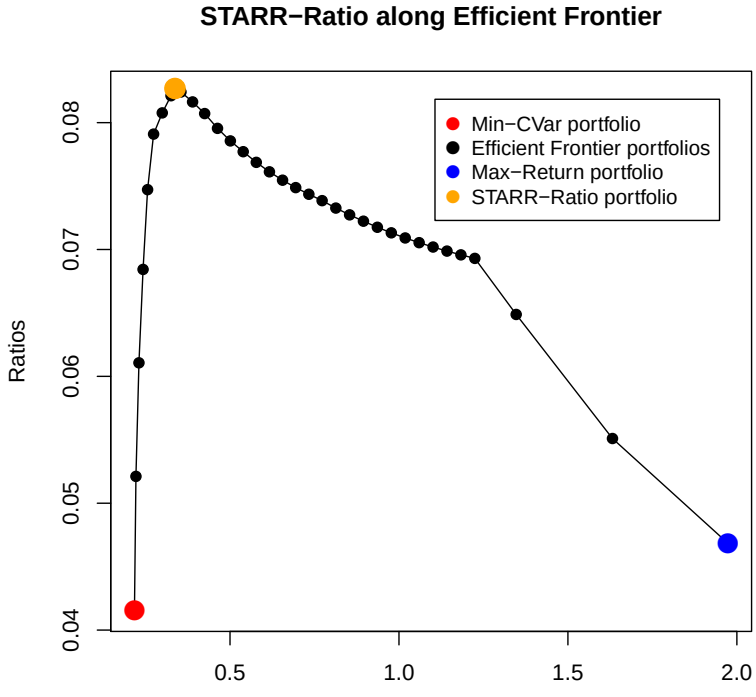


FIGURE 10.4: Position of the STARR-Ratio Portfolio



10.8 MEAN-CVAR HULL

In this section we show how to write an R function, that computes the hull of the unconstrained mean-CVaR portfolio. The hull has a *left hand side*, LHS, and a *right hand side*, RHS. The LHS is composed of the efficient frontier and the minimum CVaR locus, and the RHS is composed of the maximum CVaR locus. The LHS is easy to calculate with the help of a linear programming solver. The RHS is much more complex, i.e. the function to be optimized is non-convex. Thus we solve the rhs by optimizing all pairwise portfolios, just like for the Markowitz Portfolio or the MAD portfolio. The union of the pairwise solutions yields the rhs of the hull.
In the first step we derive the efficient mean-CVaR frontier. Let us start with the R/AMPL model file. For this we can use the model file from the CVAREDR (equi-distant return) portfolio.

```
> modelCVARMINHULL <- c(
   "param N ;",
   "param S ;",
   "param Scenarios{1..S,1..N} ;",
   "param mu{1..N} ;",
   "param alpha ;",
   "param targetReturn ;",
   "param minReturn ;",
   "param maxReturn ;",
   "param nReturn ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var VaR ;",
   "var x{1..S} >= 0 ;",
   "maximize Objective: VaR - ( sum{s in 1..S} x[s] ) / ( alpha*S ) ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to Reward: sum{i in 1..N} mu[i] * w[i] = targetReturn ;",
   "subject to CVaR{s in 1..S}: sum{i in 1..N} Scenarios[s,i] * w[i] - VaR + x[s] >= 0 ;")
> amplModelFile(model=modelCVARMINHULL, project="myPortfolio")
```

To complete the specification we add the R/AMPL run file

```
> runCVARMINHULL <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex;",
   "for {i in 0..nReturn} {",
   "  let targetReturn := minReturn + i*(maxReturn-minReturn)/nReturn ;",
   "  solve ;",
   "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
   "};",
   "exit ;")
> amplRunFile(run=runCVARMINHULL, project="myPortfolio")
```

and the R/AMPL data file

```
> requiredData(modelCVARMINHULL)
[1] "N"            "S"            "Scenarios"    "mu"           "alpha"
[6] "targetReturn" "minReturn"    "maxReturn"    "nReturn"
```

```
>   Scenarios <- 100*LPP2005REC[1:250,1:6]
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   alpha <- 0.05
>   targetReturn <- 999
>   minReturn <- min(mu)
>   maxReturn <- max(mu)
>   nReturn <- 33
> dataCVARMINHULL <- dataAUTO(modelCVARMINHULL)
> amplDataFile(data=dataCVARMINHULL, project="myPortfolio")
```

Now we are ready to optimize the portfolio and to extract the optimal weights and CVaR value.

```
>   system("ampl myPortfolio.run > myPortfolio.out")
>   weightsCVARMINHULL <- matrix(as.numeric(scan("myPortfolio.txt")), byrow=TRUE, ncol=N)
>   colnames(weightsCVARMINHULL) <- colnames(Scenarios)
>   rownames(weightsCVARMINHULL) <- paste0("CVARHULL-", 0:nReturn)
>   for (i in 1:(nReturn+1)){
     Risk <- c(Risk,
               -as.numeric(pfolioCVaR(Scenarios,weightsCVARMINHULL[i,],alpha=alpha)[1]))}
>    Return <- (mu %*% t(weightsCVARMINHULL))[1,]
```

The input for the `CVaRHull()` are the column means (`mu`) of the assets, the `Scenarios`, the percentile α, and the `Return` and `Risk` values along the minimum variance locus and the efficient frontier.

```
> CVaRHull <- function(mu,  Scenarios,alpha, Return, Risk) {
   # Minimum Risks:
   minRisks <- Risk
   Returns <- risks <- Return
   # Maximum Risks:
   maxRisks <- rep(-Inf, length(Returns))
   nAssets <- ncol(Scenarios)
   for (i in 1:(nAssets - 1)) {
     for (j in (i + 1):nAssets) {
       mu2 <- mu[c(i, j)]
       Scenarios2 <- Scenarios[, c(i, j)]
       Index <- which(Returns >= min(mu2) & Returns <= max(mu2))
       if (length(Index) > 0) {
         Index <- (1:length(Returns))[Index]
         for (k in Index) {
           weights <- (Returns[k] - mu2[2])/(mu2[1] - mu2[2])
           weights <- c(weights, 1 - weights)
           Risk <- -pfolioCVaR(Scenarios2,weights,alpha)[1]
           maxRisks[k] <- max(maxRisks[k], Risk) }
       }
     }
   }
   # Hull:
   risk <- c(minRisks, rev(maxRisks[-1])[-1], minRisks[1])
   return <- c(Returns, rev(Returns[-1])[-1], Returns[1])
   hull <- cbind(Risks = risk, Returns = return)
   # Return Value:
```

```
    hull
}
```

Now we only have to execute the function with the minimum CvaR locus as input in order to retrieve the whole hull:

```
> hull <- CVaRHull(mu, Scenarios, alpha, Return, Risk)
```

CHAPTER 11

MINIMAX PORTFOLIOS

11.1 INTRODUCTION

In this chapter we introduce the the *MiniMax Portfolio*, also called the minimum regret portfolio. The topics are:

- MinRegret Portfolio
- Efficient minRegret Portfolio

Throughout this chapter we use as an example the Swiss pension fund benchmark portfolio. The data are part of the Rmetrics package `timeSeries` and are loaded together with the `fPortfolio` package.

11.2 MINIMAX PORTFOLIO

The Minimax Portfolio maximizes the minimum portfolio return for a set of return scenarios (historical data in our case). This minimum return is also regret. Given the portfolio weights w and the matrix of asset returns R, the regret can be easily calculated using

$$\text{Regret} = \min\left(R \cdot w'\right) \tag{11.1}$$

Note that the Minimax Portfolio is a special case of the CVaR portfolio, namely the portfolio with $\alpha = 1/S$.

The regret of a portfolio can be maximizes by solving the following linear program.

$$\begin{aligned} & \max_{w,R_{\min}} \quad R_{\min} \qquad (11.2) \\ s.t. & \\ \forall s \in \{1..S\} : \sum_{i=1}^{N} r_{s,i} w_i \quad & \geq \quad R_{\min} \\ 1'w \quad & = \quad 1 \\ w_i \quad & \geq \quad 0 \end{aligned}$$

Here N is the number of assets, S is the number of scenarios, $r_{s,i}$ are the daily financial returns of asset i, w are the portfolio weights. Note that the first constraint means that $R_{\min}$ is at least as small as the minimal portfolio return $\min(r\,w)$.

It is now straightforward to define the R/AMPL model file:

```
> modelMINIMAXGLOB <- c(
   "param N ;",
   "param S ;",
   "param Scenarios{1..S,1..N} ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var Rmin ;",
   "maximize Objective: Rmin ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to Min{s in 1..S}: sum{i in 1..N} Scenarios[s,i] * w[i] >= Rmin ;")
> amplModelFile(model=modelMINIMAXGLOB, project="myPortfolio")
```

Since the model is linear, we can as usual use our standard run file:

```
> runMINIMAXGLOB <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "printf \"%16.6f\", Objective > myPortfolio.par ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runMINIMAXGLOB, project="myPortfolio")
```

The data file needs the following input:

```
> requiredData(modelMINIMAXGLOB)
[1] "N"         "S"         "Scenarios"

>   Scenarios <- 100*LPP2005REC[, 1:6]
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
> dataMINIMAXGLOB <- dataAUTO(modelMINIMAXGLOB)
> amplDataFile(data=dataMINIMAXGLOB, project="myPortfolio")
```

Optimize the model and extract the optimal weights:

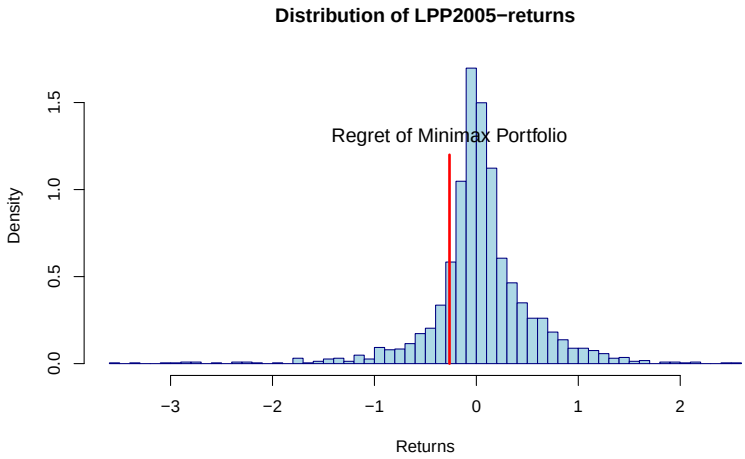


FIGURE 11.1: The distribution of the assets in the LPP2005 index, and the maximal regret of the computed Minimax Portfolio.

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMINIMAXGLOB <- as.numeric(scan("myPortfolio.txt"))
> names(weightsMINIMAXGLOB) <- colnames(Scenarios)
```

Summarize the results:

```
> SummaryMINIMAXGLOB <- c(
 Return = mu %*% weightsMINIMAXGLOB,
 Regret = min(Scenarios%*%weightsMINIMAXGLOB),
 CovarianceRisk = sqrt ( weightsMINIMAXGLOB %*% Sigma %*% weightsMINIMAXGLOB ),
 HerfindahlIndex = 1 - weightsMINIMAXGLOB %*% weightsMINIMAXGLOB)
> SummaryMINIMAXGLOB

         Return          Regret  CovarianceRisk HerfindahlIndex
      0.0094479      -0.2635741       0.1093068       0.2230009
```

In figure 11.1, the position of the regret of the computed Minimax Portfolio compared to the return distribution of the individual assets is depicted. It is visible that the maximal regret of the portfolio is significantly reduced.

11.3 EFFICIENT MINIMAX PORTFOLIO

For the efficient Minimax Portfolio, we just have to add the constraint on the expected return.

$$\max_{w,R_{min}} \quad R_{min} \tag{11.3}$$

$$s.t.$$

$$\begin{aligned} \sum_{i=1}^{N} r_{s,i} w_i &\geq R_{min} \\ \mu' w &\geq \overline{r} \\ 1' w &= 1 \\ w_i &\geq 0 \end{aligned}$$

Here $\overline{r}$ is the target return, N is the number of assets, S is the number of scenarios, $r_{s,i}$ are the daily financial returns of asset i, w are the portfolio weights.

It is now straightforward to define the R/AMPL model file:

```
> modelMINIMAX1 <- c(
   "param N ;",
   "param S ;",
   "param Scenarios{1..S,1..N} ;",
   "param mu{1..N} ;",
   "param targetReturn ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var Rmin ;",
   "maximize Objective: Rmin ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to Reward: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;",
   "subject to Min{s in 1..S}: sum{i in 1..N} Scenarios[s,i] * w[i] >= Rmin ;")
> amplModelFile(model=modelMINIMAX1, project="myPortfolio")
```

The model remains linear, thus we do not have the change the solver.

```
> runMINIMAX1 <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "printf \"%16.6f\", Objective > myPortfolio.par ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runMINIMAX1, project="myPortfolio")
```

The data file needs the following input:

```
> requiredData(modelMINIMAX1)
[1] "N"           "S"           "Scenarios"    "mu"          "targetReturn"
>   Scenarios <- 100*LPP2005REC[, 1:6]
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   targetReturn <- mean(Scenarios)
```

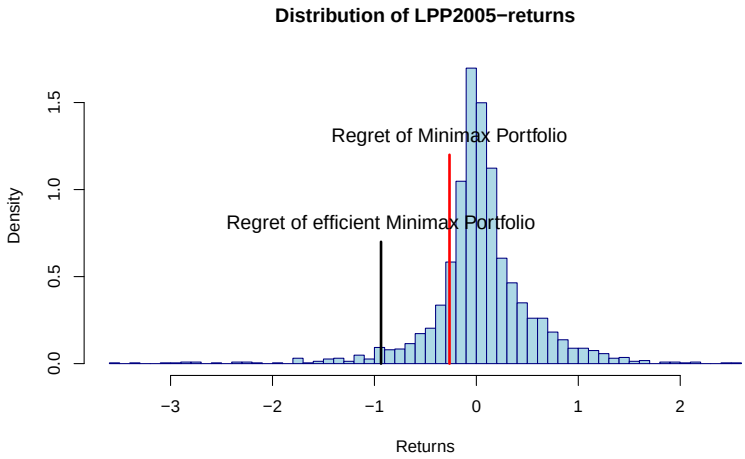


FIGURE 11.2: The distribution of the assets in the LPP2005 index, and the maximal regret of the computed Minimax Portfolio and efficient Minimax Portfolio.

```
>   mu <- colMeans(Scenarios)
> dataMINIMAX1 <- dataAUTO(modelMINIMAX1)
> amplDataFile(data=dataMINIMAX1, project="myPortfolio")
```

As usual we have taken the expected return of the equal-weights-portfolio as the desired target return. Optimize the model and extract the optimal weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMINIMAX1 <- as.numeric(scan("myPortfolio.txt"))
> names(weightsMINIMAX1) <- colnames(Scenarios)
```

Summarize the results:

```
> SummaryMINIMAX1 <- c(
 Return = mu %*% weightsMINIMAX1,
 Regret = min(Scenarios%*%weightsMINIMAX1),
 CovarianceRisk = sqrt ( weightsMINIMAX1 %*% Sigma %*% weightsMINIMAX1 ),
 HerfindahlIndex = 1 - weightsMINIMAX1 %*% weightsMINIMAX1)
> SummaryMINIMAX1
        Return          Regret  CovarianceRisk HerfindahlIndex
      0.043077       -0.935348        0.279717        0.427825
```

In figure 11.2, it is visible that the maximal regret of the Minimax Portfolio decreases significantly when we introduce a constraint on the expected return.

11.4 EQUI-DISTANT RETURN FRONTIER

As usual, we add the three additional parameters to the model file to calculate the efficient frontier:

```
> modelMINIMAXEDR <- c(
   "param minReturn ;",
   "param maxReturn ;",
   "param nReturn ;",

   "param N ;",
   "param S ;",
   "param Scenarios{1..S,1..N} ;",
   "param mu{1..N} ;",
   "param targetReturn ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var Rmin ;",
   "maximize Objective: Rmin ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to Reward: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;",
   "subject to Min{s in 1..S}: sum{i in 1..N} Scenarios[s,i] * w[i] >= Rmin ;")
> amplModelFile(model=modelMINIMAXEDR, project="myPortfolio")
```

We use the same run-file as for the CVaR return frontier:

```
> runMINIMAXEDR <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex;",
   "let targetReturn := -999;",
   "solve;",
   "let minReturn := sum{i in 1..N} w[i]*mu[i];",
   "let maxReturn := max{i in 1..N} mu[i];",
   "for {i in 0..nReturn} {",
   "  let targetReturn := minReturn + i*(maxReturn-minReturn)/nReturn ;",
   "  solve ;",
   "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
   "};",
   "exit ;")
> amplRunFile(run=runMINIMAXEDR, project="myPortfolio")
```

In order to get a value for the `minReturn`-parameter, we solve the problem first for a target return set to $-\infty$ in order to calculate the expected return of the minimum risk portfolio. We then define the values for the `minReturn`- and `maxReturn`-parameter. Afterwards, we introduce a loop and replace the value of $\overline{r}$ for every iteration.
The data file needs the following input:

```
> requiredData(modelMINIMAXEDR)
[1] "minReturn"    "maxReturn"    "nReturn"      "N"            "S"
[6] "Scenarios"    "mu"           "targetReturn"

>   Scenarios <- 100*LPP2005REC[, 1:6]
>   N <- ncol(Scenarios)
```

```
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   targetReturn <- NA
>   minReturn <- NA
>   maxReturn <- NA
>   nReturn <- 33
> dataMINIMAXEDR <- dataAUTO(modelMINIMAXEDR)
> amplDataFile(data=dataMINIMAXEDR, project="myPortfolio")
```

Optimize the model and extract the optimal weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMINIMAXEDR <- matrix(as.numeric(scan("myPortfolio.txt")), byrow=TRUE, ncol=N)
> colnames(weightsMINIMAXEDR) <- colnames(Scenarios)
> rownames(weightsMINIMAXEDR) <- 1:(nReturn+1)
```

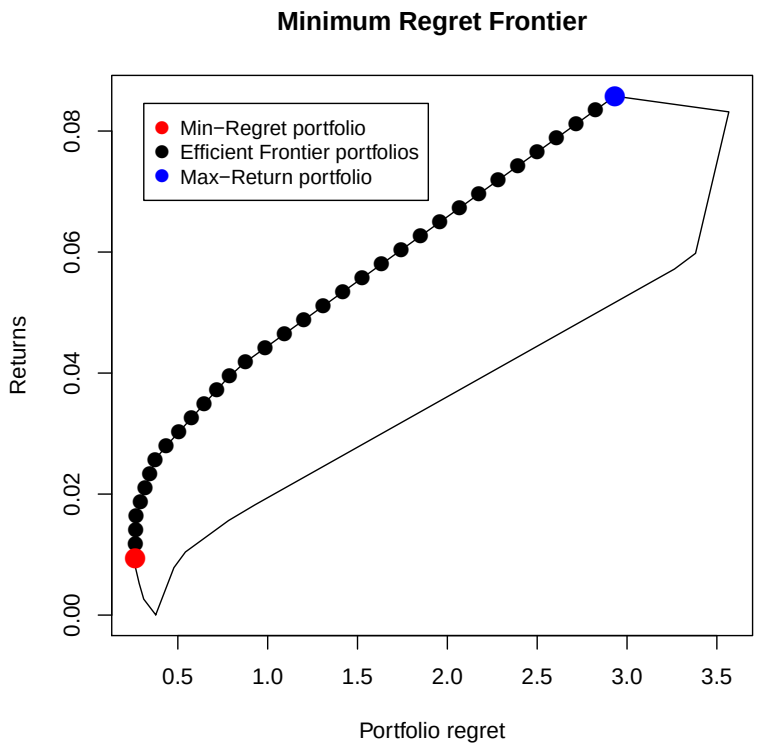


FIGURE 11.3: Regret hull, minimum regret portfolio and efficient frontier portfolios

11.5 HULL

As always, we calculate the hull in two steps:

- First, we calculate the minimum regret locus with AMPL.
- Then we calculate the maximum regret frontier by comparing the pairwise blinds between each asset.

Step 1: Minimum regret locus

For the first step, we use the modified EDR-model file with the target return now being am equality constraint:

```
> modelMINIMAXMINHULL <- c(
   "param minReturn ;",
   "param maxReturn ;",
   "param nReturn ;",
   "param N ;",
   "param S ;",
   "param Scenarios{1..S,1..N} ;",
   "param mu{1..N} ;",
   "param targetReturn ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var Rmin ;",
   "maximize Objective: Rmin ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to Reward: sum{i in 1..N} mu[i] * w[i] = targetReturn ;",
   "subject to Min{s in 1..S}: sum{i in 1..N} Scenarios[s,i] * w[i] >= Rmin ;")
> amplModelFile(model=modelMINIMAXMINHULL, project="myPortfolio")
```

To complete the specification, we add the R/AMPL run file;

```
> runMINIMAXMINHULL <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex;",
   "for {i in 0..nReturn} {",
   "  let targetReturn := minReturn + i*(maxReturn-minReturn)/nReturn ;",
   "  solve ;",
   "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
   "};",
   "exit ;")
> amplRunFile(run=runMINIMAXMINHULL, project="myPortfolio")
```

and the R/AMPL data file:

```
> requiredData(modelMINIMAXMINHULL)
[1] "minReturn"    "maxReturn"    "nReturn"      "N"            "S"
[6] "Scenarios"    "mu"           "targetReturn"
```

```
>   Scenarios <- 100*LPP2005REC[,1:6]
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   targetReturn <- NA
>   minReturn <- min(mu)
>   maxReturn <- max(mu)
>   nReturn <- 33
```

```
> dataMINIMAXMINHULL <- dataAUTO(modelMINIMAXMINHULL)
> amplDataFile(data=dataMINIMAXMINHULL, project="myPortfolio")
```

Now we are ready to optimize the portfolio and to extract the optimal weights and the regret values.

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMINIMAXMINHULL <- matrix(as.numeric(scan("myPortfolio.txt")), byrow=TRUE, ncol=N)
> colnames(weightsMINIMAXMINHULL) <- colnames(Scenarios)
> rownames(weightsMINIMAXMINHULL) <- paste0("MINIMAXHULL-", 0:nReturn)
> Return <- weightsMINIMAXMINHULL%*%mu
> Risk <- -colMins(Scenarios%*%t(weightsMINIMAXMINHULL))
```

Step 2: Maximum regret frontier

Here we write a function that calculates the maximum regret frontier by pairwise blind calculation and finding the largest line elements afterwards. We use the minimum regret locus as the input:

```
> MinimaxHull <- function(mu,  Scenarios,Return, Risk) {
   # Minimum Risks:
   minRisks <- Risk
   Returns <- risks <- Return
   # Maximum Risks:
   maxRisks <- rep(-Inf, length(Returns))
   nAssets <- ncol(Scenarios)
   for (i in 1:(nAssets - 1)) {
     for (j in (i + 1):nAssets) {
       mu2 <- mu[c(i, j)]
       Scenarios2 <- Scenarios[, c(i, j)]
       Index <- which(Returns >= min(mu2) & Returns <= max(mu2))
       if (length(Index) > 0) {
         Index <- (1:length(Returns))[Index]
         for (k in Index) {
           weights <- (Returns[k] - mu2[2])/(mu2[1] - mu2[2])
           weights <- c(weights, 1 - weights)
           Risk <- -min(Scenarios2%*%weights)
           maxRisks[k] <- max(maxRisks[k], Risk) }
       }
     }
   }
   # Hull:
   risk <- c(minRisks, rev(maxRisks[-1])[-1], minRisks[1])
   return <- c(Returns, rev(Returns[-1])[-1], Returns[1])
   hull <- cbind(Risks = risk, Returns = return)
   # Return Value:
   hull
 }
```

We can now calculate the hull with the minimum regret locus as the function input:

```
> hull <- MinimaxHull(mu, Scenarios,Return, Risk)
```

PART V

MEAN-CDAR DESIGNS

CHAPTER 12

MEAN-CDAR PORTFOLIOS

12.1 INTRODUCTION

In this chapter we introduce the Mean-CDaR Portfolio concept and show how to solve the following types of portfolios

- CDAR1 - Minimum Risk Mean-CDaR Efficient Portfolio
- CDARGLOB - Global Minimum Risk Efficient Portfolio
- CDAR2 - Maximum Return Mean-CDaR Efficient Portfolio
- CDAREDR - Equi-distant Return Mean-CDaR Frontier
- CDAR3 - Mean-CDaR Critical Line Algorithm
- CDARSORTINO - Reward/CDaR Ratio Portfolio
- CDARDIV - Herfindahl Risk Diversification
- CDARHULL - Mean-CDaR Hull
- CDARSET - Mean-CDaR Feasible Set

In contrast to the mean-variance and mean-CVaR pprtfolio concepts drawdown risk is a strongly path dependent measure. With the exception of Brownian motion with zero drift, there is no closed form solution for the distribution of this measure available, see Douady, Yor, and Shiryaev et al. [1999], cited by Ghalanos [2005].
The portfolio considered here and its optimization was introduced by Chekhlov, Uryasev, and Zabarankin [2005]. The concept used by the authors originates from the *Conditional Value-at-Risk* approach, CVaR. Cheklov et al. propose a one-parameter family of risk measures defined on the portfolio's return sample path and call it *Conditional Drawdown-at-Risk*,

CDaR. Their risk measures depend on the portfolio drawdown or "underwater" curve considered in active portfolio management. For some value of the tolerance parameter α, the CDaR is defined as the mean of the worst $(1-\alpha)$ times 100% drawdowns. They also showed that the CDaR risk function contains the *Maximal Drawdown* and the *Average Drawdown* as its limiting cases.
Please note that although the theoretical definition of the CDaR is well defined, there exist different techniques to estimate the CDaR, which do not necessarily yield the exact same value. During this chapter, we will use the following function to estimate the CDaR from existing portfolio weights :

```
> pfolioCDaR <-  function (x, weights = NULL, alpha = 0.05)
 {
   data = as.matrix(cumulated(x))
   if (is.null(weights)) {
     weights = rep(1/dim(data)[[2]], dim(data)[[2]])
   }
   n = dim(data)[1]
   Rfp = apply(t(t(data) * weights), 1, sum)
   downs <- NULL
   for (i in 1:length(Rfp)){
     downs <- c(downs, Rfp[i]-max(Rfp[1:i]))
   }
   sorted = sort(downs)
   n.alpha = floor(n * alpha)
   DaR = sorted[n.alpha]
   n.alpha = max(1, floor(n * alpha) - 1)
   CDaRplus = mean(sorted[1:n.alpha])
   lambda = 1 - floor(n * alpha)/(n * alpha)
   ans = - as.vector(lambda * DaR + (1 - lambda) * CDaRplus)
   names(ans) = "CDaR"
   attr(ans, "control") = c(CDaRplus = -CDaRplus, lambda = lambda)
   ans
   }
```

Please also note that in the previous chapters, we multiplied the returns with 100 to enhance the numerical stability of the calculations. For the conditional drawdown at risk, it is necessary that the **returns are left unchanged**, since the drawdown is dependent on the accumulated wealth. Any modification in the magnitude of the returns leads to exponential deviations of the wealth, and therefore drastically decreases the precision of the portfolio estimation.

In this chapter we show how to implement various flavors of mean-CDaR portfolios. These include, minimum risk and maximum return formulations, optimal portfolios along the efficient frontier, the calculation of the hull and the feasible set.[1]

[1] Throughout this chapter we use as an example the Swiss pension fund benchmark

12.2 GLOBAL MINIMUM RISK EFFICIENT PORTFOLIO

Similarly to the *CVaR* global minimum risk model, we can use linear programming to minimize the conditional drawdown at risk within our portfolio constraints:

$$\min_{w,\mathrm{DaR},x,u} \quad \mathrm{DaR} + \frac{1}{\alpha S}\sum_{s=1}^{S} x_s \tag{12.1}$$

$$\begin{aligned} s.t. & \\ 1'w &= 1 \\ w_i &\geq 0 \\ x_s - u_s + \mathrm{DaR} &\geq 0 \\ \sum_{i=1}^{N} r_{s,i} w_i + u_s - u_{s-1} &\geq 0 \\ u_0 &= 0 \\ x_s &\geq 0 \end{aligned}$$

Please note that we have to introduce additional variables u to the problem in order to model the path of the drawdowns.
The R/AMPL model file is then defined similarly to the `CDARGLOB` model:

```
> modelCDARGLOB <- c(
   "param N ;",
   "param S ;",
   "param Scenarios{1..S,1..N} ;",
   "param alpha ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var DaR ;",
   "var u{0..S} >= 0 ;",
   "var x{1..S} >= 0 ;",
   "minimize Objective: DaR + ( sum{s in 1..S} x[s] ) / ( alpha*S ) ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to Risk1{s in 1..S}: x[s] -u[s] + DaR >= 0; ",
   "subject to Risk2: u[0] = 0 ;",
   "subject to Risk3{s in 2..S}: u[s] - u[s-1] + (sum{i in 1..N} Scenarios[s,i] * w[i]) >= 0 ;")
> amplModelFile(model=modelCDARGLOB, project="myPortfolio")
```

Again, we use our standard linear run file:

```
> runCDARGLOB <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
```

portfolio. The data are part of the Rmetrics package `timeSeries` and are loaded together with the `fPortfolio` package.

```
   "exit ;")
> amplRunFile(run=runCDARGLOB, project="myPortfolio")
```

From the portfolio settings we construct the R/AMPL data file:

```
> requiredData(modelCDARGLOB)
[1] "N"         "S"         "Scenarios" "alpha"
>   Scenarios <- LPP2005REC[1:200,1:6]
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   alpha <- 0.05
> dataCDARGLOB <- dataAUTO(model=modelCDARGLOB)
> amplDataFile(data=dataCDARGLOB, project="myPortfolio")
```

Optimize and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsCDARGLOB <- as.numeric(scan("myPortfolio.txt"))
> names(weightsCDARGLOB) <- colnames(Scenarios)
> weightsCDARGLOB
     SBI      SPI      SII      LMI      MPI      ALT
0.000000 0.000000 0.142108 0.845540 0.000000 0.012353
```

Summarize the results:

```
> Sigma <- cov(Scenarios)
> SummaryCDARGLOB <- c(
 Return = mu %*% weightsCDARGLOB,
 CDaR = as.numeric(pfolioCDaR(Scenarios,weightsCDARGLOB,alpha=alpha)[1]),
 CovarianceRisk = sqrt ( weightsCDARGLOB %*% Sigma %*% weightsCDARGLOB ),
 HerfindahlIndex = 1 - weightsCDARGLOB %*% weightsCDARGLOB)
> SummaryCDARGLOB
         Return            CDaR  CovarianceRisk HerfindahlIndex
     1.4871e-05      2.2240e-02      1.2103e-03      2.6471e-01
```

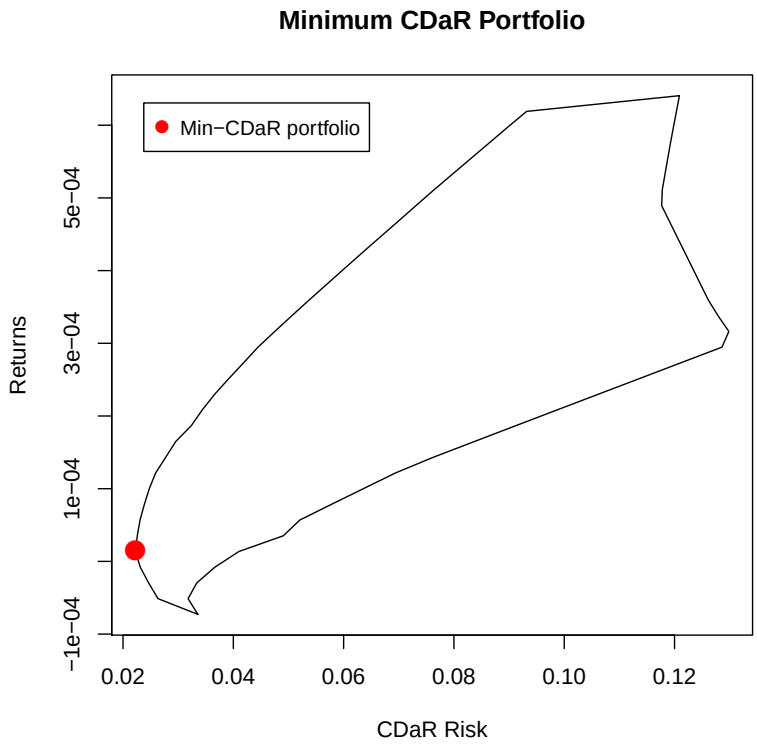


FIGURE 12.1: Feasible set and the position of the minimum CDaR portfolio.

12.3 MINIMUM RISK MEAN-CDAR EFFICIENT PORTFOLIOS

To get the efficient mean-CDaR portfolio, we just have to add our target-return constraint:

$$\min_{w,\mathrm{DaR},x,u} \quad \mathrm{DaR} + \frac{1}{\alpha S}\sum_{s=1}^{S} x_s \tag{12.2}$$

$$\begin{aligned} s.t. & \\ 1'w &= 1 \\ \mu' w &\geq \overline{r} \\ w_i &\geq 0 \\ x_s - u_s + \mathrm{DaR} &\geq 0 \\ \sum_{i=1}^{N} r_{s,i} w_i + u_s - u_{s-1} &\geq 0 \\ u_0 &= 0 \\ x_s &\geq 0 \end{aligned}$$

Compose R/AMPL Model File:

```
> modelCDAR1 <- c(
   "param N ;",
   "param S ;",
   "param Scenarios{1..S,1..N} ;",
   "param mu{1..N} ;",
   "param targetReturn ;",
   "param alpha ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var DaR ;",
   "var u{0..S} >= 0 ;",
   "var x{1..S} >= 0 ;",
   "minimize Objective: DaR + ( sum{s in 1..S} x[s] ) / ( alpha*S ) ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to Reward: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;",
   "subject to Risk1{s in 1..S}: x[s] -u[s] + DaR >= 0; ",
   "subject to Risk2: u[0] = 0 ;",
   "subject to Risk3{s in 2..S}: u[s] - u[s-1] + (sum{i in 1..N} Scenarios[s,i] * w[i]) >= 0 ;")
> amplModelFile(model=modelCDAR1, project="myPortfolio")
```

R/AMPL run file:

```
> runCDAR1 <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "printf \"%16.6f\", Objective > myPortfolio.par ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runCDAR1, project="myPortfolio")
```

From the portfolio settings we construct the R/AMPL data file:

```
> Scenarios <- LPP2005.RET[1:200, 1:6]
> requiredData(modelCDAR1)
```

```
[1] "N"            "S"            "Scenarios"    "mu"           "targetReturn"
[6] "alpha"
```

```
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   alpha <- 0.05
>   mu <- colMeans(Scenarios)
>   targetReturn <- mean(mu)
> dataCDAR1 <- dataAUTO(model=modelCDAR1)
> amplDataFile(data=dataCDAR1, project="myPortfolio")
```

Optimize the CDaR1 model for the grand mean and extract the optimal weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsCDAR1 <- as.numeric(scan("myPortfolio.txt"))
> names(weightsCDAR1) <- colnames(Scenarios)
> weightsCDAR1
     SBI      SPI      SII      LMI      MPI      ALT
0.000000 0.087128 0.230992 0.375565 0.000000 0.306315
```

Summarize the results:

```
> Sigma <- cov(Scenarios)
> SummaryCDaR1 <- c(
 Return = mu %*% weightsCDAR1,
 CDaR = as.numeric(pfolioCDaR(Scenarios,weightsCDAR1,alpha=alpha)[1]),
 CovarianceRisk = sqrt ( weightsCDAR1 %*% Sigma %*% weightsCDAR1 ),
 HerfindahlIndex = 1 - weightsCDAR1 %*% weightsCDAR1)
> SummaryCDaR1
         Return            CDaR  CovarianceRisk HerfindahlIndex
     0.00025375      0.03947891      0.00237210      0.70417346
```

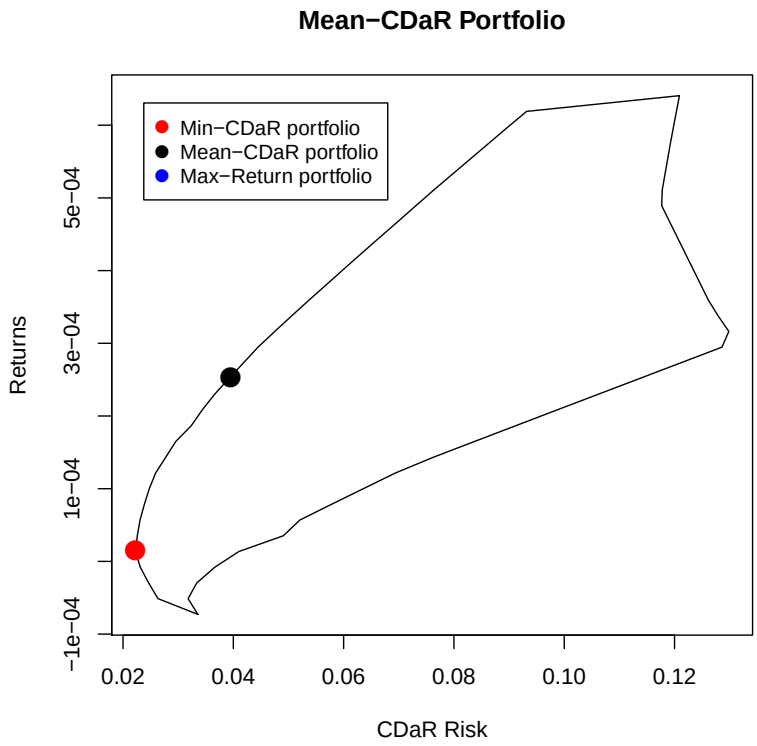


FIGURE 12.2: Feasible set and the position of an efficient Mean-CDaR portfolio.

12.4 MAXIMUM RETURN MEAN-CDAR EFFICIENT PORTFOLIO

For this case we fix the CDaR risk and maximize the portfolio's return. In a similar manner as for the conditional value at risk, we can set the CDaR as a constraint and maximize the return while keeping the linear programming approach:

$$\max_{w,VaR,x,u} \quad \mu' w \tag{12.3}$$

$$s.t.$$

$$\begin{aligned} 1'w &= 1 \\ \mu' w &\leq \overline{r} \\ x_s - u_s + \text{DaR} &\geq 0 \\ \sum_{i=1}^{N} r_{s,i} w_i + u_s - u_{s-1} &\geq 0 \\ u_0 &= 0 \\ x_s &\geq 0 \end{aligned}$$

In the R/AMPL file of the `CDAR1` model we have to interchange the objective with the return constraints.

```
> modelCDAR2 <- c(
   "param N ;",
   "param S ;",
   "param Scenarios{1..S,1..N} ;",
   "param mu{1..N} ;",
   "param targetRisk ;",
   "param alpha ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var DaR ;",
   "var u{0..S} >= 0 ;",
   "var x{1..S} >= 0 ;",
   "maximize Objective: sum{i in 1..N} mu[i] * w[i] ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to Risk1:  DaR + ( sum{s in 1..S} x[s] ) / ( alpha*S ) <= targetRisk ;",
   "subject to Risk2{s in 1..S}: x[s] -u[s] + DaR >= 0; ",
   "subject to Risk3: u[0] = 0 ;",
   "subject to Risk4{s in 2..S}: u[s] - u[s-1] + (sum{i in 1..N} Scenarios[s,i] * w[i]) >= 0 ;")
> amplModelFile(model=modelCDAR2, project="myPortfolio")
```

R/AMPL run file:

```
> runCDAR2 <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "printf \"%16.6f\", DaR + ( sum{s in 1..S} x[s] ) / ( alpha*S ) > myPortfolio.par ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runCDAR2, project="myPortfolio")
```

From the portfolio settings we construct the R/AMPL data file. As always, we use the risk of the portfolio calculated in the `CDAR1` approach as target risk:

```
> Scenarios <- LPP2005REC[1:200,1:6]
> requiredData(modelCDAR2)
[1] "N"          "S"          "Scenarios"  "mu"         "targetRisk"
[6] "alpha"

>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   alpha <- 0.05
>   mu <- colMeans(Scenarios)
>   targetRisk <- SummaryCDaR1[2] # Grand mean CDaR Risk
> dataCDAR2 <- dataAUTO(model=modelCDAR2)
> amplDataFile(data=dataCDAR2, project="myPortfolio")
```

Optimize and extract the optimal weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsCDAR2 <- as.numeric(scan("myPortfolio.txt"))
> names(weightsCDAR2) <- colnames(Scenarios)
> weightsCDAR2
    SBI     SPI     SII     LMI     MPI     ALT
0.00000 0.10053 0.10778 0.42608 0.00000 0.36560
```

Similary to the CVaR-constrained maximum return portfolio, we only get approximately the same weights as for the CDaR1-approach:

```
> weightsCDAR1
     SBI      SPI      SII      LMI      MPI      ALT
0.000000 0.087128 0.230992 0.375565 0.000000 0.306315
```

This is because of the differences in estimating the CDaR of a portfolio. To feed the CDaR constraint into the AMPL data file, we use our above defined pfolioCDaR function. In the AMPL model file, the CDaR constraint might correspond to a slightly different portfolio since equation 12.3 estimates the CDaR-value slightly lower than our function. If we would use the same CDaR-estimator as in the model, we would receive exactly the same weights.

Summarize the results:

```
> Sigma <- cov(Scenarios)
> SummaryCDaR2 <- c(
 Return = mu %*% weightsCDAR2,
 CDaR = as.numeric(scan("myPortfolio.par")),
 CovarianceRisk = sqrt ( weightsCDAR2 %*% Sigma %*% weightsCDAR1 ),
 HerfindahlIndex = 1 - weightsCDAR2 %*% weightsCDAR2)
> SummaryCDaR2
         Return            CDaR  CovarianceRisk HerfindahlIndex
     0.00029553      0.03947900      0.00250585      0.66306384
```

12.5 EQUI-DISTANT RETURN FRONTIER

To compute a set of optimal mean-CDaR portfolios along the efficient frontier we divide the frontier in `nReturn` pieces. The range starts at the minimum return `minReturn` given by the worst performing single asset and ends at the maximum return `maxReturn` given by the best performing single asset. Then we use the CDAR1 model with additional parameters `minReturn`, `maxReturn`, and `nReturn`.

```
> modelCDAREDR <- c(
   "param minReturn ;",
   "param maxReturn ;",
   "param nReturn ;",

   "param N ;",
   "param S ;",
   "param Scenarios{1..S,1..N} ;",
   "param mu{1..N} ;",
   "param alpha ;",
   "param targetReturn ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var DaR ;",
   "var u{0..S} >= 0 ;",
   "var x{1..S} >= 0 ;",
   "minimize Objective: DaR + ( sum{s in 1..S} x[s] ) / ( alpha*S ) ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to Reward: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;",
   "subject to Risk1{s in 1..S}: x[s] -u[s] + DaR >= 0; ",
   "subject to Risk2: u[0] = 0 ;",
   "subject to Risk3{s in 2..S}: u[s] - u[s-1] + (sum{i in 1..N} Scenarios[s,i] * w[i]) >= 0 ;")
> amplModelFile(model=modelCDAREDR, project="myPortfolio")
```

or in short:

```
> modelCDAREDR <- c(
   "param minReturn ;",
   "param maxReturn ;",
   "param nReturn ;",
   modelCDAR1)
> amplModelFile(model=modelCDAREDR, project="myPortfolio")
```

We can use the same run-file as for the CVaR return frontier:

```
> runCDAREDR <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex;",
   "let targetReturn := -999;",
   "solve;",
   "let minReturn := sum{i in 1..N} w[i]*mu[i];",
   "let maxReturn := max{i in 1..N} mu[i];",
   "for {i in 0..nReturn} {",
   "  let targetReturn := minReturn + i*(maxReturn-minReturn)/nReturn ;",
   "  solve ;",
```

```
    "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
    "};",
    "exit ;")
> amplRunFile(run=runCDAREDR, project="myPortfolio")
```

In order to get a value for the `minReturn`-parameter, we solve the problem first for a target return set to $-\infty$ in order to calculate the expected return of the minimum risk portfolio. We then define the values for the `minReturn`- and `maxReturn`-parameter. Afterwards, we introduce a loop and replace the value of $\overline{r}$ for every iteration.
The data file needs the following input:

```
> requiredData(modelCDAREDR)
[1] "minReturn"    "maxReturn"    "nReturn"      "N"            "S"
[6] "Scenarios"    "mu"           "targetReturn" "alpha"
```

```
>   Scenarios <- LPP2005REC[1:200,1:6]
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   alpha <- 0.05
>   targetReturn <- NA
>   minReturn <- NA
>   maxReturn <- NA
>   nReturn <- 33
> dataCDAREDR <- dataAUTO(modelCDAREDR)
> amplDataFile(data=dataCDAREDR, project="myPortfolio")
```

Optimize the portfolio and extract weights

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsCDAREDR <- matrix(as.numeric(scan("myPortfolio.txt")), byrow=TRUE, ncol=N)
> colnames(weightsCDAREDR) <- colnames(Scenarios)
> rownames(weightsCDAREDR) <- 1:(nReturn+1)
```

Summarize the results:

```
> SummaryEDR <- NULL
> CDaR <- as.numeric(scan("myPortfolio.par"))
> for (i in 0:nReturn) {
   summary <- c(
     Return = mu %*% weightsCDAREDR[i+1, ],
     CDaR = as.numeric(pfolioCDaR(Scenarios,weightsCDAREDR[i+1,],alpha=alpha)[1]),
     CovarianceRisk = sqrt ( weightsCDAREDR[i+1, ] %*% Sigma %*% weightsCDAREDR[i+1, ] ),
     HerfindahlIndex = 1 - weightsCDAREDR[i+1, ] %*% weightsCDAREDR[i+1, ])
   SummaryEDR <- rbind(SummaryEDR, summary)
 }
```

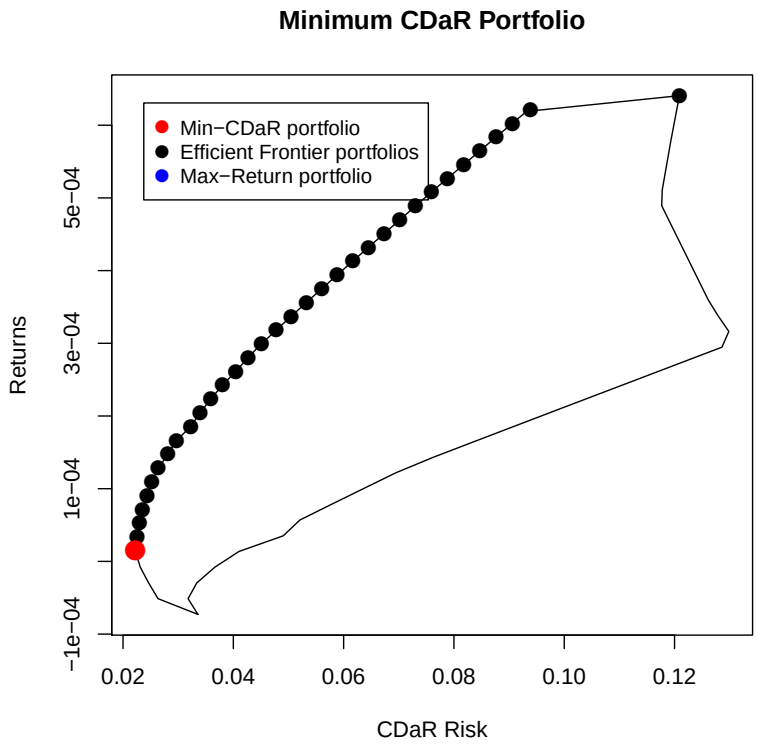


FIGURE 12.3: Feasible set and the position of the minimum CDaR portfolio.

12.6 THE CDAR CRITICAL LINE ALGORITHM

See chapter ...

12.7 REWARD/RISK RATIO PORTFOLIO

It is straightforward to define a Reward/Risk ratio for the conditional drawdown at risk in a similar manner as in the previous chapters:

$$\frac{\mu' w}{\text{CDaR}_\alpha(w)} \tag{12.4}$$

The Reward/Risk ratio portfolio is the portfolio that maximizes this ratio. Similar to the Sharpe ratio, it can be found by the following linear optimization problem:

$$\begin{aligned}
\min_{w,\text{DaR},x,t,u} \quad & \frac{1}{\alpha S}\sum_{s=1}^{S} x_s + \text{DaR} \\
s.t. \quad & \\
1' w &= t \\
\mu' w &= 1 \\
w_i &\geq 0 \\
x_s - u_s + \text{DaR} &\geq 0 \\
\sum_{i=1}^{N} r_{s,i} w_i + u_s - u_{s-1} &\geq 0 \\
u_0 &= 0 \\
x_s &\geq 0
\end{aligned} \tag{12.5}$$

R/AMPL model file

```
> modelCDARRATIO <- c(
   "param N ;",
   "param S ;",
   "param Scenarios{1..S,1..N} ;",
   "param mu{1..N} ;",
   "param alpha ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var DaR ;",
   "var x{1..S} >= 0 ;",
   "var u{0..S} >= 0 ;",
   "var t >= 0 ;",
   "minimize Objective: DaR + ( sum{s in 1..S} x[s] ) / ( alpha*S ) ;",
   "subject to Reward : sum{k in 1..N} mu[k] * w[k] = 1 ;",
   "subject to Budget: sum{i in 1..N} w[i] = t ;",
   "subject to Risk1{s in 1..S}: x[s] -u[s] + DaR >= 0; ",
   "subject to Risk2: u[0] = 0 ;",
   "subject to Risk3{s in 2..S}: u[s] - u[s-1] + (sum{i in 1..N} Scenarios[s,i] * w[i]) >= 0 ;")
> amplModelFile(model=modelCDARRATIO, project="myPortfolio")
```

R/AMPL run file:

```
> runCDARRATIO <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m]/t > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(runCDARRATIO, "myPortfolio")
```

R/AMPL data file:

```
> requiredData(modelCDARRATIO)
```

```
[1] "N"         "S"         "Scenarios" "mu"        "alpha"
```

```
>   Scenarios <- LPP2005REC[1:200,1:6]
>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   alpha <- 0.05
> dataCDARRATIO <- dataAUTO(modelCDARRATIO)
> amplDataFile(data=dataCDARRATIO, project="myPortfolio")
```

Optimize and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsCDARRATIO <- as.numeric(scan("myPortfolio.txt"))
> names(weightsCDARRATIO) <- colnames(Scenarios)
```

Summarize

```
> Sigma <- cov(Scenarios)
> SummaryCDARRATIO <- c(
 Return = mu %*% weightsCDARRATIO,
 CDaR = as.numeric(pfolioCDaR(weights=weightsCDARRATIO,x=Scenarios,alpha=0.05)[1]),
 CovarianceRisk = sqrt ( weightsCDARRATIO %*% Sigma %*% weightsCDARRATIO ),
 HerfindahlIndex = 1 - weightsCDARRATIO %*% weightsCDARRATIO,
 Ratio = mu %*% weightsCDARRATIO /(as.numeric(pfolioCDaR(weights=weightsCDARRATIO,x=Scenarios,alpha=0.05)[1]))
> SummaryCDARRATIO
```

```
         Return            CDaR  CovarianceRisk HerfindahlIndex           Ratio
     0.00051037      0.07627366      0.00466697      0.30500496      0.00669133
```

Plot the Risk/Reward Diagram:

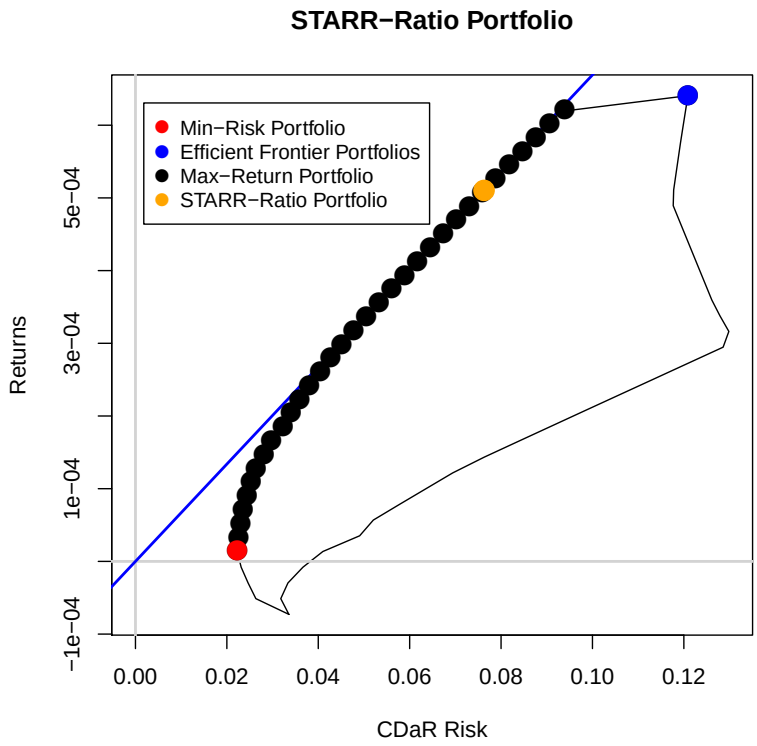
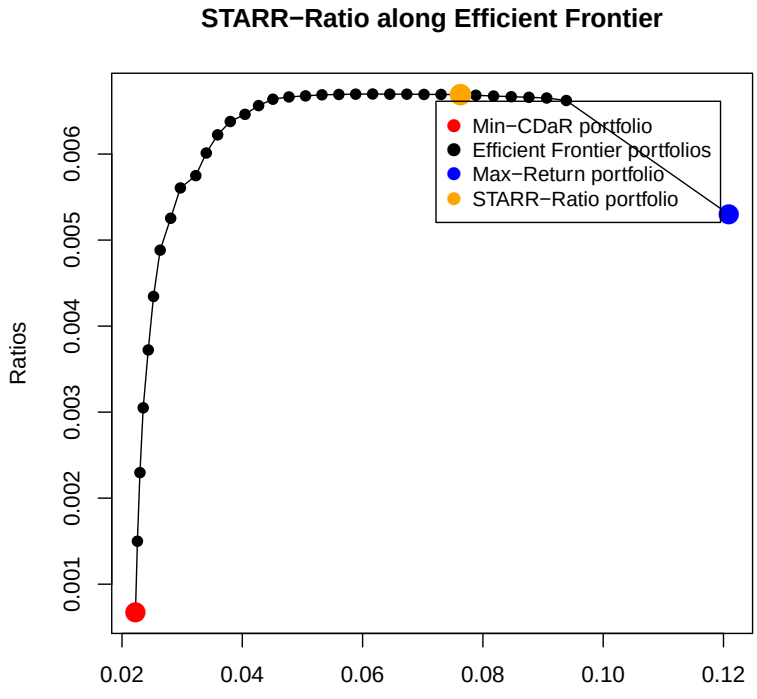


FIGURE 12.4: Position of the Reward/Risk-Ratio Portfolio



12.8 HULL

In this section we show how to write an R function, that computes the hull of the unconstrained mean-CDaR portfolio. The hull has a *left hand side*, LHS, and a *right hand side*, RHS. The LHS is composed of the efficient frontier and the minimum CDaR locus, and the RHS is composed of the maximum CDaR locus. The LHS is easy to calculate with the help of a linear programming solver. The RHS is much more complex, i.e. the function to be optimized is non-convex. Thus we solve the rhs by optimizing all pairwise portfolios, just like for the Markowitz Portfolio or the MAD portfolio. The union of the pairwise solutions yields the rhs of the hull.
In the first step we derive the efficient mean-CDaR frontier. Let us start with the R/AMPL model file. For this we can use the model file from the CDAREDR (equi-distant return) portfolio.
In the end, compose everything to get the polygon for the mean-CDaR hull.

```
> modelCDARMINHULL <- c(
   "param N ;",
   "param S ;",
   "param Scenarios{1..S,1..N} ;",
   "param mu{1..N} ;",
   "param alpha ;",
   "param Return ;",
   "param minReturn ;",
   "param maxReturn ;",
   "param nReturn ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var DaR ;",
   "var u{0..S} >= 0 ;",
   "var x{1..S} >= 0 ;",
   "minimize Objective: DaR + ( sum{s in 1..S} x[s] ) / ( alpha*S ) ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to Reward: sum{i in 1..N} mu[i] * w[i] = Return ;",
   "subject to Risk1{s in 1..S}: x[s] -u[s] + DaR >= 0; ",
   "subject to Risk2: u[0] = 0 ;",
   "subject to Risk3{s in 2..S}: u[s] - u[s-1] + (sum{i in 1..N} Scenarios[s,i] * w[i]) >= 0 ;")
> amplModelFile(model=modelCDARMINHULL, project="myPortfolio")
```

To complete the specification we add the R/AMPL run file

```
> runCDARMINHULL <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex;",
   "for {i in 0..nReturn} {",
   "  let Return := minReturn + i*(maxReturn-minReturn)/nReturn ;",
   "  solve ;",
   "  printf \"%16.12f\", Objective > myPortfolio.par ;",
   "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
   "};",
   "exit ;")
```

```
> amplRunFile(run=runCDARMINHULL, project="myPortfolio")
```

and the R/AMPL data file

```
> requiredData(modelCDARMINHULL)
[1] "N"         "S"         "Scenarios" "mu"        "alpha"     "Return"
[7] "minReturn" "maxReturn" "nReturn"

>   N <- ncol(Scenarios)
>   S <- nrow(Scenarios)
>   mu <- colMeans(Scenarios)
>   alpha <- 0.05
>   Return <- NA
>   minReturn <- min(mu)
>   maxReturn <- max(mu)
>   nReturn <- 33
> dataCDARMINHULL <- dataAUTO(modelCDARMINHULL)
> amplDataFile(data=dataCDARMINHULL, project="myPortfolio")
```

Now we are ready to optimize the portfolio and to extract the optimal weights and CDaR value.

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsCDARMINHULL <- matrix(as.numeric(scan("myPortfolio.txt")), byrow=TRUE, ncol=N)
> colnames(weightsCDARMINHULL) <- colnames(Scenarios)
> rownames(weightsCDARMINHULL) <- 1:(nReturn+1)
> Risk <- NULL
>   for (i in 1:(nReturn+1)){
     Risk <- c(Risk,
               as.numeric(pfolioCDaR(Scenarios,weightsCDARMINHULL[i,],alpha=alpha)[1]))}
>   Return <- (mu %*% t(weightsCDARMINHULL))[1,]
```

The input for the `CDaRHull()` are the column means (`mu`) of the assets, the `Scenarios`, the percentile α, and the `Return` and `Risk` values along the minimum variance locus and the efficient frontier.

```
> CDaRHull <- function(mu,  Scenarios, alpha, Return, Risk) {
   # Minimum Risks:
   minRisks <- Risk
   Returns <- risks <- Return
   # Maximum Risks:
   maxRisks <- rep(-Inf, length(Returns))
   nAssets <- ncol(Scenarios)
   for (i in 1:(nAssets - 1)) {
     for (j in (i + 1):nAssets) {
       mu2 <- mu[c(i, j)]
       Scenarios2 <- Scenarios[, c(i, j)]
       Index <- which(Returns >= min(mu2) & Returns <= max(mu2))
       if (length(Index) > 0) {
         Index <- (1:length(Returns))[Index]
         for (k in Index) {
           weights <- (Returns[k] - mu2[2])/(mu2[1] - mu2[2])
           weights <- c(weights, 1 - weights)
           Risk <- pfolioCDaR(Scenarios2,weights,alpha)[1]
           maxRisks[k] <- max(maxRisks[k], Risk) }
```

```
      }
    }
  }
  # Hull:
  risk <- c(minRisks, rev(maxRisks[-1])[-1], minRisks[1])
  return <- c(Returns, rev(Returns[-1])[-1], Returns[1])
  hull <- cbind(Risks = risk, Returns = return)
  # Return Value:
  hull
}
```

Now we only have to execute the function with the minimum CDaR locus as input in order to retrieve the whole hull:

```
> hull <- CDaRHull(mu, Scenarios, alpha, Return, Risk)
```

PART VI

DIVERSIFICATION

CHAPTER 13

PORTFOLIO DIVERSIFICATION

13.1 INTRODUCTION

In this chapter we present concepts and ideas for risk parity portfolios. This includes the following topics:

- Herfindhal diversification
- Entropy diversification
- Dependence diversification

The goal is to diversify the investment or the risk over all individual assets included in the portfolio as best as possible. Since the measure we have in mind is the variance of the investment and/or risk budgets we can express this by a quadratic form that describes the objective portfolio function. In section 2 we show that the best diversified weights portfolio is the *Equal Weights Diversified* (EWD) portfolio. In section 3 we discuss the covariance risk parity (CRP) budgeting problem. The *sufficient diversification portfolio* is introduced in section 4. *Tail Risk Parity* (TRP) budgeting will be considered in section 5.

Throughout this chapter we use as an example the Swiss pension fund benchmark portfolio. The data are part of the Rmetrics package `timeSeries` and are loaded together with the `fPortfolio` package. As the benchmark portfolio we use the equal weights portfolio which is characterized by the following settings.

```
> Scenarios <- 100*LPP2005REC[, 1:6]
> N <- ncol(Scenarios)
> S <- nrow(Scenarios)
> mu <- colMeans(Scenarios)
> Sigma <- cov(Scenarios)
```

The `targetReturn` for the equal weights portfolio is defined by the *grand mean* of the portfolio scenarios, and the `targetRisk` is defined by the *grand variance* of the portfolio. `mu` is the vector of the sample means of the assets, and `Sigma` the sample covariance matrix.

13.2 DIVERSIFICATION

For centuries, investment diversification has been a method to reduce the risk of significant losses. Most investment professionals agree that while diversification does not guarantee safety from a loss, it is the most important component to helping investors reach their long-range financial goals while minimizing their risk.
Under normal circumstances, it is pointless to optimize only the diversification since the maximum diversification portfolio is usually trivial. Rather than that, the diversification is often taken as an additional constraint to the portfolio optimization model. Two obvious choices of implementing diversification constraints are

- constraining the diversification of the maximum-return portfolio

  or

- constraining the diversification of a convex risk measure.

Since there exist different risk measures, many combinations between a risk measure and a diversification measure are possible. Fortunately, it is very simple to add diversification constraints to an optimization model in AMPL, since these constraints are independent of remaining problem. We will explain how to do this for the Herfindhal index, and will then proceed to give just the implemented diversification constraints.
Another approach is to include the diversification measure into the objective function. Such approaches are discussed in section ....

13.3 HERFINDAHL DIVERSIFIED PORTFOLIOS

In this modification of the variance portfolio we search for the best diversified model with respect to the weights. This is descibed by the Herfindahl Index, which is a meaure for the variance of the portfolio weights. We get a best Herfindahl diversified portfolio when the variance of the weights takes its minimum. This is the same as to minimize the quadratic form $w'1w$ instead of the sample covariance matrix.
The globally most diversified portfolio is obviously given by the equal weights portfolio with $w_i = 1/N$.
Taking the constraint on the sum of the portfolio weights into account, the Herfindhal index can take numbers between 1 **for a the worst diversification** (all investment in one asset), and $1/N$ **for the most diversified portfolio** (equal weights portfolio).
We will now demonstrate how the process of adding diversification constraints to an existing model works on the example of the markowitz portfolio to which we add constraints on the portfolio's Herfindhal index.

Markowitz portfolio with Herfindhal constraints

The Markowitz problem with Herfindhal constraints looks as following:

$$\min_w w\prime\Sigma w \tag{13.1}$$

$$\begin{aligned} s.t. \\ 1'w &= 1 \\ w_i &\geq 0 \\ w\prime w &\leq \overline{D} \end{aligned} \tag{13.2}$$

where $\overline{D}$ is the constraint we impose on the diversification. Since the constraint on the Herfindhal index is a positive-definite constraint, the problem remains convex.
The R/AMPL Model File for is just the Markowitz model with an additional constraint:

```
> modelMVHERF <- c(
   # Markowitz model
   "param N ;",
   "param Sigma{1..N,1..N} ;",
   "var w{1..N} >= 0, default 1/N ;",
   "minimize Objective: sum{i in 1..N} sum{j in 1..N} w[i] * Sigma[i,j] * w[j] ;",
   "subject to Budget: sum{i in 1..N} w[i] >= 1 ;",

   # Diversification constraint
   "param D ;",
```

```
   "subject to Diversification: sum{i in 1..N} w[i] * w[i] <= D ;")
> amplModelFile(model=modelMVHERF, project="myPortfolio")
```

We can also patch the original Markowitz model file and the diversification constraints together:

```
> modelMVHERF <- c(
   modelMVGLOB,
   "param D ;",
   "subject to Diversification: sum{i in 1..N} w[i] * w[i] <= D ;")
> amplModelFile(model=modelMVHERF, project="myPortfolio")
```

We see that we can introduce the diversification constraint on our portfolio independent of the chosen risk measure.
Since both the diversification and the risk are quadratic in w, as usual we use the CPLEX-solver:

```
> runMVHERF <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex;",
   "solve ;",
   "for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runMVHERF, project="myPortfolio")
```

Construct the R/AMPL Data File with the desired parameters. Remember that the Herfindhal index can take values between 1 and $1/N$. Thus an appropriate value for our diversification constraint to ensure a sufficient diversification is $D = 1/N \cdot 1.5$:

```
> Scenarios <- 100*LPP2005REC[, 1:6]
> requiredData(modelMVHERF)
[1] "N"     "Sigma" "D"

>   N <- ncol(Scenarios)
>   Sigma <- cov(Scenarios)
>   D <- 1/N *1.5
> dataMVHERF <- dataAUTO(modelMVHERF)
> amplDataFile(data=dataMVHERF, project="myPortfolio")
```

Optimize the Portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMVHERF <- as.numeric(scan("myPortfolio.txt"))
> names(weightsMVHERF) <- colnames(Scenarios)
```

Return-Diversification portfolio:

Instead of minimizing a risk measure while constraining the diversification, we can also maximize the expected return. This is exactly the same approach as when constraining the risk while maximizing the expected return:

$$\min_{w} w'\mu \tag{13.3}$$
$$s.t.$$
$$1'w = 1$$
$$w_i \geq 0$$
$$w\prime w \leq \overline{D}$$

The R/AMPL Model file looks as following:

```
> modelHERF2 <- c(
   "param N ;",
   "param mu{1..N} ;",
   "var w{1..N} >= 0, default 1/N ;",
   "maximize Objective: sum{i in 1..N} w[i] * mu[i] ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",

   # Diversification constraint ########
   "param D ;",
   "subject to Diversification: sum{i in 1..N} w[i] * w[i] <= D ;")
> amplModelFile(model=modelHERF2, project="myPortfolio")
```

Again, we just attached the diversification constraints to an existing model file.
R/AMPL Run File:

```
> runHERF2 <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runHERF2, project="myPortfolio")
```

Again, we choose the same value for the diversification constraint:

```
> requiredData(modelHERF2)
[1] "N"  "mu" "D"

>   Scenarios <- 100*LPP2005REC[,1:6]
>   N <- ncol(Scenarios)
>   Sigma <- cov(Scenarios)
>   mu <- colMeans(Scenarios)
>   D <- 1/N*1.5
> dataHERF2 <- dataAUTO(modelHERF2)
> amplDataFile(data=dataHERF2, project="myPortfolio")
```

Optimize the Portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsHERF2 <- as.numeric(scan("myPortfolio.txt"))
> names(weightsHERF2) <- colnames(Scenarios)
> weightsHERF2
      SBI       SPI       SII       LMI       MPI       ALT
0.022213 0.304614 0.102275 0.040644 0.220292 0.309961
```

Since the addition of diversification constraints is independent of the original portfolio model, naturally, this procedure was exactly the same as for the Markowitz problem.

Efficient Frontier calculation:

Since the constraints on our portfolio are now imposed on the diverification instead of the expected return, our AMPL run file for computing the efficient frontier now changes. We now have to loop over different target values for the diversification instead of the expected return.
Naturally, we again have to add the three usual parameters *minimum value*, *maximum value* and *number of steps*, to describe the loop to our model:

```
> modelMVHERFEF <- c(
   "param minD ;",
   "param maxD ;",
   "param nD ;",
   modelMVHERF)
> amplModelFile(model=modelMVHERFEF, project="myPortfolio")
```

In our AMPL run file, we now have to replace the expected return with the constraint on the diversification. For that, we first have to find and specify the maximum and the minimum value for the Herfindhal index. The minimum value is of course given by $1/N$. To find the maximum value, we set the constraint value very high such that the portfolio is not constrained anymore, and then solve the portfolio. We can then loop the diversification constraint between these two values:

```
> runMVHERFEF <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex;",
   "let D := 1;",
   "solve;",
   "let maxD := log(sum{i in 1..N} w[i]*w[i]) ;",
   "let minD := log(1/N) ;",
   "for {i in 0..nD} {",
   "  let D := exp(minD + i*(maxD-minD)/nD) ;",
   "  solve ;",
   "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
   "};",
   "exit ;")
> amplRunFile(run=runMVHERFEF, project="myPortfolio")
```

We have to specify the data file:

```
> requiredData(modelMVHERFEF)
[1] "minD"  "maxD"  "nD"    "N"     "Sigma" "D"

>   N <- ncol(Scenarios)
>   Sigma <- cov(Scenarios)
>   D <- NA
>   minD <- NA
>   maxD <- NA
>   nD <- 20
> dataMVHERFEF <- dataAUTO(modelMVHERFEF)
> amplDataFile(data=dataMVHERFEF, project="myPortfolio")
```

Optimize the portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMVHERFEF <- matrix(as.numeric(scan("myPortfolio.txt")), byrow=TRUE, ncol=N)
> colnames(weightsMVHERFEF) <- colnames(Scenarios)
> rownames(weightsMVHERFEF) <- paste0("MVHERFEF-", 0:nD)
```

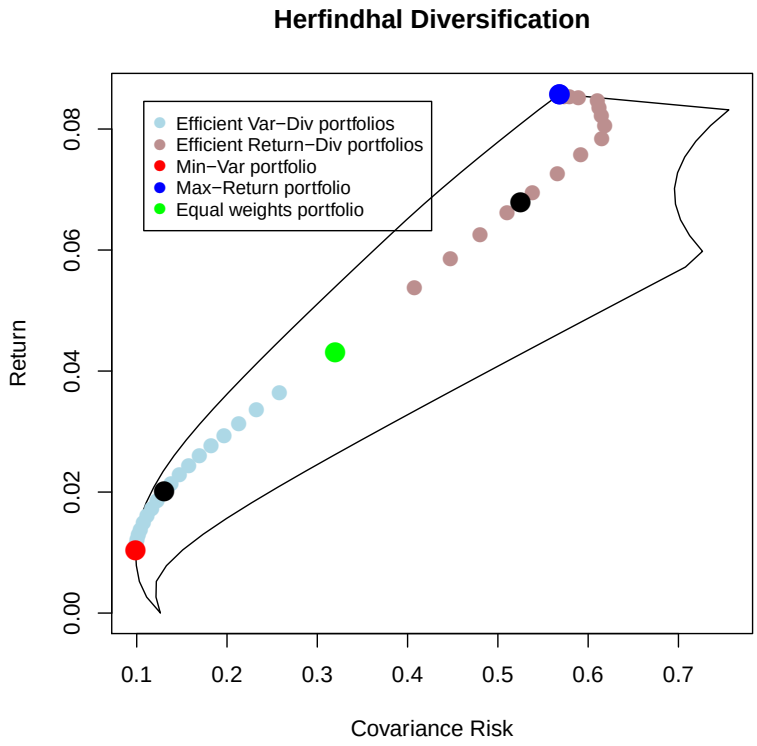


FIGURE 13.1: Herfindahl Diversification line of weights. The portfolios with our chosen target diversification of $1/N \cdot 1.5$ are depicted by the two black dots.

As we have seen, it is very easy to add diversification constraints to an existing portfolio model. In the following, we will only give the necessary AMPL code to implement these constraints for different diversification models, as well as the particular AMPL run file to compute the efficient frontier.

13.4 ENTROPY DE-CONCENTRATED PORTFOLIOS

Another useful diversification measure is the entropy of the weights. Originating from thermodynamics, in order to improve the diversification of our portfolio, we can maximize the weight's entropy, i.e.

$$\min_{w} \sum_{i}^{N} w_i \log(w_i) \tag{13.4}$$

$\lim_{w \to 0} w \log(w)$ converges to 0, but the function is not actually defined on 0 because of the logarithm term. In order to prevent any computational errors, we will add 10^{-14} to the weight inside the logarithm. This will have no noticable effect on the function, but ensures that the function is defined (and equal to zero) for $w = 0$.
The entropy values span from 0 **for the most concentrated portfolio** (one-asset portfolio) $log(1/N)$ **for the most diversified portfolio** (equal-weights portfolio).
The AMPL entropy constraints are simply given by the following two AMPL lines:

```
> ConstraintsEnt <- c(
   "param D ;",
   "subject to Diversification: sum{i in 1..N} w[i] * log(w[i]+10e-6) <= D ;")
```

Please keep in mind that due to the logarithm, this constraint is of non-linear nature and requires a non-linear solver such as minos.
To find the corresponding AMPL run file for the efficient frontier, we just replace the Herfindhal index in the specification of the `maxD`-parameter with the entropy, and adjust the other values to be in the right range:

```
> runEntEF <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver minos;",
   "let D := 0;",
   "solve;",
   "let maxD := sum{i in 1..N} w[i] * log( w[i]+10e-6) ;",
   "let minD := log(1/N) ;",
   "for {i in 0..nD} {",
   "  let D := minD + i*(maxD-minD)/nD ;",
   "  solve ;",
   "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
   "};",
   "exit ;")
```

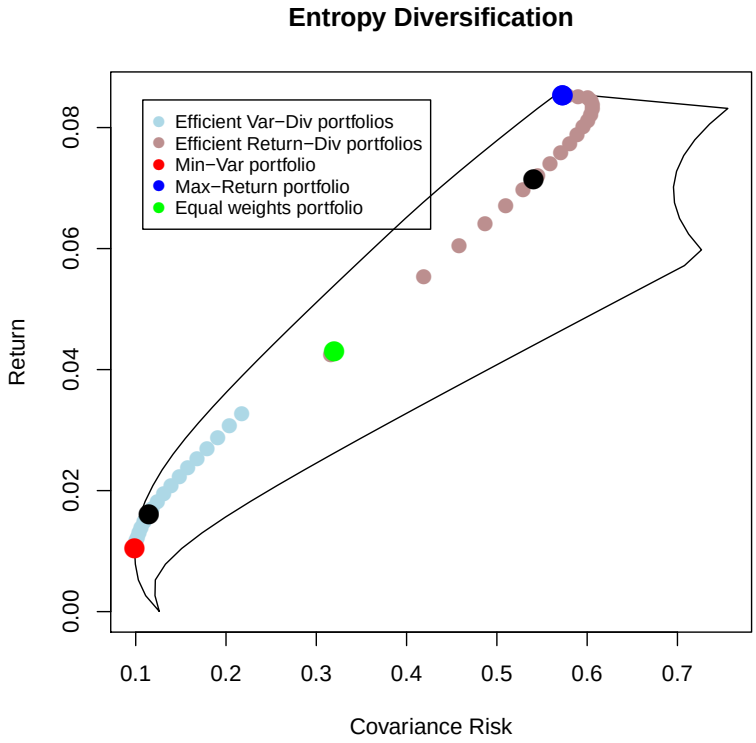


FIGURE 13.2: Entropy Diversification line of weights.

13.5 DEPENDENCE DIVERSIFIED PORTFOLIOS

The aim of the dependence diversification approach is to reduce the total dependence between the individual assets in the portfolio. Most concepts of dependence are usually measured by a matrix product, such as the Pearson correlation:

$$\rho = w' \mathrm{P} w \tag{13.5}$$

Another example of portfolio dependence is the pairwise tail dependence portfolio model. It was introduced by Würtz et al. [20xx]. A convenient way to compute the tail dependency of a portfolio is by using the function `gldDependencyFit` from the R-package `fPortfolio`:

```
> require(fCopulae)
> require(fPortfolio)
```

```
> Scenarios <- 100*LPP2005REC[,1:6]
> TailDepMatrix <- gldDependencyFit(Scenarios,
                                    doplot=FALSE,trace=FALSE)$lower+
                                    diag(1, nrow=6)
> TailDepMatrix
         SBI     SPI      SII      LMI      MPI     ALT
SBI 1.000000 0.00000 0.041913 0.215355 0.018325 0.00000
SPI 0.000000 1.00000 0.000000 0.000000 0.356763 0.36039
SII 0.041913 0.00000 1.000000 0.084666 0.000000 0.00000
LMI 0.215355 0.00000 0.084666 1.000000 0.015633 0.00000
MPI 0.018325 0.35676 0.000000 0.015633 1.000000 0.46247
ALT 0.000000 0.36039 0.000000 0.000000 0.462469 1.00000
```

The exact procedure to estimate the tail dependency would extend the scope of this handbook. For further information, see ....
A dependence matrix is generally positive semi-definite, therefore the portfolio dependence is **always bigger than** 0. The most diversified portfolio in terms of dependence is not trivial, but depends on the particular asset distributions. Therefore the dependence value for the most diversified portfolio has to be found via optimization. This can be done by just replacing the covariance matrix in the Markowitz approach with the particular dependence matrix.
The value for the most dependent portfolio is always given by 1. This value always realized for a one-asset portfolio.
The constraint is very simple:

```
> ConstraintsDep <- c(
   "param D ;",
   "param Rho{1..N,1..N} ;",
   "subject to Diversification:
           sum{i in 1..N} sum{j in 1..N} w[i] *Rho[i,j]*w[j]  <= D ;")
```

Please keep in mind that this is a quadratic constraint, and not all solvers are capable of solving problems with quadratic constraints.
Unlike before, we do not know the value of the most diversified portfolio for dependence diversification a priori. Therefore our AMPL Run script might terminate while calculating the efficient frontier, when we feed our model a too low constraint value such that the problem becomes infeasible. To prevent that, we can set the AMPL option *eexit* to a positive value. The loop now skips iteration steps that are infeasible. We also have to add the condition that the portfolio weights are only printed when the problem was feasible:

```
> runDepEF <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex;",
   "let D := 1;",
   "solve;",
   "option eexit 1 ;",
```

```
"let maxD := sum{i in 1..N} sum{j in 1..N} w[i] * Rho[i,j]* w[j] ;",
"let minD := 0 ;",
"for {i in 0..nD} {",
"  let D := minD + i*(maxD-minD)/nD ;",
"  solve ;",
"  if solve_result_num == 0 then ",
"  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
"};",
"exit ;")
```

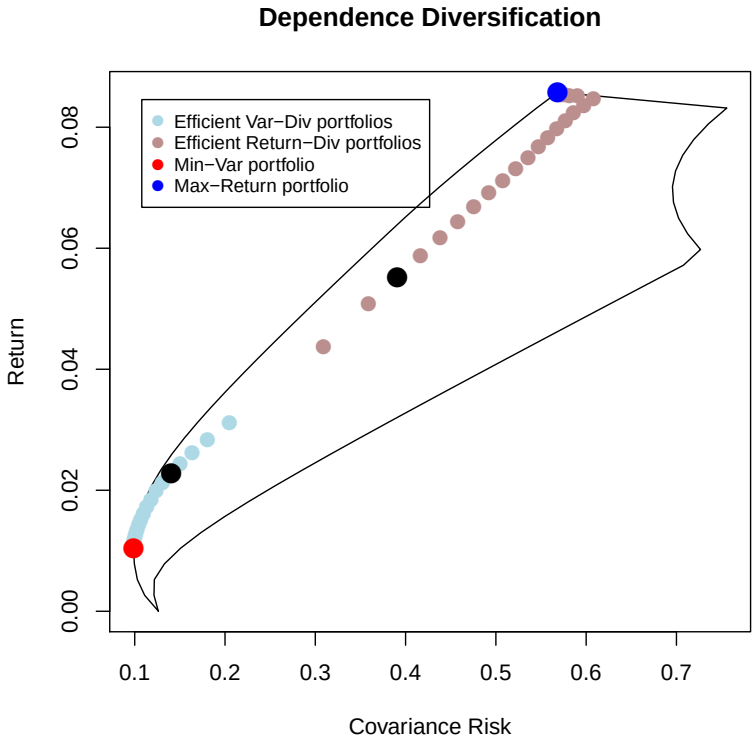


FIGURE 13.3: Tail dependence Diversification line of weights.

CHAPTER 14

COVARIANCE RISK PARITY

14.1 INTRODUCTION

Covariance risk parity is the keyword when it comes to the managing of risk exposures using the *risk budgeting approach*. For an introduction we refer the article of Benjamin Bruder and Thierry Roncalli [2012].
Instead of diversifying the portfolio weights, in this approach we try to diversify the contributions of individual assets to the total covariance risk. The covariance risk contributions are defined as following:

$$b_i := \frac{w_i(\Sigma w)_i}{w' \Sigma w}$$

Throughout this chapter we use as an example the Swiss pension fund benchmark portfolio. The data are part of the Rmetrics package `timeSeries` and are loaded together with the `fPortfolio` package. As the benchmark portfolio we use the equal weights portfolio which is characterized by the following settings.

```
> Scenarios <- 100*LPP2005REC[, 1:6]
> N <- ncol(Scenarios)
> S <- nrow(Scenarios)
> mu <- colMeans(Scenarios)
> Sigma <- cov(Scenarios)
```

The `targetReturn` for the equal weights portfolio is defined by the *grand mean* of the portfolio scenarios, and the `targetRisk` is defined by the *grand variance* of the portfolio. `mu` is the vector of the sample means of the assets, and `Sigma` the sample covariance matrix.

14.2 RISK PARITY PORTFOLIO

The most diversified portfolio is the *Equal-Risk-Contribution* portfolio (ERC) with $b_i = 1/N$. If we use the Herfindhal-index to measure the diversification of the Risk Budgets, we can write our model as

$$\begin{aligned} \min_{w} & \sum_{i=1}^{N} (w_i(\Sigma w)_i - \frac{w' \Sigma w}{N})^2 \\ s.t. & \\ & \mu' w \geq \overline{r} \\ & w_i \geq 0 \end{aligned}$$

A computationally more simple method to solve this problem is to add another variable to the expression to replace the average risk contribution:

$$\begin{aligned} \min_{w,\theta} & \sum_{i=1}^{N} (w_i(\Sigma w)_i - \theta)^2 \\ s.t. & \\ & \mu' w \geq \overline{r} \\ & w_i \geq 0 \end{aligned}$$

These two problems are equivalent and yield the same results. Due to the less complex formulation, the latter one is significantly faster to solve. We can now define the AMPL model file:

```
> modelPARITY <- c(
   "param N ;",
   "param Sigma{1..N, 1..N} ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var Theta ;",
   "minimize Objective: sum{k in 1..N} ( w[k] * sum{j in 1..N} Sigma[k,j] * w[j] - Theta )^2 ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;")
> amplModelFile(model=modelPARITY, project="myPortfolio")
```

Please note that the model is non-linear and requires a solver capable of solving non-linear problems such as *MINOS*:

```
> runPARITY <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver minos;",
   "solve ;",
   "for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runPARITY, project="myPortfolio")
```

Construct the R/AMPL Data File with the desired parameters.

```
> Scenarios <- 100*LPP2005REC[, 1:6]
> requiredData(modelPARITY)
[1] "N"     "Sigma"
>   N <- ncol(Scenarios)
>   Sigma <- cov(Scenarios)
> dataPARITY <- dataAUTO(modelPARITY)
> amplDataFile(data=dataPARITY, project="myPortfolio")
```

Optimize the Portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsPARITY <- as.numeric(scan("myPortfolio.txt"))
> names(weightsPARITY) <- colnames(Scenarios)
> weightsPARITY
     SBI      SPI      SII      LMI      MPI      ALT
0.332104 0.042509 0.144077 0.376393 0.045027 0.059890
```

Finally, we summarize our results:

```
> mu <- colMeans(Scenarios)
> SummaryPARITY <- c(
   TargetReturn = mu %*% weightsPARITY,
   CovarianceRisk = sqrt ( weightsPARITY %*% Sigma %*% weightsPARITY ),
   HerfindahlIndex = 1 - weightsPARITY %*% weightsPARITY)
> SummaryPARITY
   TargetReturn  CovarianceRisk HerfindahlIndex
       0.016912        0.119612        0.719856
```

14.3 EFFICIENT RISK-PARITY PORTFOLIO

To compute the efficient risk-parity portfolio, we just add the constraint on the expected return:

$$\begin{aligned} \min_{w,\theta} & \sum_{i=1}^{N} (w_i(\Sigma w)_i - \theta)^2 \\ s.t. & \\ & 1'w = 1 \\ & \mu' w \geq \overline{r} \\ & w_i \geq 0 \end{aligned}$$

We have to add the μ-vector as well as the the target return as parameters to our AMPL model before adding the constraint on the expected return:

```
> modelPARITY1 <- c(
   "param N ;",
   "param Sigma{1..N, 1..N} ;",
   "var w{1..N} >= 0, default 1/N ;",
   "var Theta ;",
   "minimize Objective: sum{k in 1..N} ( w[k] * sum{j in 1..N} Sigma[k,j] * w[j] - Theta )^2 ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",

   "param mu{1..N} ;",
   "param targetReturn ;",
   "subject to Reward: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;")
> amplModelFile(model=modelPARITY1, project="myPortfolio")


> runPARITY1 <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver minos;",
   "solve ;",
   "for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runPARITY1, project="myPortfolio")
```

Construct the R/AMPL Data File with the desired parameters. As usual, we optimize for the grand-mean as the target return:

```
> Scenarios <- 100*LPP2005REC[, 1:6]
> requiredData(modelPARITY1)
[1] "N"            "Sigma"        "mu"           "targetReturn"
>   N <- ncol(Scenarios)
>   Sigma <- cov(Scenarios)
>   mu <- colMeans(Scenarios)
>   targetReturn <- mean(mu)
> dataPARITY1 <- dataAUTO(modelPARITY1)
> amplDataFile(data=dataPARITY1, project="myPortfolio")
```

Optimize the Portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsPARITY1 <- as.numeric(scan("myPortfolio.txt"))
> names(weightsPARITY1) <- colnames(Scenarios)
> weightsPARITY1
      SBI       SPI       SII       LMI       MPI       ALT
0.000000 0.119946 0.436322 0.162807 0.091326 0.189599
```

Finally, we summarize our results:

```
> mu <- colMeans(Scenarios)
> SummaryPARITY1 <- c(
    TargetReturn = mu %*% weightsPARITY1,
    CovarianceRisk = sqrt ( weightsPARITY1 %*% Sigma %*% weightsPARITY1 ),
    HerfindahlIndex = 1 - weightsPARITY1 %*% weightsPARITY1)
> SummaryPARITY1
    TargetReturn  CovarianceRisk HerfindahlIndex
        0.043077        0.286121        0.724442
```

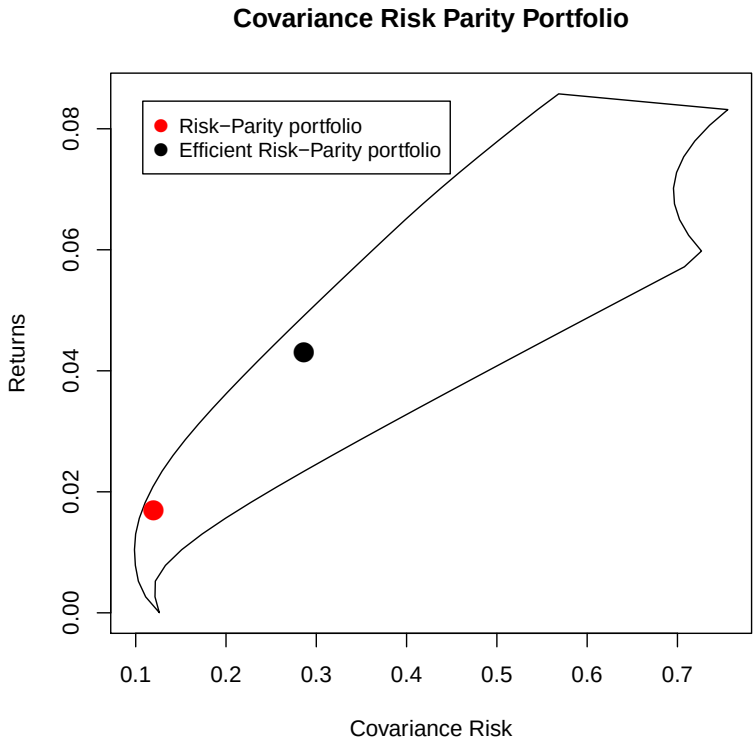


FIGURE 14.1: Feasible set and the position of covariance risk-parity portfolio as well as the efficient covariance risk-parity portfolio.

14.4 EFFICIENT FRONTIER

The computation of the mean-risk-parity efficient frontier is very similar to the Markowitz efficient frontier computation. First, we add the three necessary looping parameters to our model file:

```
> modelPARITYEF <- c(
   "param minReturn ;",
   "param maxReturn ;",
   "param nReturn ;",
   modelPARITY1)
> amplModelFile(model=modelPARITYEF, project="myPortfolio")
```

We can use the same run-file as for the Markowitz efficient frontier, although we have to replace the specified solver with a non-linear solver:

```
> runPARITYEF <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver minos;",
   "let targetReturn := -999;",
   "solve;",
   "let minReturn := sum{i in 1..N} w[i]*mu[i];",
   "let maxReturn := max{i in 1..N} mu[i];",
   "for {i in 0..nReturn} {",
   "  let targetReturn := minReturn + i*(maxReturn-minReturn)/nReturn ;",                    "  solve ;",
   "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
   "};",
   "exit ;")
> amplRunFile(run=runPARITYEF, project="myPortfolio")
```

We have to specify the exact same parameters as for the Markowitz efficient frontier:

```
> requiredData(modelPARITYEF)
[1] "minReturn"    "maxReturn"    "nReturn"      "N"            "Sigma"
[6] "mu"           "targetReturn"
>   N <- ncol(Scenarios)
>   Sigma <- cov(Scenarios)
>   mu <- colMeans(Scenarios)
>   targetReturn <- NA
>   minReturn <- NA
>   maxReturn <- NA
>   nReturn <- 30
> dataPARITYEF <- dataAUTO(modelPARITYEF)
> amplDataFile(data=dataPARITYEF, project="myPortfolio")
```

Optimize the portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsPARITYEF <- matrix(as.numeric(scan("myPortfolio.txt")), byrow=TRUE, ncol=N)
> colnames(weightsPARITYEF) <- colnames(Scenarios)
> rownames(weightsPARITYEF) <- paste0("PARITYEF-", 0:nReturn)
```

Plot the Risk/Reward Diagram:

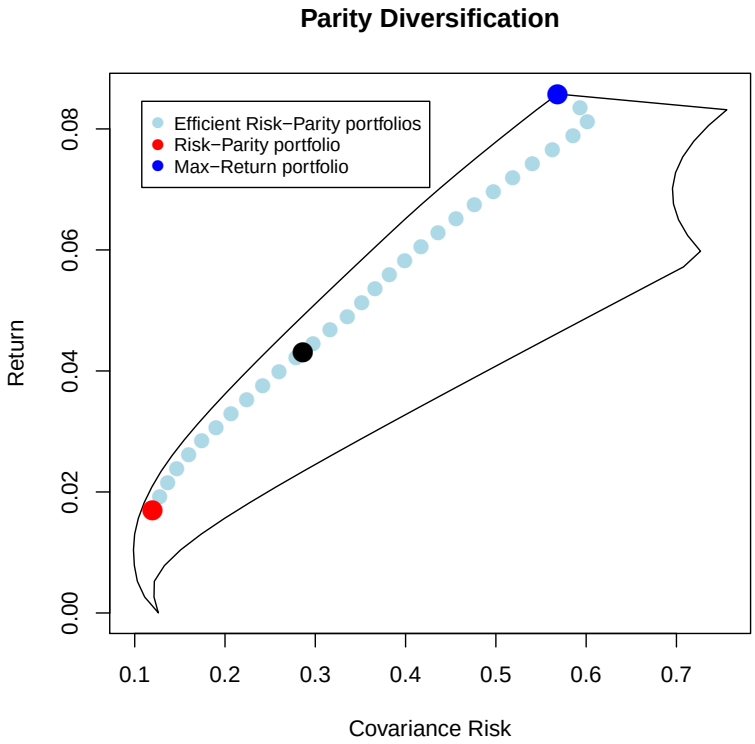


FIGURE 14.2: Equi-distant return frontier for the Covariance Risk-Parity portfolio

PART VII

MULTIOBJECTIVE PROGRAMMING

CHAPTER 15

MULIT-OBJECTIVE VARIANCE PORTFOLIOS

15.1 INTRODUCTION

In this chapter ...
We will show the implementation and the scaling of multi-objective portfolio models.

15.2 CRITICAL LINE ALGORITHM

A different approach to portfolio optimization, named the critical line algorithm, takes in the objective function both return and risk into account. For the Mean-Variance portfolio, an implementation looks as following:

$$\begin{aligned} &\max_{w}(1-\lambda)\mu' w - \lambda w'\Sigma w \\ &s.t. \\ &\qquad 1' w = 1 \\ &\qquad w_i \geq 0 \end{aligned}$$

Here λ is a risk aversion parameter and is allowed to take values between zero and one. If $\lambda = 1$ then we minimize the risk, if $\lambda = 0$ then we optimize the return.

The AMPL model file is very easy to implement:

```
> modelMV3 <- c(
  "param N ;",
  "param lambda ;",
  "param mu{1..N} ;",
  "param L ;",
  "param Sigma{1..N, 1..N} ;",
```

```
    "var w{1..N} >= 0, default 1/N;",
    "maximize Objective: (1-lambda) * sum{i in 1..N} mu[i] * w[i] - ",
    "  lambda * sum{i in 1..N} sum{j in 1..N} w[i] * Sigma[i,j] * w[j] ;",
    "subject to Budget: sum{i in 1..N} w[i] = 1 ;")
> amplModelFile(model=modelMV3, project="myPortfolio")
```

What is an appropriate value for λ? Note that if $\lambda = 1$ the critical line algorithm searches for the global minimum risk portfolio, and on the opposite hand, when $\lambda = 0$ the solution for the portfolio is the asset with the highest individual return.

When we want to calculate the efficient frontier, we have to modify our AMPL run file, since now we are looping over λ and not $\overline{r}$ anymore:

```
> runMV3 <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex;",
   "for {i in 0..L} {",
   "  let lambda := i/L ;",
   "  solve ;",
   "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
   "};",
   "exit ;")
> amplRunFile(run=runMV3, project="myPortfolio")
```

The run file optimizes a fixed number of portfolios (here 34) along the critical line. The loop is part of the run file, this makes the optimization most time efficient.
Finally, do not forget to declare λ as an additional parameter in the AMPL data file. Set the value to `NA` since the parameter is defined inside the run file.

```
> Scenarios <- 100*LPP2005REC[,1:6]
> requiredData(modelMV3)
[1] "N"      "lambda" "mu"     "L"      "Sigma"

>   N <- ncol(Scenarios)
>   mu <- colMeans(Scenarios)
>   Sigma <- cov(Scenarios)
>   lambda <- NA
>   L <- 33
> dataMV3 <- dataAUTO(modelMV3)
> amplDataFile(data=dataMV3, project="myPortfolio")
```

Optimize the Portfolio:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMV3 <- matrix(as.numeric(scan("myPortfolio.txt")), byrow=TRUE, ncol=N)
> colnames(weightsMV3) <- colnames(Scenarios)
```

Plot the result in a risk/return diagram:

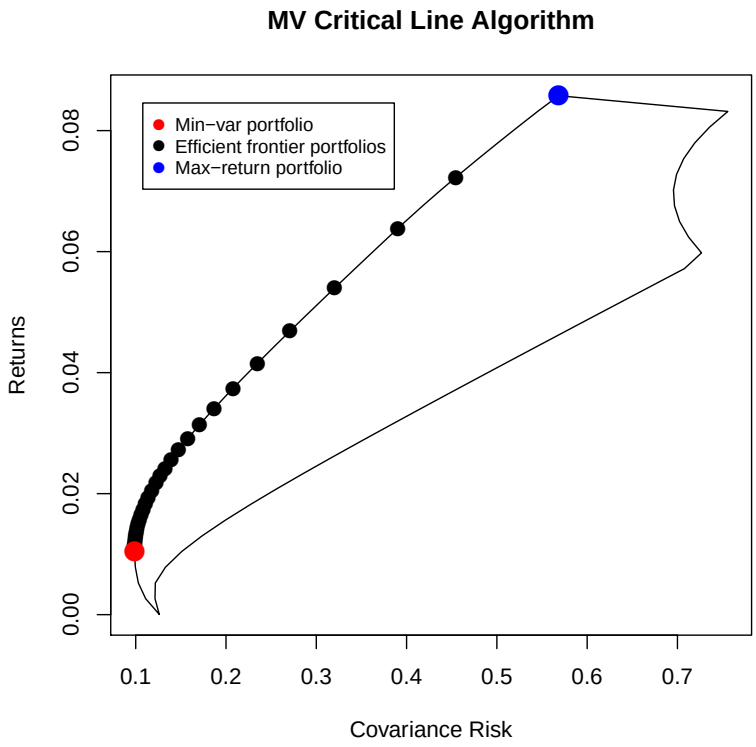


FIGURE 15.1: Critical line algorithm vean-variance portfolio

Only 3 points are well separated, the remaining 31 are agglomerated very close together. The reason for this is that the two objectives for the return and risk are not well scaled.

Let us consider a better scaling by introducing the scaling parameters `Norm1` and `Norm2`. These will normalize the maximal change in the components (the expected return and the covariance risk) of the objective function to be equal to one:

```
> modelMV3SCALED <- c(
  "param N ;",
  "param lambda ;",
  "param L ;",
  "param mu{1..N} ;",
  "param Sigma{1..N, 1..N} ;",
  "param Norm1 ;",
  "param Norm2 ;",
```

```
  "var w{1..N} >= 0, default 1/N;",
  "maximize Objective: (1-lambda) * ( sum{i in 1..N} mu[i] * w[i] )/( Norm1) ",
  " -lambda*( sum{i in 1..N} sum{j in 1..N} w[i] * Sigma[i,j] * w[j] )/( Norm2 ) ;",
  "subject to Budget: sum{i in 1..N} w[i] = 1 ;")
> amplModelFile(model=modelMV3SCALED, project="myPortfolio")
```

We presolve the portfolio for $\lambda = 1$ in order to measure the maximal change of the risk and the expected return. We only have to do this for $\lambda = 1$ in order to find the minimum variance portfolio since the maximum return portfolio is trivial. We can then specify our scaling parameters:

```
> runMV3SCALED <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex;",
   "set I ordered := {s in 1..N: mu[s] == max{i in 1..N} mu[i]} ;",
   "let Norm1 :=   mu[first(I)] ;",
   "let Norm2 :=  Sigma[first(I),first(I)] ;",
   "let lambda :=1 ;",
   "solve ;",
   "let Norm1 := Norm1 - sum{i in 1..N} mu[i] * w[i] ;",
   "let Norm2 := Norm2 - sum{i in 1..N} sum{j in 1..N} w[i]*Sigma[i,j]*w[j] ;",
   "for {i in 0..L} {",
   "  let lambda := i/L ;",
   "  solve ;",
   "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
   "};",
   "exit ;")
> amplRunFile(run=runMV3SCALED, project="myPortfolio")
```

We have to add the scaling parameters to the AMPL data file:

```
> Scenarios <- 100*LPP2005REC[,1:6]
> requiredData(modelMV3SCALED)
[1] "N"      "lambda" "L"      "mu"     "Sigma"  "Norm1"  "Norm2"

>   N <- ncol(Scenarios)
>   mu <- colMeans(Scenarios)
>   Sigma <- cov(Scenarios)
>   lambda <- NA
>   L <- 33
>   Norm1 <- NA
>   Norm2 <- NA
> dataMV3SCALED <- dataAUTO(modelMV3SCALED)
> amplDataFile(data=dataMV3SCALED, project="myPortfolio")
```

Optimize the portfolio:s

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMV3SCALED <- matrix(as.numeric(scan("myPortfolio.txt")), byrow=TRUE, ncol=6)
> colnames(weightsMV3SCALED) <- colnames(Scenarios)
```

Plot the Results:

The portfolios are now buch better distributed along the efficient frontier.

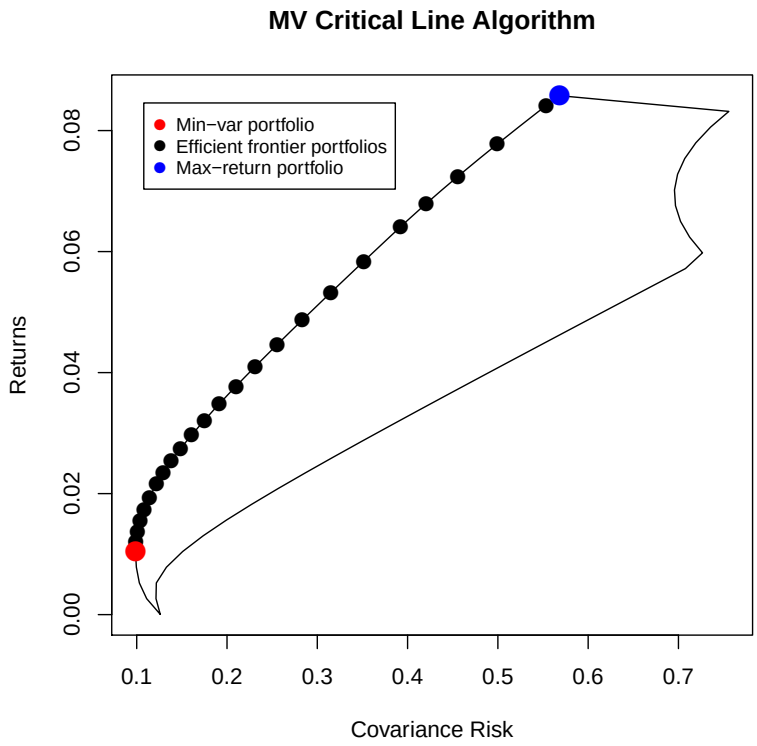


FIGURE 15.2: Critical line algorithm vean-variance portfolio

15.3 MEAN - COVARIANCE - ENTROPY DIVERSIFICATION

Obviously, we can add more than two portfolio measures into the objective function instead of constraining them in constraints. As an example for a multi-objective portfolio design, we look at the simultaneuous optimization of the expected return, the portfolio variance and the portfolio diversification, characterized by the entropy diversification measure. Our optimal portfolios are then spanned between the minimum variance portfolio, the maximum return portfolio and the equal weights portfolio.

We have to introduce an additional parameter λ_2 in order to quantify our emphasis on the diversification. It is natural to then impose the condition that $\lambda_1 + \lambda_2 \leq 1$. The mathematical formulation then looks as following:

$$\max_w (1-\lambda_1-\lambda_2)\mu' w - \lambda_1 w'\Sigma w - \lambda_2 \sum_i^N w_i \log(w_i)$$
$$s.t.$$
$$1'w = 1$$
$$w_i \geq 0$$

In the implementation, we will add another scaling parameter `Norm3` in order to scale the entropy diversification:

```
> modelMV4 <- c(
  "param N ;",
  "param lambda1 ;",
  "param lambda2 ;",
  "param L ;",
  "param mu{1..N} ;",
  "param Sigma{1..N, 1..N} ;",
  "param Norm1 ;",
  "param Norm2 ;",
  "param Norm3 ;",
  "var w{1..N} >= 0, default 1/N;",
  "maximize Objective: (1-lambda1-lambda2) * ( sum{i in 1..N} mu[i] * w[i] ) / Norm1",
  "-lambda1 * ( sum{i in 1..N} sum{j in 1..N} w[i] * Sigma[i,j] * w[j] ) / Norm2 ",
  "-lambda2* (sum{i in 1..N} w[i] * log(w[i]+10e-6)) / Norm3 ;",
  "subject to Budget: sum{i in 1..N} w[i] = 1 ;")
> amplModelFile(model=modelMV4, project="myPortfolio")
```

We use the same approach in the AMPL run file to set our scaling parameters. Since the equal weights portfolio is trivial, we again only have to presolve the portfolio for $\lambda_1 = 1$ in order to find the minimum variance portfolio. For more complicated diversification measures, we might have to presolve again for $\lambda_2 = 1$ in order fully estimate the maximal change of the risk, the expected return and the diversification.
We can then easily find the maximum change of the risk, the expected return and the diversification, and set our scaling parameters:

```
> runMV4 <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver minos;",
   "set I ordered := {s in 1..N: mu[s] == max{i in 1..N} mu[i]} ;",
   "let Norm1 :=   mu[first(I)] ;",
   "let Norm2 :=  Sigma[first(I),first(I)] ;",
   "let Norm3 := -log(1/N) ;",
   "let lambda1 :=1 ;",
   "let lambda2 :=0 ;",
   "solve ;",
   "let Norm1 := Norm1 - min(sum{i in 1..N} mu[i] * w[i], sum{i in 1..N} mu[i] /N) ;",
```

```
    "let Norm2 := Norm2
    - max(sum{i in 1..N} sum{j in 1..N} w[i]*Sigma[i,j]*w[j],
    1/N^2 sum{i in 1..N} sum{j in 1..N} Sigma[i,j]);",
    "for {i in 0..L} {",
      "for {j in 0..(L-i)}{",
        "  let lambda1 := i/L ;",
        "  let lambda2 := j/L ;",
        "  solve ;",
        "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
    "}};",
    "exit ;")
> amplRunFile(run=runMV4, project="myPortfolio")
```

Add the two additional parameters λ_2 and `Norm3` to the AMPL Data File:

```
> Scenarios <- 100*LPP2005REC[,1:6]
> requiredData(modelMV4)
[1] "N"       "lambda1" "lambda2" "L"       "mu"      "Sigma"   "Norm1"
[8] "Norm2"   "Norm3"
>   N <- ncol(Scenarios)
>   mu <- colMeans(Scenarios)
>   Sigma <- cov(Scenarios)
>   lambda1 <- NA
>   lambda2 <- NA
>   L <- 20
>   Norm1 <- NA
>   Norm2 <- NA
>   Norm3 <- NA
>   dataMV4 <- dataAUTO(modelMV4)
> amplDataFile(data=dataMV4, project="myPortfolio")
```

Optimize the Portfolio:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMV4 <- matrix(as.numeric(scan("myPortfolio.txt")), byrow=TRUE, ncol=N)
> colnames(weightsMV4) <- colnames(Scenarios)
```

Plot the Results:

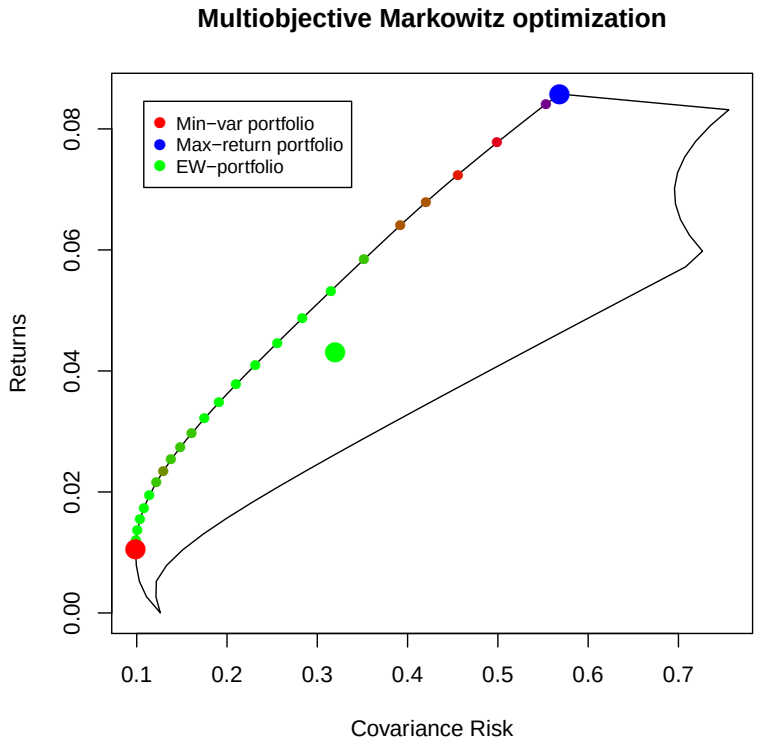


FIGURE 15.3: Critical Line Algorithm Mean-Variance Benchmark Portfolio

PART VIII

CONSTRAINTS

CHAPTER 16

CONSTRAINED PORTFOLIOS

16.1 INTRODUCTION

In this chapter we optimize the efficient mean-variance Markowitz portfolio with box and/or group constraints. This includes

- short selling constraints,
- box constraints ,
- group constraints
- turnover constraints
- tracking error constraints

Section 2 shows how to add box constraints. In section 3 we discuss how to add group constraints. And in section 4 we combine both, box and group constraints. Note, although all examples use the efficient mean-variance portfolio, the principle of adding box and/or group constraints holds for other types of portfolios: the global minimum risk portfolio, portfolios optimized by the critical line algorithms, robust portfolios, and many others.

Throughout this chapter we use as an example the Swiss pension fund benchmark portfolio. The data are part of the Rmetrics package `timeSeries` and are loaded together with the `fPortfolio` package. As the benchmark portfolio we use the equal weights portfolio which is characterized by the following settings.

16.2 SHORT SELLING PORTFOLIOS

Let us consider the unconstrained short selling portfolio. We have no longer any constraints on the individual weights. In other words, *short selling* will be allowed regardless of a desired target risk or target return. The budget $1'w = 1$ is fully invested, and this is the only constraint. We call this portfolio problem the unrestricted short selling problem.

If we consider the unconstrained short selling mean-variance Markowitz portfolio, the problem reads:

$$\begin{aligned} & \min_{w} \quad w'\Sigma w \\ s.t. & \\ & 1'w = 1 \\ & \mu\prime w = \bar{r} \end{aligned}$$

The only difference to the long-only portfolio model is the lack of the $w_i \geq 0$ constraint.
Let us look again at the MV1 implementation:

```
> modelMV1 <- c(
  "param N ;",
  "param mu{1..N} ;",
  "param Sigma{1..N,1..N} ;",
  "param targetReturn ;",
  "var w{1..N} >= 0, default 1/N ;",
  "minimize Objective: sum{i in 1..N} sum{j in 1..N} w[i] * Sigma[i,j] * w[j] ;",
  "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
  "subject to Return: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;")
```

In the R/AMPL model file for the unrestricted short selling Markowitz problem, we only have to change the following line:

```
>  "var w{1..N}>=0, default 1/N ;"
```

The R/AMPL model file becomes:

```
> modelMV1SS <- c(
  "param N ;",
  "param mu{1..N} ;",
  "param Sigma{1..N,1..N} ;",
  "param targetReturn ;",
  "var w{1..N}, default 1/N ;",
  "minimize Objective: sum{i in 1..N} sum{j in 1..N} w[i] * Sigma[i,j] * w[j] ;",
  "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
  "subject to Return: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;")
> amplModelFile(model=modelMV1SS, project="myPortfolio")
```

Both the R/AMPL Run File as well as the Data File stay the same as for the long-only portfolio problem `MV1`:
R/AMPL Run File:

```
> runMV1SS <-  c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(runMV1SS, "myPortfolio")
```

R/AMPL Data File:

```
> Scenarios <- 100*LPP2005REC[,1:6]
> requiredData(modelMV1SS)
[1] "N"            "mu"           "Sigma"        "targetReturn"
>   N <- ncol(Scenarios)
>   Sigma <- cov(Scenarios)
>   mu <- colMeans(Scenarios)
>   targetReturn <- mean(mu)
> dataMV1SS <- dataAUTO(modelMV1SS)
> amplDataFile(data=dataMV1SS, project="myPortfolio")
```

Optimize it and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMV1SS <- as.numeric(scan("myPortfolio.txt"))
> names(weightsMV1SS) <- colnames(Scenarios)
> weightsMV1SS
       SBI        SPI        SII        LMI        MPI        ALT
-0.225673   0.073365   0.218793   0.706155 -0.309798   0.537157
```

The implementation of short-selling portfolios is not restricted to just the Markowitz portfolio. For every risk measure, we can remove the $w_i \geq 0$-condition in the R/AMPL model file to allow short-selling.

Efficient Frontier

It is very straightforward to implement the EDR-approach for the short-selling portfolio model. Again we again just add the additional parameters to the unresticted R/AMPL model file:

```
> modelMVEDRSS <- c(
   "param minReturn ;",
   "param maxReturn ;",
   "param nReturn ;",
   modelMV1SS)
> amplModelFile(model=modelMVEDRSS, project="myPortfolio")
```

The R/AMPL run file is as usual the same as for the long-only approach. If we would like to compute the efficient frontier for non-linear risk measure,

we might have to replace the solver specification with the appropriate solver.

```
> runMVEDRSS <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver minos;",
   "for {i in 0..nReturn} {",
   "  let targetReturn := minReturn + i*(maxReturn-minReturn)/nReturn ;",
   "  solve ;",
   "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
   "};",
   "exit ;")
> amplRunFile(run=runMVEDRSS, project="myPortfolio")
```

Again we use the already calculated return of the Global Minimum Variance portfolio as the minimal target return.
A big difference to the long-only approach is the fact that the short-selling model can achieve higher return expections than the long-only approach. In fact, there is no limit on the expected return. Therefore we just pick a value in order to specify the maximum target return in the data file. The R/AMPL Data File then reads:

```
> requiredData(modelMVEDRSS)
[1] "minReturn"    "maxReturn"    "nReturn"      "N"            "mu"
[6] "Sigma"        "targetReturn"

>   N <- ncol(Scenarios)
>   Sigma <- cov(Scenarios)
>   mu <- colMeans(Scenarios)
>   targetReturn <- NA
>   minReturn <- (mu %*% weightsMVGLOB)[1]
>   maxReturn <- 1.25*max(mu)
>   nReturn <- 33
> dataMVEDRSS <- dataAUTO(modelMVEDRSS)
> amplDataFile(data=dataMVEDRSS, project="myPortfolio")
```

In figure 16.1 we can see the efficient frontier for the short selling model:

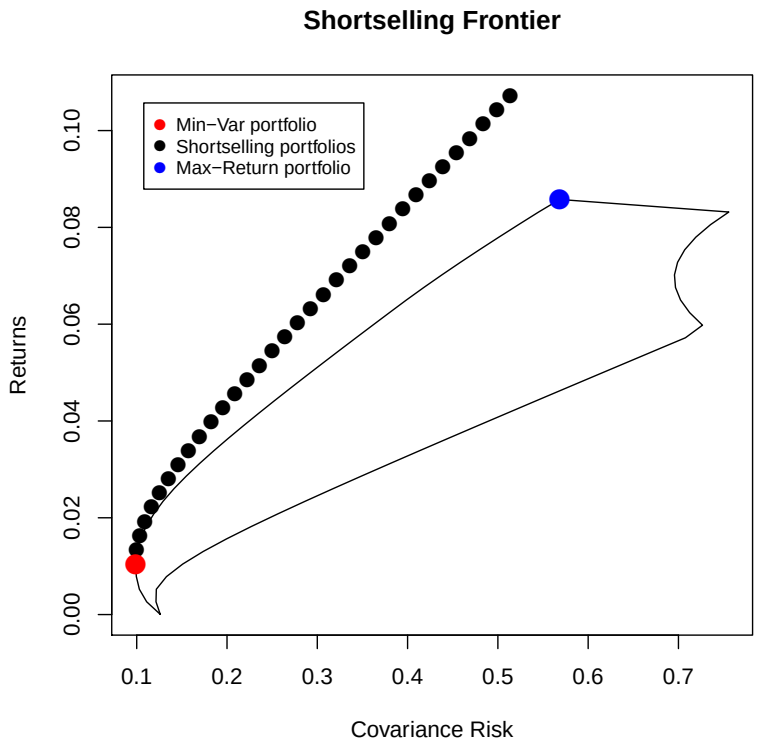


FIGURE 16.1: Short-selling Efficient Markowitz Frontier

16.3 BOX CONSTRAINED MARKOWITZ PORTFOLIO

Box constraints are used to confine the values the individual asset weights can take. For instance, if an investor does not want to invest more than 30% in one asset, and wants to invest at least 10 % in another asset, he can impose box constrains on these assets.

Box constraints consist of an upper bound vector and a lower bound vector. Each entry of these vectors specifies the upper/lower bound of the corresponding asset. For example, the Markowitz portfolio model with box constraints can by written as following:

$$\begin{aligned} \min_{w} \quad & w'\Sigma w \\ s.t. \quad & \\ & 1'w = 1 \\ & l_i \leq w_i \leq u_i \\ & \mu' w = \bar{r} \end{aligned} \tag{16.1}$$

l_i and u_i are the vectors of lower and upper bound on the weights.
To construct our model file, we only have to add the additional constraints to the existing long-only Markowitz portfolio model that we specified above:

```
> modelMV1BOX <- c(
   #Original model
   "param N ;",
   "param mu{1..N} ;",
   "param Sigma{1..N,1..N} ;",
   "param targetReturn ;",
   "var w{1..N} >= 0, default 1/N ;",
   "minimize Objective: sum{i in 1..N} sum{j in 1..N} w[i] * Sigma[i,j] * w[j] ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",
   "subject to Return: sum{i in 1..N} mu[i] * w[i] >= targetReturn ;",

   # Box Constraints
   "param lower{1..N} ;",
   "param upper{1..N} ;",
   "subject to Lower {i in 1..N}:  w[i] >= lower[i] ;",
   "subject to Upper {i in 1..N}:  w[i] <= upper[i] ;")
> amplModelFile(model=modelMV1BOX, project="myPortfolio")
```

or in short:

```
> modelMV1BOX <- c(
   modelMV1,
   "param lower{1..N} ;",
   "param upper{1..N} ;",
   "subject to Lower {i in 1..N}:  w[i] >= lower[i] ;",
   "subject to Upper {i in 1..N}:  w[i] <= upper[i] ;")
> amplModelFile(model=modelMV1BOX, project="myPortfolio")
```

As for the diversification constraints, the box constraints are independent of the chosen risk measure or the existing portfolio model. We can just attach the constraints to the existing portfolio model.
The R/AMPL run file is the the same as for the MV1 model without box constraints:

```
> runMV1BOX <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
```

```
    "option solver cplex ;",
    "solve ;",
    "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
    "exit ;")
> amplRunFile(runMV1BOX, "myPortfolio")
```

Let us take as an example the constraints that will take on the following values: Swiss Equities SPI max 40%, Foreign Equities MPI and Swiss Real Estate SII max 30%, Alternative Instruments ALT max 10%, and Swiss SBI at least 20%. Foreign Bonds LMI are unconstrained. These bounds define the `lower` and `upper` limits on the weights.

```
> Scenarios <- 100*LPP2005.RET[, 1:6]
> requiredData(modelMV1BOX)
[1] "N"            "mu"           "Sigma"        "targetReturn" "lower"
[6] "upper"
>   N <- ncol(Scenarios)
>   mu <- colMeans(Scenarios)
>   Sigma <- cov(Scenarios)
>   targetReturn <- mean(mu)
>   # Box Constraints:
>   lower <- c(SBI=0.2, SPI=0.01, SII=0.1, LMI=0.1, MPI=0.1, ALT=0.1)
>   upper <- c(SBI=1.0, SPI=0.4, SII=0.3, LMI=1.0, MPI=0.3, ALT=0.1)
> dataMV1BOX <- dataAUTO(modelMV1BOX)
> amplDataFile(data=dataMV1BOX, project="myPortfolio")
```

The solution for the box-constrained Markowitz portfolio is:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMV1BOX <- as.numeric(scan("myPortfolio.txt"))
> names(weightsMV1BOX) <- colnames(Scenarios)
> weightsMV1BOX
    SBI     SPI     SII     LMI     MPI     ALT
0.20000 0.26686 0.23314 0.10000 0.10000 0.10000
```

Note:

- If you like to allow for short-selling, then you just define negative box constraints.
- It is important to note that the specification of box constraints is completely independent of the portfolio model. We can add the constraints to any existing model file in the same manner as we did above for the MV1 model file, as long as the constraints do not interfere with any other imposed constraints like the target return. The modification can always be written in the following way

```
> ExampleModelBox <- c(
   ExampleModel,
   "param lower{1..N} ;",
   "param upper{1..N} ;",
   "subject to Lower {i in 1..N}:  w[i] >= lower[i] ;",
   "subject to Upper {i in 1..N}:  w[i] <= upper[i] ;")
```

just as we did for the diversification constraints.

Efficient Frontier

As mentioned, the box constrained might be set such that some target returns yield infeasible portfolios, and the solver is then not able to solve the problem. Therefore we need to modify our standard frontier run file:

```
> runBOXEDR <- c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex;",
   "option eexit 100 ;",
   "let targetReturn := -999;",
   "solve;",
   "let minReturn := sum{i in 1..N} w[i]*mu[i];",
   "let maxReturn := max{i in 1..N} mu[i];",
   "for {i in 0..nReturn} {",
   "  let targetReturn := minReturn + i*(maxReturn-minReturn)/nReturn ;",
   "  solve ;",
   "  if solve_result_num == 0 then ",
   "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
   "};",
   "exit ;")
> amplRunFile(run=runBOXEDR, project="myPortfolio")
```

Here we have two modifications. (i) Since AMPL bails out warnings after 10 messages, the solver will stop its work before it was looping through all portfolios. To prevent that, we can set the AMPL option *eexit* to a positive value. The loop now skips iteration steps that are infeasible. (ii) In addition weights are only printed to the output file `"myPortfolio.txt"` when the solution was feasible, i.e. when `solve_result_num == 0`.

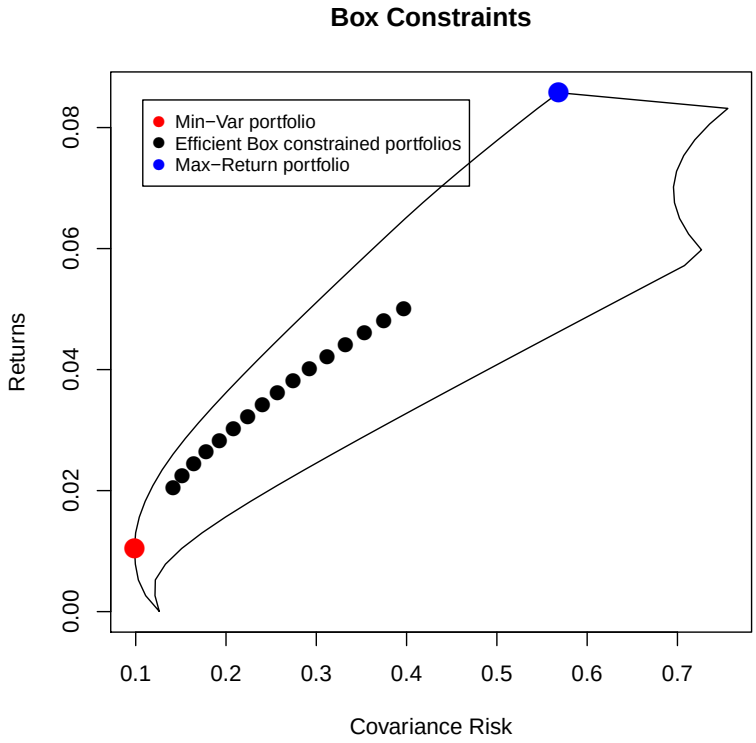


FIGURE 16.2: Box Constrained Mean-Variance Portfolio

16.4 GROUP CONSTRAINTS

In this section we discuss Group Constrainted EDR Portfolios. Group constraints are lower and upper bounds on the weights of a subgroup of assets. They can be expressed by a set of linear equations.

$$l \leq Aw \leq u \tag{16.2}$$

Where A is the constraints matrix and l and u are vectors. Their lengths is given by the number of subgroups.
Group constraints can be applied to portfolios as additional linear constraints. Here is an example for three subgroups, (10) local equities, (2) foreign equities, and (3) bonds.
Group 1 - all Swiss equities SPI, SII, MPI and ALT together not more than 70%
Group 2 - all foreign equities LMI, SPI, ALT together not more than 20%
Group 3 - all bonds SBI, LMI together at least 30%.

The corresponding matrix and constraint vectors are then given by

$$\begin{pmatrix} 0 \\ 0 \\ 0.3 \end{pmatrix} \leq \overset{SBI\ SPI\ SII\ LMI\ MPI\ ALT}{\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}} \cdot w \leq \begin{pmatrix} 0.7 \\ 0.2 \\ 1 \end{pmatrix} \tag{16.3}$$

The AMPL implementation of group constraints looks as following:

```
> GroupConstraints <- c(
   "param G ;",
   "param lower{1..G} ;",
   "param upper{1..G} ;",
   "param A{1..G,1..N} ;",
   "subject to Lower{i in 1..G}: sum{j in 1..N} A[i,j]*w[j]>=lower[i] ;",
   "subject to Upper{i in 1..G}: sum{j in 1..N} A[i,j]*w[j]<=upper[i] ;")
```

These constraints can be attached to any existing AMPL-model in the same manner as the box constraints. Similarly, the Run-File has to be modified if the efficient frontier with group constraints is to be calculated:

```
> runGROUPEDR <- c(
  "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex;",
   "option eexit 100 ;",
   "let targetReturn := -999;",
   "solve;",
   "let minReturn := sum{i in 1..N} w[i]*mu[i];",
   "let maxReturn := max{i in 1..N} mu[i];",
   "for {i in 0..nReturn} {",
   "  let targetReturn := minReturn + i*(maxReturn-minReturn)/nReturn ;",
   "  solve ;",
   "  if solve_result_num == 0 then ",
   "  for {m in 1..N} printf \"%16.12f\", w[m] > myPortfolio.txt ;",
   "};",
   "exit ;")
> amplRunFile(run=runBOXEDR, project="myPortfolio")


 [1] "minReturn"    "maxReturn"    "nReturn"      "N"            "mu"
 [6] "Sigma"        "targetReturn" "G"            "lower"        "upper"
[11] "A"
```

Again, it is important to note that we can impose group constraints not just on the Markowitz portfolio model, but on any model, just as for the Box constraints. We can always just append the constraints onto the existing model:

```
> ExampleModelGROUP <- c(
   ExampleModel,
   "param G ;",
   "param lower{1..G} ;",
   "param upper{1..G} ;",
   "param A{1..G,1..N} ;",
```

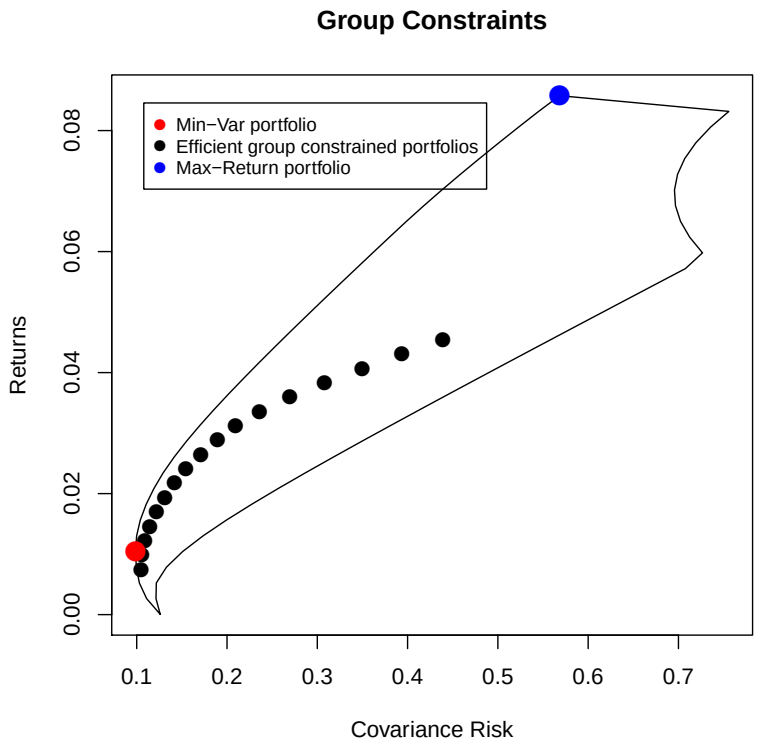


FIGURE 16.3: Group Constrained Mean-Variance Portfolio

```
"subject to Lower{i in 1..G}: sum{j in 1..N} A[i,j]*w[j]>=lower[i] ;",
"subject to Upper{i in 1..G}: sum{j in 1..N} A[i,j]*w[j]<=upper[i] ;")
```

For non-linear, non-quadratic risk measures, the solver specification in the run file that is given above has of course to be corrected.

16.5 TURNOVER CONSTRAINTS

The total portfolio turnover measures how frequently assets within a portfolio are bought and sold. Turnover constraints are normally imposed to limit the total amount of trades, i.e. to limit the overall change of the portfolio.
The portfolio turnover is defined as following:

$$|w - v| \tag{16.4}$$

where v is the vector of held assets. It is usually implemented as a constraint to a problem in the following form:

$$\begin{aligned} & \min_{w} \ \text{Risk} \\ s.t. & \\ & 1'w = 1 \\ & w_i \geq 0 \\ & \sum_{i=1}^{N} |w_i - v_i| \leq T \end{aligned} \tag{16.5}$$

where T is the limit to the turnover. It is very easy to linearize the turnover constraint in the following form:

$$\begin{aligned} & \min_{w,\delta} \ \text{Risk} \\ s.t. & \\ & 1'w = 1 \\ & w_i \geq 0 \\ & w_i - v_i \leq \delta_i \\ & v_i - w_i \leq \delta_i \\ & \sum_{i=1}^{N} \delta_i \leq T \end{aligned} \tag{16.6}$$

The AMPL implementation of the turnover constraints then looks as following:

```
> TurnoverConstraints <- c(
                "param v ;",
                "param turnover ;",
                "var delta{1..N} ;",
                "subject to Turnover1:sum{i in 1..N} delta[i] <= turnover ;",
                "subject to Turnover2{i in 1..N}: w[i]-v[i] - delta[i] <= 0 ;",
                "subject to Turnover3{i in 1..N}: v[i]-w[i]- delta[i] <= 0 ;")
```

Similar to the box and group constraint, the turnover constraints can just be attached to any existing AMPL-model.

16.6 TRACKING ERROR

Often investors want to model a specific benchmark index. The tracking error indicates how closely a portfolio follows the index to which it is benchmarked. The most common definition of the tracking error is the variance of the two portfolios' return differences:

$$\sum_{s=1}^{S}(\sum_{i=1}^{N} r_s,i(w_i-v_i))^2-(\sum_{s=1}^{S}\sum_{i=1}^{N} r_s,i(w_i-v_i))^2=(w-v)'\Sigma(w-v) \quad (16.7)$$

where v is the vector of the benchmark index. The AMPL implementation of tracking error constraints is very straightforward:

```
> TurnoverConstraints <- c(
                "param v ;",
                "param trackingerror ;",
                "param Sigma ;",
                "subject to Trackingerror1: sum{i in 1..N}
                sum{j in 1..N} (w[i]-v[i])*Sigma[i,j]*(w[j]-v[j]) <= trackingerror ;")
```

Please note that the introduction of the `Sigma`-parameter is redundant if it was already defined in the existing portfolio model.

CHAPTER 17

INTEGER CONSTRAINS

17.1 INTRODUCTION

Jobst, Horniman, Lucas, and GMitra [2001] have examined the effects of applying

- Buy-in Thresholds,
- Cardinality Constraints, and
- Transaction Roundlot Restrictions

to the portfolio selection problem. These constraint are discrete and require the use of integer variables. They are therefore called *mixed-integer problems*. It is well known that such discrete constraints are of practical importance. But their disadvantage is that these kind of constraints make the efficient frontier discontinuous, and the problems are therefore much more difficult to optimize.
The definitions of these constraints are given in the article of Jobst et. al. [2001]:

> (i) "A buy-in threshold is defined as the minimum level below which an asset is not purchased.This requirement eliminates unrealistically small trades that can otherwise be included in an optimum portfolio."
>
> (ii) "Cardinality constraints: investors may wish to specify the number of assets in their portfolio for the purpose of monitoring and control."
>
> (iii) "Roundlots are defined as discrete numbers of assets which are taken as the basic unit of investment. Investors are restricted to making transactions only in multiples of these roundlots.This overcomes the assumption of the infinite divisibility of assets inherent in the Mean-Variance rule."

AMPL is capable of implementing mixed-integer problems, but only certain solvers such as `CPLEX, XPRESS` and `Gurobi` are capable of solving these problems. In this chapter, we will see how to implement the above constraints.

17.2 BUY-IN CONSTRAINTS

As described by Jobst et al. [2001] "in the presence of buy-in constraints the portfolio weights behave as semicontinuous variables (see Beale and Forrest [1976]) and are modelled using variable upper and lower bounds in the following way. A binary variable, δ_i , and finite lower bounds l_i are associated with each asset $i = 1,\ldots,N$. The buy-in thresholds are represented by the constraint pair $l_i\delta_i \leq w_i \leq \delta_i$ and $\delta_i \in \{0,1\}$".
The mean-variance Markowitz portfolio model with buy-in constraints for example is then given by the following problem:

$$\begin{aligned} & \min_{w,\delta\in\{0,1\}} w'\Sigma w \\ s.t. & \\ & 1'w = 1 \\ & \mu' w = \overline{r} \\ & w_i \geq 0 \\ & l_i\delta_i \leq w_i \leq \delta_i \end{aligned}$$

We can introduce binary variables in AMPL by declaring them as binary right at their declaration:

```
> "var delta{1..N} binary ;"
```

We can implement the buy-in constraints in AMPL in a very straightforward manner:

```
> BuyInConstraints <- c(
   "param lower{1..N} ;",
   "var delta{1..N} binary ;",
   "subject to BuyinLower{i in 1..N}: w[i] - lower[i] * delta[i] >= 0 ;",
   "subject to BuyinUpper{i in 1..N}: w[i] -  delta[i] <= 0 ;")
```

Like in the previous chapter, we can just attach these constraints to any existing model.

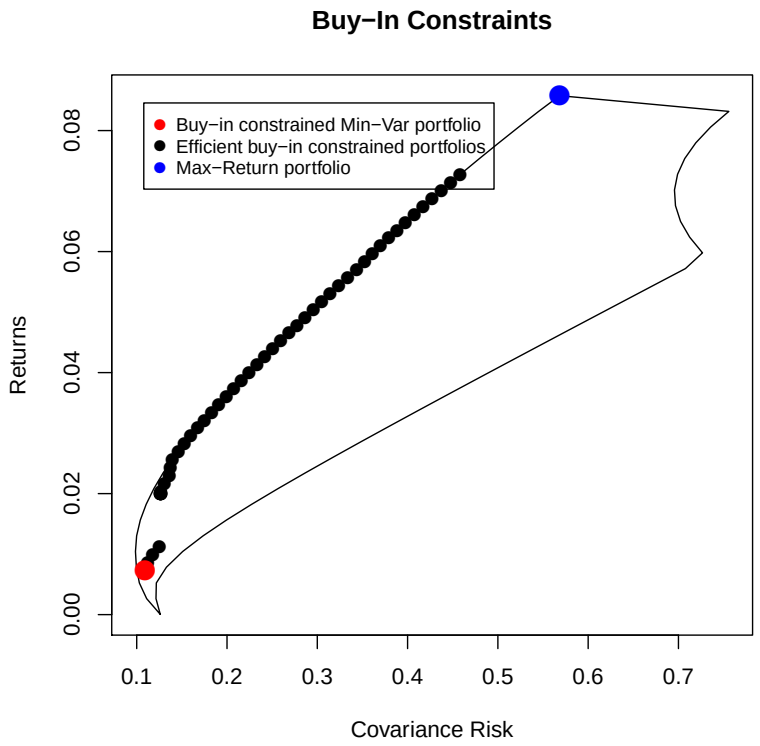


FIGURE 17.1: Buy-In Constrained Mean-Variance Portfolio

17.3 CARDINALITY CONSTRAINTS

Cardinality constraints are not exclusively used in finance, but have a wide application to reduce the number of actively used variables, for example in database design. The cardinality is originally defined as the number of elements in a set, but this definition is often extended such that the cardinality denotes the number of active elements in a vector. The cardinality of a vector with three elements unequal to zero for example is three.
The cardinality of a portfolio is defined to be the number of actively held assets:

$$\text{Card}(w) := \sum_{i=1}^{N} \mathbb{1}_{\{w_i \neq 0\}} \tag{17.1}$$

It is possible to model the cardinality with binary variables, similar to the buy-in constraints. The mean-variance Markowitz portfolio model with cardinality constraints for example is then given by the following problem:

$$\min_{w,\delta\in\{0,1\}} w'\Sigma w \tag{17.2}$$

$$s.t.$$

$$1'w = 1$$
$$\mu' w = \overline{r}$$
$$w_i \geq 0$$
$$w_i \leq \delta_i$$
$$\sum_{i=1}^{N} \delta_i \leq C$$

Again we just declare the δ variable to be binary:

```
> "var delta{1..N} binary ;"
```

We can implement the cardinality constraints in AMPL in a very straightforward manner:

```
> CardinalityConstraints <- c(
   "param C ;",
   "var delta{1..N} binary ;",
   "subject to Cardinality1{i in 1..N}: w[i] -   delta[i] <= 0 ;",
   "subject to Cardinality2: sum{i in 1..N}  delta[i] <= C ;")
```

Like in the previous chapter, we can just attach these constraints to any existing model.

17.4 ROUND LOT CONSTRAINTS

`http://edoc.hu-berlin.de/series/speps/2007-1/PDF/1.pdf`
Often, investments cannot be done continuously but in quantized manner. A stock portion for example can only be purchased in even multiples of price of one share. Round lot constraints take into account these discontinouities of investments. The implementation of round lot constraints in AMPL is more complicated than cardinality or buy-in constraints, and has to be done in the existing portfolio model. It is not possible to attach round lot constraints to an existing model.
To quantize the holding of assets, we have to change our portfolio weight variables w_i to have integer values instead of continuous ones:

$$w_i \in [0,1] \rightarrow w_i \in \{0,1,2,\dots\} \tag{17.3}$$

We then have to rescale the weights by multiplying them with the relative price of a single share (the price is of course divided by the total investment volume to be scaled correctly):

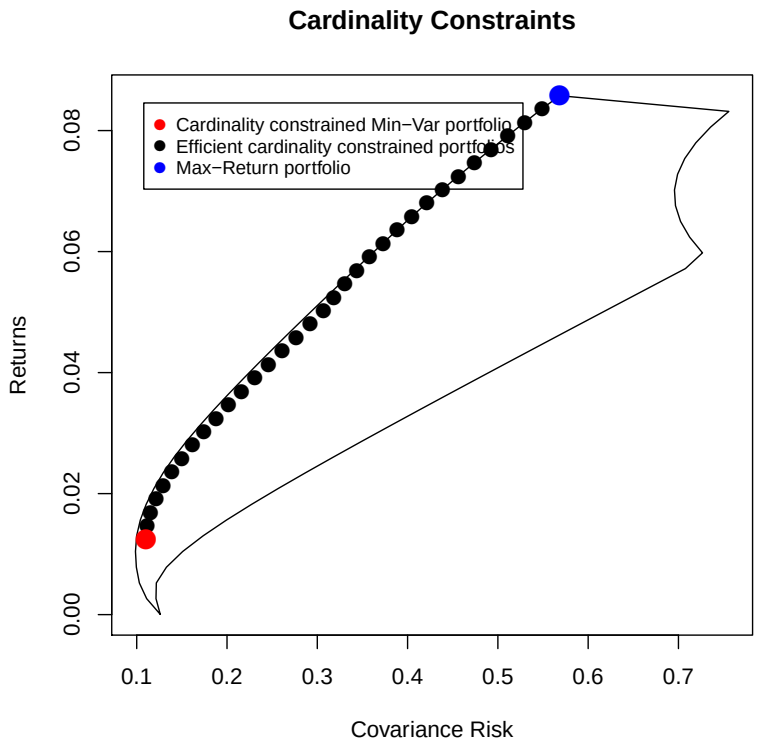


FIGURE 17.2: Cardinality Constrained Mean-Variance Portfolio

$$w_i \rightarrow w_i \cdot \zeta_i \tag{17.4}$$

Since the investments are now quantized, they do not necessarily scale to one, but may over- or undershoot the total investment volume. To soften our full-investment constraint, we add two variables $e_+ \geq 0$ and $e_- \geq 0$ to the constraints which then allow to over- and undershoot the constraint. To keep these as small as possible, they are added to the objective in order to be minimized:

$$1'w = 1 \rightarrow w'\zeta + e_+ - e_- = 1 \tag{17.5}$$

As an example, we will show the efficient Markowitz problem with round lot constraints and its implementation:

$$\min_{w_i \in \mathbb{Z}, e_+, e_-} (w \odot \zeta)\Sigma(w \odot \zeta) + e_+ + e_- \qquad (17.6)$$
$$s.t.$$
$$w'\zeta + e_+ - e_- = 1$$
$$\mu'(w \odot \zeta) \geq \overline{r}$$

where $\odot$ denotes pairwise vector multiplication.
Declaring a variable to have integer values in AMPL is very similar to binary ones:

```
> "var w{1..N} integer ;"
```

The implementation of the efficient Markowitz problem with round lot constraints in AMPL looks as following:

```
> modelMVRoundlot <- c(
   "param N ;",
   "param mu{1..N} ;",
   "param zeta{1..N} ;",
   "param Sigma{1..N,1..N} ;",
   "param targetReturn ;",
   "var w{1..N}>=0, integer;",
   "var eplus >= 0, default 0 ;",
   "var eminus >= 0, default 0 ;",
   "minimize Objective: sum{i in 1..N} sum{j in 1..N} zeta[i]*w[i] * Sigma[i,j] * w[j]*zeta[j] + eplus + emin
   "subject to Budget: sum{i in 1..N} w[i]*zeta[i] + eplus - eminus = 1 ;",
   "subject to Reward: sum{i in 1..N} mu[i] * w[i] * zeta[i] >= targetReturn ;")
> amplModelFile(model=modelMVRoundlot, project="myPortfolio")
```

We take the grand-mean as the target return. As mentioned, the ζ-vector contains the prices of a single share of the respective asset, which has to be scaled to the total amount of investment. For this demonstration purpose, we choose the odd value 0.013 for all entries of the vector:

```
> Scenarios <- 100*LPP2005.RET[, 1:6]
> requiredData(modelMVRoundlot)
[1] "N"            "mu"           "zeta"         "Sigma"        "targetReturn"

>   N <- ncol(Scenarios)
>   Sigma <- cov(Scenarios)
>   mu <- colMeans(Scenarios)
>   targetReturn <- mean(mu)
>   zeta <- rep(0.00103,N)
> dataMVRoundlot <- dataAUTO(modelMVRoundlot)
> amplDataFile(data=dataMVRoundlot, project="myPortfolio")
```

Since now the product of the weight variables w_i and the ζ_i represent the portfolio weights, we replace this expression in the AMPL Run-File:

```
> runMVRoundlot <-  c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m]*zeta[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runMVRoundlot, project="myPortfolio")
```

Optimize the portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMVRoundlot <- as.numeric(scan("myPortfolio.txt"))
> names(weightsMVRoundlot) <- colnames(Scenarios)
> weightsMVRoundlot
     SBI      SPI      SII      LMI      MPI      ALT
0.00000 0.00721 0.25853 0.33269 0.00000 0.40170
```

The weights sum almost to one:

```
> sum(weightsMVRoundlot)
[1] 1.0001
```

To get the number of asset shares, we have to divide the weights-vector by the scaled ζ-vector:

```
> assetshares <- weightsMVRoundlot/zeta
> names(assetshares) <- colnames(Scenarios)
> assetshares
SBI SPI SII LMI MPI ALT
  0   7 251 323   0 390
```

Mixed round lot constrainted and continuous asset

Sometimes only a set of assets require round lot constraints whereas the remaining assets can be treated continuously. To implement this consideration into our problem, we have to leave some of our weight variables w_i continuous and set their corresponding price to be equal to one in order treat them as before. Our existing AMPL implementation does not need much modification.
In our model file, we add the number y of assets that **do not** need round lot constraints, and a vector Y of the indices of those assets as parameters:

```
> modelMVRoundlot2 <- c(
   "param N ;",
   "param mu{1..N} ;",
   "param zeta{1..N} ;",
   "param Sigma{1..N,1..N} ;",
   "param targetReturn ;",
   "var w{1..N}>=0, integer;",
   "var eplus >= 0, default 0 ;",
```

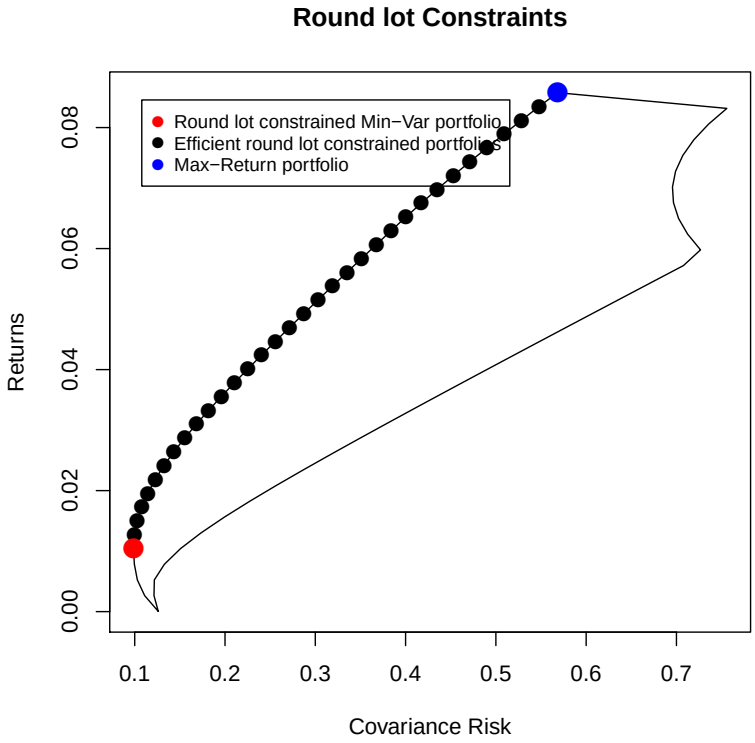


FIGURE 17.3: Round lot Constrained Mean-Variance Portfolio. As visible, the effect on the efficient frontier is small when imposing round lot constraints.

```
    "var eminus >= 0, default 0 ;",
    "minimize Objective: sum{i in 1..N} sum{j in 1..N} zeta[i]*w[i] * Sigma[i,j] * w[j]*zeta[j]   + eplus + em
    "subject to Budget: sum{i in 1..N} w[i]*zeta[i] + eplus - eminus = 1 ;",
    "subject to Reward: sum{i in 1..N} mu[i] * w[i] * zeta[i] >= targetReturn ;",

    #New Parameters
    "param y ;",
    "param Y{1..y} ;")
> amplModelFile(model=modelMVRoundlot2, project="myPortfolio")
```

We include these additional parameters in the creation of the AMPL data-file where we specify which assets are not subject to round lot constraints. As an example, we chose the SPI, the LMI and the ALT indices, so our Y-vector is $(2,4,6)$. We also have to set the ζ-values of these assets to be equal to one.

```
> Scenarios <- 100*LPP2005.RET[, 1:6]
> requiredData(modelMVRoundlot2)
[1] "N"           "mu"          "zeta"        "Sigma"       "targetReturn"
```

```
[6] "y"            "Y"
```

```
>   N <- ncol(Scenarios)
>   Sigma <- cov(Scenarios)
>   mu <- colMeans(Scenarios)
>   targetReturn <- mean(mu)
>   zeta <- rep(0.00103,N)
>   Y <- c(2,4,6)
>   y <- length(Y)
>   zeta[Y] <- 1
> dataMVRoundlot2 <- dataAUTO(modelMVRoundlot2)
> amplDataFile(data=dataMVRoundlot2, project="myPortfolio")
```

In the AMPL run-file we can then relax the specific weight variables w_i of the corresponding assets to be continuous. This can be done by the AMPL *relax*-command which removes the integer value requirement of these variables:

```
> runMVRoundlot2 <-  c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "let {i in 1..y} w[Y[i]].relax := 1;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m]*zeta[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runMVRoundlot2, project="myPortfolio")
```

We can now again optimize the portfolio and extract the weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMVRoundlot2 <- as.numeric(scan("myPortfolio.txt"))
> names(weightsMVRoundlot2) <- colnames(Scenarios)
> weightsMVRoundlot2
     SBI      SPI      SII      LMI      MPI      ALT
0.000000 0.008623 0.254410 0.335706 0.000000 0.401261
```

The weights only differ slightly from the first calculation.

CHAPTER 18

TRANSACTION COSTS

18.1 INTRODUCTION

There is no doubt, controlling transaction costs is an elementary aspect in each investment process, see Lobo, Fazel, and Boyd [2007], and Kopman, and Liu [2011]. In this chapter we will show how to manage transaction costs. The topics are:

- Linear Costs
- Piecewise Linear Costs
- Nonlinear Transaction Costs
- Fixed Transaction Costs

Throughout this chapter we use as an example the Swiss pension fund benchmark portfolio. The data are part of the Rmetrics package `timeSeries` and are loaded together with the `fPortfolio` package. As the benchmark portfolio we use the equal weights portfolio which is characterized by the following settings.
The `targetReturn` for the equal weights portfolio is defined by the *grand mean* of the portfolio scenarios, and the `targetRisk` is defined by the *grand variance* of the portfolio. `mu` is the vector of the sample means of the assets, and `Sigma` the sample covariance matrix.

18.2 LINEAR TRANSACTION COST CONSTRAINTS

Linear transaction costs are directly proportional to the amount of an asset that is bought or sold. The individual cost $c_{i,b}$ of buying asset i is not necessarily the same as the cost $c_{i,s}$ of selling the same asset i. A simple form of the linear transaction cost function, see Kopman and Liu [2011], is:

$$\mathrm{TC}_p(w,v) = \sum_{j=1}^{n}\Big[c_{i,b}\max(w_i - v_i, 0) + c_{i,s}\max(v_i - w_i, 0)\Big] \tag{18.1}$$

where v is again the vector of held assets and w_i are the new asset weights after transactions. To ensure convexity, it must be true that $c_{i,buy} > 0$, $c_{i,sell} > 0$. More complicated piece-wise linear transaction cost functions may include multiple break points at the buy side as well as the sell side. We will look at this scenario later.
Normally, transaction costs are constrained together with the expected future return. As an example, the standard Markowitz portfolio problem can be represented as:

$$\begin{aligned} &\min_w w'\Sigma w \\ s.t.& \\ &1'w = 1 \\ &\mu' w - \mathrm{TC}_p(w,v) \geq \overline{r} \\ &w_i \geq 0 \end{aligned}$$

$\mathrm{TC}_p(w,v)$ is a non-linear function, but it is easily possible to linearize it in a similar manner as the turnover constraints:

$$\begin{aligned} &\min_w w'\Sigma w \\ s.t.& \\ &1'w = 1 \\ &\mu' w - c_b\prime\delta_b - c_s\prime\delta_s \geq \overline{r} \\ &w_i \geq 0 \\ &\delta_{i,b} \geq 0 \\ &\delta_{i,s} \geq 0 \\ &\delta_{i,b} \geq w_i - v_i \\ &\delta_{i,s} \geq v_i - w_i \end{aligned}$$

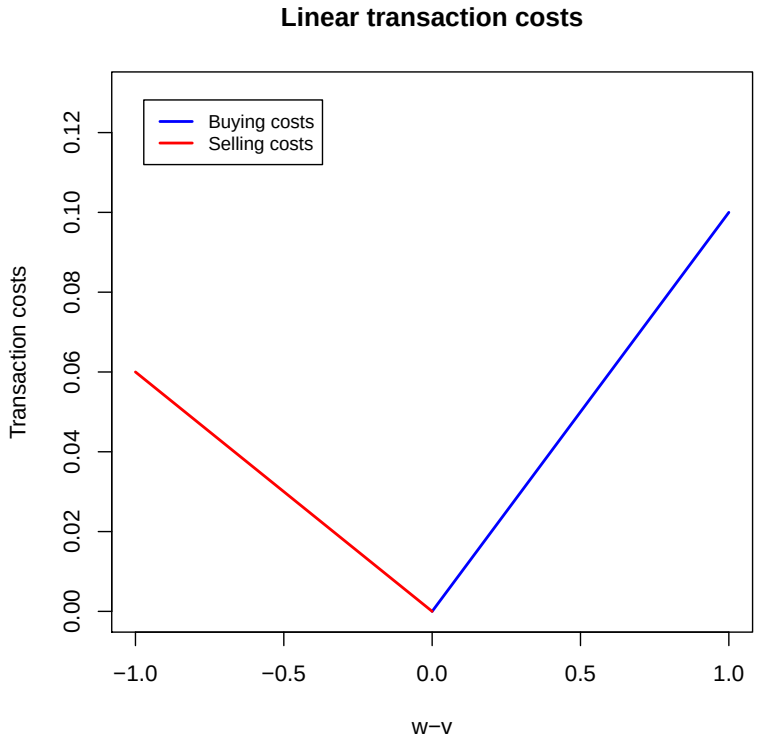


FIGURE 18.1: Visualization of linear transaction costs

We can now easily formulate an AMPL model file.

```
> modelMVtrans <- c(
   "param N ;",
   "param Sigma{1..N,1..N} ;",
   "var db{1..N} >= 0, default 0 ;",
   "var ds{1..N} >= 0, default 0 ;",
   "var w{1..N} >= 0, default 1/N ;",
   "minimize Objective: sum{i in 1..N} sum{j in 1..N} w[i] * Sigma[i,j] * w[j] ;",
   "subject to Budget: sum{i in 1..N} w[i] = 1 ;",

   # Transaction costraints
   "param mu{1..N} ;",
   "param cb{1..N} ;",
   "param cs{1..N} ;",
   "param v{1..N} ;",
   "param targetReturn ;",
   "subject to Return: sum{i in 1..N} (mu[i]*w[i] -db[i]*cb[i] -ds[i]*cs[i] ) >= targetReturn ;",
   "subject to Buy{i in 1..N}:  db[i] >= w[i]-v[i] ;",
```

```
   "subject to Sell{i in 1..N}:  ds[i] >= v[i]-w[i] ;" )
> amplModelFile(model=modelMVtrans, project="myPortfolio")
```

The rest of the procedure is the same as for the Markowitz portfolio:
We use the same AMPL run-file:

```
> runMVtrans <-  c(
   "model myPortfolio.mod ;",
   "data myPortfolio.dat ;",
   "option solver cplex ;",
   "solve ;",
   "for {m in 1..N} printf \"%16.6f\", w[m] > myPortfolio.txt ;",
   "exit ;")
> amplRunFile(run=runMVtrans, project="myPortfolio")
```

We add the v-vector and the individual buy and sell costs to the AMPL data file:

```
> Scenarios <- 100*LPP2005.RET[, 1:6]
>   N <- ncol(Scenarios)
>   Sigma <- cov(Scenarios)
>   mu <- colMeans(Scenarios)
>   targetReturn <- mean(mu)
>   v <- rep(1/N,N)
>   cb <- rep(0.02*max(mu),N)
>   cs <- rep(0.02*max(mu),N)
> dataMVtrans <- dataAUTO(modelMVtrans)
> amplDataFile(data=dataMVtrans, project="myPortfolio")
```

We can then optimize the portfolio and extract the portfolio weights:

```
> system("ampl myPortfolio.run > myPortfolio.out")
> weightsMVtrans <- as.numeric(scan("myPortfolio.txt"))
> names(weightsMVtrans) <- colnames(Scenarios)
```

Again, these are constraints that can in be attached to every portfolio model in the same manner as turnover or box constraints.

```
> LinearTransactionConstraints <- c(
   "param mu{1..N} ;",
   "param cb{1..N} ;",
   "param cs{1..N} ;",
   "param v{1..N} ;",
   "param targetReturn ;",
   "subject to Return: sum{i in 1..N} (mu[i]*w[i] -db[i]*cb[i] -ds[i]*cs[i] ) >= targetReturn ;",
   "subject to Buy{i in 1..N}:  db[i] >= w[i]-wo[i] ;",
   "subject to Sell{i in 1..N}:  ds[i] >= wo[i]-w[i] ;" )
```

If a constraint on the expected return already exists, this of course has to be removed before attaching transaction cost constraints.
Since the market impact cost is a nonlinear function, users can model it by utilizing either a multiple breakpoint piece-wise linear transaction cost described in the next subsection, or a nonlinear transaction cost function.

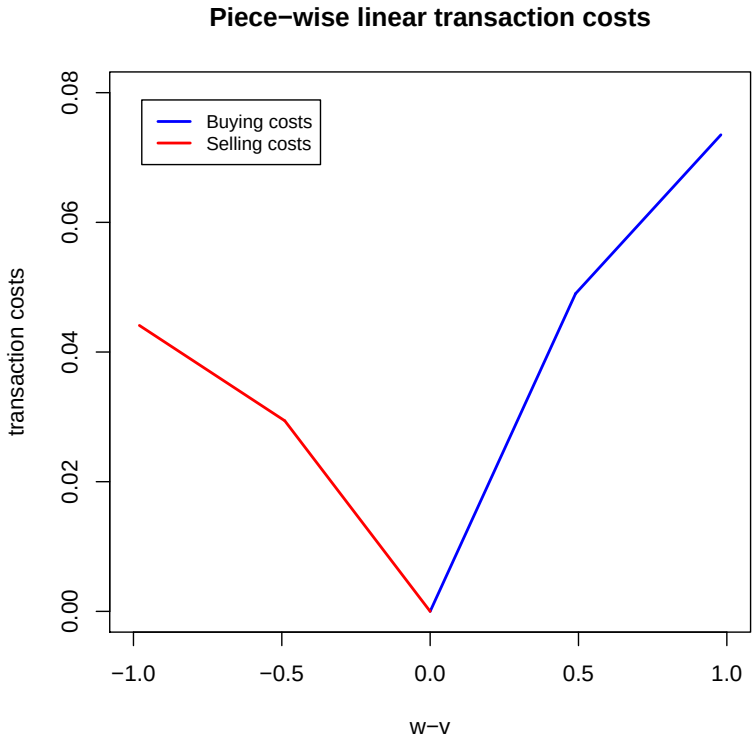


FIGURE 18.2: Piece-wise linear transaction costs

18.3 PIECE-WISE LINEAR TRANSACTION CONSTRAINTS

Often, transaction cost often have break points where the cost rate changes. These are so called piece-wise linear transaction costs. Piece-wise lienar functions are also often utilized to approximate non-linear cost functions. AMPL allows the implementation of simple piece-wise linear functions. The necessary parameters for this are the break points, and the corresponding slopes of the function before and after a break point. For example, a simple function with one breakpoint is implemented in the following way:

```
> "<<break;slope1, slope2>> * x "
```

where x is the variable. Note that you only have to give the different slopes and the breakpoint(s) of the function, no offset parameter has to be given. If we have three instead of two linear pieces, we just have to add the additional breakpoint and slope to the expression:

```
> "<<break1, break2 ;slope1, slope2, slope3>> * x "
```

If we want to make our transaction costs piece-wise linear with one break-point on both the buy and the sell side, we have to add two or more additional cost parameters to specify the cost slopes as well as the two break points to our constraints:

```
> PiecewiseTransactionConstraints <- c(
   "param mu{1..N} ;",
   "param targetReturn ;",
   "param cb1{1..N} ;",
   "param cs1{1..N} ;",
   "param cb2{1..N} ;",
   "param cs2{1..N} ;",
   "param breakb{1..N} ;",
   "param breaks{1..N} ;",
   "param v{1..N} ;",
   "var db{1..N} >= 0, default 0 ;",
   "var ds{1..N} >= 0, default 0 ;",
   "subject to Return: sum{i in 1..N} (mu[i]*w[i] ",
   "- <<breakb[i];cb1[i], cb2[i]>> db[i] ",
   "- <<breakb[i];cs1[i], cs2[i]>> ds[i] ) >= targetReturn ;",
   "subject to Buy{i in 1..N}:  db[i] >= w[i]-v[i] ;",
   "subject to Sell{i in 1..N}:  ds[i] >= v[i]-w[i] ;" )
```

It is very important to note that AMPL handles convex and non-convex piecewise linear functions differently: If the transaction costs are convex, AMPL is able to convert the model into a linear problem which can then be solved efficiently. If the transaction costs are non-convex, i.e. one of the slopes is lower than its predecessor, the portfolio model is not convex anymore, and there will be more than one localy optimal solution. AMPL then uses mixed-integer programming to solve the problem.

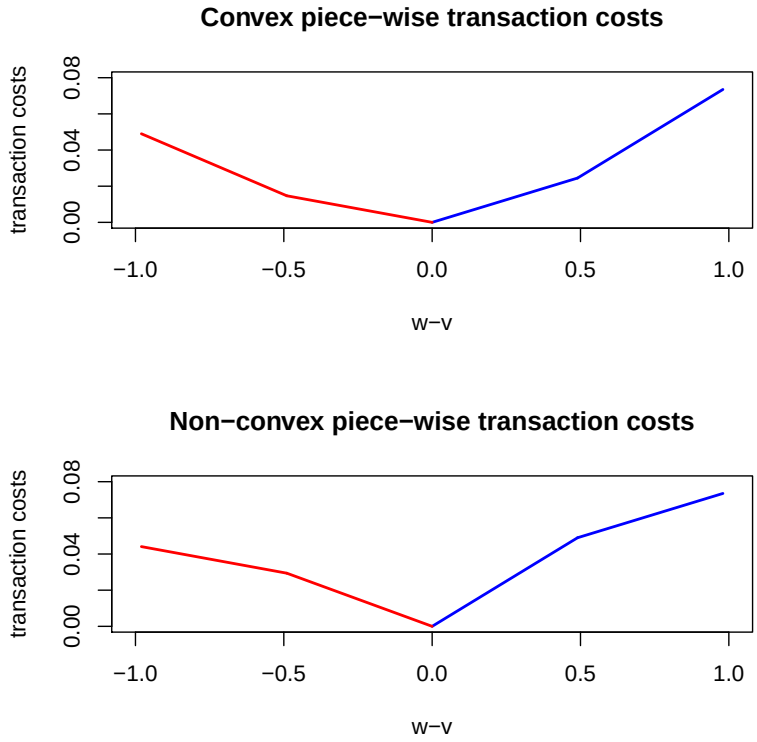


FIGURE 18.3: Comparison of convex and non-convex piece-wise linear transaction costs

18.4 FIXED TRANSACTION COSTS

Fixed transaction costs are independent of the traded amount of an asset. Instead, a fixed amount is imposed every time a trade is done. With different costs for sells and purchases, a mathematical formulation for fixed transaction costs looks as following:

$$TC_f(w,v) = \sum_{i=1} \left(\mathbb{1}_{\{w_m > v_m\}} c_{i,b} + \mathbb{1}_{\{w_m < v_m\}} c_{i,s} \right) \tag{18.2}$$

where $c_{i,fix\ buy}$ and $c_{i,fix\ sell}$ are fixed costs of buying and selling asset i, respectively.

This is a very similar constraint as the cardinality constraint we encountered in the last section since we are only interested in the number of trades, not in the trade volume. We therefore again deal with a mixed-integer problem. We can use the same approach to formulate the constraints:

$$
\begin{aligned}
w_i - v_i &\leq \delta_{i,b} \\
v_i - w_i &\leq \delta_{i,s} \\
\mu' w - \Big(\sum_{i=1}^{N} \delta_{i,b} \cdot c_{i,b} + \delta_{i,s} \cdot c_{i,s} \Big) &\geq \overline{r}
\end{aligned}
\tag{18.3}
$$

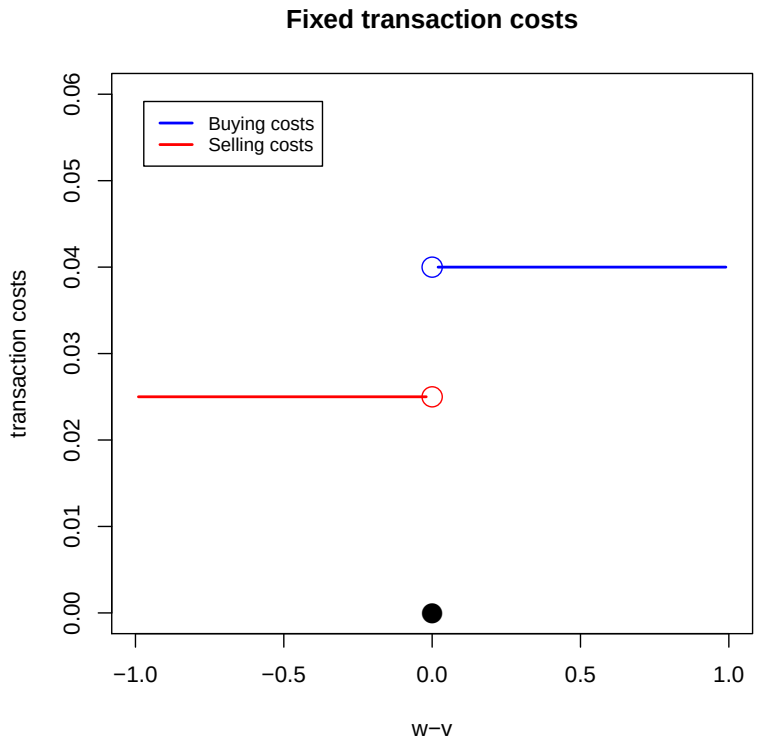


FIGURE 18.4: Fixed transaction costs

The portfolio constraints can then be implemented as:

```
> FixedTransactionConstraints <- c(
   "param mu{1..N} ;",
   "param targetReturn ;",
   "param cb{1..N} ;",
   "param cs{1..N} ;",
   "param v{1..N} ;",
   "var db{1..N} binary ;",
   "var ds{1..N} binary ;",
   "var w{1..N} >= 0, default 1/N ;",
   "subject to Reward: sum{i in 1..N} (mu[i] * w[i]- db[i]*cb[i] -ds[i]*cs[i])>= targetReturn ;",
```

```
"subject to Buy1{i in 1..N}:  w[i]-wo[i] <= db[i] ;",
"subject to Sell2{i in 1..N}:  wo[i]-w[i] <= ds[i] ;")
```

18.5 COMBINATION OF FIXED AND PIECE-WISE LINEAR TRANSACTION COSTS

Of course, transaction costs can appear in a mixed form of fixed and linear transaction costs. It is not difficult to combine these two forms of transaction costs:

$$
\begin{aligned}
w_i - v_i &\leq \delta_{i,b,l} && (18.4)\\
v_i - w_i &\leq \delta_{i,s,l}\\
w_i - v_i &\leq \delta_{i,b,f}\\
v_i - w_i &\leq \delta_{i,s,f}\\
\mu' w & && (18.5)\\
-\Big(\sum_{i=1}^{N} \delta_{i,b,f} \cdot c_{i,b,f} + \delta_{i,s,f} \cdot c_{i,s,f} & && (18.6)\\
+\delta_{i,b,l} \cdot c_{i,b,l} + \delta_{i,s,l} \cdot c_{i,s,l}\Big) &\geq \overline{r}
\end{aligned}
$$

The AMPL implementation of these constraints looks as following:

```
> LinearFixedTransactionConstraints <- c(
   "param mu{1..N} ;",
   "param targetReturn ;",
   "param cbf{1..N} ;",
   "param csf{1..N} ;",
   "param cbl{1..N} ;",
   "param csl{1..N} ;",
   "param v{1..N} ;",
   "var dbf{1..N} binary ;",
   "var dsf{1..N} binary ;",
   "var dbl{1..N} >= 0, default 0 ;",
   "var dsl{1..N} >= 0, default 0 ;",
   "var w{1..N} >= 0, default 1/N ;",
   "subject to Reward: sum{i in 1..N} (mu[i] * w[i]",
   " -dbf[i]*cbf[i] -dsf[i]*csf[i] ",
   " -dbl[i]*cbl[i] -dsl[i]*csl[i] >= targetReturn ;",
   "subject to Buy1{i in 1..N}:  w[i]-v[i] <= dbf[i] ;",
   "subject to Sell1{i in 1..N}:  v[i]-w[i] <= dsf[i] ;",
   "subject to Buy2{i in 1..N}:  w[i]-v[i] <= dbl[i] ;",
   "subject to Sell2{i in 1..N}:  v[i]-w[i] <= dsl[i] ;")
```

If we want to have piece-wise linear constraints, we have to add the break-points and the additional cost slopes to the constraints:

```
> PiecewiseLinearFixedTransactionConstraints <- c(
   "param mu{1..N} ;",
   "param targetReturn ;",
   "param cbf{1..N} ;",
   "param csf{1..N} ;",
   "param cbl1{1..N} ;",
```

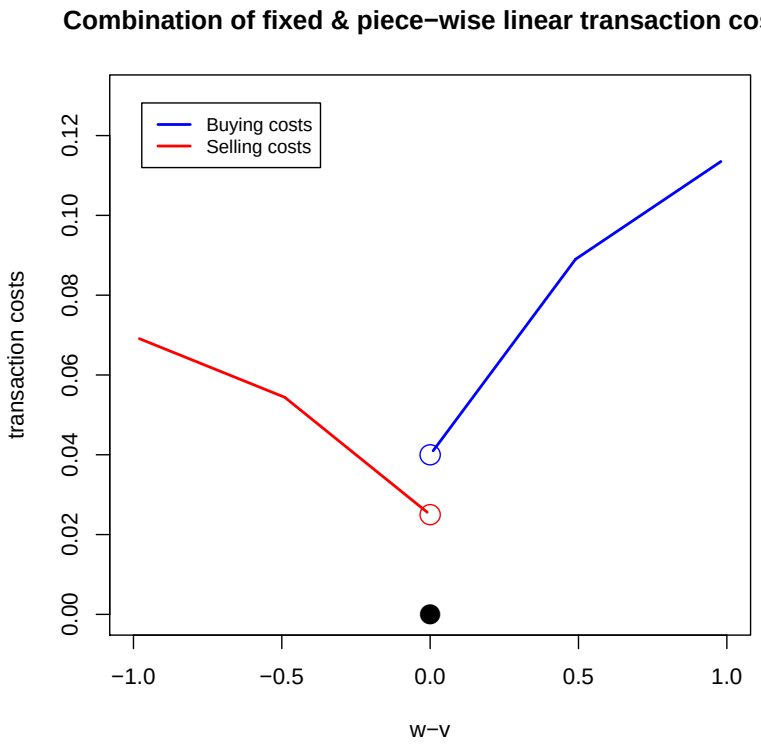


FIGURE 18.5: Combination of fixed and piece-wise linear transaction costs

```
"param csl1{1..N} ;",
"param cbl2{1..N} ;",
"param csl2{1..N} ;",
"param breakb{1..N} ;",
"param breaks{1..N} ;",
"param v{1..N} ;",
"var dbf{1..N} binary ;",
"var dsf{1..N} binary ;",
"var dbl{1..N} >= 0, default 0 ;",
"var dsl{1..N} >= 0, default 0 ;",
"var w{1..N} >= 0, default 1/N ;",
"subject to Reward: sum{i in 1..N} (mu[i] * w[i]",
" -dbf[i]*cbf[i] -dsf[i]*csf[i] ",
" -<<breakb[i];cbl1[i], cbl2[i]>> dbl[i] ",
" -<<breakb[i];csl1[i], csl2[i]>> dsl[i])>= targetReturn ;",
"subject to Buy1{i in 1..N}:  w[i]-v[i] <= dbf[i] ;",
"subject to Sell1{i in 1..N}:  v[i]-w[i] <= dsf[i] ;",
"subject to Buy2{i in 1..N}:  w[i]-v[i] <= dbl[i] ;",
"subject to Sell2{i in 1..N}:  v[i]-w[i] <= dsl[i] ;")
```

PART IX

APPENDIX

CHAPTER 19

R/AMPL API

19.1 R/AMPL API 1.0

The first R/AMPL interface was written in November 2011 and presented at the Rmetrics Meielisalp Workshop 2012. The functions are part of the *fPortfolio* package. They can be find in the package R folder in the following scripts:

- amplExec.R
- amplRxtractors.R
- amplInterface.R
- ampLibrary.Y

19.2 R/AMPL API 2.0

Is the new extended R/AMPL interface used in this documentation. In the following the R functions are briefly described.
ALMP File Creation Functions

```
> amplModelFile <- function(model, project)
 {
   # Create the AMPL Model File:
   amplModelOpen(project)
   amplModelAdd(model, project)
   invisible()
 }


> amplDataFile <- function(data, project)
 {
   # Create the AMPL Data File:
   amplDataOpen(project)
   amplDataAdd2(data, project)
```

```
  invisible()
 }


> amplRunFile <- function(run, project)
 {
   # Create the AMPL Run File:
   amplRunOpen(project)
   amplRunAdd(run, project)
   invisible()
 }
```

The functions called are those from API 1.0 beside the function

```
> amplDataAdd2 <- function(data, project)
 {
   funName <- paste0("amplDataAdd", c("Value", "Vector", "Matrix"))
   dataName <- names(data)
   for (i in 1:length(data)) {
     TEST <- 1 +
       as.integer(length(data[[i]])!=1) +
       as.integer(is.matrix(data[[i]]) )
     fun <- match.fun(funName[TEST])
     fun(data = dataName[i], data[[i]], project) }
   invisible()
 }
```

that replaces the function `amplDataAdd2()` from API 1.0.

Printing Functions
The API 2.0 printing functions are:

```
> printModel <- function(x) print(as.data.frame(x), right=FALSE)
> printRun <- function(x) print(as.data.frame(x), right=FALSE)
> printData <- function(x) print(x)
```

Utility Functions
New are also the following draft untility functions:

- `makeGroupConstraints()` - creates constraints matrix for group constraints using the “Constraints Generator Language” from Rmetrics package fPortfolio
- `optimize()` - a simple draft function that optimizes and returns the results from portfolio design
- `portfolioFigures()` - a simple draft function that computes some selected key figures for an optimized portfolio
- `portfolioPlot()` - a simple draft function that computes some selected key figures for an optimized portfolio

This functions are still prelininary and will be replaced in the future by more elaborate functions.

Constraints Generator:

```
> makeGroupConstraints <-
 function(data, groups, BIG=1e6)
 {
   # Draft Group Constraints Generator:
   constraints <- portfolioConstraints(data, portfolioSpec(), groups)
   sumW <- rbind(constraints@minsumWConstraints, constraints@maxsumWConstraints)
   ineqA <- sumW[, -1]
   ineqA.bounds <- sumW[, 1]
   ineqA.lower <- rep(-BIG, times = nrow(ineqA))
   ineqA.upper <- rep(+BIG, times = nrow(ineqA))
   directions <- rownames(ineqA)
   for (i in 1:nrow(ineqA)) {
     if (directions[i] == "lower") ineqA.lower[i] <- ineqA.bounds[i]
     if (directions[i] == "upper") ineqA.upper[i] <- ineqA.bounds[i] }
   list(G=nrow(ineqA), Groups=ineqA, lowerG=ineqA.lower, upperG=ineqA.upper)
 }
```

Optimizer:

```
> optimize <- function(project)
 {
   # Draft Optimizer and Result Extractor:
   system(paste0("ampl ", project, ".run"))
   result <- readLines(paste0(project, ".txt"))
   cat(t(t(result)),  sep="\n" )
   result <- result[2:(length(result)-2)]
   for (i in 1:3) result <- gsub("  ", " ", result)
   weights <- matrix(unlist(strsplit(result, split=" ")), byrow=TRUE, ncol=2)
   weights <- as.numeric(weights[,2])
   weights
 }
```

Portfolio Key Figures:

```
> portfolioFigures <- function(assetReturns, weights)
 {
   # Draft Portfolio Figures:
   .pfolioTargetReturn <- function (x, weights) as.vector(colMeans(x) %*% weights)
   .pfolioTargetRisk <- function (x, weights)  as.vector(sqrt(weights %*% cov(x) %*% weights))
   targetReturn <- .pfolioTargetReturn(assetReturns, weights)
   targetRisk <- .pfolioTargetRisk(assetReturns, weights)
   maxLoss <- pfolioMaxLoss(assetReturns, weights)
   CVaR <- pfolioCVaR(assetReturns, weights)
   ans <- c(targetRisk, targetReturn, maxLoss, CVaR)
   names(ans) <- c("Risk", "Return", "MaxLoss", "CVaR")
   ans
 }
```

Portfolio Plots:

```
> portfolioPlot <- function(assetReturns, weights, hull)
 {
   # Draft Portfolio Plots:
   par(mfrow=c(2, 2))
   # Hull:
   x <- 100 * assetReturns
   nAssets <- ncol(x)
   x1 <- pfolioReturn(x, weights)
   x2 <- pfolioReturn(x, rep(1/nAssets, nAssets))
   plot(hull, type="l", main="Feasible Set")
   points(colSds(x1)/100, colMeans(x1)/100, pch=19)
   points(colSds(x2)/100, colMeans(x2)/100, pch=19, col="orange")
   grid()
   # Wealth:
   X <- cbind(x1, x2)
   plot(100*cumulated(X/100), main="Wealth", plot.type="s", col=c("black", "orange"))
   # Drawdowns:
   dd <- 100*drawdowns(X/100)
   plot(dd, plot.type="s", main="Drawdowns", col=c("black", "orange"))
   abline(h=min(dd[, 1]), lty=3)
   abline(h=min(dd[, 2]), lty=1, col="orange")
   # Histogram:
   hist(x, col="steelblue", border="white", main="Density")
   box()
   invisible()
 }
```