Vrije Universiteit Brussel

Faculty of Science and Bio-Engineering Sciences
Master of Science in Actuarial Sciences

Generating correlation matrices with constraints

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Actuarial Sciences.

Jan Tuitman

Promotor: prof. dr. S. Vanduffel

VRIJE UNIVERSITEIT BRUSSEL • SCIENTIA VINCERE TENEBRAS •

August 2016

Preface

This thesis was written to obtain the degree of Master of Sciences in Actuarial Sciences at the Vrije Universiteit Brussel (VUB).

I want to thank my promotor prof. dr. Steven Vanduffel and dr. Jing Yao for proposing the topic of this thesis, for providing very useful intuition and insight and for their feedback on early drafts. Most of the results in this thesis will appear in a joint research publication with them.

Finally, I want to thank my family for putting up with me spending so much time on this project and my studies and work in general.

Jan Tuitman

Abstract

In this thesis we develop new algorithms to generate random correlation matrices satisfying certain constraints. We show how to generate random correlation matrices that have given average correlation, given total variance, given blocks along the diagonal or satisfying combinations of these constraints. We have implemented all of these algorithms in R and include an explicit example in each case.

Samenvatting

In deze thesis ontwikkelen we nieuwe algoritmen om willekeurige correlatie matrices voort te brengen die aan bepaalde voorwaarden voldoen. We laten zien hoe willekeurige correlatie matrices gevonden kunnen worden die gegeven gemiddelde correlatie, gegeven totale variantie, gegeven blokken langs de diagonaal of combinaties van deze eigenschappen hebben. We hebben deze algoritmen allemaal geïmplementeerd in R en geven een expliciet voorbeeld voor elk geval.

Contents

Chapter 1

Introduction

1.1 Correlation matrices

Let X and Y be random variables with expected values μ_X, μ_Y and standard deviations σ_X, σ_Y, respectively. The linear correlation coefficient of X and Y is then defined as

$$\rho_{X,Y} = \frac{E[X - \mu_X]E[Y - \mu_Y]}{\sigma_X \sigma_Y}.$$

The correlation coefficient is a real number contained in the interval $[-1, 1]$. We say that X, Y are uncorrelated if $\rho_{X,Y} = 0$. Independent random variables are uncorrelated but the converse is only true in certain special cases (e.g. joint normally distributed X, Y). A positive or negative correlation coefficient indicates a positive or negative linear relationship between X and Y.

Let $X_1, \ldots, X_n$ be random variables where X_i has expected value μ_i and standard deviation σ_i. Then we can consider the correlation matrix C with entries given by $C_{ij} = \rho_{X_i,X_j}$. This matrix is symmetric, positive semidefinite (i.e. nonnegative eigenvalues) with ones on the diagonal. Conversely, any symmetric, positive semidefinite matrix with ones on the diagonal can appear as a correlation matrix. Note that C contains a lot of information about the joint distribution of $X_1, \ldots, X_n$. For example, the variance of $X_1 + \ldots + X_n$ is given by

$$\text{Var}(X_1 + \ldots + X_n) = \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} \sigma_{X_i} \sigma_{X_j}. \tag{1.1}$$

This thesis is about generating correlation matrices satisfying constraints. To motivate our work, we start by giving some applications of correlation matrices in finance and insurance.

1.2 Some applications

1.2.1 Portfolio optimisation

Modern portfolio theory is a widely used method to optimise asset portfolios. It was introduced in 1952 by H. Markowitz [6], who (much later) got the Nobel prize in economics for this work. We give a very brief overview of this theory.

Suppose that there are N risky assets such that the return of asset i (say the yearly return, as a percentage) is a random variable X_i with expected value μ_i and standard deviation $\sigma_i > 0$. Moreover, assume that the correlation matrix C between the returns X_i is known. Usually, one also assumes that there exists a risk-free asset X_0 with standard deviation $\sigma_0 = 0$, but in this case the portfolio optimisation problem can be split up into two parts: first find an optimal portfolio consisting of risky assets (the one with the highest Sharpe ratio) and then take an appropriate linear combination of this portfolio and the risk-free asset. For simplicity, we will only consider risky asssets.

We have to choose the fractions w_i of our portfolio to be invested in asset i. We assume that short selling is allowed, so that the w_i can be negative as well. As we can only spend our money once, we have

$$w_1 + \cdots + w_N = 1. \tag{1.2}$$

Moreover, the expected value of the return X of the portfolio is given by the weighted average

$$\mu = w_1\mu_1 + \ldots + w_N\mu_N. \tag{1.3}$$

Finally, the variance σ^2 of X can be expressed as

$$\sigma^2 = \sum_{i=1}^{N}\sum_{j=1}^{N} C_{ij}\sigma_i\sigma_j w_i w_j. \tag{1.4}$$

We take the standard deviation σ of the return of the portfolio as measure for its risk and suppose that investors are risk averse. This means that for a given value of μ investors prefer the portfolio with lowest value of σ. Similarly, for a given value of σ they prefer the portfolio with the highest value of μ. Minimising (1.4) under the constraints (1.3) and (1.4), or equivalently maximising (1.3) under the constraints (1.2) and (1.4) are standard optimisation problems that can be solved using Lagrange multipliers. The optimal portfolio (i.e. the w_i for a given value of μ or σ) of course depend on the μ_i, σ_i and on the correlation matrix C.

In practice, the main problem with this method of portfolio optimisation is that the μ_i, σ_i and the matrix C are not known a priori but have to be estimated, e.g. from historical data. This is very hard already for the μ_i, σ_i, but especially problematic for C, since in this case $N(N-1)/2$ entries have to be estimated from N time series of asset returns. To estimate C we would need time series of past returns of length at least $(N-1)/2$, but N is typically very large. Moreover, correlation matrices of asset returns have also been observed to change over time, complicating matters further. Using an inaccurate correlation matrix might lead to sub-optimal porfolios and underestimating of the portfolio variance. This kind of model risk is sometimes called correlation risk.

As it is hard to know the exact correlation matrix, we would like to repeat the computation of the optimal portfolio with various correlation matrices to see how the results vary. However, if we do not assume anything about the correlation matrix, we do not expect to be able to draw any interesting conclusions either. Suppose that we know some properties of the correlation matrix, like the average correlation over all assets or over some classes of assets, or the exact correlations within a certain class of assets. For example we might be fairly sure about the average correlation of the returns between stocks within the same industry. Or we might know the variance of a certain portfolio (say some well known index) to a reasonable degree of precision. It would be interesting to be able to generate random correlation matrices satisfying these contraints, compute the optimal portfolios for each of them and see if any interesting conclusions can be drawn.

1.2.2 Solvency II: SCR aggregation

Since January 1st 2016, capital requirements for (re)insurance companies within the EU are subject to the Solvency II directive (2009/138/EC) [1]. It would be possible to write a whole thesis (probably even more) about this directive and its implications, but that is not our aim here. We only want to give a quick example of how correlation matrices have very important applications in the insurance context.

The central concept in (the quantitative part of) Solvency II is the so called Solvency Capital Requirement (SCR). The SCR of an insurance company is defined to be the Value at Risk (VaR) over the next 12 months of the company at the 99.5 percent probability level. In other words, it is the amount of capital the company should hold to be able to meet its obligations over the next 12 months with a probability of 99.5 percent. When its capital falls below this SCR, the company will usually get some time from supervisory authorities to remedy the situation. There is also a (significantly lower) Minimal Capital Requirement (MCR) below which supervisory authorities will usually intervene directly.

An insurance company can compute its SCR either by using the standard formula as specified in the directive, or by developing an alternative (partial) internal model (subjective to approval by supervisory authorities). In what follows we will restrict attention to the standard formula. Since not all insurance companies are able to build their own models, and the ones that are still have to compare it against the standard formula anyway, this is reasonable.

The risk that an insurance company faces is divided up into modules with various submodules which again have submodules etc. The modules at the top level are:

1. market risk
2. (counterparty) default risk
3. life insurance risk
4. health insurance risk
5. non-life insurance risk

As an example, the market risk module (which is usually the largest top level module) consists of the following submodules:

1. interest rate risk
2. equity risk
3. property risk
4. currency risk
5. spread risk
6. concentration risk
7. illiquidity premium risk

Every (sub)module has its own SCR (the VaR at the 99.5 percent probability level) and the submodules at a given level are combined (or aggregated) into the module one level above by a formula of the form

$$\mathrm{SCR}_{above} = \sqrt{\sum_{i,j} C_{ij}\mathrm{SCR}_i\mathrm{SCR}_j} \qquad (1.5)$$

where C is a correlation matrix! Note that this formula is based on the idea that diversification leads to lower risk. Only when all correlations are equal to 1 do we have that $\mathrm{SCR}_{above} = \sum \mathrm{SCR}_i$.

The correlation matrices for the various (sub)modules can be found in the directive and its annexes [1]. For example, the correlation matrix for aggregating the top level modules into the total SCR of the company is given by

$$C_{top} = \begin{pmatrix} 1.00 & 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 1.00 & 0.25 & 0.25 & 0.50 \\ 0.25 & 0.25 & 1.00 & 0.25 & 0.00 \\ 0.25 & 0.25 & 0.25 & 1.00 & 0.00 \\ 0.25 & 0.50 & 0.00 & 0.00 & 1.00 \end{pmatrix}.$$

Quite a lot can (and has) been argued against this approach. First, some would already say that VaR is not an adequate risk measure, for example because it is not subadditive. Next, a formula to aggregate VaR as in (1.5) is only valid mathematically when the submodules are normally (or elliptically) distributed and do not have tail dependencies, conditions which are often not satisfied. Finally, looking at the example correlation matrix above (all entries are integer multiples of 0.25) it is clear that it is at best an educated guess.

There are various reasons why it is hard to aggregate the different SCR. To get a good estimate for the correlation matrices in the presence of tail dependencies for example, we would need some data from tail events. However, by definition such events only happen once every 200 years! In addition, different insurance companies can have very different risk profiles, so that one correlation matrix probably does not fit all.

However, as in the example about portfolio optimisation, it could be that even though we do not know the exact correlation matrix, we still know some its properties like the average correlation between all modules or the exact correlations between a couple of them. So again it would be interesting to generate random correlation matrices satisfying these properties, compute the resulting SCR for each of them and see if any interesting conclusions can be drawn.

1.3 Generating correlation matrices

The aim of this thesis is to develop algorithms to generate random correlation matrices with certain properties, e.g. such that the average correlation, total variance (i.e. the expression in (1.1)) or some blocks along the diagonal of the correlation matrix are fixed. Note that it is not hard to generate symmetric matrices with ones on the diagonal satisfying these properties, it is the positive semidefiniteness that makes the problem nontrivial.

We are not the first to investigate the problem of generating random correlation matrices. However, existing literature has mainly focused on the following two problems:

1. given a correlation matrix C, add a sufficiently small random perturbation X to it, so that $C + X$ is still a correlation matrix,
2. given non-negative real numbers $\lambda_1, \ldots, \lambda_n$ generate random correlation matrices C with these numbers as eigenvalues.

For the first problem we refer to [7, 9, 5]. Note that in [9] the authors start from a matrix C that is not necessarily positive semidefinite but numerically find the closest one (in some sense) that is, so their problem is slightly more general . For the second problem we refer to [2, 7, 3]. Note that the algorithm in [3] has also been implemented in SAS.

We have not been able to find anything in the literature on generating correlation matrices with given average correlation or total variance. Therefore, we will have to start more or less from scratch.

1.4 This thesis

In this thesis we will develop and implement algorithms to generate random correlation matrices satisfying some constraints. The title of each of the following four chapters refers to the relevant constraint: in the second chapter we generate correlation matrices with given average correlation, in the third chapter with given total variance, in the fourth chapter with given blocks along the diagonal and in the fifth chapter with both given blocks along the diagonal and given total variance.

Each chapter has a similar structure. In the first section we define the problem at hand. Then in the second section we explain its solution. Finally, in the third section we give some examples computed with our implementation in R [8]. Note that the R code for all algorithms in this thesis (some 700 lines in total) can be found in Appendix A and is available at the webpage:

`https://perswww.kuleuven.be/jan_tuitman/correlations/`

All our algorithms have in common that (in principle) they generate all correlation matrices satisfying the relevant constraint. This is to be contrasted with for example [9, 5] for which it is not clear which correlation matrices will be attained. However, we have not really analysed the distributions of the correlation matrices that we generate, apart from some experiments with various random choices of parameters in Chapters 2 and 3 to obtain correlation matrices that are not too singular.

Chapter 2

Average correlation

2.1 Problem

Definition 2.1.1. Let $n \geq 2$ be an integer. A correlation matrix $C \in M_{n\times n}(\mathbb{R})$ is a matrix that:

1. is symmetric ($C = C^t$),
2. is positive semidefinite (all eigenvalues of C are ≥ 0),
3. has ones on the diagonal ($C_{ii} = 1$ for all $1 \leq i \leq n$).

We let $\langle \cdot, \cdot \rangle$ denote the standard inner product on $\mathbb{R}^n$, i.e. such that

$$\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i$$

for all $x, y \in \mathbb{R}^n$. Moreover, we let $\|\cdot\|$ denote the (Euclidean) norm on $\mathbb{R}^n$, i.e. such that $\|x\| = \sqrt{\langle x, x \rangle}$. We will also refer to the norm of a vector as its length. For $x, y \in \mathbb{R}^n$ we have

$$|\ \|x\| - \|y\|\ | \leq \|x - y\| \leq \|x\| + \|y\|. \tag{2.1}$$

We will refer to these inequalities as the triangle inequalities.

Theorem 2.1.2. *A matrix $C \in M_{n\times n}(\mathbb{R})$ is a correlation matrix if and only if there exists a matrix $T \in M_{n\times n}(\mathbb{R})$ such that $C = TT^t$ where the rows of T considered as vectors in $\mathbb{R}^n$ have length 1. Moreover, one can assume the matrix T to be lower triangular, i.e. $T_{ij} = 0$ for all $i < j$.*

Proof. This is called Cholesky decomposition, see [4]. □

Definition 2.1.3. The average correlation ρ of a correlation matrix $C \in M_{n\times n}(\mathbb{R})$ is defined as

$$\rho = \frac{1}{(n^2-n)} \sum_{1\leq i\neq j\leq n} C_{ij}$$

The main problem in this chapter can now be stated as follows:

Problem 2.1.4. *Given an integer $n \geq 2$ and a real number ρ, generate random correlation matrices C with average correlation ρ.*

Note that such correlation matrices do not always exist. However, we will characterise the admissible values of ρ in Theorem 2.1.7 below. The following theorem will be essential in solving Problem 2.1.4.

Definition 2.1.5. To simplify notation we denote $L = \sqrt{n + \rho(n^2 - n)}$.

Theorem 2.1.6. *Let $T \in M_{n\times n}(\mathbb{R})$ be a matrix with rows t_i satisfying $\|t_i\| = 1$ for all $1 \leq i \leq n$. We then have the following equivalence:*

$$C = TT^t \textit{ has average correlation } \rho \Leftrightarrow \|t_1 + \ldots + t_n\| = L.$$

Proof. This follows easily from:

$$\begin{aligned}\langle\sum_{i=1}^n t_i, \sum_{i=1}^n t_i\rangle &= \sum_{i=1}^n \langle t_i, t_i\rangle + 2\sum_{i<j}\langle t_i, t_j\rangle \\ &= n + 2\sum_{i<j} C_{ij}\end{aligned}$$

□

This theorem gives a nice geometric interpretation of Problem 2.1.4. We need to generate n vectors $t_1, \ldots, t_n$ in $\mathbb{R}^n$ of length 1 that sum to a vector of length L.

Theorem 2.1.7. *Problem 2.1.4 has a solution if and only if $-1/(n-1) \leq \rho \leq 1$.*

Proof. It is clear that the length $\|t_1 + \ldots + t_n\|$ of the sum of n vectors $\in \mathbb{R}^n$ of length 1 takes precisely any value between 0 and n. The corresponding bound for the average correlation ρ now follows from Theorem 2.1.6. □

2.2 Solution

Instead of generating the correlation matrix C directly, we are going to generate a matrix T as in Theorem 2.1.6.

To generate the vectors $t_1, \ldots, t_n$ we will first generate the lengths of their partial sums, i.e. $l_i = \|t_1 + \ldots + t_i\|$. The following theorem characterises these.

Theorem 2.2.1. *Let $t_1, \ldots t_n$ in $\mathbb{R}^n$ be a sequence of vectors such that $\|t_i\| = 1$ for all $1 \leq i \leq n$. We define a sequence $l_1, \ldots, l_n$ of non-negative real numbers by $l_i = \|t_1 + \ldots + t_i\|$. We then have:*

1. *$l_1 = 1$,*
2. *$|l_i - l_{i-1}| \leq 1 \leq l_i + l_{i-1}$ for all $2 \leq i \leq n$.*

Conversely, for any sequence $l_1, \ldots, l_n$ of non-negative real numbers satisfying 1. and 2., there exist $t_1, \ldots, t_n \in \mathbb{R}^n$ such that $\|t_i\| = 1$ and $l_i = \|t_1 + \ldots + t_i\|$ for all $1 \leq i \leq n$.

Proof. First, it is clear that $l_1 = \|t_1\| = 1$ implies *1.* Second, applying the triangle inequalities (2.1) with $x = t_1 + \ldots + t_i$ and $y = t_1 + \ldots + t_{i-1}$ yields *2.*

For the converse, we construct the t_i inductively, the base case $i = 1$ being trivial. If $l_{i-1} \geq 1$, then by taking t_i either parallel or anti-parallel to $t_1 + \ldots + t_{i-1}$, we see that $l_i = l_{i-1} \pm 1$ can both be achieved. By continuity all values of l_i in between are then also possible. If $l_{i-1} < 1$, the same argument holds with the lower bound $l_{i-1} - 1$ replaced by $1 - l_{i-1}$. □

Now we want to generate $l_1, \ldots, l_n$ as in Theorem 2.2.1. The next theorem tells us when a partial sequence $l_1, \ldots, l_k$ can be extended to a complete one.

Theorem 2.2.2. *Let $1 \leq k \leq n - 2$ be an integer and $l_1, \ldots, l_k$ non-negative real numbers such that $l_1 = 1$ and $|l_i - l_{i-1}| \leq 1 \leq l_i + l_{i-1}$ for all $2 \leq i \leq k$. Then the following are equivalent:*

1. *There exist non-negative real numbers $l_{k+1}, \ldots, l_n$ with $l_n = L$ such that $|l_i - l_{i-1}| \leq 1 \leq l_i + l_{i-1}$ for all $2 \leq i \leq n$.*
2. *$|L - l_k| \leq (n - k)$.*

Proof.
(*1.* $\Rightarrow$ *2.*) If $l_{k+1}, \ldots, l_n$ satisfying the conditions exist, then

$$|L - l_k| = |l_n - l_k| \leq |l_{k+1} - l_k| + \ldots + |l_n - l_{n-1}| \leq n - k.$$

(*2.* $\Rightarrow$ *1.*) We construct the l_i inductively for $k + 1 \leq i \leq n - 2$, by choosing

$$l_i \in [\max\{L - (n - i), |l_{i-1} - 1|\}, \min\{L + (n - i), l_{i-1} + 1\}]$$

arbitrarily. Note that this closed interval has been obtained by imposing all conditions from *1.* and *2.* on l_i. From the assumption

$$|L - l_{i-1}| \leq n - (i - 1)$$

it follows that this interval is non-empty. Moreover, it follows from the definition of l_i that $|l_i - l_{i-1}| \leq 1 \leq l_i + l_{i-1}$ and $|L - l_i| \leq (n-i)$, completing the induction step.

For $i = n-1$ we get an extra condition since $l_n = L$ is fixed, namely $1 \leq L + l_{n-1}$, but in this case we can choose

$$l_{n-1} \in [\max\{|L-1|, |l_{n-2} - 1|\}, \min\{L+1, l_{n-2}+1\}]$$

arbitrarily. It follows from the assumption

$$|L - l_{n-2}| \leq 2$$

that this interval is non-empty.

Finally, using the definition of l_{n-1} it is easily checked that $l_n = L$ satisfies $|l_n - l_{n-1}| \leq 1 \leq l_n + l_{n-1}$. □

As we will soon see, Theorem 2.2.2 can be used to generate the l_i starting from $l_1 = 1$. However, it will also be useful to be able to start from $l_n = L$ and work our way backwards to $l_1 = 1$. For this we will use the following (very similar) theorem.

Theorem 2.2.3. *Let $2 \leq k \leq n$ be an integer and $l_k, \ldots, l_n$ non-negative real numbers such that $l_n = L$, $|l_i - l_{i-1}| \leq 1 \leq l_i + l_{i-1}$ for all $k+1 \leq i \leq n$. Then the following are equivalent:*

1. *There exist non-negative real numbers $l_1, \ldots, l_{k-1}$ with $l_1 = 1$ such that $|l_i - l_{i-1}| \leq 1 \leq l_i + l_{i-1}$ for all $2 \leq i \leq n$.*

2. *$l_k \leq k$.*

Proof.
(*1.* ⇒ *2.*) If $l_1, \ldots, l_{k-1}$ satisfying the conditions exist, then

$$|l_k| \leq |l_k - l_{k-1}| + \ldots + |l_2 - l_1| + |l_1| \leq k.$$

(*2.* ⇒ *1.*) We construct the l_i inductively (downwards) for $2 \leq i \leq k-1$, by choosing

$$l_i \in [|l_{i+1} - 1|, \min\{l_{i+1}+1, i\}]$$

arbitrarily. Note that this closed interval has been obtained by imposing all conditions from *1.* and *2.* on l_i. From the assumption

$$l_{i+1} \leq (i+1)$$

it follows that this interval is non-empty. Moreover, it follows from the definition of l_i that $|l_{i+1} - l_i| \leq 1 \leq l_{i+1} + l_i$ and $l_i \leq i$, completing the induction step.

Finally, using the definition of l_2 it is clear that $l_1 = 1$ satisfies $|l_2 - l_1| \leq 1 \leq l_2 + l_1$. □

Now we want to turn the last two theorems into an explicit algorithm for generating random correlation matrices C with average correlation ρ.

The following pseudocode generates random $l_1, \dots, l_n$ using Theorem 2.2.2:

Algorithm 2.2.4.

```
// generate $l_1, \dots, l_n$
$l_1 := 1$
for $i := 2$ to $n - 2$ do
    $l_i :=$ random in $[\max\{L - (n - i), |l_{i-1} - 1|\}, \min\{L + (n - i), l_{i-1} + 1\}]$
end for
$l_{n-1} :=$ random in $[\max\{|L - 1|, |l_{n-2} - 1|\}, \min\{L + 1, l_{n-2} + 1\}]$
$l_n := L$
return $l_1, \dots, l_n$
```

Proof of correctness. See the proof of Theorem 2.2.2. □

Alternatively, we can work backwards using Theorem 2.2.3, leading to the following pseudocode:

Algorithm 2.2.5.

```
// generate $l_1, \dots, l_n$ (backwards)
$l_n := L$
for $i := n - 1$ downto $2$ do
    $l_i :=$ random in $[|l_{i+1} - 1|, \min\{l_{i+1} + 1, i\}]$
end for
$l_1 := 1$
return $l_1, \dots, l_n$
```

Proof. See the proof of Theorem 2.2.3. □

Now that we have generated $l_1, \dots, l_n$, we still need to generate vectors $t_1, \dots, t_n$ in $\mathbb{R}^n$ such that $\|t_i\| = 1$ and $\|t_1 + \dots + t_i\| = l_i$ for all $1 \leq i \leq n$.

Definition 2.2.6. To simplify notation we denote $s_i = t_1 + \dots + t_i$.

The following lemma tells us how to compute a vector t_k given the lengths $l_1, \dots, l_k$ and the vectors $t_1, \dots, t_{k-1}$.

Lemma 2.2.7. *Assume that $t_1, \dots, t_{k-1}$ are vectors in $\mathbb{R}^n$ such that $\|t_i\| = 1$ and $\|s_i\| = l_i$ for all $1 \leq i \leq k - 1$. Then we have the following equivalence:*

$$\|s_k\| = l_k \Leftrightarrow \langle t_k, s_{k-1} \rangle = (l_k^2 - l_{k-1}^2 - 1)/2$$

Proof. This follows easily from:

$$\begin{aligned}\|s_k\|^2 &= \|s_{k-1} + t_k\|^2 \\ &= \|s_{k-1}\|^2 + \|t_k\|^2 + 2\langle t_k, s_{k-1}\rangle \\ &= l_{k-1}^2 + 1 + 2\langle t_k, s_{k-1}\rangle \end{aligned}$$

□

Finding the vector t_k from this lemma is a rather standard problem which is solved (for example) by the following pseudocode:

Algorithm 2.2.8.

```
// generate $t_1, \ldots, t_n$, given $l_1, \ldots, l_n$
$t_1$ := random in $\mathbb{R}^n$ of length $1$
$s_1 := t_1$
for $i := 2$ to $n$ do
    $x$ := random in $\mathbb{R}^n$ of length $1$
    $y := x - s_{i-1}\langle x, s_{i-1}\rangle / \|s_{i-1}\|^2$
    $z := (s_{i-1}/\|s_{i-1}\|^2)(l_i^2 - l_{i-1}^2 - 1)/2$
    $t_i := z + (\sqrt{1 - \|z\|^2}/\|y\|)y$
    $s_i := s_{i-1} + t_i$
end for
return $t_1, \ldots, t_n$
```

Proof of correctness. Note that in the loop y is orthogonal to s_{i-1} which is parallel to z and $\langle z, s_{i-1}\rangle = (l_i^2 - l_{i-1}^2 - 1)/2$. Since t_i differs from z by a multiple of y, we have $\langle t_i, s_{i-1}\rangle = \langle z, s_{i-1}\rangle$. Finally, one easily checks that $\langle t_i, t_i\rangle = 1$. □

We now let T be the matrix with rows $t_1, \ldots, t_n$ and compute the correlation matrix $C = TT^t$ which has average correlation ρ. This finishes the solution to Problem 2.1.4.

Remark 2.2.9. In the algorithms above, when we chose random elements, we did not specify the distribution. For the random vectors in Algorithm 2.2.8 we will always take the uniform distribution on the unit sphere. However, for the choice of l_i in Algorithms 2.2.4 and 2.2.5, a uniform distribution on the indicated interval will lead to particularly singular (i.e. not so random) correlation matrices, so we will experiment with another distribution in the next section.

2.3 Implementation

We have implemented the algorithms from this chapter in R. The code can be found in the file `cor.r`.

Recall that in Algorithms 2.2.4 and 2.2.5 we still have to choose a distribution on the intervals of allowed values for l_i. A natural first choice is to take the uniform distribution on these intervals. Let us give an example of how to compute a random 6×6 correlation matrix C with average correlation $\rho = 0.2$ using Algorithm 2.2.4 with uniform distributions:

```
n=6
rho=0.2
li=gen_li_forward_uniform(n,rho)
C=gen_C_from_li(n,li)
```

This gives

$$C = \begin{pmatrix} 1.0000000 & -0.7669058 & 0.1442050 & 0.1372040 & 0.3284773 & 0.6463319 \\ -0.7669058 & 1.0000000 & -0.6059352 & -0.2850269 & -0.3895216 & -0.6645325 \\ 0.1442050 & -0.6059352 & 1.0000000 & 0.7252801 & 0.7406292 & 0.6270028 \\ 0.1372040 & -0.2850269 & 0.7252801 & 1.0000000 & 0.9604118 & 0.6341328 \\ 0.3284773 & -0.3895216 & 0.7406292 & 0.9604118 & 1.0000000 & 0.7682472 \\ 0.6463319 & -0.6645325 & 0.6270028 & 0.6341328 & 0.7682472 & 1.0000000 \end{pmatrix}$$

Note that to use Algorithm 2.2.5 instead, we simply replace `forward` by `backward`. The problem with using uniform distributions for the l_i is that often one obtains a very singular correlation matrix. For example, for the matrix C above, the (geometric) mean of the eigenvalues $\det(C)^{1/6}$ is only about 0.11. This can be understood as follows.

In the early steps of Algorithm 2.2.4, condition *2.* from Theorem 2.2.2 is usually irrelevant since $(n-k)$ is still very large. Therefore, the interval of admissible values for l_i is roughly symmetric around l_{i-1} and choosing a uniform distribution on this interval means that the expected value of l_i does not generally increase with i. However, for an honestly random correlation matrix the expected value of l_i should be about $(i/n)L$. In later steps, Algorithm 2.2.4 realises it has to catch up to still make it to distance L away from the origin, leading it to choose the remaining $l_i - l_{i-1}$ close to 1, for which the remaining t_i have to be almost parallel, so that the correlation matrix becomes highly singular.

Therefore, we have experimented a bit with another distribution. Let us first explain and motivate our choice of this distribution. If l_{i-1} has been chosen already, then we expect that l_i is about

$$\mu_i = l_{i-1} + \frac{(L - l_{i-1})}{(n - i + 1)}.$$

Let $[a_i, b_i]$ be the interval of admissible values for l_i. It seems natural to draw l_i from a normal distribution with mean μ_i, cut off at a_i and b_i. Note that since μ_i does not necessarily lie in the middle of the interval $[a_i, b_i]$, the expected value for l_i is not exactly μ_i either. Actually, in rare cases μ_i is not even contained in this interval, in which case we draw l_i from a Gaussian with mean $(a_i + b_i)/2$ cut off at a_i and b_i. For the standard deviation of the Gaussian, we have made the rather arbitrary choices $\min(\mu_i - a_i, b_i - \mu_i)/2$ when μ_i does lie in $[a_i, b_i]$ and $(b_i - a_i)/4$ if not.

Let us again give an example of how to compute a random 6×6 correlation matrix C with average correlation $\rho = 0.2$ using Algorithm 2.2.4, but now with Gaussian distributions:

```
n=6
rho=0.2
li=gen_li_forward_gaussian(n,rho)
C=gen_C_from_li(n,li)
```

This gives

$$C = \begin{pmatrix} 1.00000000 & -0.04047704 & -0.1368799 & -0.1865553 & 0.2714462 & 0.1613068 \\ -0.04047704 & 1.00000000 & 0.4464875 & 0.1216332 & 0.5057594 & 0.1689500 \\ -0.13687987 & 0.44648746 & 1.0000000 & 0.1853779 & 0.5645701 & 0.6552757 \\ -0.18655529 & 0.12163319 & 0.1853779 & 1.0000000 & 0.4899899 & -0.4408905 \\ 0.27144617 & 0.50575942 & 0.5645701 & 0.4899899 & 1.0000000 & 0.2340061 \\ 0.16130676 & 0.16895000 & 0.6552757 & -0.4408905 & 0.2340061 & 1.0000000 \end{pmatrix}$$

Note that to use Algorithm 2.2.5 instead, we again simply replace `forward` by `backward`. For this matrix we find $\det(C)^{1/6} = 0.59$ which is indeed a lot less singular than for the matrix obtained above using uniform distributions.

Of course, one example is not very convincing. Therefore, we repeated the above computations 1000 times. Using uniform distributions, the average of $\det(C)^{1/6}$ was 0.21 and using Gaussian distributions it was 0.38. Therefore, we conclude that the correlation matrices obtained using Gaussian distributions are significantly less singular than the ones obtained using uniform distributions. For larger n the difference becomes more pronounced. For example, for $n = 100$ using uniform distributions we virtually always obtain $\det(C) = 0$ while using Gaussian distributions, the average of $\det(C)^{1/100}$ was 0.25 over 1000 runs. To give some idea about runtimes, computing 1000 examples with $n = 100$ took about a minute.

For convenience of the user, the code contains a function `gen_C_from_rho` that generates the correlation matrix directly from n and ρ using the function `gen_li_forward_gaussian`.

Chapter 3

Total variance

3.1 Problem

We will now consider a slightly different problem, which generalises the one from the previous chapter.

Let $n \geq 2$ be an integer, and $\sigma_1, \ldots, \sigma_n$ positive real numbers. For $1 \leq i \leq n$ let X_i denote a real random variable with standard deviation σ_i and suppose that C denotes the correlation matrix of the X_i. Recall that

$$\text{Var}(X_1 + \ldots + X_n) = \sum_{i=1}^{n} \sigma_i^2 + 2 \sum_{i<j} C_{ij} \sigma_i \sigma_j.$$

Instead of generating correlation matrices with given average correlation, we now want to generate correlation matrices for which the sum $X_1 + \ldots + X_n$ has given variance.

Problem 3.1.1. *Given an integer $n \geq 2$, positive real numbers $\sigma_1, \ldots, \sigma_n$ and a non-negative real number S, generate random correlation matrices C such that*

$$S^2 = \sum_{i=1}^{n} \sigma_i^2 + 2 \sum_{i<j} C_{ij} \sigma_i \sigma_j. \tag{3.1}$$

Note that such correlation matrices do not always exist. However, we will characterise the admissible values of S in Theorem 3.1.4 below.

Remark 3.1.2. Note that when $\sigma_i = 1$ for all $1 \leq i \leq n$ and ρ denotes the average correlation of C, as defined in the previous chapter, then:

$$S^2 = n + \rho(n^2 - n).$$

Hence in this case, fixing S is equivalent to fixing ρ, so that Problem 3.1.1 reduces to Problem 2.1.4, which we solved in the previous chapter.

We still let $\langle\cdot,\cdot\rangle$ and $\|\cdot\|$ denote the standard inner product and norm on $\mathbb{R}^n$. Recall that we refer to the inequalities

$$|\,\|x\| - \|y\|\,| \leq \|x-y\| \leq \|x\| + \|y\|. \tag{3.2}$$

which hold for all $x, y \in \mathbb{R}^n$ as the triangle inequalities. The following theorem will be essential in solving Problem 3.1.1.

Theorem 3.1.3. *Let $T \in M_{n\times n}(\mathbb{R})$ be a matrix with rows t_i satisfying $\|t_i\| = 1$ for all $1 \leq i \leq n$. We then have the following equivalence:*

$$C = TT^t \text{ satisfies } (3.1) \Leftrightarrow \|\sigma_1 t_1 + \ldots + \sigma_n t_n\| = S.$$

Proof. This follows easily from:

$$\begin{aligned}\langle\sum_{i=1}^n \sigma_i t_i, \sum_{i=1}^n \sigma_i t_i\rangle &= \sum_{i=1}^n \sigma_i^2 \langle t_i, t_i\rangle + 2\sum_{i<j} \sigma_i\sigma_j\langle t_i, t_j\rangle \\ &= \sum_{i=1}^n \sigma_i^2 + 2\sum_{i<j} C_{ij}\sigma_i\sigma_j\end{aligned}$$

□

The next theorem characterises the values of S for which Problem 3.1.1 has a solution.

Theorem 3.1.4. *Problem 3.1.1 has a solution if and only if*

$$\max\{\sigma_{i_{max}} - \sum_{i\neq i_{max}} \sigma_i, 0\} \leq S \leq \sum_{i=1}^n \sigma_i,$$

where $1 \leq i_{max} \leq n$ denotes an index for which $\sigma_{i_{max}}$ is maximal.

Proof. We use the criterion from Theorem 3.1.3. Suppose there exists a solution, then $S = \|\sum_{i=1}^n \sigma_i t_i\|$ is clearly non-negative and by the triangle inequalities satisfies

$$\sigma_{i_{max}} - \sum_{i\neq i_{max}} \sigma_i \leq S \leq \sum_{i=1}^n \sigma_i.$$

Conversely, it is clear that for all S satisfying these conditions a solution exists.

□

3.2 Solution

Again, instead of generating the correlation matrix C directly, we are going to generate a matrix T as in Theorem 2.1.2. Recall that we have to generate vectors $t_i \in \mathbb{R}^n$ that are the rows of a matrix T such that $C = TT^t$ solves Problem 3.1.1. For this we will first generate the lengths $l_i = \|\sigma_1 t_1 + \ldots + \sigma_i t_i\|$. The following Theorem characterises these.

Theorem 3.2.1. *Let $t_1, \ldots t_n$ in $\mathbb{R}^n$ be a sequence of vectors such that $\|t_i\| = 1$ for all $1 \leq i \leq n$. We define a sequence $l_1, \ldots, l_n$ of non-negative real numbers by $l_i = \|\sigma_1 t_1 + \ldots + \sigma_i t_i\|$. We then have:*

1. $l_1 = \sigma_1$,
2. $|l_i - l_{i-1}| \leq \sigma_i \leq l_i + l_{i-1}$ *for all* $2 \leq i \leq n$.

Conversely, for any sequence $l_1, \ldots, l_n$ of non-negative real numbers satisfying 1. and 2., there exist $t_1, \ldots, t_n \in \mathbb{R}^n$ such that $\|t_i\| = 1$ and $l_i = \|\sigma_1 t_1 + \ldots + \sigma_i t_i\|$ for all $1 \leq i \leq n$.

Proof. First, it is clear that $l_1 = \|\sigma_1 t_1\| = 1$ implies *1*. Second, applying the triangle inequalities (3.2) with $x = \sigma_1 t_1 + \ldots + \sigma_i t_i$ and $y = \sigma_1 t_1 + \ldots + \sigma_{i-1} t_{i-1}$ yields *2*.

For the converse, we construct the t_i inductively, the base case $i = 1$ being trivial. If $l_{i-1} \geq \sigma_i$, then by taking t_i either parallel or anti-parallel to the vector $\sigma_1 t_1 + \ldots + \sigma_{i-1} t_{i-1}$, we see that $l_i = l_{i-1} \pm \sigma_i$ can both be achieved. By continuity all values of l_i in between are then also possible. If $l_{i-1} < \sigma_i$, the same argument holds with the lower bound $l_{i-1} - \sigma_i$ replaced by $\sigma_i - l_{i-1}$. □

Now we want to generate $l_1, \ldots, l_n$ as Theorem 3.2.1. The following proposition tells us when a partial sequence $l_1, \ldots, l_k$ can be extended to a complete one.

Theorem 3.2.2. *Suppose that $\sigma_1 \leq \sigma_2 \leq \ldots \leq \sigma_n$. Let $1 \leq k \leq n - 2$ be an integer and $l_1, \ldots, l_k$ non-negative real numbers that satisfy $l_1 = \sigma_1$ and $|l_i - l_{i-1}| \leq \sigma_i \leq l_i + l_{i-1}$ for all $2 \leq i \leq k$. Then the following are equivalent:*

1. *There exist non-negative real numbers $l_{k+1}, \ldots, l_n$ with $l_n = S$ such that $|l_i - l_{i-1}| \leq \sigma_i \leq l_i + l_{i-1}$ for all $2 \leq i \leq n$.*
2. $|S - l_k| \leq \sigma_{k+1} + \ldots + \sigma_n$ *and* $S + l_k \geq \sigma_n - (\sigma_{k+1} + \ldots + \sigma_{n-1})$.

Proof.
(*1.* $\Rightarrow$ *2.*) If $l_{k+1}, \ldots, l_n$ satisfying the conditions exist, then

$$|S - l_k| = |l_n - l_k| \leq |l_{k+1} - l_k| + \ldots + |l_n - l_{n-1}| \leq \sigma_{k+1} + \ldots + \sigma_n,$$
$$S + l_k = (l_n + l_{n-1}) + (l_k - l_{k+1}) + \ldots + (l_{n-2} - l_{n-1}) \geq \sigma_n - (\sigma_{k+1} + \ldots + \sigma_{n-1}).$$

(*2.* $\Rightarrow$ *1.*) We construct the l_i inductively for $k+1 \leq i \leq n-1$, by choosing

$$l_i \in \left[\max\{|l_{i-1} - \sigma_i|, S - \sum_{j=i+1}^{n} \sigma_j, \sigma_n - \sum_{j=i+1}^{n-1} \sigma_j - S\}, \min\{l_{i-1} + \sigma_i, S + \sum_{j=i+1}^{n} \sigma_j\}\right]$$

arbitrarily. Note that this closed interval has been obtained by imposing all conditions from *1.* and *2.* on l_i. From the assumptions

$$|S - l_{i-1}| \leq \sum_{j=i}^{n} \sigma_j \text{ and } S + l_{i-1} \geq \sigma_n - \sum_{j=i}^{n-1} \sigma_j$$

it follows that this interval is non-empty. Here we are also using that the σ_i are increasing. It follows from the definition of l_i that

1. $|l_i - l_{i-1}| \leq \sigma_i \leq l_i + l_{i-1}$,
2. $|S - l_i| \quad \leq \sigma_{i+1} + \ldots + \sigma_n$,
3. $S + l_i \quad \geq \sigma_n - (\sigma_{i+1} + \ldots + \sigma_{n-1})$.

completing the induction step.

Finally, using the definition of l_{n-1} it is easily checked that $l_n = S$ satisfies $|l_n - l_{n-1}| \leq \sigma_n \leq l_n + l_{n-1}$. □

As we will soon see, Theorem 3.2.2 can be used to generate the l_i starting from $l_1 = \sigma_1$. However, it will also be useful to be able to start from $l_n = S$ and work our way backwards to $l_1 = \sigma_1$. For this we will use the following (very similar) Theorem.

Theorem 3.2.3. *Suppose that $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$. Let $2 \leq k \leq n$ be an integer and $l_k, \ldots, l_n$ non-negative real numbers that satisfy $l_n = S$ and $|l_i - l_{i-1}| \leq \sigma_i \leq l_i + l_{i-1}$ for all $k+1 \leq i \leq n$. Then the following are equivalent:*

1. *There exist non-negative real numbers $l_1, \ldots, l_{k-1}$ with $l_1 = \sigma_1$ such that $|l_i - l_{i-1}| \leq \sigma_i \leq l_i + l_{i-1}$ for all $2 \leq i \leq n$.*
2. *$\sigma_1 - (\sigma_2 + \ldots + \sigma_k) \leq l_k \leq \sigma_1 + \ldots + \sigma_k$.*

Proof.
(*1.* $\Rightarrow$ *2.*) If $l_1, \ldots, l_{k-1}$ satisfying the conditions exist, then

$$|l_k| \leq |l_k - l_{k-1}| + \ldots + |l_2 - l_1| + |l_1| \leq \sigma_1 + \ldots + \sigma_k.$$

(*2.* $\Rightarrow$ *1.*) We construct the l_i inductively (downwards) for $2 \leq i \leq k-1$, by choosing

$$l_i \in \left[\max\{|l_{i+1} - \sigma_{i+1}|, \sigma_1 - \sum_{j=2}^{i} \sigma_j\}, \min\{l_{i+1} + \sigma_{i+1}, \sum_{j=1}^{i} \sigma_j\}\right]$$

arbitrarily. Note that this closed interval has been obtained by imposing all conditions from *1.* and *2.* on l_i. From the assumption

$$\sigma_1 - \sum_{j=2}^{i+1} \sigma_j \leq l_{i+1} \leq \sum_{j=1}^{i+1} \sigma_j$$

it follows that this interval is non-empty. Here we are also using that the σ_i are decreasing. It follows from the definition of l_i that $|l_{i+1} - l_i| \leq \sigma_{i+1} \leq l_{i+1} + l_i$ and $\sigma_1 - (\sigma_2 + \ldots + \sigma_{i-1}) \leq l_i \leq \sigma_1 + \ldots + \sigma_i$, completing the induction step.

Finally, using the definition of l_2 it is easily checked that $l_1 = \sigma_1$ satisfies $|l_2 - l_1| \leq \sigma_2 \leq l_2 + l_1$. □

Remark 3.2.4. In Theorem 3.2.2 we supposed that the σ_i were increasing, and in Theorem 3.2.3 that they were decreasing (for technical reasons). Note that this does not lead to any loss of generality, since we can first apply a permutation to sort the σ_i whichever way we want, then use the theorems and at the end apply the inverse permutation to the rows of the matrix T before computing $C = TT^t$.

Now we want to turn the last two theorems into an explicit algorithm for generating random correlation matrices C solving Problem 3.1.1.

The following pseudocode generates random $l_1, \ldots, l_n$ using Theorem 3.2.2:

Algorithm 3.2.5.

```
// given  $\sigma_1, \ldots, \sigma_n$ increasing and $S$ such that
// $\sigma_n - (\sigma_1 + \ldots + \sigma_{n-1}) \leq S \leq \sigma_1 + \ldots + \sigma_n$
// generate $l_1, \ldots, l_n$
$l_1 := 1$
for $i := 2$ to $n-1$ do
    $l_i :=$ random in $[\max\{S - \sum_{j=i+1}^{n} \sigma_j, \sigma_n - \sum_{j=i+1}^{n-1} \sigma_j - S, |l_{i-1} - \sigma_i|\},$
                     $\min\{S + \sum_{j=i+1}^{n} \sigma_j, l_{i-1} + \sigma_i\}]$
end for
$l_n := S$
return $l_1, \ldots, l_n$
```

Proof of correctness. See the proof of Theorem 3.2.2. □

Alternatively, we can work backwards using Theorem 2.2.3, leading to the following pseudocode:

Algorithm 3.2.6.

```
// given  $\sigma_1, \ldots, \sigma_n$ decreasing and $S$ such that
// $\sigma_n - (\sigma_1 + \ldots + \sigma_{n-1}) \leq S \leq \sigma_1 + \ldots + \sigma_n$
// generate $l_1, \ldots, l_n$ (backwards)
$l_n := S$
for $i := n-1$ downto $2$ do
    $l_i :=$ random in $[\max\{|l_{i+1} - \sigma_{i+1}|, \sigma_1 - \sum_{j=2}^{i} \sigma_j\}, \min\{l_{i+1} + \sigma_{i+1}, \sum_{j=1}^{i} \sigma_j\}]$
end for
$l_1 := 1$
return $l_1, \ldots, l_n$
```

Proof. See the proof of Theorem 3.2.3. □

Now that we have generated $l_1, \ldots, l_n$, we still need to generate vectors $t_1, \ldots, t_n$ in $\mathbb{R}^n$ such that $\|t_i\| = 1$ and $\|\sigma_1 t_1 + \ldots + \sigma_i t_i\| = l_i$ for all $1 \leq i \leq n$.

Definition 3.2.7. To simplify notation we denote $s_i = \sigma_1 t_1 + \ldots + \sigma_i t_i$.

The following lemma tells us how to compute a vector t_k given the lengths $l_1, \ldots, l_k$ and the vectors $t_1, \ldots, t_{k-1}$.

Lemma 3.2.8. *Assume that $t_1, \ldots, t_{k-1}$ are vectors in $\mathbb{R}^n$ such that $\|t_i\| = 1$ and $\|s_i\| = l_i$ for all $1 \leq i \leq k-1$. Then we have the following equivalence:*

$$\|s_k\| = l_k \Leftrightarrow \langle t_k, s_{k-1} \rangle = \frac{l_k^2 - l_{k-1}^2 - \sigma_k^2}{2\sigma_k}$$

Proof. This follows easily from:

$$\begin{aligned}
\|s_k\|^2 &= \|s_{k-1} + \sigma_k t_k\|^2 \\
&= \|s_{k-1}\|^2 + \|\sigma_k t_k\|^2 + 2\langle \sigma_k t_k, s_{k-1} \rangle \\
&= l_{k-1}^2 + \sigma_k^2 + 2\sigma_k \langle t_k, s_{k-1} \rangle
\end{aligned}$$ □

Finding a vector t_k from this lemma is a rather standard problem which is solved (for example) by the following pseudocode:

Algorithm 3.2.9.

```
// generate $t_1, \ldots, t_n$, given $l_1, \ldots, l_n$
$t_1 :=$ random in $\mathbb{R}^n$ of length $1$
```

```
$s_1 := \sigma_1 t_1$
for $i := 2$ to $n$ do
    $x :=$ random in $\mathbb{R}^n$ of length $1$
    $y := x - s_{i-1}\langle x, s_{i-1}\rangle / \|s_{i-1}\|^2$
    $z := (s_{i-1}/\|s_{i-1}\|^2)(l_i^2 - l_{i-1}^2 - \sigma_i^2)/(2\sigma_i)$
    $t_i := z + (\sqrt{1 - \|z\|^2}/\|y\|)y$
    $s_i := s_{i-1} + \sigma_i t_i$
end for
return $t_1, \ldots, t_n$
```

Proof of correctness. Note that in the loop y is orthogonal to s_{i-1} which is parallel to z and $\langle z, s_{i-1}\rangle = (l_i^2 - l_{i-1}^2 - \sigma_i^2)/(2\sigma_i)$. Since t_i differs from z by a multiple of y, we have $\langle t_i, s_{i-1}\rangle = \langle z, s_{i-1}\rangle$. Finally, one easily checks that $\langle t_i, t_i\rangle = 1$. $\square$

We now let T be the matrix with rows $t_1, \ldots, t_n$ and compute the correlation matrix $C = TT^t$ which solves Problem 3.1.1.

3.3 Implementation

We have implemented the algorithms from this chapter in R. The code can be found in the file `var.r`. Let us give an example of how to compute a random 6×6 correlation matrix with $\sigma_i = i$ for $i = 1, \ldots, 6$ and $S = 10$ using Algorithm 3.2.5 with uniform distributions for the l_i:

```
n=6
sigmai=seq(1,6)
S=10
li=gen_li_forward_uniform(n,sigmai,S)
C=gen_C_from_li(n,sigmai,li)
```

This gives

$$C = \begin{pmatrix} 1.00000000 & 0.40489182 & -0.76723212 & -0.08523962 & 0.4605829 & 0.35101758 \\ 0.40489182 & 1.00000000 & -0.63772837 & -0.18420237 & 0.2461405 & 0.09452355 \\ -0.76723212 & -0.63772837 & 1.00000000 & 0.55205732 & -0.7066766 & -0.03967874 \\ -0.08523962 & -0.18420237 & 0.55205732 & 1.00000000 & -0.6505148 & 0.52180633 \\ 0.46058294 & 0.24614051 & -0.70667664 & -0.65051479 & 1.0000000 & 0.29349251 \\ 0.35101758 & 0.09452355 & -0.03967874 & 0.52180633 & 0.2934925 & 1.00000000 \end{pmatrix}$$

Note that to use Algorithm 3.2.6 instead, we simply replace `forward` by `backward`. This matrix is quite singular since $\det(C)^{1/6} = 0.15$. As in the previous chapter we have also experimented with drawing l_i from a Gaussian distribution. Let $[a_i, b_i]$ be the interval of admissible values for l_i. This time we draw l_i from a normal distribution with mean

$$\mu_i = l_{i-1} + \frac{\sigma_i}{\sum_{j=i}^n \sigma_j}(S - l_{i-1})$$

cut off at a_i and b_i. In rare cases where μ_i is not contained in the interval $[a_i, b_i]$, we take a normal distribution with mean $(a_i + b_i)/2$ instead. For the standard deviations of the Gaussians, we have made the rather arbitrary choices $\min(\mu_i - a_i, b_i - \mu_i)/2$ when μ_i does lie in $[a_i, b_i]$ and $(b_i - a_i)/4$ if not.

Let us again give an example of how to compute a random 6×6 correlation matrix with $\sigma_i = i$ for $i = 1, \ldots, 6$ and $S = 10$ using Algorithm 3.2.5, but now with Gaussian distributions for the l_i:

```
n=6
sigmai=seq(1,6)
S=10
li=gen_li_forward_gaussian(n,sigmai,S)
C=gen_C_from_li(n,sigmai,li)
```

This gives

$$C = \begin{pmatrix} 1.00000000 & 0.40489182 & -0.76723212 & -0.08523962 & 0.4605829 & 0.35101758 \\ 0.40489182 & 1.00000000 & -0.63772837 & -0.18420237 & 0.2461405 & 0.09452355 \\ -0.76723212 & -0.63772837 & 1.00000000 & 0.55205732 & -0.7066766 & -0.03967874 \\ -0.08523962 & -0.18420237 & 0.55205732 & 1.00000000 & -0.6505148 & 0.52180633 \\ 0.46058294 & 0.24614051 & -0.70667664 & -0.65051479 & 1.0000000 & 0.29349251 \\ 0.35101758 & 0.09452355 & -0.03967874 & 0.52180633 & 0.2934925 & 1.00000000 \end{pmatrix}$$

This matrix is less singular than the one obtained above, since $\det(C)^{1/6} = 0.43$. Since one example is not very convincing, we have repeated each of the examples above 1000 times. Using uniform distributions, $\det(C)^{1/6}$ was 0.30 on average and using Gaussian distributions 0.40. For larger n the difference becomes more pronounced. For example for $n = 100$, $\sigma_i = i$ for $i = 1, \ldots, 100$ and $S = 1000$, using uniform distributions the average of $\det(C)^{1/n}$ was 0.13 while using Gaussian distributions it was 0.27. To give an idea about runtimes, computing 1000 examples for $n = 100$ took about a minute.

For convenience of the user, the code contains a function `gen_C_from_S` that generates the correlation matrix directly from n, σ_i and ρ using the function `gen_li_forward_gaussian`.

Chapter 4

Block structure

4.1 Problem

We now want to generate random correlation matrices with some fixed blocks. To make this more precise, suppose that $n = n_1 + \ldots + n_k$ are all positive integers, that we are given an $n_i \times n_i$ correlation matrix C_i for each $1 \leq i \leq k$ and that we want to generate a random $n \times n$ correlation matrix C which contains the C_i as consecutive blocks along the diagonal.

Equivalently, we have n random variables partitioned into k groups such that the correlations between random variables within the same group is known and we want to generate the remaining correlations. For example, the C_i could have been generated using the algorithms from the previous chapters, if the average correlation or total variance within each group were known.

Problem 4.1.1. *Generate random correlation matrices C containing the C_i as consecutive blocks along the diagonal.*

4.2 Solution

Definition 4.2.1. For all $1 \leq i \leq k$, let $T_i \in M_{n \times n}(\mathbb{R})$ be a matrix such that $C_i = T_i T_i^t$. Note that the existence of such a matrix follows from Theorem 2.1.2. For $1 \leq i \leq k$ and $1 \leq j \leq n_i$ let $t_{ij} \in \mathbb{R}^{n_i}$ denote the j-th row of T_i.

Definition 4.2.2. Let $m < n$ be positive integers. We will always consider $\mathbb{R}^m$ to be a vector subspace of $\mathbb{R}^n$ by embedding it onto the first m components, i.e. by sending the vector $(x_1, \ldots, x_m)$ to the vector $(x_1, \ldots, x_m, 0, \ldots, 0)$. Note that this choice of embedding preserves the standard inner product, by which we mean that $\langle x, y \rangle$ does not depend on whether we consider $x, y \in \mathbb{R}^m$ to be elements of $\mathbb{R}^m$ or $\mathbb{R}^n$.

Definition 4.2.3. For n a positive integer, we let $O(n)$ denote the orthogonal group in dimension n, i.e. the subset of matrices $M \in M_{n\times n}(\mathbb{R})$ for which $M^t = M^{-1}$. Equivalently, a matrix $M \in M_{n\times n}(\mathbb{R})$ is element of $O(n)$ if and only it it preserves the standard inner product on $\mathbb{R}^n$, by which we mean that $\langle Mx, My\rangle = \langle x, y\rangle$ for all $x, y \in \mathbb{R}^n$.

Lemma 4.2.4. *Let m, n be positive integers. Suppose that $v_1, \ldots, v_m$ and $w_1, \ldots, w_m$ are vectors in $\mathbb{R}^n$ such that $\langle v_i, v_j\rangle = \langle w_i, w_j\rangle$ for all $1 \le i, j \le m$. Then there exists an orthogonal matrix $M \in O(n)$ such that $Mv_i = w_i$ for all $1 \le i \le m$.*

Proof. Let V be the vector space spanned by $v_1, \ldots, v_m$ and W the vector space spannned by $w_1, \ldots, w_m$. Note that if $c_1v_1 + \ldots + c_mv_m = 0$ for some $c_1, \ldots, c_m \in \mathbb{R}$ then $c_1w_1 + \ldots + c_mw_m = 0$ as well. This follows from the fact that a vector $u \in \mathbb{R}^n$ is zero if and only if $\langle u, u\rangle = 0$. In particular this implies that $\dim(V) = \dim(W)$. So there exists a unique linear transformation $\mathcal{L} : V \to W$ such that $\mathcal{L}(v_i) = w_i$ for all $1 \le i \le m$.

Let $e_1, \ldots, e_l$ be an orthonormal basis for the orthogonal complement of V in $\mathbb{R}^n$ and $f_1, \ldots, f_l$ an orthonormal basis for the orthogonal complement of W in $\mathbb{R}^n$. We extend $\mathcal{L}$ to all of $\mathbb{R}^n$ by setting $\mathcal{L}(e_i) = f_i$ for all $1 \le i \le l$. By construction, $\mathcal{L}$ preserves all inner products on $\mathbb{R}^n$. Therefore, its matrix M is indeed an element of $O(n)$. □

Theorem 4.2.5. *Let $M_1, \ldots, M_k$ be k elements of $O(n)$. For $1 \le i \le k$ and $1 \le j \le n_i$ let $\tau_{ij} = M_it_{ij} \in \mathbb{R}^n$ (where we consider t_{ij} as an element of $\mathbb{R}^n$). Let T be the matrix with rows τ_{ij} (ordered first by i then j) and $C = TT^t$. Then C is a solution to Problem 4.1.1. Conversely, any solution C to Problem 4.1.1 can be obtained from k elements $M_1, \ldots, M_k$ of $O(n)$ this way.*

Proof. Since C is of the form TT^t, it is clearly a correlation matrix. Therefore, to check that C is solution to Problem 4.1.1, we only have to check that it contains the C_i as consecutive blocks along the diagonal. Note that because $M_i \in O(n)$, we have $\langle \tau_{ij}, \tau_{ik}\rangle = \langle t_{ij}, t_{ik}\rangle$ for all $1 \le i \le k$ and $1 \le j, k \le n_i$. However, the $\langle t_{ij}, t_{ik}\rangle$ are exactly the entries of C_i and the $\langle \tau_{ij}, \tau_{ik}\rangle$ form the k-th diagonal block of C.

For the converse, suppose that C is a solution to Problem 4.1.1 and that $T \in M_{n\times n}(\mathbb{R})$ is a matrix such that $C = TT^t$. Then we can number the rows of T as τ_{ij} with $1 \le i \le k$ and $1 \le j \le n_i$ such that $(C_i)_{jk} = \langle \tau_{ij}, \tau_{ik}\rangle$. By the definition of T_i we have $(C_i)_{jk} = \langle t_{ij}, t_{ik}\rangle$ as well. Therefore, by Lemma 4.2.4 for all $1 \le i \le k$ there exists an orthogonal matrix $M_i \in O(n)$ such that $M_it_{ij} = \tau_{ij}$ for all $1 \le j \le n_i$. □

Now we want to turn Theorem 4.2.5 into a concrete algorithm to compute correlation matrices C that solve Problem 4.1.1. The pseudocode for this algorithm is as follows.

Algorithm 4.2.6.

```
// generate $C$ with blocks $C_1, \dots, C_k$ along the diagonal
for $i := 1$ to $k$ do
    Compute $T_i$ (with rows $t_{ij}$) from $C_i$
    $M_i :=$ random in $O(n)$
    $\tau_{ij} := M_i t_{ij}$
end for
$T :=$ matrix with rows $\tau_{ij}$
$C := TT^t$
return $C$
```

Proof of correctness. See the proof of Theorem 4.2.5. □

Remark 4.2.7. As usual, we have not specified the distribution of a random element of $O(n)$. However, in this case (unlike in previous chapters) there is a very natural choice of distribution. Since $O(n)$ is a compact topological group, it carries a unique (left) translation invariant probability measure called the Haar measure. Note that there are good algorithms available to generate random orthogonal matrices according to this distribution [10].

Remark 4.2.8. Algorithm 4.2.6 can be simplified somewhat in practice. First, computing Cholesky decompositions is often not necessary since the T_i are known already, e.g. when the C_i are generated as in the previous chapters. Second, multiplying all M_i with the same $M \in O(n)$ leaves C invariant. Therefore, we may assume without loss of generality that $M_1 = I$.

4.3 Implementation

We have implemented the algorithm from this chapter in R. The code can be found in the file `blocks.r`. Let us give an example of how to use the code. Suppose we are looking for a 6×6 correlation matrix C such that the 3×3 upper left corner has average correlation 0.2 and the 3×3 lower right corner average correlation 0. We start by generating the 3×3 blocks using the code from `cor.r`:

For the upper left block:

```
n1=3
rho1=0.2
C1=gen_C_from_rho(n1,rho1)
```

and for the lower right block:

```
n2=3
rho2=0
C2=gen_C_from_rho(n2,rho2)
```

For the upper left block this gives

$$C_1 = \begin{pmatrix} 1.0000000 & 0.1307564 & 0.8562048 \\ 0.1307564 & 1.0000000 & -0.3869612 \\ 0.8562048 & -0.3869612 & 1.0000000 \end{pmatrix}$$

and for the lower right block

$$C_2 = \begin{pmatrix} 1.00000000 & 0.03949805 & 0.4043501 \\ 0.03949805 & 1.00000000 & -0.4438482 \\ 0.40435012 & -0.44384816 & 1.0000000 \end{pmatrix}$$

Note that so far we have only used algorithms and code from Chapter 2. To find a correlation matrix C with C_1 and C_2 as blocks along the diagonal, we now use the code from `blocks.r`.

```
Ci=list(C1,C2)
C=gen_C_from_Ci(Ci)
```

This gives

$$C = \begin{pmatrix} 1.0000000 & 0.1307564 & 0.85620478 & 0.73725058 & 0.51956587 & 0.0912944 \\ 0.1307564 & 1.0000000 & -0.38696122 & -0.43638596 & 0.84869870 & -0.7474391 \\ 0.8562048 & -0.3869612 & 1.00000000 & 0.87710025 & 0.06164776 & 0.5356547 \\ 0.7372506 & -0.4363860 & 0.87710025 & 1.00000000 & 0.03949805 & 0.4043501 \\ 0.5195659 & 0.8486987 & 0.06164776 & 0.03949805 & 1.00000000 & -0.4438482 \\ 0.0912944 & -0.7474391 & 0.53565467 & 0.40435012 & -0.44384816 & 1.0000000 \end{pmatrix}$$

Chapter 5

Block structure and total variance

5.1 Problem

We now study a variation on the problem from the previous chapter, where apart from the blocks along the diagonal, we also fix the total variance. To make this precise, let $n = n_1 + \ldots + n_k$ be positive integers. For each $1 \leq i \leq k$ and $1 \leq j \leq n_i$ let σ_{ij} be a positive real number and X_{ij} a real random variable with standard deviation σ_{ij}. Suppose that we are given a non-negative real number S and for each $1 \leq i \leq k$ an $n_i \times n_i$ correlation matrix C_i. We now want to generate a random $n \times n$ correlation matrix for the X_{ij}, such that the correlations between the X_{ij} for any fixed value of i are given by the entries of the matrix C_i and the variance of the sum of X_{ij} over all $1 \leq i \leq k$ and $1 \leq j \leq n_i$ is equal to S^2. We then say that C has total variance S^2.

Problem 5.1.1. *Generate random correlation matrices C containing the C_i as consecutive blocks along the diagonal with total variance S^2.*

Remark 5.1.2. Note that a solution C to Problem 5.1.1 is also a solution to Problem 4.1.1, since we have just imposed an extra condition (on the total variance). Therefore, we have to work out what choices of orthogonal matrices $M_1, \ldots, M_k$ in Theorem 4.2.5 give a correlation matrix C with total variance S^2.

5.2 Solution

Let us first recall some notation and terminology from the previous chapter.

Definition 5.2.1. For all $1 \leq i \leq k$, let $T_i \in M_{n \times n}(\mathbb{R})$ denote a matrix such that $C_i = T_i T_i^t$ as in Definition 4.2.1. Moreover, for all $1 \leq i \leq k$ and $1 \leq j \leq n_i$, let $t_{ij} \in \mathbb{R}^{n_i}$ denote the j-th row of T_i (note that $\|t_{ij}\| = 1$). We also consider t_{ij} to be an element of $\mathbb{R}^n$ as in Definition 4.2.2.

We need some new definitions as well.

Definition 5.2.2. We let $v_i \in \mathbb{R}^{n_i}$ be defined as $v_i = \sigma_{i1}t_{i1} + \ldots + \sigma_{in_i}t_{in_i}$ and denote $S_i = \|v_i\|$ for all $1 \leq i \leq k$.

Theorem 5.2.3. *Let $w_1, \ldots, w_k \in \mathbb{R}^n$ be vectors such that $\|w_1 + \ldots + w_k\| = S$ and $\|w_i\| = S_i$ for all $1 \leq i \leq k$. Then for all $1 \leq i \leq k$ there exists an orthogonal matrix $M_i \in O(n)$ such that $w_i = M_i v_i$. For all $1 \leq i \leq k$ and $1 \leq j \leq n_i$ define $\tau_{ij} \in \mathbb{R}^n$ by $\tau_{ij} = M_i t_{ij}$. Let T be the matrix with rows τ_{ij} (first ordered by i then j). Then the matrix $C = TT^t$ is a solution to Problem 5.1.1.*

Conversely, suppose that the matrix C is a solution to Problem 5.1.1 and let T be a matrix such that $C = TT^t$. Let $\tau_{ij} \in \mathbb{R}^n$ be the rows of T (first ordered by $1 \leq i \leq k$ then $1 \leq j \leq n_i$). For all $1 \leq i \leq k$, let $w_i \in \mathbb{R}^n$ denote the vector $w_i = \sigma_{i1}\tau_{i1} + \ldots + \sigma_{in_i}\tau_{in_i}$. Then $\|w_1 + \ldots + w_k\| = S$ and $\|w_i\| = S_i$ for all $1 \leq i \leq k$. Moreover, for each $1 \leq i \leq k$ there exists an orthogonal matrix $M_i \in O(n)$ such that $M_i t_{ij} = \tau_{ij}$ for all $1 \leq j \leq n_i$.

Proof. If $\|w_i\| = S_i$, then $\langle v_i, v_i \rangle = \langle w_i, w_i \rangle$. Therefore, by Lemma 4.2.4 for all $1 \leq i \leq k$ there exists $M_i \in O(n)$ such that $M_i v_i = w_i$. Since M_i is orthogonal and $\tau_{ij} = M_i t_{ij}$ the inner products between the τ_{ij} are the same as between the t_{ij} for any fixed value of $1 \leq i \leq k$. Therefore, the matrix C contains the C_i as consecutive blocks along the diagonal. By Theorem 3.1.3, the total variance of C is equal to

$$\| \sum_{\substack{1 \leq i \leq k \\ 1 \leq j \leq n_i}} \sigma_{ij}\tau_{ij} \|^2 = \| \sum_{i=1}^{k} w_k \|^2 = S^2,$$

so that the matrix C is indeed a solution to Problem 5.1.1.

For the converse, let C be a solution to Problem 5.1.1 and let T be a matrix such that $C = TT^t$. Moreover, let τ_{ij} be the rows of T (first ordered by $1 \leq i \leq k$ then $1 \leq j \leq n_i$). Since the matrix C contains the C_i as consecutive blocks along the diagonal, the inner products between the τ_{ij} are the same as between the t_{ij} for any fixed value of $1 \leq i \leq k$. Therefore, by Lemma 4.2.4 for all $1 \leq i \leq k$ there exists an orthogonal matrix $M_i \in O(n)$ such that $M_i t_{ij} = \tau_{ij}$ for all $1 \leq j \leq n_i$. In particular, we have that $\|w_i\| = \|v_i\| = S_i$ for all $1 \leq i \leq k$. The equality $\|w_1 + \ldots + w_k\| = S$ follows directly from the fact that the matrix C has total variance S. □

Summarising, we can solve Problem 5.1.1 in two steps:

1. Generate vectors $w_1, \ldots, w_k \in \mathbb{R}^n$ such that $\|w_1 + \ldots + w_k\| = S$ and $\|w_i\| = S_i$ for all $1 \leq i \leq k$.

2. Generate $M_1, \ldots, M_k \in O(n)$ such that $M_i v_i = w_i$ (this is always possible). Let T be the matrix with rows $\tau_{ij} = M_i t_{ij}$ (first ordered by $1 \leq i \leq k$ then $1 \leq j \leq n_i$) and define $C = TT^t$.

Then the matrix C is a solution to Problem 5.1.1 and any solution is obtained this way.

However, part 1. is almost the same as the problem that we solved in Chapter 3. There we generated $t_1, \ldots, t_n \in \mathbb{R}^n$ such that $\|\sigma_1 t_1 + \ldots + \sigma_n t_n\| = S$ and $\|t_i\| = 1$ for all $1 \leq i \leq n$. Here we want to generate $w_1, \ldots, w_k \in \mathbb{R}^n$ with $k \leq n$ such that $\|w_1 + \ldots + w_k\| = S$ and $\|w_i\| = S_i$. Taking $n = k$, $\sigma_i = S_i$ and $t_i = w_i / S_i$ this is equivalent, apart from the fact that $w_1, \ldots, w_k$ are to be contained in $\mathbb{R}^n$ and not $\mathbb{R}^k$. However, this last problem is easily solved by generating $w_1, \ldots, w_k \in \mathbb{R}^k$, embedding them into $\mathbb{R}^n$ as in Definition 4.2.2 and multiplying all by a single random orthogonal matrix $M \in O(n)$.

Theorem 5.2.4. *Problem 5.1.1 has a solution if and only if*

$$S_{i_{max}} - \sum_{i \neq i_{max}} S_i \leq S \leq \sum_{i=1}^{k} S_i,$$

where i_{max} is an index for which S_i is maximal.

Proof. We know from (the proof of) Theorem 5.2.3 that Problem 5.1.1 has a solution if and only if there exist $w_1, \ldots, w_k \in \mathbb{R}^n$ such that $\|w_1 + \ldots + w_k\| = S$ and $\|w_i\| = S_i$ for all $1 \leq i \leq k$. The result now follows from the argument from the previous paragraph and Theorem 3.1.4. □

We now give the pseudocode for the Algorithm that we have described above to solve Problem 5.1.1.

Algorithm 5.2.5.

```
// Generate $C$ with blocks $C_1, \ldots, C_k$ along the diagonal and total variance $S$
Generate $t_1, \ldots, t_k \in \mathbb{R}^k$ with $\|t_i\| = 1$ such that $\|S_1 t_1 + \ldots + S_k t_k\| = S$ (as in
Chapter 3) and set $w_i := S_i t_i$ for all $1 \leq i \leq k$
$M :=$ random in $O(n)$.
for $i := 1$ to $k$ do
    $w_i := M w_i$ // consider $w_i$ as an element of $\mathbb{R}^n$
    Compute $T_i$ (with rows $t_{ij}$) from $C_i$
    $v_i := \sigma_{i1} t_{i1} + \ldots + \sigma_{in_i} t_{in_i}$
    $M_i :=$ random in $O(n)$ such that $M_i v_i = w_i$
    for $j := 1$ to $n_i$ do
        $\tau_{ij} := M_i t_{ij}$
    end for
end for
$T :=$ matrix with rows $\tau_{ij}$
$C := TT^t$
return $C$
```

Remark 5.2.6. There is again a natural distribution on orthogonal matrices $M \in O(n)$ satisfying $Mv = w$ for fixed vectors $v, w \in \mathbb{R}^n$. To see this let $G \subset O(n)$ be the set of orthogonal matrices that fix w, i.e. such that $gw = w$ for all $g \in G$. In the terminology of group theory, G is the stabilizer of w and is therefore a closed subgroup of $O(n)$.

Now let $M_0 \in O(n)$ be some fixed orthogonal matrix such that $M_0 v = w$. Then for every $M \in O(n)$ such that $Mv = w$, the matrix MM_0^{-1} is contained in G. In the terminology of group theory, the set of matrices M we want to draw from is a coset of G in $O(n)$.

Therefore, after finding a single orthogonal matrix M_0 such that $M_0 v = w$, we can draw from the set of all such matrices by drawing an element $g \in G$ and setting $M = gM_0$. However, G is a compact topological group (actually it is isomorphic to $O(n-1)$) so again carries a unique (left) translation invariant probability measure (the Haar measure).

5.3 Implementation

We have implemented the algorithm from this chapter in R. The code can be found in the file `blocksvar.r`. Let us give an example of how to use the code. Suppose we are given random variables $X_1, X_2, \dots, X_9$ with standard deviations $1, 2, \dots, 9$ respectively, and that we are looking for a 9×9 correlation matrix C such that:

1. the correlation matrix between X_1, X_2, X_3 (i.e. the 3×3 top left block of C) is given by

$$C_1 = \begin{pmatrix} 1.00000000 & 0.8000636 & 0.05492573 \\ 0.80006356 & 1.0000000 & -0.55498928 \\ 0.05492573 & -0.5549893 & 1.00000000 \end{pmatrix},$$

2. the correlation matrix between X_4, X_5, X_6 has average correlation 0.15,

3. the standard deviation of $X_7 + X_8 + X_9$ is equal to 8,

4. the standard deviation of $X_1 + \ldots + X_9$ is equal to 10.

First, we input the correlation matrix between X_1, X_2, X_3:

```
C1=matrix(c(1.00000000, 0.8000636, 0.05492573, 0.80006356,
1.0000000, -0.55498928, 0.05492573,-0.5549893,1.00000000),
nrow=3,ncol=3)
```

Second, we generate a 3×3 correlation matrix with average correlation 0.15 using `cor.r`:

```
n=3
rho=0.15
C2=gen_C_from_rho(n,rho)
```

This gives

$$C_2 = \begin{pmatrix} 1.00000000 & 0.01658719 & -0.08584799 \\ 0.01658719 & 1.00000000 & 0.51926080 \\ -0.08584799 & 0.51926080 & 1.00000000 \end{pmatrix}.$$

Third, we generate a 3×3 correlation matrix between X_7, X_8, X_9 such that the standard deviation of $X_7 + X_8 + X_9$ is equal to 8 using `var.r`:

```
n=3
sigmai=c(7,8,9)
S=8
C3=gen_C_from_S(n,sigmai,S)
```

This gives

$$C_3 = \begin{pmatrix} 1.0000000 & -0.8368862 & -0.7882645 \\ -0.8368862 & 1.0000000 & 0.4378652 \\ -0.7882645 & 0.4378652 & 1.0000000 \end{pmatrix}$$

Finally, we generate a 9×9 correlation matrix C between $X_1, \ldots, X_9$, that contains C_1, C_2, C_3 as consecutive blocks along the diagonal such that the sum $X_1 + \ldots + X_9$ has standard deviation 10 using `blocksvar.r`:

```
Ci=list(C1,C2,C3)
sigma1=c(1,2,3)
sigma2=c(4,5,6)
sigma3=c(7,8,9)
sigmaij=list(sigma1,sigma2,sigma3)
S=10
C=gen_C_from_Ci_S(Ci,sigmaij,S)
```

This gives

$$C = \begin{pmatrix} 1.000000 & 0.800063 & 0.054925 & -0.2569282 & -0.584775 & -0.848993 & -0.40177 & 0.6239221 & 0.179799 \\ 0.800063 & 1.000000 & -0.554989 & -0.0609691 & -0.109986 & -0.606159 & -0.54606 & 0.5419313 & 0.604220 \\ 0.054925 & -0.554989 & 1.000000 & -0.2645339 & -0.628666 & -0.161378 & 0.35170 & -0.0353828 & -0.754479 \\ -0.256928 & -0.060969 & -0.264533 & 1.0000000 & 0.016587 & -0.085847 & -0.40774 & 0.0060713 & 0.421747 \\ -0.584775 & -0.109986 & -0.628666 & 0.0165871 & 1.000000 & 0.519260 & 0.16461 & -0.2861101 & 0.141843 \\ -0.848993 & -0.606159 & -0.161378 & -0.0858479 & 0.519260 & 1.000000 & 0.37820 & -0.5714897 & -0.024925 \\ -0.401771 & -0.546060 & 0.351701 & -0.4077487 & 0.164616 & 0.378201 & 1.00000 & -0.8368862 & -0.788264 \\ 0.623922 & 0.541931 & -0.035382 & 0.0060713 & -0.286110 & -0.571489 & -0.83688 & 1.0000000 & 0.437865 \\ 0.179799 & 0.604220 & -0.754479 & 0.4217473 & 0.141843 & -0.024925 & -0.78826 & 0.4378651 & 1.000000 \end{pmatrix}$$

Remark 5.3.1. Note that this example is using just about every algorithm developed in this thesis!

Chapter 6

Conclusion

In this thesis we have developed and implemented new algorithms to generate correlation matrices with certain properties, e.g. such that the average correlation, total variance or some blocks along the diagonal of the correlation matrix are fixed.

When working on a thesis, one has to stop somewhere. However, there are a number of ways in which the work in this thesis could be extended.

First, it would be interesting to apply our algorithms to problems of practical interest. One could think of the problems presented in the introduction, but as correlation matrices appear in many different fields, there are a lot of potential applications.

Second, one could try to extend the algorithms in this thesis by adding constraints. For example, fixing both the average correlation and either some signs of entries or eigenvalues of the correlation matrix.

Appendix A

Code

A.1 cor.r

```
gen_li_forward_uniform <- function(n,rho){

  L <- sqrt(n+rho*(n^2-n))

  li <- 1
  if(2 <= n-2){
    for (i in 2:(n-2)){
      a <- max(L-(n-i),abs(li[i-1]-1))
      b <- min(L+(n-i),li[i-1]+1)
      li[i] <- runif(1,min=a,max=b) # uniform distribution
    }
  }
  a <- max(abs(li[n-2]-1),abs(L-1))
  b <- min(li[n-2]+1,L+1)
  li[n-1] <- runif(1,min=a,max=b)
  li[n] <- L
  return(li)
}


gen_li_backward_uniform <- function(n,rho){

  li <- 0
  for (i in 2:n){
    li[i] <- 0 # initialisation
  }

  L <- sqrt(n+rho*(n^2-n))

  li[n] <- L
  if(2 <= n-1){
    for (i in (n-1):2){
      a <- abs(li[i+1]-1)
      b <- min(li[i+1]+1,i)
      li[i] <- runif(1,min=a,max=b)
    }
  }
  li[1] <- 1
  return(li)
```

```
}


gen_li_forward_gaussian <- function(n,rho){

  L <- sqrt(n+rho*(n^2-n))

  li <- 1
  if(2 <= n-2){
    for (i in 2:(n-2)){
      a <- max(L-(n-i),abs(li[i-1]-1))
      b <- min(L+(n-i),li[i-1]+1)
      ex <- li[i-1]+(L-li[i-1])/(n-i+1)
      if(ex >= a & ex <= b){
        done <- FALSE
        while(done == FALSE){
          r=rnorm(1,ex,min(ex-a,b-ex)/2)
          if(r >= a & r <= b){
            done <- TRUE
          }
        }
    } else{
        done <- FALSE
        while(done == FALSE){
          r=rnorm(1,(a+b)/2,(b-a)/4)
          if(r >= a & r <= b){
            done <- TRUE
          }
        }
      }
      li[i] <- r
    }
  }
  a <- max(abs(li[n-2]-1),abs(L-1))
  b <- min(li[n-2]+1,L+1)
  done <- FALSE
  while(done == FALSE){
    r=rnorm(1,(a+b)/2,(b-a)/4)
    if(r >= a & r <= b){
      done <- TRUE
    }
  }
  li[n-1] <- r
  li[n] <- L
  return(li)
}


gen_li_backward_gaussian <- function(n,rho){

  li <- 0
  for(i in (1:n)){
    li[i] <- 0 # initialisation
  }

  L <- sqrt(n+rho*(n^2-n))

  li[n] <- L

  if(2 <= n-1){
    for(i in ((n-1):2)){
      a <- abs(li[i+1]-1)
      b <- min(li[i+1]+1,i)
      ex <- li[i+1]-(li[i+1]-1)/i
      if(ex >=a & ex <= b){
        done <- FALSE
        while(done == FALSE){
          r=rnorm(1,ex,min(ex-a,b-ex)/2)
          if(r >= a & r <= b){
            done <- TRUE
          }
        }
      }
      else{
        done <- FALSE
        while(done == FALSE){
          r=rnorm(1,(a+b)/2,(b-a)/4)
          if(r >= a & r <= b){
            done <- TRUE
          }
        }
      }
      li[i] <- r
    }
  }

  li[1] <- 1

  return(li)

}


gen_T_from_li <- function(n,li){

  rows <- vector(mode="list", length=n)
  s <- vector(mode="list", length=n)

  x <- rnorm(n)
  x <- x/sqrt(sum(x^2)) # random vector of length 1
  rows[[1]] <- x
  s[[1]] <- x

  for (i in 2:n){
    p <- (li[i]^2-li[i-1]^2-1)/2
    x <- rnorm(n)
    y <- x - s[[i-1]]*sum(x*s[[i-1]])/sum(s[[i-1]]^2) # y is a random vector orthogonal to s[[i-1]]
    z <- s[[i-1]]*p/sum(s[[i-1]]^2)                   # <z,s[[i-1]]> is already what it should be,
                                                      # but z not of length 1
    if(sum(z^2)<=1){
      ti <- z+y*sqrt((1-sum(z^2))/sum(y^2))           # add the right multiple of y to z, so that
                                                      # ti is of length 1
    }
    else{
      ti <- z
    }
    rows[[i]]<-ti
    s[[i]]<-s[[i-1]]+ti
  }

  T <- matrix(,n,n)
  for (i in 1:n){
    for (j in 1:n){
      T[i,j] <- rows[[i]][j] # the matrix T with rows given by 'rows'
    }
  }

  return(T)
}


gen_C_from_T <- function(T){

  C <- T %*% t(T) # correlation matrix C=T*transpose(T)

  return(C)

}


gen_C_from_li <- function(n, li){

  T <- gen_T_from_li(n, li)
  C <- T %*% t(T) # correlation matrix C=T*transpose(T)

  return(C)

}


gen_C_from_rho <- function(n,rho){

  li <- gen_li_forward_gaussian(n,rho)
  C <- gen_C_from_li(n, li)

  return(C)

}


test_cor <- function(n,rho,C){

  S <- 0
  for (i in 1:n){
    for (j in 1:n){
      if(i != j){S <- S + C[i,j]}
    }
  }
  return(S-rho*(n^2-n)) # should be close to zero
}
```

A.2 var.r

```
my_sum <- function(v,i,j){

  s <- 0

  if(i<=j){
    for (k in i:j){
      s <- s + v[k]
    }
  }

  return(s)
}
```

```
gen_li_forward_uniform <- function(n,sigmai,S){

  li <- sigmai[1]
  if(2 <= n-1){
    for (i in 2:(n-1)){
      a <- max(S-my_sum(sigmai,i+1,n),sigmai[n]-my_sum(sigmai,i+1,n-1)-S,abs(li[i-1]-sigmai[i]))
      b <- min(S+my_sum(sigmai,i+1,n),li[i-1]+sigmai[i])
      li[i] <- runif(1,min=a,max=b)
    }
  }
  li[n] <- S
  return(li)
}


gen_li_backward_uniform <- function(n,sigmai,S){

  li <- 0
  for (i in 2:n){
    li[i] <- 0 # initialisation
  }

  li[n] <- S
  if(2 <= n-1){
    for (i in (n-1):2){
      a <- max(sigmai[1]-my_sum(sigmai,2,i),abs(li[i+1]-sigmai[i+1]))
      b <- min(li[i+1]+sigmai[i+1],my_sum(sigmai,1,i))
      li[i] <- runif(1,min=a,max=b)
    }
  }
  li[1] <- sigmai[1]
  return(li)
}


gen_li_forward_gaussian <- function(n,sigmai,S){

  li <- sigmai[1]

  if(2 <= n-1){
    for (i in 2:(n-1)){
      a <- max(S-my_sum(sigmai,i+1,n),sigmai[n]-my_sum(sigmai,i+1,n-1)-S,abs(li[i-1]-sigmai[i]))
      b <- min(S+my_sum(sigmai,i+1,n),li[i-1]+sigmai[i])
      ex <- li[i-1]+sigmai[i]*(S-li[i-1])/(my_sum(sigmai,i,n))
      if(ex >= a & ex <= b){i
        done <- FALSE
        while(done == FALSE){
          r=rnorm(1,ex,min(ex-a,b-ex)/2)
          if(r >= a & r <= b){
            done <- TRUE
          }
        }
      }
      else{
        done <- FALSE
        while(done == FALSE){
          r=rnorm(1,(a+b)/2,(b-a)/4)
          if(r >= a & r <= b){
            done <- TRUE
          }
        }
      }
      li[i] <- r
    }
  }
  li[n] <- S
  return(li)
}


gen_li_backward_gaussian <- function(n,sigmai,S){

  li <- 0
  for(i in (1:n)){
    li[i] <- 0 # initialisation
  }

  li[n] <- S

  if(2 <= n-1){
    for(i in ((n-1):2)){
      a <- max(abs(li[i+1]-sigmai[i+1]),sigmai[1]-my_sum(sigmai,2,i))
      b <- min(li[i+1]+sigmai[i+1],my_sum(sigmai,1,i))
      ex <- li[i+1]-(li[i+1]-sigmai[1])/my_sum(sigmai,1,i)
      if(ex >=a & ex <= b){
        done <- FALSE
        while(done == FALSE){
          r=rnorm(1,ex,min(ex-a,b-ex)/2)
          if(r >= a & r <= b){
            done <- TRUE
          }
        }
      } else{
        done <- FALSE
        while(done == FALSE){
          r=rnorm(1,(a+b)/2,(b-a)/4)
          if(r >= a & r <= b){
            done <- TRUE
          }
        }
      }
      li[i] <- r
    }
  }
  li[1] <- sigmai[1]

  return(li)

}


gen_T_from_li <- function(n,sigmai,li){

  rows <- vector(mode="list", length=n)
  s <- vector(mode="list", length=n)

  x <- rnorm(n)
  x <- x/sqrt(sum(x^2)) # random vector of length 1
  rows[[1]] <- x
  s[[1]] <- sigmai[1]*x
  for (i in 2:n){
    p <- (li[i]^2-li[i-1]^2-sigmai[i]^2)/(2*sigmai[i])
    x <- rnorm(n)
    y <- x - s[[i-1]]*sum(x*s[[i-1]])/sum(s[[i-1]]^2) # y is a random vector orthogonal to s[[i-1]]
    z <- s[[i-1]]*p/sum(s[[i-1]]^2)                   # <z,s[[i-1]]> is already what it should be,
                                                      # but z not of length 1
    if(sum(z^2)<=1){
      ti <- z+y*sqrt((1-sum(z^2))/sum(y^2))           # add the right multiple of y to z, so that
                                                      # ti is of length 1
    } else{
      ti <- z
    }
    rows[[i]]<-ti
    s[[i]]<-s[[i-1]]+sigmai[i]*ti
  }

  T <- matrix(,n,n)
  for (i in 1:n){
    for (j in 1:n){
      T[i,j] <- rows[[i]][j] # the matrix T with rows given by 'rows'
    }
  }

  return(T)
}


gen_C_from_T <- function(T){


  C <- T %*% t(T) # correlation matrix C=T*transpose(T)

  return(C)
}


gen_C_from_li <- function(n,sigmai,li){

  T <- gen_T_from_li(n,sigmai,li)
  C <- T %*% t(T) # correlation matrix C=T*transpose(T)

  return(C)

}


gen_C_from_S <- function(n,sigmai,S){

  li <- gen_li_forward_gaussian(n,sigmai,S)
  C <- gen_C_from_li(n,sigmai,li)

  return(C)

}


test_cor <- function(n,sigmai,C,S){

  s <- 0
  for (i in 1:n){
    for (j in 1:n){
      s <- s+C[i,j]*sigmai[i]*sigmai[j]
    }
  }
  return(S-abs(s)^(1/2)) # should be close to zero
}
```

A.3 blocks.r

```
random_On <- function(n){

  A=matrix(,n,n)
  for(i in 1:n){
    for(j in 1:n){
      A[i,j]=rnorm(1)
    }
  }

  qrdata=qr(A)
  Q=qr.Q(qrdata)
  R=qr.R(qrdata)

  for(i in 1:n){
    if(R[i,i] < 1){
      for(j in 1:n){
        Q[j,i]=-Q[j,i]
      }
    }
  }

  return(Q)

}


gen_C_from_Ti <- function(Ti){

  k=length(Ti)
  n <- 0
  for(i in 1:k){
    n = n + nrow(Ti[[i]])
  }

  T<-matrix(,n,n)
  cnt<-0

  for(i in 1:k){
    Mi <- random_On(n)
    Tii <- Ti[[i]]
    ni <- nrow(Tii)
    for(j in 1:ni){
      cnt=cnt+1
      tij=Tii[j,]
      for(l in (ni+1):n){
        tij[l]=0
      }
      for(l in 1:n){
        T[cnt,l]=(Mi%*%tij)[l]
      }

    }
  }
  C <- T%*%t(T)

  return(C)

}


gen_C_from_Ci <- function(Ci){

  k=length(Ci)
  Ti=Ci

  for(i in 1:k){
    Ti[[i]] = t(chol(Ci[[i]]))
  }

  C = gen_C_from_Ti(Ti)

  return(C)
}
```

A.4 blocksvar.r

```
library(far)


my_sum <- function(v,i,j){

  s <- 0

  if(i<=j){
    for (k in i:j){
      s <- s + v[k]
    }
  }

  return(s)
}


gen_li_forward_gaussian <- function(n,sigmai,S){


  li <- sigmai[1]
  if(2 <= n-1){
    for (i in 2:(n-1)){
      a <- max(S-my_sum(sigmai,i+1,n),sigmai[n]-my_sum(sigmai,i+1,n-1)-S,abs(li[i-1]-sigmai[i]))
      b <- min(S+my_sum(sigmai,i+1,n),li[i-1]+sigmai[i])
      ex <- li[i-1]+sigmai[i]*(S-li[i - 1])/(my_sum(sigmai,i,n))
      if(ex >= a & ex <= b){
        done <- FALSE
        while(done == FALSE){
          r=rnorm(1,ex,min(ex-a,b-ex)/2)
          if(r >= a & r <= b){
            done <- TRUE
          }
        }
      }
      else{
        done <- FALSE
        while(done == FALSE){
          r=rnorm(1,(a+b)/2,(b-a)/4)
          if(r >= a & r <= b){
            done <- TRUE
          }
        }
      }
      li[i] <- r
    }
  }
  li[n] <- S
  return(li)
}


gen_T_from_li <- function(n,sigmai,li){

  rows <- vector(mode="list", length=n)
  s <- vector(mode="list", length=n)

  x <- rnorm(n)
  x <- x/sqrt(sum(x^2)) # random vector of length 1
  rows[[1]] <- x
  s[[1]] <- sigmai[1]*x


  for (i in 2:n){
    p <- (li[i]^2-li[i-1]^2-sigmai[i]^2)/(2*sigmai[i])
    x <- rnorm(n)
    y <- x - s[[i-1]]*sum(x*s[[i-1]])/sum(s[[i-1]]^2) # y is a random vector orthogonal to s[[i-1]]
    z <- s[[i-1]]*p/sum(s[[i-1]]^2)                   # <z,s[[i-1]]> is already what it should be,
                                                      # but z not of length 1
    if(sum(z^2)<=1){
      ti <- z+y*sqrt((1-sum(z^2))/sum(y^2))           # add the right multiple of y to z, so that
                                                      # ti is of length 1
    }
    else{
      ti <- z
    }
    rows[[i]]<-ti
    s[[i]]<-s[[i-1]]+sigmai[i]*ti
  }

  T <- matrix(,n,n)
  for (i in 1:n){
    for (j in 1:n){
      T[i,j] <- rows[[i]][j] # the matrix T with rows given by 'rows'
    }
  }

  return(T)
}


random_On <- function(n){

  A=matrix(,n,n)
  for(i in 1:n){
    for(j in 1:n){
      A[i,j]=rnorm(1)
    }
  }
  qrdata=qr(A)
  Q=qr.Q(qrdata)
  R=qr.R(qrdata)

  for(i in 1:n){
    if(R[i,i] < 1){
      for(j in 1:n){
        Q[j,i]=-Q[j,i]
      }
    }
  }

  return(Q)

}


random_On_v_to_w <- function(n,v,w){

  V = orthonormalization(v) # Gramm-Schmidt orthogonalization from package "far"
  W = orthonormalization(w)

  g = random_On(n-1)

  M = matrix(0,nrow=n,ncol=n) # initialisation
  M[1,1] = 1
  for(i in 2:n){
    for(j in 2:n){
      M[i,j] = g[i-1,j-1]
    }
  }

  return(W %*% M %*% solve(V)) # solve is matrix inverse, confusing R syntax!

}


gen_w_from_Si_S <-function(k,Si,S){

  li = gen_li_forward_gaussian(k,Si,S)


  W = gen_T_from_li(k,Si,li)

  for(i in 1:k){
    for(j in 1:k){
      W[i,j]=Si[i]*W[i,j]
    }
  }

  return(W)
}


gen_C_from_Ti_S<-function(Ti,sigmaij,S){

  k=length(Ti)
  n <- 0
  for(i in 1:k){
    n = n + nrow(Ti[[i]])
  }
  V=matrix(0,nrow=k,ncol=n)
  Si=0
  for(i in 1:k){
    vec=0
    for(j in 1:nrow(Ti[[i]])){
      vec=vec+sigmaij[[i]][j]*Ti[[i]][j,]
    }
    for(j in 1:nrow(Ti[[i]])){
      V[i,j]=vec[j]
    }
    Si[[i]]=sqrt(sum(vec^2))
  }

  W = gen_w_from_Si_S(k,Si,S)

  Wnew=matrix(0,nrow=k,ncol=n)
  for(i in 1:k){
    for(j in 1:k){
      Wnew[i,j] = W[i,j]
    }
  }

  M = random_On(n)
  W = Wnew %*% M

  T = matrix(0,nrow=n,ncol=n)
  ct=0
  for(i in 1:k){
    Mi=random_On_v_to_w(n,V[i,],W[i,])
    for(j in 1:nrow(Ti[[i]])){
      ct = ct+1
      tij=matrix(0,nrow=n,ncol=1)
      for(l in 1:nrow(Ti[[i]])){
        tij[l,1]=Ti[[i]][j,l]
      }
      T[ct,]=Mi%*%tij
    }
  }

  C = T%*%t(T)

  return(C)

}


gen_C_from_Ci_S<-function(Ci,sigmaij,S){

  k = length(Ci)
  Ti = Ci

  for(i in 1:k){
    Ti[[i]] = t(chol(Ci[[i]]))
  }

  C = gen_C_from_Ti_S(Ti,sigmaij,S)

  return(C)

}
```

```
test_cor <- function(n,sigmai,C,S){

  s <- 0
  for (i in 1:n){
    for (j in 1:n){
      s <- s+C[i,j]*sigmai[i]*sigmai[j]
    }
  }
  return(S-abs(s)^(1/2)) # should be close to zero
}
```

Bibliography

[1] The Solvency II Directive (2009/138/EC). `http://ec.europa.eu/finance/insurance/solvency/solvency2/`.

[2] C.P. Chalmers. Generation of correlation matrices with a given eigenstructure. *Journal of Statistical Computation and Simulation*, 4(2), 1975.

[3] Philip I. Davies and Nicholas J. Higham. Numerically stable generation of correlation matrices and their factors. *BIT*, 40(4):640–651, 2000.

[4] J. E. Gentle. *Numerical Linear Algebra for Applications in Statistics*, pages 93–95. Berlin: Springer-Verlag, 1998.

[5] Johanna Hardin, Stephan Ramon Garcia, and David Golan. A method for generating realistic correlation matrices. *Ann. Appl. Stat.*, 7(3):1733–1762, 2013.

[6] H. Markowitz. Portfolio selection. *The Journal of Finance*, 7, 1952.

[7] George Marsaglia and Ingram Olkin. Generating correlation matrices. *SIAM J. Sci. Statist. Comput.*, 5(2):470–475, 1984.

[8] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014.

[9] R. Rebonato and P. Jaeckel. The most general methodology to create a valid correlation matrix for risk management and option pricing purposes. `http://www.quarchome.org/correlationmatrix.pdf`, 1999.

[10] G. W. Stewart. The efficient generation of random orthogonal matrices with an application to condition estimators. *SIAM J. Numer. Anal.*, 17(3):403–409 (loose microfiche suppl.), 1980.