A Newton Algorithm for the Nearest Correlation Matrix

Borsdorf, Rudiger

2007

MIMS EPrint: **2008.49**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: http://eprints.maths.manchester.ac.uk/
And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097

A NEWTON ALGORITHM FOR THE NEAREST CORRELATION MATRIX

A dissertation submitted to the University of Manchester
for the degree of Master of Science
in the Faculty of Engineering and Physical Sciences

2007

Ruediger Borsdorf

School of Mathematics

Contents

Word count 28239

List of Tables

List of Figures

The University of Manchester

Ruediger Borsdorf
Master of Science
A Newton Algorithm for the Nearest Correlation Matrix
September 5, 2007

Firstly, we describe and investigate the algorithm of Qi and Sun which solves the problem of finding the nearest correlation matrix to a symmetric matrix. This algorithm claims a quadratic convergence. We discuss improving this algorithm's efficiency and reliability and detect a problem when we are aiming at a nearest correlation matrix with a high accuracy, using small error tolerences. As a consequence, we suggest a modified version, based on the algorithm of Qi and Sun, which is also a quadratically convergent algorithm, has improved efficiency and is modified so that the algorithm can return the nearest correlation matrix to high accuracy showing a robust and reliable behaviour.

Secondly, we investigate the general alternating projections method and also Higham's alternating projections method for the nearest correlation matrix. We discuss variations of the latter and include a further projection which allows more constraints to be added to the problem. We introduce a new algorithm and compare its convergence behaviour with Higham's alternating projections method.

Declaration

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright Statement

Acknowledgements

I would like to thank my supervisor Prof. Nick Higham, for his support, useful comments and guidance of this thesis. Thanks also go to Prof. Marcos Raydán for helpful discussions on global convergence strategies and Prof. David Silvester for his explanations of the MINRES routine and his implementation of MINRES. I thank Dr David Sayers for his comments and also NAG for their support and providing access to the MATLAB-NAG Toolbox. I would like to thank my friends Chris Munro, Chi-Wai Chiu and Kyriakos Parides for their companionship and enriching my stay here in Manchester. Finally, I thank my family for their help in every respect.

Chapter 1

Introduction

Our thesis concerns the problem of finding the nearest correlation matrix to a given symmetric matrix. Before focusing our attention on solving this problem, we clarify what it covers and for which applications it is used. Basically, the keywords in our problem are the words 'nearest' and 'correlation matrix'. The first one means we look for the smallest distance measured in the Frobenius norm (see Section 1.14) between an input matrix and a matrix which has all the properties of a correlation matrix. We now explain the meaning of the second keyword.

1.1 Correlation Matrices

A correlation matrix is a symmetric positive semidefinite matrix with ones on the diagonal. The term 'correlation matrix' comes from statistics, describing a matrix in which the (i,j) entry indicates the correlation between two random variables ξ_i and ξ_j. This explains immediately why correlation matrices have ones on the diagonal and are symmetric. The reason is that the correlation between a random variable and itself is always 1 and the correlation between ξ_i and ξ_j is the same as between ξ_j and ξ_i. We also deduce the semidefiniteness from the original meaning of a correlation matrix. Therefore, let $A = (a_{ij})_{i,j=1}^n$ be a 'statistical' correlation matrix corresponding to a

random vector $\xi = (\xi_1, \ldots, \xi_n)$ and let $x \in \mathbb{R}^n$ be arbitrary. Then we have

$$\begin{aligned} x^T A x &= \sum_{i,j=1}^{n} x_i a_{ij} x_j \\ &= \sum_{i,j=1}^{n} x_i \frac{E\left((\xi_i - \mu_i)(\xi_j - \mu_j)\right)}{\sqrt{\nu_i \nu_j}} x_j \\ &= E\left(\sum_{i=1}^{n} \frac{x_i(\xi_i - \mu_i)}{\sqrt{\nu_i}} \sum_{j=1}^{n} \frac{x_j(\xi_j - \mu_j)}{\sqrt{\nu_j}} \right) \\ &= E\left(\left(\sum_{i=1}^{n} \frac{x_i(\xi_i - \mu_i)}{\sqrt{\nu_i}} \right)^2 \right) \\ &\geq 0 \end{aligned} \tag{1.1}$$

where $E(X)$ denotes the mean value for a random variable X, ν_i is the variance of ξ_i and $\mu_i := E(\xi_i)$. Since x was arbitrary we obtain that the matrix A is positive semidefinite.

Now conversely, we prove that every correlation matrix is a 'statistical' correlation matrix. Therefore, let A be symmetric and positive semidefinite matrix with ones on the diagonal. Since A is symmetric, there exists a factorization of A with

$$A = P \Lambda P^T = R R^T, \tag{1.2}$$

where P is orthogonal, Λ is a diagonal matrix with the eigenvalues $(\lambda_1, \ldots, \lambda_n)$ of A on the diagonal and

$$R := P \begin{bmatrix} \sqrt{\lambda_1} & & \\ & \ddots & \\ & & \sqrt{\lambda_n} \end{bmatrix}. \tag{1.3}$$

Now let $\eta = (\eta_1, \ldots, \eta_n)$ be a random vector with $\eta_1, \ldots, \eta_n$ linearly independent and η_i normally distributed with a mean value of 0 and a variance of 1. Then we can define a random vector ξ as $\xi := R\eta$ where this vector is the random vector which

corresponds to the matrix A since

$$\begin{aligned} E(\xi\xi^T) &= E((R\eta)(R\eta)^T) \\ &= E(R\eta\eta^T R^T) \\ &= RE(\eta\eta^T)R^T \\ &= RI_nR^T \\ &= A \end{aligned} \tag{1.4}$$

with I_n the identity in $\mathbb{R}^{n\times n}$. It follows that A is a 'statistical' correlation matrix.

1.2 Applications

The problem (1.14) arises for example in the financial world, where for prediction purposes the correlations between pairs of stocks are measured. Unfortunately, in practice only an approximate correlation matrix is obtained, since not all values of all stocks of interest can be recorded at the same time over a long period. This obtained matrix does not necessarily satisfy all the properties of a correlation matrix. Thus, the nearest matrix satisfying these properties is desired. More details, in particular how such approximate correlation matrices are computed, can be found in [22].

If we would like to change some entries of a correlation matrix computed historically for example, then it will also be likely that a solution of our problem is required, because the modified matrix will not generally have the properties of a correlation matrix. Reasons for changing the matrix are that perhaps a user has more insight which tells him that a particular correlation may not be up to date, or stress testing purposes, where a user wishes to examine the effect of an extreme market on a portfolio.

Note that using matrices which do not satisfy the properties of a correlation matrix can lead to a breakdown of a Value-at-Risk methodology if these matrices are used to calculate the Value-at-Risk of a portfolio; see [8].

Further applications occur in robust quadratic optimization and numerical linear algebra involving preconditioning of linear systems and error analysis of Jacobi

methods for the symmetric eigenvalue problem; see [5] for more details.

1.3 Problem Formulation

1.3.1 Some Definitions and a Lemma

We want to express the problem of finding the nearest correlation matrix, a convex optimization problem, mathematically. Therefore, we need to define a space where we can measure distances. For that purpose, we start with the definition of an inner product and continue with the definition of a Hilbert space. Then we define our particular Hilbert space equipped with an inner product giving rise to a norm. This norm will be the Frobenius norm and the norm which we will use to measure distances. Since the Cauchy-Schwarz inequality is required later and linked with the definition of an inner product, we also state here a lemma which proves this inequality for all elements in a real Hilbert space.

Finally, we define what convex means and what a convex optimization problem is. In this context, convex refers to functions or sets. We provide both definitions. Beyond that, we also define the terms "cone" and "linear" since these definitions are also used later.

Definition 1.3.1. Let $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$ be a field and V a $\mathbb{K}$-vector space. A mapping $\langle \cdot, \cdot \rangle$: $V \times V \mapsto \mathbb{K}$ is called an *inner product*, if

- $\langle \cdot, w \rangle : V \mapsto \mathbb{K}$ is linear (see Definition (1.3.8)) $\forall w \in V$
- $\langle v, w \rangle = \overline{\langle w, v \rangle} \quad \forall v, w \in V$
- $\langle v, v \rangle \geq 0$ and $\langle v, v \rangle = 0 \Leftrightarrow v = 0 \quad \forall v \in V$

Now we have defined the inner product and we can give the definition of a Hilbert space.

Definition 1.3.2. A space H is a *Hilbert space* if H is a real or complex vector space equipped with an inner product where this vector space is also complete and normed

under the norm defined by the inner product:

$$\|\cdot\| := \sqrt{\langle \cdot, \cdot \rangle}. \tag{1.5}$$

Now we define an operator $\langle \cdot, \cdot \rangle : \mathbb{R}^{n\times n} \mapsto \mathbb{R}^{n\times n}$ with

$$\langle X, Y \rangle := \operatorname{trace}\left(X^T Y\right) = \sum_{i,j=1}^{n} X_{ij} Y_{ij}, \quad \forall X, Y \in \mathbb{R}^{n\times n}. \tag{1.6}$$

The operator $\langle \cdot, \cdot \rangle$ clearly satisfies all conditions of Definition 1.3.1 with $\mathbb{R}$ as field and $\mathbb{R}^{n\times n}$ as vector space so that we can now specify our particular Hilbert space as $\mathbb{R}^{n\times n}$ equipped with this inner product $\langle \cdot, \cdot \rangle$. Note that this Hilbert space will be our general framework for our problem later. Our associated norm is then defined as

$$\|X\|_F := \sqrt{\langle X, X \rangle}, \quad \forall X \in \mathbb{R}^{n\times n}$$

which defines the Frobenius norm and is thus our measurement of distance in our Hilbert space $\mathbb{R}^{n\times n}$.

Next, we state a lemma which proves the Cauchy-Schwarz inequality

$$|\langle v, w \rangle| \leq \|v\| \, \|w\| \tag{1.7}$$

for all v, w in a real Hilbert space.

Lemma 1.3.3. *(see [6, Theorem 1.2 (Schwarz Inequality)]) Let H be a real Hilbert space (need not to be complete) and $\langle \cdot, \cdot \rangle$ the corresponding inner product with $\mathbb{R}$ as field. Furthermore, let $\|\cdot\| := \sqrt{\langle \cdot, \cdot \rangle}$ be the associated norm. Then for any $v, w \in H$,*

$$|\langle v, w \rangle| \leq \|v\| \, \|w\|. \tag{1.8}$$

Moreover, equality holds in (1.8) if and only if v and w are linearly dependent (there exists a $\lambda \in \mathbb{R}$ so that $v = \lambda y$).

Proof. "⇒":

Let v and w be linearly dependent, then there exists a scalar λ with $v = \lambda w$. It immediately follows that both sides of (1.8) are equal to $|\lambda| \, \|w\|^2$.

"⇐":

Now conversely, let v and w be linearly independent, then $v - \lambda w \neq 0$ for every scalar λ, therefore using the third property in Definition 1.3.1 yields

$$0 < \langle v - \lambda w, v - \lambda w \rangle = \|v\|^2 - 2\lambda \langle v, w \rangle + \lambda^2 \|w\|^2. \tag{1.9}$$

Setting $\lambda := \frac{\langle v,w \rangle}{\|w\|^2}$ leads to

$$0 < \|v\|^2 - \frac{|\langle v, w \rangle|^2}{\|w\|^2} \tag{1.10}$$

which implies strict inequality in (1.8) and we obtain our assertion. □

We now give the definition of convex functions and subsequently the definition of convex sets.

Definition 1.3.4. A function $f : I \mapsto \mathbb{R}$ is called *convex* if $\forall x, y \in I$ and $\forall t \in (0, 1)$:

$$f(tx + (1 - t)y) \leq tf(x) + (1 - t)f(y) \tag{1.11}$$

where I is an arbitrary real interval.

Definition 1.3.5. A set C in a complex or real vector space is called *convex* if for all $x, y \in C$ the line segment connecting x and y is also in C, i.e.

$$\forall x, y \in C : x + t(y - x) \in C \quad \forall t \in [0, 1]. \tag{1.12}$$

Now we can define a convex optimization problem.

Definition 1.3.6. Let $\mathcal{E}$ and $\mathcal{I}$ be arbitrary index sets and Ω an arbitrary set. Consider the following problem.

$$\begin{aligned} &\min f(x) \\ \text{s.t.} \quad &x \in \Omega \\ &h_i(x) = c_i, \quad \forall i \in \mathcal{E} \\ &g_j(x) \leq 0, \quad \forall j \in \mathcal{I} \end{aligned} \tag{1.13}$$

where $f : \Omega \mapsto \mathbb{R}$ is the objective function, $h_i : \Omega \mapsto \mathbb{R}$ are the equality constraints with c_i a constant $\forall i \in \mathcal{E}$ and $g_j : \Omega \mapsto \mathbb{R}$ are the inequality constraints $\forall j \in \mathcal{I}$. The problem (1.13) is called a *convex optimization problem* if Ω is a convex set, f, g_j convex functions and h_i linear $\forall i \in \mathcal{E}$ and $\forall j \in \mathcal{I}$ [24].

Finally, we provide the two remaining definitions.

Definition 1.3.7. A subset C of a real vector space is a *cone* if

$$\lambda x \in C, \quad \forall x \in C \text{ and } \forall \lambda \geq 0.$$

Definition 1.3.8. Let $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$ be a field and V a $\mathbb{K}$-vector space. A function $f : V \mapsto \mathbb{K}$ is called *linear* if

- $f(x + y) = f(x) + f(y), \quad \forall x, y \in V$ and
- $f(\lambda x) = \lambda f(x), \quad \forall \lambda \in \mathbb{K}$ and $\forall x \in V$.

1.3.2 The Problem

We express our problem mathematically as the convex optimization problem

$$\begin{aligned} &\min \frac{1}{2}\|G - X\|_F^2, \quad G \in \mathcal{S}^n \text{ given} \\ &\text{s.t.} \quad \operatorname{diag}(X) = e \quad \text{and} \quad X \in \mathcal{S}_+^n \end{aligned} \tag{1.14}$$

where $\mathcal{S}^n$ is the set of symmetric matrices in our underlying Hilbert space $\mathbb{R}^{n\times n}$, $\mathcal{S}_+^n \subset \mathcal{S}^n$ denotes the set of symmetric positive semidefinite matrices, $\operatorname{diag}(X)$ is a linear operator which produces a vector of the diagonal elements of X and e is a vector of all ones. The constraints of this problem guarantee that our matrix X is a correlation matrix.

1.3.3 Why It Is a Convex Optimization Problem

Here, we outline why this problem is a convex optimization problem. Consider $\Omega := \mathcal{S}_+^n$ in Section 1.3.1. Ω is convex since the set of symmetric positive semidefinite matrices is a closed convex cone. Now we look at our objective function. We prove that it is convex. Let $X, Y \in \mathcal{S}_+^n$ be arbitrary. Then

$$\begin{aligned} \frac{1}{2}\|G - (tX + (1-t)Y)\|_F^2 &= \frac{1}{2}\|tG - tX + (1-t)(G-Y)\|_F^2 \\ &\leq \frac{1}{2}(t\|G - X\|_F + (1-t)\|G - Y\|_F)^2 \quad \forall t \in (0,1). \end{aligned} \tag{1.15}$$

Since the function x^2 is convex on the positive real axis, we have that

$$\begin{aligned}\frac{1}{2}(t\|G-X\|_F+(1-t)\|G-Y\|_F)^2 &\leq \frac{1}{2}(t\|G-X\|_F^2+(1-t)\|G-Y\|_F^2)\\ &= t\frac{1}{2}\|G-X\|_F^2+(1-t)\frac{1}{2}\|G-Y\|_F^2 \quad \forall t\in(0,1).\end{aligned} \tag{1.16}$$

Hence our objective function is convex. Now consider our linear operator $\mathrm{diag}(X)$ as $\mathrm{diag}(X)=[h_1(X),\ldots,h_n(X)]^T$ with $h_i:\mathcal{S}_+^n\mapsto\mathbb{R}$ and $h_i(X)=X_{ii}$. Then we can rewrite our constraint as

$$h_i(X)=1, \quad \forall i=1,\ldots,n.$$

Since $h_i(X)$ is linear (recall $\mathrm{diag}(X)$ is linear) and $c_i:=1$ is constant, we have a convex optimization problem.

1.4 Some Properties of the Problem

In this section, we show that a solution of our problem exists and is unique. We start with the existence. Our objective function is continuous, the feasible set $\mathcal{S}_+^n\cap\{X\in\mathcal{S}^n:\mathrm{diag}(X)=e\}$ is closed and convex (see [17]) and the feasible set is not empty (consider X equal to the identity). Moreover, since we can intersect our feasible set with a closed bounded set and obtain an equivalent problem, a solution exists.

Such a bounded set is for example $\mathcal{B}:=\{X\in\mathcal{S}^n:\|X\|_F\leq 2\|G\|_F+\sqrt{n}\}$, as we now prove. Therefore, we denote the original problem as Problem 1 and the problem where we intersect our feasible set with $\mathcal{B}$ by Problem 2. Let U be a solution of Problem 1. Since we have

$$\begin{aligned}\|U\|_F-\|G\|_F &\leq \|G-U\|_F\\ &\leq \|G-I_n\|_F\\ &\leq \|G\|_F+\sqrt{n}.\end{aligned} \tag{1.17}$$

with I_n the identity in $\mathbb{R}^{n\times n}$, it follows by adding $\|G\|_F$ on both sides that $U\in\mathcal{B}$ and hence U is also a solution of Problem 2. We obtain one direction of the proof. Now conversely, let U be a solution of Problem 2 and assume that U is not a solution

of Problem 1, i.e. there exists a matrix $\hat{U}$ in the feasible set of Problem 1 with $\|G - \hat{U}\|_F < \|G - U\|_F$ and $\hat{U} \notin \mathcal{B}$. From (1.17) and the fact that $\|\hat{U}\|_F - \|G\|_F \leq \|G - \hat{U}\|_F \leq \|G - U\|_F$ it follows that $\|\hat{U}\|_F \leq 2\|G\|_F + \sqrt{n}$. However, this is a contradiction to $\hat{U} \notin \mathcal{B}$ and we obtain our assertion.

Now we consider the uniqueness. Therefore, we provide first a lemma which will be helpful to show the uniqueness.

Lemma 1.4.1. *If $\|X + Y\|_F = \|X\|_F + \|Y\|_F$ for $X, Y \in \mathcal{S}_+^n$ and $X, Y \neq 0$ then it holds that $Y = \lambda X$ for a $\lambda > 0$.*

Proof. Let $X, Y \in \mathcal{S}_+^n$ with $X, Y \neq 0$ and $\|X + Y\|_F = \|X\|_F + \|Y\|_F$. Then from

$$\|X + Y\|_F^2 = \langle X + Y, X + Y \rangle = \|X\|_F^2 + 2\langle X, Y \rangle + \|Y\|_F^2, \tag{1.18}$$

and

$$(\|X\|_F + \|Y\|_F)^2 = \|X\|_F^2 + 2\|X\|_F\|Y\|_F + \|Y\|_F^2, \tag{1.19}$$

we obtain that

$$\langle X, Y \rangle = \|X\|_F \|Y\|_F. \tag{1.20}$$

Since the equality holds in the Cauchy-Schwarz inequality ($\langle X, Y \rangle \leq \|X\|_F \|Y\|_F$) it follows by Lemma 1.3.3 that $Y = \lambda X$ and thus, it only remains to show that $\lambda > 0$. Obviously,

$$\begin{aligned} |1 + \lambda| \, \|X\|_F &= \|X + \lambda X\|_F \\ &= \|X + Y\|_F \\ &= \|X\|_F + \|Y\|_F \\ &= \|X\|_F + \|\lambda X\|_F \\ &= (1 + |\lambda|)\|X\|_F \end{aligned} \tag{1.21}$$

which implies that $\lambda > 0$ and hence, we obtain our assertion. □

We move back to showing the uniqueness, we know from Section 1.3.3 that our objective function is convex and hence that every local minimum is the global minimum [24, Theorem 2.5]. However, we still have to show that the minimum is attained

only once. We assume that we have two matrices X, Y which minimize our objective function with $X, Y \neq G$ (if $X = G$ or $Y = G$ then obviously G is our unique solution), $X \neq Y$ and X, Y in the feasible set. We show that $X = Y$. Since we have a convex optimization problem $\frac{1}{2}(X + Y)$ also minimizes the objective function and lies in the feasible set. Hence, we deduce that

$$\frac{1}{2}\|G - X + G - Y\|_F = \|G - \frac{1}{2}(X + Y)\|_F = \frac{1}{2}\|G - X\|_F + \frac{1}{2}\|G - Y\|_F \quad (1.22)$$

and by Lemma 1.4.1 it follows that $G - X = \lambda(G - Y)$ for a $\lambda > 0$ since $G - X \neq 0$ and $G - Y \neq 0$. From

$$\|G - X\|_F = \|G - Y\|_F = \|\lambda(G - X)\|_F = |\lambda| \, \|G - X\|_F \quad (1.23)$$

and the fact that $\|G - X\|_F \neq 0$ we obtain that $\lambda = 1$ and subsequently that $X = Y$. This implies our uniqueness, the minimum of (1.14) is achieved and it is achieved at a unique matrix X.

1.5 Approaches

In the financial world, many different methods are applied to solve our problem. For example one approach is to solve the problem by using an angles parametrization of a correlation matrix. That is,

$$\begin{aligned} &\min \left\|G - B(\Theta)B(\Theta)^T\right\|_F^2 \\ &\text{with } \Theta \in \mathbb{R}^{n \times (n-1)} \end{aligned} \quad (1.24)$$

and $B \in \mathbb{R}^{n \times n}$ defined as

$$B(\Theta)_{ij} := \begin{cases} \cos(\Theta_{ij}) \prod_{k=1}^{j-1} \sin(\Theta_{ik}) & \text{if } j < n \\ \prod_{k=1}^{j-1} \sin(\Theta_{ik}) & \text{if } j = n \end{cases} \quad (1.25)$$

where $B(\Theta)B(\Theta)^T$ is our correlation matrix which is by construction always positive semidefinite and the latter condition (1.25) guarantees that this matrix has ones on its diagonal. However, whether *all* correlation matrices can be represented in this way is unclear. Unfortunately, the authors in [29] leave this as an open question.

To carry on listing approaches, Finger [8] proposes an algorithm to solve our problem that we summarize now. Let $C \in \mathbb{R}^{m\times m}$ be a given correlation matrix with the block structure

$$C := \begin{bmatrix} C_{11} & C_{12} \\ C_{12}^T & C_{22} \end{bmatrix} \tag{1.26}$$

with $C_{11} \in \mathbb{R}^{k\times k}$, $C_{12} \in \mathbb{R}^{k\times(m-k)}$ and $C_{22} \in \mathbb{R}^{(m-k)\times(m-k)}$. Furthermore, let C_{11} contain all correlations which are desired to be changed and let $\nu \in [0,1]$ be a given scalar. Then the new matrix C_F obtained by the Finger algorithm is

$$C_F := \begin{pmatrix} \Pi C_{11}\Pi^T & \Pi C_{12} \\ C_{12}^T\Pi^T & C_{22} \end{pmatrix} = \begin{pmatrix} \Pi & 0 \\ 0 & I \end{pmatrix} \cdot \begin{pmatrix} C_{11} & C_{12} \\ C_{12}^T & C_{22} \end{pmatrix} \cdot \begin{pmatrix} \Pi & 0 \\ 0 & I \end{pmatrix}^T \tag{1.27}$$

where $\Pi := \Gamma A$ with $\Gamma = \mathrm{diag}(\delta_{11}^{-1/2}, \ldots, \delta_{kk}^{-1/2})$ and δ_{ii} the (i,i) element of $AC_{11}A^T$ and $A = (a_{ij})_{i,j=1}^k$ with

$$a_{ij} := \begin{cases} 1-\nu+\frac{\nu}{k} & \text{if } \ j = i \\ \frac{\nu}{k} & otherwise \end{cases}. \tag{1.28}$$

The right-hand side of (1.27) clarifies why the matrix C_F preserves the symmetry and positive semidefiniteness of C and since we rescale the diagonal of the matrix $AC_{11}A^T$ to 1 by multiplying Γ from both sides, we guarantee all conditions of a correlation matrix. Hence, our new matrix C_F is also a correlation matrix. The matrix A is chosen in this way since applying the matrix A to C as shown in (1.27) is equivalent to modifying the random variables corresponding to the entries of C_{11} by taking a convex combination of the original random variable and the average of all other random variables corresponding to C_{11} (see therefore [8]). That is,

$$\hat{\xi}_i := (1-\nu)\xi_i + \nu\frac{1}{k}\sum_{j=1}^{k}\xi_j \quad \forall i = 1, \ldots, k, \tag{1.29}$$

where $\xi_1, \ldots, \xi_k$ are the random variables corresponding to C_{11} and $\hat{\xi}_i$ are the changed random variables. Hence, the value of ν is operative and has to be computed in advance.

In [21] an improved version of Finger's algorithm is proposed where the modification of the blocks C_{12} and C_{21} is reduced, a new correlation matrix

$$C^* := \begin{pmatrix} C_{11}^* & C_{12}^* \\ {C_{12}^*}^T & C_{22}^* \end{pmatrix} \tag{1.30}$$

is computed with $C_{11}^* = \Pi C_{11} \Pi^T, C_{22}^* = C_{22}$, $C_{12}^* = \Pi C_{12} + B_{12}$ where $B_{12} \in \mathbb{R}^{k \times (m-k)}$ is chosen to minimize $\|C_{12}^* - C_{12}\|_F$ subject to C^* positive semidefinite. In other words, we look for a matrix B with

$$B := \begin{pmatrix} 0 & B_{12} \\ B_{12}^T & 0 \end{pmatrix} \tag{1.31}$$

and $C^* = C_F + B$ such that the matrix C^* is a correlation matrix having a top right block C_{12}^* which is nearest to C_{12}.

Another approach is also to project the original matrix onto the set of positive semidefinite matrices and to rescale the diagonal elements to 1 by multiplying a diagonal matrix from both sides afterwards [29].

Unfortunately, all these methods have the problem that they are either too inaccurate or inapplicable for a large dimension of the matrix G. Obviously, for this reason our problem has recently been investigated quite frequently and other approaches have been proposed. However, the approach to solve the problem by means of semidefinite programming techniques, for example to reformulate and solve it by using the interior point method, has so far been unsuccessful. Higham [17, Section 3.3] outlined some reasons. He introduced some approaches using such techniques and showed that they all suffer, mainly from either requiring too many variables or involving too many constraints and, hence, that solving our problem with such techniques becomes prohibitively expensive, especially for n large. An approach that may be competitive is described in [3].

Higham [17] proposes an alternating projections method with correction due to Dykstra to solve our problem. This method guarantees to converge to the nearest correlation matrix with high accuracy, however, at best linearly. In other respects, this method is quite flexible. So for example, variations of the problem can easily be

adapted because the method is essentially based on carrying out only two projections onto closed convex sets in each iteration. Therefore, we only have to introduce a further projection in each iteration (potentially with a correction [17, Section 3.2]) which projects onto a closed convex set which is due to our variations and the algorithm is applied as before. Moreover, because of the procedure of alternating projections the algorithm is fairly easy to code and it is particularly preferable if the matrix G is highly rank deficient since the spectral properties of G are exploited.

The next two approaches are not as flexible as the last one, but they claim a faster convergence. One of them is to apply a quasi-Newton method to the Lagrangian dual function of the problem as proposed by Malick [23]. The other one, recently proposed by Qi and Sun [27], is also based on the dual problem but, in contrast to the approach of Malick, an inexact Newton method is applied to the dual function. Qi and Sun proved the quadratic convergence for this approach and also provided an algorithm which we focus our attention on in the next chapters and which we take as the basis for our algorithm.

In the next chapter we point out the basic theoretical background of the approach proposed by Qi and Sun. In Chapter 3 we introduce an algorithm to compute the nearest correlation matrix by using this approach which is essentially the same as the algorithm of Qi and Sun in [27, Algorithm 5.1]. We analyse and discuss possible improvements to this algorithm in Chapter 4 and introduce our modified version in Chapter 5. We perform some numerical tests in Chapter 6. Chapter 7 concerns the alternating projections method: we introduce Higham's method, discuss its variations and perform relevant numerical tests. We draw our concluding remarks in Chapter 8.

Chapter 2

Theoretical Background

We now introduce the idea presented by Qi and Sun [27] to formulate a quadratically convergent algorithm for (1.14) and outline the proof. Furthermore, we state all notations and formulas which are important for the later chapters. Further details of the proofs can be found in [27]. We start with the dual problem.

2.1 The Dual Problem

2.1.1 Dual Formulation

The dual problem to (1.14) is also a convex optimization problem which, unlike the primal problem, is unconstrained and can be posed as

$$\min_{y\in\mathbb{R}^n} \theta(y) := \frac{1}{2}\|(G + \mathrm{Diag}(y))_+\|_F^2 - e^T y. \tag{2.1}$$

Here, $\mathrm{Diag}(y)$ is a linear operator which puts the entries of y on the diagonal of a matrix which consists apart from that of zero elements. The operator $(\cdot)_+ : \mathcal{S}^n \mapsto \mathcal{S}^n_+$ projects onto the set $\mathcal{S}^n_+$ (see [17, Theorem 3.2]):

$$(C)_+ := P \begin{bmatrix} \max(\lambda_1, 0) & & \\ & \ddots & \\ & & \max(\lambda_n, 0) \end{bmatrix} P^T, \tag{2.2}$$

where C is any symmetric matrix having the spectral decomposition

$$C = P\,\mathrm{Diag}(\lambda)P^T, \tag{2.3}$$

with P orthogonal and $\lambda = (\lambda_1, \ldots, \lambda_n)$ the vector of eigenvalues. For the definition of a projection operator onto closed convex sets see Definition 7.1.2 in Chapter 7. Note that our underlying Hilbert space is now the Euclidean space $\mathbb{R}^n$ equipped with the Euclidean inner product

$$\langle x, y \rangle := x^T y \quad \text{for } x, y \in \mathbb{R}^n \tag{2.4}$$

and the corresponding 2-norm

$$\|x\|_2 := \sqrt{x^T x} \quad \text{for } x \in \mathbb{R}^n. \tag{2.5}$$

2.1.2 Derivation of the Dual Problem

In this section we point out how to derive the dual problem (2.1) from the primal problem (1.14). First we form the partial Lagrangian of our objective function in (1.14), which is

$$L(X, y) := \frac{1}{2}\|G - X\|_F^2 - y^T (\operatorname{diag}(X) - e) \quad \text{for } y \in \mathbb{R}^n, X \in \mathcal{S}_+^n \tag{2.6}$$

and transform it into the following expression (see [23], using properties of the inner product $\langle \cdot, \cdot \rangle$ of Definition 1.3.1)

$$L(X, y) = \frac{1}{2}\|(G + \operatorname{Diag}(y)) - X\|_F^2 - \left(\frac{1}{2}\|\operatorname{Diag}(y) + G\|_F^2 - \frac{1}{2}\|G\|_F^2\right) + y^T e. \tag{2.7}$$

Now we place (2.7) in the definition of the corresponding dual function which is defined as

$$\widetilde{\theta}(y) := \min_{X \in \mathcal{S}_+^n} L(X, y) \tag{2.8}$$

and obtain with $C := G + \operatorname{Diag}(y)$

$$\widetilde{\theta}(y) = \min_{X \in \mathcal{S}_+^n} \left(\frac{1}{2}\|C - X\|_F^2 - \frac{1}{2}\|C\|_F^2 + \frac{1}{2}\|G\|_F^2 + y^T e\right), \tag{2.9}$$

where only the first term depends on X. The minimizer of the first term is the projection of C onto the set $\mathcal{S}_+^n$ since this is one property of the projection operator

$(\cdot)_+$ (see Definition 7.1.2). We have

$$\begin{aligned}\widetilde{\theta}(y) &= \min_{X\in\mathcal{S}_+^n}\left(\frac{1}{2}\|C-X\|_F^2\right) - \frac{1}{2}\|C\|_F^2 + \frac{1}{2}\|G\|_F^2 + y^T e \\ &= \left(\frac{1}{2}\|C-(C)_+\|_F^2\right) - \frac{1}{2}\|C\|_F^2 + \frac{1}{2}\|G\|_F^2 + y^T e \\ &= -\frac{1}{2}\|(C)_+\|_F^2 + \frac{1}{2}\|G\|_F^2 + y^T e,\end{aligned} \tag{2.10}$$

where the latter equality holds since

$$\begin{aligned}\frac{1}{2}\left(\|C-(C)_+\|_F^2 - \|C\|_F^2\right) &= \frac{1}{2}(\langle C-(C)_+, C-(C)_+\rangle - \langle C, C\rangle) \\ &= -\langle C, (C)_+\rangle + \frac{1}{2}\langle (C)_+, (C)_+\rangle \\ &= \langle (C)_+ - C, (C)_+\rangle - \frac{1}{2}\langle (C)_+, (C)_+\rangle \\ &= -\frac{1}{2}\|(C)_+\|_F^2,\end{aligned} \tag{2.11}$$

and where $\langle (C)_+ - C, (C)_+\rangle = 0$ because a projected matrix onto a closed convex cone is orthogonal to the difference of this matrix and its projection in the sense of an inner product [18, Prop. 3.2.3].

Now, we can define our dual problem, which is

$$\sup_{y\in\mathbb{R}^n} \widetilde{\theta}(y) \tag{2.12}$$

and using (2.10) equivalent to

$$\min_{y\in\mathbb{R}^n} \theta(y) := \frac{1}{2}\|(G+\mathrm{Diag}(y))_+\|_F^2 - e^T y, \tag{2.13}$$

since $\frac{1}{2}\|G\|_F^2$ is independent of y. For more details see [23].

2.1.3 Some Properties of the Dual Problem

The dual problem is well posed, a solution of the problem exists and is unique. Furthermore, our dual function $\theta(y)$ has the following properties:

- θ is convex and continuously differentiable.
- The gradient $\nabla\theta(y)$ is Lipschitz-continuous with the Lipschitz-constant 1.

- The gradient is given by

$$\nabla\theta(y) = \operatorname{diag}(G + \operatorname{Diag}(y))_+ - e. \tag{2.14}$$

We consider here only the second property. For all other derivations see [23]. In order to prove the second property we anticipate the Lemma 7.3.1 which we state in a later chapter when we consider projections onto closed convex sets more generally. This lemma says that for any Y in a Hilbert space H with an inner product $\langle\cdot,\cdot\rangle$ and the corresponding norm $\|\cdot\|$

$$\langle Y - P_C(Y), Z - P_C(Y)\rangle \leq 0, \quad \forall Z \in C \tag{2.15}$$

with P_C the projection operator onto a closed convex set $C \subset H$ (see Definition 7.1.2). The subsequent lemma uses this property and proves that the projection $P_C(\cdot)$ is Lipschitz continuous with constant 1 [18, Proposition 3.1.3].

Lemma 2.1.1. *Let C be a closed convex set of a Hilbert space H and $P_C : H \mapsto C$ the projection operator onto C. For all $Y_1, Y_2 \in H$ it holds that*

$$\|P_C(Y_1) - P_C(Y_2)\| \leq \|Y_1 - Y_2\|. \tag{2.16}$$

Proof. Let $Y_1, Y_2 \in H$. From (2.15) with $Z = P_C(Y_2) \in C$ we have that

$$\langle Y_1 - P_C(Y_1), P_C(Y_2) - P_C(Y_1)\rangle \leq 0 \tag{2.17}$$

likewise,

$$\langle Y_2 - P_C(Y_2), P_C(Y_1) - P_C(Y_2)\rangle \leq 0. \tag{2.18}$$

Adding (2.17) to (2.18) yields

$$\langle Y_2 - Y_1 + P_C(Y_1) - P_C(Y_2), P_C(Y_1) - P_C(Y_2)\rangle \leq 0. \tag{2.19}$$

By using the linearity of the inner product (see Definition 1.3.1) and applying the Cauchy-Schwarz inequality (see Lemma 1.3.3) we obtain

$$\|P_C(Y_1) - P_C(Y_2)\|^2 \leq \|P_C(Y_1) - P_C(Y_2)\|\,\|Y_1 - Y_2\| \tag{2.20}$$

which completes the proof. □

Note that the best Lipschitz constant is 1 because the projection of $Y_1, Y_2 \in C$ onto C is the element Y_1 and Y_2, respectively so that we obtain the equality in (2.16). The remaining part is to show that $\nabla\theta(y)$ is also Lipschitz continuous with the constant 1. Let $y_1, y_2 \in \mathbb{R}^n$ be arbitrary. We apply our Lemma 2.1.1 with $H := \mathcal{S}^n \subset \mathbb{R}^{n\times n}$ and the inner product as defined in (1.6), $C := \mathcal{S}^n_+$, $P_C(\cdot) := (\cdot)_+$ and obtain

$$\begin{aligned} \|\nabla\theta(y_1) - \nabla\theta(y_2)\|_2 &= \|\mathrm{diag}(G + \mathrm{Diag}(y_1))_+ - \mathrm{diag}(G + \mathrm{Diag}(y_2))_+\|_2 \\ &\leq \|(G + \mathrm{Diag}(y_1))_+ - (G + \mathrm{Diag}(y_2))_+\|_F \\ &\leq \|\mathrm{Diag}(y_1) - \mathrm{Diag}(y_2)\|_F \\ &\leq \|y_1 - y_2\|_2 \end{aligned} \tag{2.21}$$

which implies that $\nabla\theta(y)$ is Lipschitz continuous with constant 1.

We know that we can find a solution y_* of the dual problem but we have not clarified yet that we can derive the solution of the primal problem X_* from the solution of the dual problem. Fortunately, by the subsequent Theorem 2.1.2 there is no duality gap between the primal and the dual and we can indeed derive X_* from y_*.

Theorem 2.1.2. *If y_* is a solution of the dual problem (2.1) then*

$$X_* = (G + \mathrm{Diag}(y_*))_+ \tag{2.22}$$

is the solution of the primal problem (1.14).

Proof. The proof consists of two parts. First, we show that

$$\frac{1}{2}\|G - X\|_F^2 \geq \widetilde{\theta}(y) \tag{2.23}$$

for all $y \in \mathbb{R}^n$ and all X in the feasible set of the primal problem in (1.14), $\widetilde{\theta}(y)$ is defined as in (2.8). Second, we prove that the lower bound in (2.23) is reached and is achieved at $\widetilde{\theta}(y_*)$, we obtain that

$$\widetilde{\theta}(y_*) = \frac{1}{2}\|G - X\|_F^2 \tag{2.24}$$

for an X. Furthermore, we prove that $X = (G + \mathrm{Diag}(y_*))_+$ and obtain our assertion. Let $y \in \mathbb{R}^n$ and X be primal-feasible. Then

$$\begin{aligned} \frac{1}{2}\|G - X\|_F^2 &= \frac{1}{2}\|G - X\|_F^2 - y^T\left(\mathrm{diag}(X) - e\right) \\ &= L(X, y) \geq \widetilde{\theta}(y) \end{aligned} \tag{2.25}$$

with $L(X, y)$ as defined in (2.6) and the first part follows.

From (2.7) and the fact that the minimizer regarding the variable X in (2.7) is the projection of $G + \mathrm{Diag}(y)$ onto the set $\mathcal{S}_+^n$ we obtain that

$$X(y) := \mathrm{argmin}_{X \in \mathcal{S}_+^n} L(X, y) = (G + \mathrm{Diag}(y))_+. \tag{2.26}$$

Since y_* is a solution of the unconstrained dual problem the gradient of $\theta(y)$ is zero at y_*. Hence, (2.26) and (2.14) imply that

$$\mathrm{diag}(X(y_*)) = e \tag{2.27}$$

and thus that $X(y_*)$ is primal-feasible. Finally, we have from (2.8) that

$$\widetilde{\theta}(y_*) := L(X(y_*), y_*) = \frac{1}{2}\|G - X(y_*)\|_F^2 \tag{2.28}$$

and hence the second part follows. That means the lower bound in (2.23) is reached at $X(y_*)$ and since $X(y_*)$ is primal-feasible $X(y_*)$ is the unique minimizer of the primal problem. □

2.2 Applying an Inexact Newton Method

Now we discuss solving the problem of finding a solution y_* of the dual problem numerically and construct a quadratically convergent method. Our target is to apply an inexact Newton method to the dual function (2.1). Applying an inexact Newton method to a function means we generate a sequence $\{x_k\}$ by applying a Newton method to that function but that the required equation

$$A_k d_k = b_k \tag{2.29}$$

is solved only inexactly. That is, we look for a $d_k \in \mathbb{R}^n$ so that

$$\|A_k d_k - b_k\| \leq \eta \|b_k\| \quad \text{with } \eta \in (0,1) \tag{2.30}$$

where k is the iteration count, $d_k := x_{k+1} - x_k$ the step direction, $b_k \in \mathbb{R}^n$ the negative gradient and $A_k \in \mathbb{R}^{n \times n}$ the second derivative of the objective function at x_k. $\|\cdot\|$ denotes any norm in $\mathbb{R}^n$.

However, a problem arises if we apply a Newton method to our dual function $\theta(y)$. The second derivative of $\theta(y)$ does not exist since the operator $(\cdot)_+$ is not continuously differentiable. Fortunately, this operator is strongly semismooth (see the definition below) and Lipschitz continuous so that in order to tackle the problem, we use the generalized Jacobian of our gradient function $\nabla\theta(y)$ in the sense of Clarke in our Newton iteration instead of the second derivative. Moreover, we aim at applying a theorem which guarantees a quadratic convergence for such a Newton iteration in the case of strongly semismooth functions.

2.2.1 Some Definitions and a Basic Theorem

Before introducing the mentioned theorem, we first provide some necessary definitions. We start with the definition of the generalized Jacobian of a Lipschitz function $\Phi : \mathbb{R}^l \mapsto \mathbb{R}^m$ in the sense of Clarke.

Definition 2.2.1. Let $\Phi : \mathbb{R}^l \mapsto \mathbb{R}^m$ be a (locally) Lipschitz function. Let D_Φ be the set of all $\hat{y} \in \mathbb{R}^l$ where the function Φ is differentiable. Then, the Jacobian of Φ denoted by Φ' exists at all $\hat{y} \in D_\Phi$. Since Φ is differentiable almost everywhere according to Redemacher's theorem [30, Section 9.J] we can define the Bouligand subdifferential of Φ at all elements $y \in \mathbb{R}^l$, which is the following set:

$$\partial_B \Phi(y) := \{V \in \mathbb{R}^{m \times l} : V \text{ accumulation point of } \Phi'(y_k), \text{ where } y_k \to y \text{ and } y_k \in D_\Phi\}.$$

Our *generalized Jacobian* $\partial\Phi$ is then defined as:

$$\partial\Phi(y) := \operatorname{conv} \partial_B \Phi(y),$$

where conv denotes the convex hull.

Note that the generalized Jacobian is now a set. The next definition is required to define the property of strongly semismoothness.

Definition 2.2.2. The function $\Phi : \mathbb{R}^l \mapsto \mathbb{R}^m$ is said to be *directionally differentiable* at $y \in \mathbb{R}^l$ if for any vector $v \in \mathbb{R}^l$ the limit,

$$\lim_{h \to 0} \frac{\Phi(y + h \cdot v) - \Phi(y)}{h}$$

exists.

Now we clarify, as we mentioned above, what strongly semismooth means.

Definition 2.2.3. The function $\Phi : \mathbb{R}^l \mapsto \mathbb{R}^m$ is said to be *strongly semismooth* at $y \in \mathbb{R}^l$ if

- Φ is directionally differentiable at y and
- for any $V \in \partial\Phi(y + h)$,

$$\Phi(y + h) - \Phi(y) - Vh = \mathcal{O}(\|h\|^2).$$

We present the theorem which we pronounced earlier and which is the basic theorem in the paper of Qi and Sun [27, Theorem 2.1].

Theorem 2.2.4. *Let y_* be a solution of $\Phi(y) = 0$ and let $\Phi : \mathbb{R}^l \mapsto \mathbb{R}^m$ be a Lipschitz-function and strongly semismooth at y_*. If all $V \in \partial\Phi(y_*)$ are nonsingular then every sequence $\{y_k\}$ generated by $y_{k+1} = y_k - V_k^{-1}\Phi(y_k)$ with $V_k \in \partial\Phi(y_k)$ converges to y_* quadratically if y_0 is sufficiently close to y_*.*

2.2.2 Applying the Theorem to our Dual Function

To facilitate our analysis in the latter Section 2.2.1, we define

$$J(y) := \operatorname{diag}((G + \operatorname{Diag}(y))_+) = \nabla\theta(y) + e \tag{2.31}$$

and denote ∂J as the generalized Jacobian of J. Note that ∂J is also the generalized Jacobian of $\nabla\theta(y)$ since e is a constant and thus, we can use the set ∂J for our Newton iteration.

We know that J is Lipschitz continuous since $\nabla\theta(y)$ is Lipschitz continuous (see Section 2.1.3) and strongly semismooth (see [27] and references therein). However, in order to apply this theorem (set $\Phi = J$), we still need to prove that all $V \in \partial J(y_*)$ are nonsingular where y_* is a solution of $J(y) = 0$. Moreover, we should be able to compute at least one element or representative $V_k \in \partial J(y_k)$ for all k, in order to construct a sequence like $y_{k+1} = y_k - V_k^{-1} J(y_k)$. Fortunately, we can prove that all $V \in \partial J(y_*)$ are nonsingular (see [27]) and we can find such a representative V_k in the set $\partial J(y_k)$ for all k, which has even the property to be at least positive semidefinite and to converge to a positive definite matrix for $y_k \to y_*$. This makes the theorem applicable.

Furthermore, Qi and Sun [27] proved by means of Theorem 2.2.4 that the sequence y_k converges quadratically for y_k sufficiently close to y_* when we apply an inexact Newton method only, i.e. we satisfy the following condition in each step:

$$\|\nabla\theta(y_k) + V_k d_k\|_2 \leq \eta_k \|\nabla\theta(y_k)\|_2 \qquad \text{for } \eta_k = \min\left(\eta, \|\nabla\theta(y_k)\|_2\right) \tag{2.32}$$

where $\eta \in (0,1)$ and $d_k := y_{k+1} - y_k$ is our step direction.

Our representative $V_y \in \partial J(y)$ is given implicitly by the following formula:

$$V_y h = \operatorname{diag}\left(P_y (W_y \circ (P_y^T H P_y)) P_y^T\right) \tag{2.33}$$

where $\circ$ denotes the Hadamard product ($X \circ Y = (x_{ij} y_{ij})$), h is a vector in $\mathbb{R}^n$ and H is the corresponding matrix given by $H := \operatorname{Diag}(h)$. P_y is an orthogonal matrix calculated by the spectral decomposition of $G + \operatorname{Diag}(y)$, i.e.

$$G + \operatorname{Diag}(y) = P_y \operatorname{Diag}(\lambda(y)) P_y^T \tag{2.34}$$

with $\lambda(y)$ the vector of all eigenvalues and W_y is a constructed matrix which depends only on the eigenvalues $\lambda(y)$ as we describe below.

Let $\lambda(y)$ be in descending order and define the sets $\alpha := \{i : \lambda_i(y) > 0\}$, $\beta := \{i : \lambda_i(y) = 0\}$ and $\gamma := \{i : \lambda_i(y) < 0\}$. Then the matrix W_y is defined by

$$W_y := \begin{bmatrix} E_{|\alpha|,|\alpha|} & E_{|\alpha|,|\beta|} & \mathcal{T}_{|\alpha|,|\gamma|} \\ E_{|\beta|,|\alpha|} & 0 & 0 \\ T_{|\gamma|,|\alpha|} & 0 & 0 \end{bmatrix}, \tag{2.35}$$

where E is the matrix of ones and $\mathcal{T} = (\tau_{ij})_{i,j=1}^{n}$ with $\tau_{ij} \in (0,1)$ for all $i \in \alpha$ and $j \in \gamma$. The first index of E and $\mathcal{T}$ in W_y denotes the number of rows of the corresponding matrix and the second the number of columns. Note that the values of τ_{ij} depend on the eigenvalues in $\lambda(y)$.

To determine the matrix V_y explicitly, we could use $h = e_i$ in (2.33) for all $i = 1, \ldots, n$, where e_i is a vector with zeros except for the ith entry which is 1. However, this would be expensive since we require $\mathcal{O}(n^4)$ operations to compute this matrix (at least 2 matrix-matrix products for each column for V_y). Hence, in order to determine the direction d_k in the inequality (2.32) we need to use a method which requires matrix-vector products only.

We are now in the situation where we can construct a method which converges quadratically if our y_k is close to our solution y_*. We combine this method with a line search strategy and use global convergence techniques leading to the algorithm specified in the next chapter.

Chapter 3

The Newton Algorithm

3.1 The Newton Algorithm

The following algorithm applies an inexact Newton method to our dual function as discussed in Chapter 2 and combines this with global convergence techniques. The algorithm is essentially the same as the algorithm of Qi and Sun in [27, Algorithm 5.1]. For Qi and Sun's implementation see Section A.1 of Appendix A.

Algorithm 1. Given a symmetric matrix $G \in \mathbb{R}^{n\times n}$ this quadratically convergent algorithm computes the nearest correlation matrix X to G in the Frobenius norm. On termination $\|\nabla\theta(y_k)\|_2 \leq$ *error_tol* (see (2.14) for the formula of $\nabla\theta(y)$) with *error_tol* the given error tolerance.

Step 1: Set the starting values: $y_0 \in \mathbb{R}^n$, $\eta \in (0,1)$, ρ, $\sigma \in (0,1/2)$ and $k := 0$.

Step 2: Calculate $\nabla\theta(y_k)$. If $\|\nabla\theta(y_k)\|_2 \leq$ *error_tol* compute X with (2.22) ($y_* := y_k$, $X := X_*$) and exit.

Step 3: Perform a spectral decomposition of $G + \mathrm{Diag}(y_k)$ and compute the constructive matrix W_{y_k} defined by (2.35).

Step 4: Determine the new direction d_k for the step k by applying an iterative method (using formula (2.33) to compute $V_k d_k$) to

$$V_k d_k = -\nabla\theta(y_k) \tag{3.1}$$

such that the conditions

$$\|\nabla\theta(y_k) + V_k d_k\|_2 \leq \eta_k \|\nabla\theta(y_k)\|_2 \qquad \text{for } \eta_k = \min(\eta, \|\nabla\theta(y_k)\|_2) \quad (3.2)$$

and

$$-\frac{\nabla\theta(y_k)^T}{\|d_k\|_2} \cdot \frac{d_k}{\|d_k\|_2} \geq \eta_k \quad (3.3)$$

are satisfied. If either one of these conditions cannot be satisfied, let

$$d_k := -B_k^{-1}\nabla\theta(y_k) \quad (3.4)$$

where B_k is any symmetric positive definite matrix with $\{\|B_k\|_2\}$ and $\{\|B_k^{-1}\|_2\}$ uniformly bounded.

Step 5: Choose an appropriate step length α_k by applying Armijo backtracking: find the smallest number $m_k \in \mathbb{N}_0$ such that

$$\theta(y_k + \rho^{m_k} d_k) - \theta(y_k) \leq \sigma\rho^{m_k}\nabla\theta(y_k)^T d_k \quad (3.5)$$

is satisfied.

Step 6: Set $\alpha_k := \rho^{m_k}$, $y_{k+1} = y_k + \alpha_k d_k$ and $k \leftarrow k+1$. Go to Step 2.

3.2 Convergence Analysis

To obtain a globally convergent algorithm, i.e. $\lim_{k\to\infty} \nabla\theta(y_k) = 0$, we have to satisfy the following three conditions [20], [24]:

- d_k is a descent direction at every iteration.
- The angle between the negative gradient direction $-\nabla\theta(y_k)$ and the direction d_k is bounded away from 90 degrees.
- The step length α_k is chosen so that the step $\alpha_k d_k$ is not too small and the descent in the function $\theta(y)$ is sufficient.

In this algorithm, the inequalities (3.2) and (3.3) guarantee the first and the second conditions. In the case of not satisfying one of the inequalities, the choice $d_k := -B_k^{-1}\nabla\theta(y_k)$ with $\{\|B_k\|_2\}$ and $\{\|B_k^{-1}\|_2\}$ uniformly bounded also guarantees these conditions. The Armijo backtracking strategy makes sure that the third condition is satisfied and, hence, Algorithm 1 is globally convergent. Furthermore, since $\theta(\cdot)$ is convex and $\{y_k\}$ is bounded, we also obtain that $y_k \to y_*$ as k goes to infinity. From the fact that the projection $(\cdot)_+$ is Lipschitz continuous with a Lipschitz constant 1 by Lemma 2.1.1, we can deduce with $X_k = (G + \mathrm{Diag}(y_k))_+$ (compare (2.26)) that

$$\begin{aligned} \|X_k - X_*\|_F &= \|(G + \mathrm{Diag}(y_k))_+ - (G + \mathrm{Diag}(y_*))_+\|_F \\ &\leq \|\mathrm{Diag}(y_k) - \mathrm{Diag}(y_*)\|_F \\ &= \|y_k - y_*\|_2 \end{aligned} \tag{3.6}$$

and we also obtain that $X_k \to X_*$ as $k \to \infty$.

Chapter 4

Discussion of the Algorithm

4.1 Issues

We have implemented a modified version of Algorithm 1 in MATLAB starting with an M-file obtained from the authors of [27] (see Section A.1 of Appendix A) which contains the implementation of Algorithm 1. We state the modified version in Chapter 5. We have changed the algorithm referring to some issues which we now discuss, we consider some weaknesses and possible improvements of the obtained algorithm. In substance, we pursue and discuss the following issues:

- Choosing η_k in inequality (3.2) and (3.3).
- Choosing the method for (3.1).
- Applying a preconditioner to (3.1).
- Using

$$\|\nabla\theta(y_k) + (V_k + \varepsilon_k I_n)d_k\|_2 \leq \eta_k \|\nabla\theta(y_k)\|_2, \qquad \text{with } \varepsilon_k = \|\nabla\theta(y_k)\|_2 \quad (4.1)$$

  instead of condition (3.2) or (3.4).
- What accuracy is achievable and how can it be improved.
- Choosing the method for the eigenvalue decomposition of $G + \mathrm{Diag}(y_k)$ in Algorithm 1.

- Strategies if the matrix G is nonsymmetric.

4.2 Regarding η_k in the First Inequality

To prove that Algorithm 1 is quadratically convergent, Qi and Sun use that η_k is equal to the $\min(\eta, \|\nabla\theta(y_k)\|_2)$ with $\eta \in (0,1)$. However, in their implementation η_k was set as constant with modulus 10^{-6} only. We change that fact according to the theory and use

$$\eta_k := \min(\eta, \|\nabla\theta(y_k)\|_2). \tag{4.2}$$

To overcome the last concern determining a reasonable value of η, we follow the suggestion of Nocedal and Wright in [24], who use 0.5 as the maximum value for η_k in a line search Newton-CG method.

4.3 Choosing the Appropriate Method

4.3.1 Reasons for Considering Different Methods

To determine a direction d_k which satisfies the condition (3.2), Qi and Sun use the unpreconditioned conjugate gradient method. However, this can lead to some problems. For example, the conjugate gradient method does not reduce the residual $r_k := \nabla\theta(y_k) + V_k d_k$ in every iteration in general and in addition the residual can oscillate, especially if no preconditioner is used. Another problem is that the matrix V_k can be positive semidefinite only and consequently the conjugate gradient method can fail to find a direction d_k in (3.2). Further details are given in the next section.

This motivates us to investigate other methods that use only matrix-vector products. We introduce five other methods and point out the pros and cons of the methods and perform tests in Section 6.3.

All methods are iterative methods which use only matrix-vector products and, likewise, their convergence behaviour can be improved enormously by using a preconditioner. Note that all specifications of storage and required flops are without

preconditioning in the subsequent sections.

4.3.2 CG

The linear conjugate gradient (CG) method is an iterative method that was proposed by Hestenes and Stiefel in the 1950s and is used for solving a linear system $Ax = b$ with $A \in \mathbb{R}^{n\times n}$ symmetric positive definite.

This method generates a basis of A-conjugate vectors p_k that span a Krylov-subspace

$$\mathcal{K}(r_0, A, k+1) = \text{span}\{r_0, Ar_0, \cdots, A^k r_0\} = \text{span}\{p_0, \ldots, p_k\} = \text{span}\{r_0, \ldots, r_k\}$$

where $r_k = Ax_k - b$ is the residual and is orthogonal to $\mathcal{K}(r_0, A, k)$. The index k denotes the current iteration count. The corresponding linear coefficients α_k are chosen to minimize $\frac{1}{2}x_k^T Ax_k - b^T x_k$ with $x_k = \sum_{i=0}^{k} \alpha_i p_i$, which is equivalent to minimizing the energy norm

$$\|Ax_k - b\|_{A^{-1}} = \sqrt{(Ax_k - b)^T A^{-1}(Ax_k - b)}$$

over x_k.

In exact arithmetic, this method converges after n iterations at the latest since the generated vectors p_k are linearly independent (note that all subsequent methods have this property, if we consider here RGMRES as GMRES). However, in finite precision arithmetic the method should be regarded as a genuinely iterative technique with a termination based on the residual norm, since rounding errors cause loss of linear independence among the vectors p_k.

This method is mathematically equivalent to generating a sequence of symmetric tridiagonal matrices $T_k = Q_k^T A Q_k$ by means of the Lanczos process with Lanczos vectors $Q_k = (q_1, \ldots, q_k)$ (equivalent to normalized r_k in CG) and carrying out an LDL^T (Cholesky) factorization of these matrices to solve the system.

Pros	Cons
• Solves systems for large and sparse matrices A efficiently.	• Method defined for A symmetric positive definite.
• Requires one matrix-vector product and only $10n$ flops per iteration.	• If the matrix is not positive definite then solving $Ax = b$ can fail or undefined CG points x_k possible (see more details in [26]).
• Requires little storage.	• No monotonic convergence of the sequence of the norms of the residuals $\|Ax_k - b\|_2$ in general, thus the residual norm can oscillate wildly.

4.3.3 SYMMLQ

The SYMMLQ is an iterative method based on the Lanczos process for solving linear systems with a symmetric matrix A and was first proposed by Paige and Saunders [26]. This method is similar to the conjugate gradient method. The main difference is that this method is mathematically equivalent to carrying out an LQ factorization of the symmetric tridiagonal matrices T_k (see Section 4.3.2). This allows the use of matrices A which are also indefinite.

Pros	Cons
• Suitable for sparse general symmetric matrices A. • SYMMLQ more suitable for singular or, in finite precision arithmetic, nearly singular matrices than CG since this method is numerically stable for indefinite systems. • CG iterate x_k^{CG} can be recovered from the SYMMLQ iterate x_k. • Requires two inner products, five vector updates and one matrix-vector product at each iteration (approximately $12n + 2n^2$ flops).	• Less efficient in comparison to CG if the matrix A is positive definite. • No monotonic convergence of the sequence of the norms of the residuals $\|Ax_k - b\|_2$ in general.

4.3.4 MINRES

This method was also proposed by Paige and Saunders in [26] and is the first method to be introduced which minimizes the residual norm $\|r_k\|_2 = \|Ax_k - b\|_2$ in every iteration monotonically. Based on the Lanczos process a sequence of symmetric triangular matrices T_k is also generated. Furthermore, this method is mathematically equivalent to carrying out a QR factorization of T_k. The matrix A can be symmetric positive definite or indefinite.

We minimize

$$\|Ax_k - b\|_2 \tag{4.3}$$

over the set of the Lanczos-vectors $\text{span}\{q_1, \ldots, q_k\}$ with $x_k = x_0 + \sum_{i=1}^{k} \alpha_k q_k$ where x_0 is the initial guess.

Pros	Cons
• Suitable for sparse general symmetric matrices A.	• Does not give as accurate results as SYMMLQ for ill-conditioned A (see [26]).
• MINRES more suitable for singular or, in finite precision arithmetic, nearly singular matrices than CG.	• If we use a preconditioner we do not have the residual norm $\|Ax_k - b\|_2$ available without an additional computation, we only have the following energy norm $\|Ax_k - b\|_{M^{-1}} = \sqrt{(Ax_k - b)^T M^{-1}(Ax_k - b)}$ where M is the preconditioning matrix, see [7, Section 6.1].
• Reduces the norm of the residual monotonically.	• Requires slightly more work than CG, requiring 2 inner products, five vector updates and 1 matrix-vector product at each iteration (approximately $14n + 2n^2$ flops).
• CG iterate x_k^{CG} can be recovered from the MINRES iterate x_k. • Can take fewer iterations than CG.	

4.3.5 RGMRES

The restarted generalized minimum residual method (RGMRES) was proposed by Saad and Schultz [31] and can be seen as a generalization of the MINRES algorithm. It solves the system $Ax = b$ for $A \in \mathbb{C}^{n\times n}$ sparse general and non-Hermitian. The residual norm $\|b - Ax_k\|_2$ is minimized for x_k in the set

$$\mathcal{S} = x_0 + \text{span}\{q_1, \ldots, q_k\}$$

in every iteration, where x_0 is the initial value of the generated sequence $\{x_i\}_{i=1}^{k}$ and $q_1, \ldots, q_k$ are vectors generated by Arnoldi's method. Unfortunately, generating the orthogonal basis $q_1, \ldots, q_k$ can require a 'long' recurrence since the matrix A can be not real and nonsymmetric. Actually, instead of generating a triangular matrix as in MINRES it produces an upper Hessenberg matrix. Hence, the computational and storage costs can become prohibitive. Therefore, in the RGMRES method (difference to GMRES) a restarting strategy is used, where the iterate x_m of a GMRES sequence computed after m iterations is taken as initial value for the next GMRES sequence. The value of m is to specify in advance. See more details in [12], [31] and [25].

Pros	**Cons**
• Suitable for complex sparse general and non-Hermitian matrices A.	• The value of m is chosen in advance and the optimal value cannot be predicted easily.
• Reduces the norm of the residual monotonically at least in each GMRES sequence.	• The choice of m depends on the problem.
• The GMRES sequence cannot break down.	• Can stagnate or converge slowly if m is too small.
• RGMRES converges for all choices of m if the matrix A is positive real, in other words, the symmetric part of the matrix A has strictly positive eigenvalues.	• RGMRES requires $(m+3+1/m)n+n_z$ flops (we assume that flops correspond with the multiplication operations in [31]) per iteration on average and a storage of $m+2$ vectors of the length n, where n_z denotes the number of nonzero elements of the matrix A.
	• GMRES cannot perform better than MINRES for symmetric matrices in general.

4.3.6 BI-CGSTAB(ℓ)

The bi-conjugate gradient stabilized method (BI-CGSTAB) (ℓ) proposed by Van der Vorst is a modification of the BCG (bi-conjugate gradient method) and solves systems $Ax = b$ with $A \in \mathbb{C}^{n\times n}$ a sparse general and non-Hermitian matrix. In the BCG method for solving $Ax = b$, two sequences of vectors $\{r_i\}_{i=0}^k$ and $\{\hat{r}_i\}_{i=0}^k$ are generated by means of an unsymmetric Lanczos process such that

$$\text{span}\{r_0, \ldots, r_k\} = \mathcal{K}(r_0, A, k+1),$$
$$\text{span}\{\hat{r}_0, \ldots, \hat{r}_k\} = \mathcal{K}(\hat{r}_0, A^*, k+1)$$

with $A^* = \bar{A}^T$ and

$$\hat{r}_j^* r_i = \begin{cases} 0 & \text{if } i \neq j \\ d_i \neq 0 & \text{if } i = j \end{cases}$$

(bi-orthogonality) and the iterate x_k is computed by $x_k = x_0 + Q_k y_k$. r_k is the residual $Ax_k - b$, $\hat{r}_0$ is any vector and k again denotes the current iteration count. Let $Q_k := (r_1, \ldots, r_k)$ and $\hat{R} := (\hat{r}_1, \ldots, \hat{r}_n)$ then $T_k = \hat{R}_k^* A Q_k$ is a triangular matrix and y_k is the solution of $T_k y_k = Q_k^* r_0$ which is computed by carrying out an LU-factorization. This procedure is done by a recursion over the iterations, i.e. r_k and x_k can be updated from its predecessor in each iteration. Note that $d_i = \langle \hat{r}_i, r_i \rangle = \langle P_i(A^*)\hat{r}_0, P_i(A)r_0 \rangle$ is required in the course of iterations and thus, A^* is needed to compute $\hat{r}_i$ where $r_i = P_i(A)r_0$ and $\hat{r}_i = P_i(A^*)\hat{r}_0$ for P_i a polynomial of degree i generated by the Lanczos process.

In the BI-CGSTAB(ℓ) method, we do not require A^*. We generate a sequence $\{\tilde{r}_i\}_{i=1}^k = \{Q_i(A)P_i(A)r_0\}_{i=0}^k$ derived from the idea that $d_i = \langle P_i(A^*)\hat{r}_0, P_i(A)r_0 \rangle = \langle \hat{r}_0, P_i(A)P_i(A)r_0 \rangle$ and substituting the first polynomial $P_i(A)$ in the second argument by a polynomial $Q_i(A)$ of degree i. That is, $\tilde{d}_i = \langle \hat{r}_0, Q_i(A)P_i(A)r_0 \rangle$ where $Q_i(A)$ is chosen to minimize the new residual $\tilde{r}_i$. This procedure is also done by recursion. The value of ℓ gives the order over which the polynomial $Q_i(A)$ is minimized in ℓ iterations. See [35] and [32] for the whole derivation and the algorithm.

Pros	Cons
• Suitable for complex sparse general and non-Hermitian matrices A. • Requires in the simple form [35, Figure 9.1.] 2 matrix-vector products, $12n$ flops for vector updates and 2 inner products. • 7 vectors are required to be stored. • Does not require a matrix-vector product with A^*. • More robust in term of rounding errors and overflow than the CGS method (which is similar to BCG but generates residual vectors of the form $\bar{r}_i = P_i(A)^2 r_0$). • Converges more smoothly than CGS.	• Does not minimize the residual norm monotonically. • BI-CGSTAB can break down since the unsymmetric Lanczos-process can break down. • Can converge considerably slower than CGS.

4.3.7 TFQMR

The transpose-free quasi-minimal residual method of Freund and Nachtigal [9] solves a linear system $Ax = b$ with $A \in \mathbb{C}^{n \times n}$ sparse and non-Hermitian and is a modification of the CGS method (which is similar to BCG but generates residual vectors of the form $\bar{r}_i = P_i(A)^2 r_0$). In contrast to CGS, the iterates x_k are chosen to minimize the norm of a part of the residual (quasi minimization property). Note that this method is not mathematically equivalent to QMR (Quasi minimal residual method). For more details see [9], [10], [32] and [25].

Pros	Cons
• Suitable for complex sparse general and non-Hermitian matrices A. • Almost monotonic convergence curves in the residual norms, iterates x_k satisfy a quasi-minimization property. • Converges at least as fast as CGS in terms of iterations. • Remedies the rather irregular convergence behaviour of CGS in the residual norm. • Requires roughly the same work and storage per iteration as BI-CGSTAB and CGS. • Bounds for the residuals are essentially the same as the bounds for GMRES.	• TFQMR in the form described in [10] can break down since the unsymmetric Lanczos-process can break down. Most of the breakdowns can be avoided by using look-ahead variants of the Lanczos process.

4.4 Using a Preconditioner

4.4.1 The Idea of a Preconditioner

The idea of preconditioning is to transform the system $Ax = b$ without significantly more additional cost into another system $\hat{A}\hat{x} = \hat{b}$ so that $\hat{A}$ is well conditioned or has clustered eigenvalues, which leads to a better convergence of an iterative method. Consider for example the following convergence bound which can be derived for the

MINRES method ([7, Section 2.4]):

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \min_{p_k \in \Pi_k, p_k(0)=1} \max_j |p_k(\lambda_j)| \tag{4.4}$$

where r_k is the residual, Π_k the set of all polynomials of degree at most k and λ_j are the eigenvalues of A. This shows clearly why clustering the eigenvalues of A leads to better convergence results, because the $\min_{p_k \in \Pi_k, p_k(0)=1} \max_j |p_k(\lambda_j)|$ will become small already for small k if the eigenvalues are close together.

In the iterative methods of Section 4.3 for A symmetric, $\hat{A}$, $\hat{b}$ and $\hat{x}$ are usually chosen as $\hat{A} = C^{-1}AC^{-T}$, $\hat{x} = C^T x$ and $\hat{b} = C^{-1}b$ with C nonsingular so that $\hat{A} = C^{-1}AC^{-T}$ is well conditioned and so that the system $My = d$ with $M = CC^T$ and $y, d \in \mathbb{R}^n$ is easily solvable. Note that using the preconditioner requires only the additional cost of solving the system $My = d$ at every iteration. M defines our preconditioner.

In all methods for A nonsymmetric, a matrix M nonsingular is chosen, where M defines our preconditioner again, so that we obtain the same conditions as in the symmetric case, however, with $\hat{A} = M^{-1}A$, $\hat{x} = x$ and $\hat{b} = M^{-1}b$. Here, using the preconditioner also requires the additional cost of solving the system $My = d$.

4.4.2 Our Choice of Preconditioner

In order to use a preconditioner for our matrix V_k, we need some information about the matrix. Consider our formula for V_k again.

$$V_k h = \mathrm{diag}(P_k(W_k \circ (P_k^T H P_k))P_k^T). \tag{4.5}$$

As can be seen in (4.5), the matrix V_k is given implicitly only and computing the whole matrix, e.g. in order to be able to apply an incomplete Cholesky preconditioner, by means of the approach described in Chapter 2, is extremely expensive and therefore, for our purpose rather inefficient. Hence, we restrict our determination of the elements of the matrix V_k to the diagonal elements only and it turns out that this can be done quite efficiently (see below). Hence, we pursue the approach to rescale the diagonal elements of V_k to 1 and apply a preconditioner which is a diagonal matrix and has

simply these diagonal entries. In other words, we define our preconditioner as

$$M_k := \text{Diag}(\text{diag}(V_k)). \tag{4.6}$$

Note that such a preconditioner is called a Jacobi preconditioner.

Referring to improving the convergence behaviour when applying the preconditioner M_k, we are motivated by the corollary in [16, Corollary 7.6] which says this:

Lemma 4.4.1. *Let $A \in \mathbb{R}^{n\times n}$ be a symmetric positive definite matrix and $D_* = \text{diag}\,(a_{ii}^{-1/2})$ the matrix which scales the diagonal of A to 1 by applying D_*AD_*. Then*

$$\kappa_2(D_*AD_*) \leq n \min_{D\in\mathcal{D}_n}(\kappa_2(DAD)) \tag{4.7}$$

with $\kappa = \|A\|_2\|A^{-1}\|_2$ and $\mathcal{D}_n$ the set of diagonal matrices in $\mathbb{R}^{n\times n}$.

Fortunately, our matrix V_y (see (2.33)) is always symmetric positive semidefinite and converges to a positive definite matrix (all diagonal elements are positive for k sufficiently large since $e_i^T V_y e_i > 0\ \forall i$), which can be proved by using a similar derivation for V_{ij} and V_{ji} to the derivation of V_{ii} below. Hence we satisfy the conditions of Lemma 4.4.1 for all k sufficiently large, the upper bound of Lemma 4.4.1 applies to our suggested preconditioner (4.6). This does not imply that the convergence behaviour improves. However, if we consider for instance a diagonal matrix with different values on the diagonal then our matrix will become the identity when we apply our preconditioner and thus our preconditioner will indeed improve the convergence behaviour and will cluster the eigenvalues. We test our preconditioner in Section 6.3. Note that if the matrix is only positive semidefinite one diagonal element can be zero. Theoretically, in such a case we cannot apply our Jacobi preconditioner but numerical rounding errors often allow it nevertheless.

Now we consider how we can compute our diagonal. Obviously, computing the diagonal by applying the formula (4.5) directly, using $H = e_i e_i^T$, $\forall i = 1, \ldots, n$ is rather impractical because for every diagonal element at least 1 matrix-matrix multiplication is required. Hence, computing the whole diagonal needs $\mathcal{O}(n^4)$ operations. Fortunately, we can find another approach. Let $h = e_i$, the corresponding matrix

$H = e_i e_i^T$ for $i \in \{1, \ldots, n\}$ and $P_k^T = [p_1 p_2 \ldots p_n]$. Then the diagonal element of V_k located in the ith column and ith row is given by (compare with (4.5))

$$\begin{aligned} v_{ii} &= e_i^T P_k (W_k \circ P_k^T H P_k) P_k^T e_i \\ &= p_i^T (W_k \circ p_i p_i^T) p_i. \end{aligned} \tag{4.8}$$

Now we can apply [19, Lemma 5.1.2] to our last term and obtain

$$\begin{aligned} v_{ii} &= p_i^T \operatorname{Diag}(p_i) W_k \operatorname{Diag}(p_i) p_i \\ &= q_i^T W_k q_i, \end{aligned} \tag{4.9}$$

where $q_i \in \mathbb{R}^n$ with $q_i(j) = p_i(j)^2$ for all $j = 1, \ldots, n$. From

$$\begin{aligned} L_k &= W_k [q_1 q_2 \ldots q_n] \\ &= W_k Q_k \ (2n^3 \text{ flops}) \end{aligned} \tag{4.10}$$

where $Q_k = P_k \circ P_k$ (n^2 flops), our diagonal element v_{ii} can then computed by

$$v_{ii} = q_i^T l_i \quad \text{for all } i = 1, \ldots, n \ (2n^2 \text{ flops}). \tag{4.11}$$

This computation requires 1 matrix-matrix multiplication plus computations with operation counts which are all of order n^2 only. Moreover, the whole computation is vectorizable and, consequently, our diagonal can be computed quite efficiently. Furthermore, note that the structure of the matrix W_k can be exploited when we multiply this matrix with Q_k since W_k contains zero and ee^T blocks — see (2.35). We apply this preconditioner in our tests in Section 6.3.

Note that a preconditioner must be positive definite. Thus we set all entries of the diagonal that are less than a predefined tolerance to our tolerance in our preconditioner.

4.5 Choosing η_k in the Second Inequality

In principle, the value of η_k in the inequality (3.3) can be chosen independently of the η_k in the inequality (3.2). In order to avoid confusion, we rename the η_k in inequality (3.3) to φ_k first so that (3.3) becomes

$$\nabla\theta(y_k)^T d_k \leq -\varphi_k \|d_k\|_2^2. \tag{4.12}$$

We show now that the value of

$$\varphi_k := \min\left(\varphi, \|\nabla\theta(y_k)\|_2^m\right) \quad \text{with } \varphi \text{ constant and } \varphi \in (0,1) \tag{4.13}$$

for any nonnegative integer m is sufficient to satisfy the two conditions on the global convergence linked with the second inequality (3.3). Recall that these two conditions are a descent direction in each iteration and that the descent direction is gradient-related (the angle between descent direction and negative gradient direction is bounded away from 90 degrees).

Let $\varphi_k = \min\left(\varphi, \|\nabla\theta(y_k)\|_2^m\right)$. Then, obviously inequality (4.12) guarantees that $d_k \neq 0$ (d_k can be assumed to be nonzero because of (3.2)) is a descent direction since

$$\nabla\theta(y_k)^T d_k \leq -\varphi_k \|d_k\|_2^2 < 0. \tag{4.14}$$

We show that the second condition is also satisfied. Let α_k be the angle between the descent direction d_k and the negative gradient direction. To arrive at a contradiction, assume that $\cos(\alpha_k) \to 0$ as $k \to \infty$ and that $\|\nabla\theta(y_k)\|_2 \not\to 0$ as $k \to \infty$. We have

$$\begin{aligned} \cos(\alpha_k)\|\nabla\theta(y_k)\|_2 &= -\frac{\nabla\theta(y_k)^T d_k}{\|d_k\|_2} \\ &\geq \varphi_k \|d_k\|_2. \end{aligned} \tag{4.15}$$

Since the function $\theta(y)$ is convex and bounded below and we have a descent direction at every iteration which follows from (4.14), $\{\|\nabla\theta(y_k)\|_2\}$ is also bounded and thus we conclude from (4.15) that

$$\|d_k\|_2 \overset{k\to\infty}{\to} 0 \text{ or } \varphi_k \overset{k\to\infty}{\to} 0 \text{ for } m > 0\ .$$

Hence, $d_k \to 0$ or $\varphi_k \to 0$ as $k \to \infty$ for $m > 0$. If $\varphi_k \to 0$, it follows immediately from Definition 4.13 that $\|\nabla\theta(y_k)\|_2$ also goes to zero if $m > 0$ and we obtain the contradiction. If $d_k \to 0$, $V_k d_k$ also goes to zero in (3.2) and since η_k is always strictly less than 1 in this inequality, $\|\nabla\theta(y_k)\|_2$ must converge to zero because otherwise there exists k so that

$$\|\nabla\theta(y_k) + V_k d_k\|_2 > \eta_k \|\nabla\theta(y_k)\|_2 \qquad \text{for } \eta_k < 1 \tag{4.16}$$

and this is a contradiction to the computed direction d_k which satisfies the inequality (3.2). Hence we also obtain our contradiction in this case. Note that our proof includes $\varphi_k := \varphi$ (take $m = 0$) but this choice can slow down the convergence of Algorithm 1 since then we do not guarantee that the inequality in (4.13) is satisfied for all k large which is necessary for a quadratic convergence assuming that B_k is the identity for example. Furthermore, it is advisable to use m less than 2 because otherwise φ_k can become numerically zero and thus infeasible directions would not be refused by the inequality (4.12).

Based on the theory (Taylor series), taking the inexact Newton direction leads to a faster convergence than choosing the steepest descent direction. However, computing the Newton direction is in general more expensive than computing the gradient direction as in our problem so that for the first steps, where y_k is far from the solution, the gradient direction can be preferable. Since the test results in Section 6.2 have shown that using the inexact Newton direction is also preferable for the first steps, we choose the constant φ small and $m = 1$, in order not to slow down the convergence rate of the algorithm and nevertheless to guarantee the global convergence.

4.6 Using a Shifting Factor

Theoretically, the problem of finding a d_k satisfying the inequality (3.2) might not be solvable since far from the solution the matrix V_k can only be positive semidefinite, i.e. the matrix V_k is singular and no direction d_k satisfying the inequality (3.2) exists. We take the steepest descent direction in this case but this can lead to slower convergence. The idea is now to shift the eigenvalues of the matrix V_k by adding a multiple of the identity I_n so that the matrix becomes positive definite. That is,

$$\tilde{V}_k := V_k + \varepsilon_k I_n. \tag{4.17}$$

Thereby, we actually press the direction d_k more to the steepest descent direction. Using this shifted matrix guarantees that we always find a direction d_k which solves the inequality $\|\nabla\theta(y_k) + \tilde{V}_k d_k\|_2 \leq \eta_k \|\nabla\theta(y_k)\|_2$. However, the question arises which

convergence rate can be obtained and how to choose ε_k. We deduce the subsequent theorem from the proof of Qi and Sun in [27, Theorem 5.3].

Theorem 4.6.1. *If ε_k is chosen as*

$$\varepsilon_k := \min\left(\varepsilon, \|\nabla\theta(y_k)\|_2\right) \text{ with } \varepsilon \in (0,1) \tag{4.18}$$

then Algorithm 1 with $V_k := V_k + \varepsilon_k I_n$ converges to the solution y_* quadratically if y_k is close to the solution.

Proof. Since all conditions for the global convergence are still guaranteed (compare with Section 3.2) we have that

$$\lim_{k\to\infty} \nabla\theta(y_k) = 0 \tag{4.19}$$

and $y_k \to y_*$ with $\nabla\theta(y_*) = 0$. Hence, we also obtain that $\{\|\tilde{V}_k^{-1}\|_2 : k \geq 0\}$ is uniformly bounded with $\tilde{V}_k := V_k + \varepsilon_k I_n$ since the matrix V_k is always positive semidefinite and converges to a positive definite matrix. This implies that the iterative method can always find a direction which satisfies the inequalities (3.2) and for k sufficiently large also (3.3) (V_k substituted by $\tilde{V}_k$ now). Hence, since $\nabla\theta(y)$ is Lipschitz continuous and by Theorem 2.2.4 we obtain for all k sufficiently large that

$$\begin{aligned}
\|y_{k+1} - y_*\|_2 &= \|y_k + d_k - y_*\|_2 \\
&= \|y_k + \tilde{V}_k^{-1}[(\nabla\theta(y_k) + \tilde{V}_k d_k) - \nabla\theta(y_k)] - y_*\|_2 \\
&= \|y_k - y_* - \tilde{V}_k^{-1}\nabla\theta(y_k)\|_2 + \|\tilde{V}_k^{-1}(\nabla\theta(y_k) + \tilde{V}_k d_k)\|_2 \\
&\leq \|\tilde{V}_k^{-1}\|_2\|\nabla\theta(y_k) - \nabla\theta(y_*) - \tilde{V}_k(y_k - y_*)\|_2 + \|\tilde{V}_k^{-1}\|_2\eta_k\|\nabla\theta(y_k)\|_2 \\
&\leq \|\tilde{V}_k^{-1}\|_2\Big(\|\nabla\theta(y_k) - \nabla\theta(y_*) - V_k(y_k - y_*)\|_2 + \|\varepsilon_k(y_k - y_*)\|_2\Big) \\
&\quad + \|\tilde{V}_k^{-1}\|_2\|\nabla\theta(y_k)\|_2^2 \\
&\leq \mathcal{O}\left(\|y_k - y_*\|_2^2\right) + \|\tilde{V}_k^{-1}\|_2\|\nabla\theta(y_k)\|_2\|y_k - y_*\|_2 \\
&\quad + \|\tilde{V}_k^{-1}\|_2\|\nabla\theta(y_k) - \nabla\theta(y_*)\|_2^2 \\
&\leq \mathcal{O}\left(\|y_k - y_*\|_2^2\right) + 2\|\tilde{V}_k^{-1}\|_2\|\nabla\theta(y_k) - \nabla\theta(y_*)\|_2\|y_k - y_*\|_2 \\
&\leq \mathcal{O}\left(\|y_k - y_*\|_2^2\right) + 2\|\tilde{V}_k^{-1}\|_2\|y_k - y_*\|_2^2 \\
&\leq \mathcal{O}\left(\|y_k - y_*\|_2^2\right).
\end{aligned} \tag{4.20}$$

Now we show that the step length 1 is indeed taken for all k sufficiently large. From (4.20) and the fact that $y_k \to y_*$ we obtain that for all k sufficiently large

$$\begin{aligned} \|y_k - y_*\|_2 &= \|y_k - y_{k+1} + y_{k+1} - y_*\|_2 \\ &\leq \|d_k\|_2 + \|y_{k+1} - y_*\|_2 \\ &\leq \|d_k\|_2 + \mathcal{O}(\|y_k - y_*\|_2^2) \end{aligned} \tag{4.21}$$

and that

$$\|d_k\|_2 \to 0 \quad \text{as } k \to \infty. \tag{4.22}$$

(4.21) implies that

$$\|y_k - y_*\|_2 \leq \|d_k\|_2 + \mathcal{O}\left(\|d_k\|_2^2\right). \tag{4.23}$$

Now let $r_k := \nabla\theta(y_k) + V_k d_k + \varepsilon_k d_k$ then for all k sufficiently large

$$\begin{aligned} -\nabla\theta(y_k)^T d_k &= \langle d_k, V_k d_k\rangle + \varepsilon_k \|d_k\|_2^2 - \langle d_k, r_k\rangle \\ &\geq \langle d_k, V_k d_k\rangle + \varepsilon_k \|d_k\|_2^2 - \|d_k\|_2 \|r_k\|_2 \\ &\geq \langle d_k, V_k d_k\rangle + \varepsilon_k \|d_k\|_2^2 - \eta_k \|d_k\|_2 \|\nabla\theta(y_k)\|_2 \\ &\geq \langle d_k, V_k d_k\rangle + \varepsilon_k \|d_k\|_2^2 - \|d_k\|_2 \|\nabla\theta(y_k)\|_2^2 \\ &\geq \langle d_k, V_k d_k\rangle + \varepsilon_k \|d_k\|_2^2 - \|d_k\|_2 \|y_k - y_*\|_2^2 \end{aligned} \tag{4.24}$$

with $\langle\cdot,\cdot\rangle$ the inner product corresponding to $\|\cdot\|_2$. From (4.22), (4.23) and (4.24) it follows that there exists $\rho > 0$ so that for all k sufficiently large

$$-\nabla\theta(y_k)^T d_k \geq \rho \|d_k\|_2^2. \tag{4.25}$$

Since the inequality (4.25) holds, by [27, Lemma 5.1] the Armijo condition can be satisfied for the step length 1 for all k sufficiently large, i.e. $y_{k+1} = y_k + d_k$ is accepted by Algorithm 1 for all k sufficiently large. Hence, from (4.20) we obtain our assertion.

□

We perform some tests by means of examples in Section 6.2, in order to determine the influence of this choice.

4.7 Achievable Accuracy

In this section, we introduce some crucial points of Algorithm 1 which lead to a restriction to the smallest achievable value of $\|\nabla\theta(y_k)\|_2$ in the course of iterations and thus to a restriction of the final accuracy of the output matrix X compared with X_*. Moreover, we discuss improvements or possibilities to reduce this restriction.

4.7.1 Armijo Backtracking Rule

The Problem

The Armijo backtracking rule can cause numerical problems if the difference of $\theta(y_k + \rho^{m_k} d_k)$ and $\theta(y_k)$ is small relative to the value of $\theta(y_k)$. In order to show this and how it mirrors the smallest achievable value of $\|\nabla\theta(y_k)\|_2$, we proceed from the assumption that $\theta(y)$ is twice continuously differentiable. Furthermore, let the iterate y_k be given and let ε_A be such that

$$\forall \tilde{y} \in \mathbb{R}^n \text{ with } |\theta(\tilde{y}) - \theta(y_k)| \leq \varepsilon_A, \quad fl(\theta(\tilde{y})) = fl(\theta(y_k)) \tag{4.26}$$

and denote the set of $\tilde{y}$ which satisfy the condition (4.26) by $\mathcal{D}$. Let $y_{k+1} := y_k + \rho^{m_k} d_k \in \mathcal{D}$. That means $\theta(y_{k+1})$ and $\theta(y_k)$ are so close together that they have the same value in machine arithmetic and thus the left-hand side of the Armijo rule becomes numerically zero. In this case, the Armijo inequality cannot be satisfied potentially, even for m_k large, consequently we stop the backtracking with m_k equal to the maximum inner iteration number. Hence our step $\rho^{m_k} d_k$ can also become numerically zero and we end up in an infinite outer loop until we reach the maximum permitted number of iterations.

When the Problem Occurs

Our target is now to find a lower bound for $\|\nabla\theta(y_k)\|_2$ when the problem described above can occur so that we can estimate the smallest value of $\|\nabla\theta(y_k)\|_2$ which can possibly be achieved with this algorithm. See also [11, Section 8.2.2].

We set $|\theta(y_{k+1}) - \theta(y_k)| = \varepsilon_A$ and let $p := \rho^{m_k} d_k / \|\rho^{m_k} d_k\|_2$ and $\tau = \|\rho^{m_k} d_k\|_2$ for simplicity. We expand $\theta(y)$ about y_k in a Taylor series and obtain the following for $y_{k+1} = y_k + \tau p$:

$$\theta(y_{k+1}) = \theta(y_k) + \tau \nabla\theta(y_k)^T p + \frac{1}{2}\tau^2 p^T H(y_k) p + \mathcal{O}(\tau^3), \tag{4.27}$$

with $H(y)$ the Hessian of $\theta(y)$. We suppose that τ is small, which leads to

$$|\theta(y_{k+1}) - \theta(y_k)| \approx \left| \tau \nabla\theta(y_k)^T p + \frac{1}{2}\tau^2 p^T H(y_k) p \right|. \tag{4.28}$$

Similarly, we expand $\nabla\theta(y)$ about y_k and obtain

$$\nabla\theta(y_{k+1}) \approx \nabla\theta(y_k) + \tau H(y_k) p. \tag{4.29}$$

Multiplying (4.29) on the left by $\frac{1}{2}\tau p^T$ and substituting in (4.28) results in

$$\begin{aligned} |\theta(y_{k+1}) - \theta(y_k)| &\approx \left| \tau \nabla\theta(y_k)^T p + \frac{1}{2}\tau \nabla\theta(y_{k+1})^T p - \frac{1}{2}\tau \nabla\theta(y_k)^T p \right| \\ &= \left| \frac{1}{2}\tau \left(\nabla\theta(y_k) + \nabla\theta(y_{k+1}) \right)^T p \right| \\ &\leq \frac{1}{2}\tau \|\nabla\theta(y_k) + \nabla\theta(y_{k+1})\|_2. \end{aligned} \tag{4.30}$$

Using (4.29) again to obtain an approximation for τ leads to

$$\varepsilon_A \leq \frac{\|-\nabla\theta(y_k) + \nabla\theta(y_{k+1})\|_2 \cdot \|\nabla\theta(y_k) + \nabla\theta(y_{k+1})\|_2}{2\|H(y_k)p\|_2}. \tag{4.31}$$

Let p now be additionally a linear combination of the eigenvectors of $H(y_k)$ corresponding to an eigenvalue λ. Then from (4.31) we obtain that

$$\varepsilon_A \lessapprox \frac{\|\nabla\theta(y_k)\|_2^2}{\|H(y_k)p\|_2} \approx \frac{\|\nabla\theta(y_k)\|_2^2}{\lambda} \tag{4.32}$$

and thus the following approximate lower bound for $\|\nabla\theta(y_k)\|_2$

$$\|\nabla\theta(y_k)\|_2 \geq \sqrt{\varepsilon_A \lambda}. \tag{4.33}$$

Note that the value of $\theta(y)$ (see (2.1)) can become greater than 1 since the positive eigenvalues of $G + \mathrm{Diag}(y)$, which can be greater than 1, are involved quadratically and thus the first term can dominate over the second. Hence, ε_A tends to be greater than the machine epsilon. The value of λ is in the range $(0, 1]$ (consider the 2-norm of

$V_y h$ in (2.33)) if we treat our generalized Jacobian V_y of $\nabla\theta(y)$ as a second derivative of $\theta(y)$.

Therefore, if we assume that $\varepsilon_A\lambda$ is about the value of the machine epsilon, say 10^{-16}, then we get into difficulties with the Armijo rule at the latest when $\|\nabla\theta(y_k)\|_2$ reaches the range of $\sqrt{\varepsilon_A\lambda} \approx \sqrt{10^{-16}} = 10^{-8}$. Consequently, we cannot use an *error_tol* (see Algorithm 1) less than 10^{-8} and expect the Armijo rule to return a reasonable step length.

Note that this derivation applies to any function $f : \mathbb{R}^m \mapsto \mathbb{R}$ that is twice differentiable, the described problem is a general problem of the Armijo backtracking rule.

Approaches to Overcome the Problem

The fact that the alternating projections method by Higham in [17] converges for small values of *error_tol* and that this method is equivalent to the gradient method ($y_{k+1} = y_k - \alpha_k\nabla\theta(y_k)$, with $\alpha_k := 1$) proved in Theorem 5.1 in [23], gives us reasons to pursue other strategies and one idea to overcome the problem, in order to obtain a smaller value of $\|\nabla\theta(y_k)\|_2$ and thus, a more accurate solution.

In the case of having a distance between $\theta(y_{k+1})$ and $\theta(y_k)$ less then ε_A, we could deactivate Armijo backtracking and use a constant step size. Bertsekas provides in [4, Proposition 1.2.3] a constant step size that guarantees convergence.

Theorem 4.7.1. *Let $f : \mathbb{R}^k \mapsto \mathbb{R}$ be a continuously differentiable function. Furthermore, let $\{x_k\}$ be a sequence in $\mathbb{R}^k$ generated by a gradient method $x_{k+1} = x_k + \alpha_k d_k$, where the angle between the gradient of f at x_k and the step direction d_k is bounded away from 90 degrees. Assume that the gradient function $\nabla f(x)$ is Lipschitz continuous with a Lipschitz constant L and that there exists a scalar ε such that for all k we have $d_k \neq 0$ and*

$$0 < \varepsilon \leq \alpha_k \leq \frac{(2-\varepsilon)\left|\nabla\theta(y_k)^T d_k\right|}{L\|d_k\|_2^2}. \tag{4.34}$$

Then every the limit point of $\{x_k\}$ is a stationary point of f.

Hence, since $L = 1$ in our case (consider $f = \theta$) we obtain convergence for any

$\alpha_k \in (0, 2)$ for the steepest descent direction. This includes the gradient method of Malick in [23]. In addition, this issue can also be used if we do not satisfy the inequality of (3.2), we skip the Armijo condition and use the steepest descent direction with a constant step size.

If we use the inexact Newton direction then we get immediately a lower bound for the right-hand side of (4.34) by using the inequality (4.12) which is $(2 - \varepsilon)\varphi_k$. However, φ_k is small, in particular for k large thus the range of a feasible step length is also small and choosing an inexact Newton direction can become disadvantageous.

Assuming that we solve the equality (3.1) exactly leads to another lower bound for the right-hand side of (4.34). Consider

$$\frac{\left|\nabla\theta(y_k)^T d_k\right|}{\|d_k\|_2^2} = \frac{\left|d_k^T V_k d_k\right|}{\|d_k\|_2^2} = |\lambda| \leq \|V_k\|_2 \leq 1 \tag{4.35}$$

with λ the eigenvalue of V_k corresponding to d_k and $\nabla\theta(y_k) := -V_k d_k$ (consider for $\|V_k\|_2 \leq 1$ the 2-norm of $V_y h$ in (2.33)). We obtain that the right-hand side of (4.34) is equal to $(2-\varepsilon)\lambda$ which is less than 2 and can also become clearly less than 2. Unfortunately, the latter fact was confirmed by some numerical tests (not included in the numerical tests in Chapter 6) so that choosing our feasible step length α_k according to Theorem 4.7.1 for our inexact Newton direction could still be disadvantageous.

Another approach to overcome the problem of the Armijo rule proceeds from the assumption that we reach a neighbourhood close to the solution when $|\theta(y_{k+1}) - \theta(y_k)|$ is small so that we can take the inexact Newton direction with step length 1 without checking the Armijo condition. In order to still guarantee convergence, we check whether

$$\frac{\|\nabla\theta(y_{k+1})\|_2}{\|\nabla\theta(y_k)\|_2} < 1 - \mu \text{ with } \mu > 0. \tag{4.36}$$

For the quadratic convergence we have to prove that the inexact Newton direction d_k with step length 1 satisfies (4.36) for all k sufficiently large. This result is obtained by the subsequent lemma since $\|d_k\|_2 \to 0$ as $k \to \infty$.

Lemma 4.7.2. *Let d_k be the computed inexact Newton direction satisfying inequality*

(3.2) for all $k \geq \hat{k}$ and $\hat{k} \geq 0$. Then

$$\frac{\|\nabla\theta(y_{k+1})\|_2}{\|\nabla\theta(y_k)\|_2} \leq \mathcal{O}\left(\|d_k\|_2\right) \tag{4.37}$$

for all k sufficiently large.

Proof. From the proof of [27, Theorem 5.3] (or proof of Theorem 4.6.1 with $\varepsilon = 0$ and k sufficiently large) we know for all k sufficiently large that

$$\|y_{k+1} - y_*\|_2 \leq \mathcal{O}\left(\|y_k - y_*\|_2^2\right) \tag{4.38}$$

and

$$\|y_k - y_*\|_2 \leq \|d_k\|_2 + \mathcal{O}\left(\|d_k\|_2^2\right) \tag{4.39}$$

hold and also that there exists a $\rho > 0$ so that

$$\|\nabla\theta(y_k)\|_2\|d_k\|_2 \geq -\nabla\theta(y_k)^T d_k \geq \rho\|d_k\|_2^2. \tag{4.40}$$

It follows from (4.39) and (4.40) that for all k sufficiently large

$$\frac{\|y_k - y_*\|_2^2}{\|\nabla\theta(y_k)\|_2} \leq \frac{\|y_k - y_*\|_2^2}{\rho\|d_k\|_2} \leq \frac{1}{\rho}\|d_k\|_2 + \mathcal{O}\left(\|d_k\|_2^2\right) \leq \mathcal{O}(\|d_k\|_2). \tag{4.41}$$

Now, from (4.38) using the Lipschitz property of $\|\nabla\theta(y)\|_2$ and (4.41) we deduce the following expression

$$\frac{\|\nabla\theta(y_{k+1})\|_2}{\|\nabla\theta(y_k)\|_2} \leq \frac{\|y_{k+1} - y_*\|_2}{\|\nabla\theta(y_k)\|_2} \leq \mathcal{O}\left(\frac{\|y_k - y_*\|_2^2}{\|\nabla\theta(y_k)\|_2}\right) \leq \mathcal{O}(\|d_k\|_2) \tag{4.42}$$

which completes our proof. □

As Lemma 4.7.2 shows, checking condition (4.36) is an alternative if we cannot use the Armijo rule so that we adopt this approach for our modified version (see Algorithm 2 in Chapter 5). However, if we do not satisfy the condition (4.36) we have to revert to the approach of taking the steepest descent direction with step length 1. Moreover, if the $\|\nabla\theta(y_{k+1})\|_2$ is negligible in $y_{k+1} = y_k + \nabla\theta(y_k)$ then we can terminate Algorithm 1 since our minimal achievable value of $\|\nabla\theta(y_{k+1})\|_2$ will be approached.

4.7.2 Accuracy of the Output Matrix

Here we discuss the accuracy of the output matrix according to $\|\nabla\theta(y_k)\|_2$ with y_k the final iterate and thus the problem that the output matrix is potentially not an element of the feasible set of the problem (1.14) in general. If Algorithm 1 has terminated without an error then the norm of the gradient $\|\nabla\theta(y)\|_2$ at the final iterate y_k is less than or equal to the error tolerance, but of course nonzero in general. Assuming that $\|\nabla\theta(y_k)\|_2 > 0$ then comparing (2.14) and the formula for X_* in Theorem 2.1.2 reveals that the entries of the diagonal of our output matrix are not all 1 whereas the error on the diagonal depends clearly on the value of $\|\nabla\theta(y_k)\|_2$. That means our output matrix X is not in the feasible set of our problem (1.14) since the constraint $\mathrm{diag}(X) = e$ is not satisfied.

Our first approach to tackle the problem is to set the diagonal of X to 1 after the computation. However, this changes the eigenvalues of X so that the modified matrix X may not be in the feasible set still. Another approach is to rescale the diagonal to 1 by multiplying $D^{-1/2}$ from both sides, that is

$$\widetilde{X} := D^{-\frac{1}{2}} X D^{-\frac{1}{2}} \tag{4.43}$$

where $D = \mathrm{Diag}(\mathrm{diag}(X))$. Note that $D \approx I_n$ with I_n the identity. This congruence transformation preserves the positive semidefiniteness (consider if $y^T X y \geq 0 \ \forall y \in \mathbb{R}^n \Rightarrow y^T D^{-\frac{1}{2}} X D^{-\frac{1}{2}} y \geq 0 \ \forall y \in \mathbb{R}^n$ for positive diagonal entries of D) and as a consequence that the eigenvalues, which are changed by the transformation, remain nonnegative. Now the difference between the diagonal elements of $\widetilde{X}$ and a unit diagonal is negligible so that we combine the second approach with the first one, we rescale the diagonal to 1 and set it to 1 afterwards. This makes sure that the diagonal is equal to 1 and that we preserve the positive semidefiniteness.

Note that both approaches can increase the error $\|X - X_*\|_F$ with X our computed solution and X_* the exact solution. We therefore consider how the difference $\|G - X\|_F$ changes when we multiply $D^{-\frac{1}{2}}$ from both sides on X. Let $X = (x_{ij})_{i,j=1}^n$ be the computed solution without changing the diagonal and *error_tol* ≤ 1 our error

tolerance. We have

$$\begin{aligned}\left\|G - D^{-\frac{1}{2}}XD^{-\frac{1}{2}}\right\|_F &= \left\|G - X + X - D^{-\frac{1}{2}}XD^{-\frac{1}{2}}\right\|_F \\ &\leq \|G - X\|_F + \left\|X - D^{-\frac{1}{2}}XD^{-\frac{1}{2}}\right\|_F .\end{aligned} \tag{4.44}$$

Our target now is to find an upper bound of the latter term of (4.44) by using that $\|\nabla\theta(y_k)\|_2 \leq error\_tol$. From (2.14) and (2.22) it follows that

$$D = \mathrm{Diag}(\nabla\theta(y_k)) + I_n. \tag{4.45}$$

Now let a_{ij} be the (i, j) entry of the matrix $X - D^{-\frac{1}{2}}XD^{-\frac{1}{2}}$ so that

$$\left\|X - D^{-\frac{1}{2}}XD^{-\frac{1}{2}}\right\|_F^2 = \sum_{i,j=1}^{n} a_{ij}^2. \tag{4.46}$$

From (4.45) we have

$$\begin{aligned}a_{ij}^2 &= \left(\left(X - D^{-\frac{1}{2}}XD^{-\frac{1}{2}}\right)_{ij}\right)^2 \\ &= \left(x_{ij} - (\nabla\theta(y_k)_i + 1)^{-\frac{1}{2}}(\nabla\theta(y_k)_j + 1)^{-\frac{1}{2}}x_{ij}\right)^2 \\ &= x_{ij}^2\left(1 - (\nabla\theta(y_k)_i + 1)^{-\frac{1}{2}}(\nabla\theta(y_k)_j + 1)^{-\frac{1}{2}}\right)^2.\end{aligned} \tag{4.47}$$

Using $\nabla\theta(y_k)_i \leq error\_tol \ \ \forall i$ yields

$$\frac{1}{1 + error\_tol} \leq (\nabla\theta(y_k)_i + 1)^{-\frac{1}{2}}(\nabla\theta(y_k)_j + 1)^{-\frac{1}{2}} \leq \frac{1}{1 - error\_tol}. \tag{4.48}$$

Hence, in order to find an upper bound for a_{ij}^2 it is enough to maximize the function $f : J \mapsto \mathbb{R}$ with

$$f(s) := \left(1 - \frac{1}{1 + s}\right)^2 \tag{4.49}$$

and $J := [-error\_tol\ , error\_tol\ ] \subset \mathbb{R}$. Since

$$\mathrm{argmax}_{s\in J}(f(s)) = \mathrm{argmax}_{s\in J}\left(\frac{s^2}{(1 + s)^2}\right) = -error\_tol \tag{4.50}$$

we obtain from (4.47)

$$\begin{aligned}a_{ij}^2 &\leq x_{ij}^2\left(1 - \frac{1}{1 - error\_tol}\right)^2 \\ &= x_{ij}^2\left(\frac{error\_tol}{1 - error\_tol}\right)^2.\end{aligned} \tag{4.51}$$

Hence, $\left\|X - D^{-\frac{1}{2}} X D^{-\frac{1}{2}}\right\|_F \leq \frac{error\_tol}{1-error\_tol}\|X\|_F$ and we obtain the following upper bound for our distance

$$\left\|G - D^{-\frac{1}{2}} X D^{-\frac{1}{2}}\right\|_F \leq \|G - X\|_F + \frac{error\_tol}{1 - error\_tol}\|X\|_F. \tag{4.52}$$

That means the smaller our error tolerance *error_tol* the smaller our upper bound and in this respect the closer the transformed matrix to the input matrix G.

4.8 Choosing the Method for the Eigendecomposition

In Algorithm 1 a full eigenvalue decomposition of $G + \mathrm{Diag}(y)$ is required for every function evaluation of $\theta(y)$, thus at least 1 eigenvalue decomposition in every iteration. Moreover, obtaining such an eigenvalue decomposition is expensive and consequently computing all required eigenvalue decompositions is an expensive part of the algorithm. Qi and Sun use the MATLAB function `eig` calling the LAPACK routine `DSYEV` to compute the spectral decompositions. In order to may reduce this computational time we investigate three different codes `F08FAF` (`DSYEV`), `F08FCF` (`SSYEVD`/`DSYEVD`) and `F08FDF` (`DSYEVR`) provided by NAG Fortran Library, Mark 21 to perform an eigenvalue decomposition of a symmetric matrix. In this section, we explain briefly these three codes and in Section 6.4 we test which algorithm performs best in terms of the used time.

All three codes reduce the symmetric matrix to a tridiagonal matrix, say T, by using orthogonal similarity transformations but then proceed differently. The code `F08FCF` uses a divide and conquer algorithm to compute the eigenvalues and eigenvectors whereas `F08FAF` applies a QR algorithm to the tridiagonal matrix T. The last code `F08FDF` computes the eigenvalues of T by the dqds algorithm and the corresponding eigenvectors from various 'good' LDL^T representations. For more details see in [25] or [2] and the references therein.

4.9 The Case of G Nonsymmetric

Here we clarify what we can do if our given matrix $G \in \mathbb{R}^{n\times n}$ is nonsymmetric. We want to find a solution of the problem

$$\min_{X\in\mathcal{C}} \|G - X\|_F \tag{4.53}$$

where G is nonsymmetric and $\mathcal{C}$ is the set of all correlation matrices in $\mathbb{R}^{n\times n}$. This problem can be solved by transforming it into an equivalent problem which has the form of our standard problem in (1.14). The equivalent problem is,

$$\min_{X\in\mathcal{C}} \left\| \frac{1}{2}\left(G + G^T\right) - X \right\|_F. \tag{4.54}$$

This follows by applying the subsequent theorem [14].

Theorem 4.9.1. *For $A \in \mathbb{R}^{n\times n}$ the solution of*

$$\min_{X\in\mathcal{C}} \|A - X\|_F \tag{4.55}$$

is also a solution of

$$\min_{X\in\mathcal{C}} \left\| \frac{1}{2}\left(A + A^T\right) - X \right\|_F \tag{4.56}$$

and vice versa.

Proof. We prove first that the following equation holds

$$\|T + K\|_F^2 = \|T\|_F^2 + \|K\|_F^2 \tag{4.57}$$

with $T = T^T \in \mathbb{R}^{n\times n}$ and $K = -K^T \in \mathbb{R}^{n\times n}$. We have

$$\|T + K\|_F^2 = \|T\|_F^2 + \|K\|_F^2 + 2\langle T, K\rangle. \tag{4.58}$$

Now we consider the last term only since it remains to prove that this term is equal

to 0. We use that $T = T^T, K = -K^T$ and obtain

$$\begin{aligned} 2\langle T, K\rangle &= \langle T, K\rangle - \langle T^T, K^T\rangle \\ &= \langle T, K\rangle - \text{trace}\left(TK^T\right) \\ &= \langle T, K\rangle - \text{trace}\left((TK^T)^T\right) \\ &= \langle T, K\rangle - \text{trace}\left(KT^T\right) \\ &= \langle T, K\rangle - \text{trace}\left(T^TK\right) \\ &= \langle T, K\rangle - \langle T, K\rangle \\ &= 0 \end{aligned} \tag{4.59}$$

Hence, the equation (4.57) holds. Let $B := \frac{1}{2}\left(A - X + (A - X)^T\right) = \frac{1}{2}\left(A + A^T\right) - X$ and $C := \frac{1}{2}\left(A - X - (A - X)^T\right) = \frac{1}{2}\left(A - A^T\right)$ be the symmetric and skew-symmetric parts of $A - X$ with X a correlation matrix. Note that $A - X = B + C$, $B = B^T$ and $C = -C^T$.

Applying the equation (4.57) with $T = B$ and $K = C$ to our problem (4.55) leads to

$$\begin{aligned} S = \text{argmin}_{X\in\mathcal{C}}\|A - X\|_F &= \text{argmin}_{X\in\mathcal{C}}\|A - X\|_F^2 \\ &= \text{argmin}_{X\in\mathcal{C}}\left(\|B\|_F^2 + \|C\|_F^2\right) \\ &= \text{argmin}_{X\in\mathcal{C}}\left(\left\|\frac{1}{2}\left(A + A^T\right) - X\right\|_F^2 + \left\|\frac{1}{2}\left(A - A^T\right)\right\|_F^2\right) \\ &= \text{argmin}_{X\in\mathcal{C}}\left\|\frac{1}{2}\left(A + A^T\right) - X\right\|_F^2 \\ &= \text{argmin}_{X\in\mathcal{C}}\left\|\frac{1}{2}\left(A + A^T\right) - X\right\|_F \end{aligned}$$

which proves our assertion. □

Chapter 5

The Modified Algorithm

The following algorithm is our modification of Algorithm 1. We have changed Algorithm 1 based on the discussion in Chapter 4 but we do not yet specify which method we use for our spectral decomposition and for solving our linear system. We will mention these particular methods in Section 6.7 where we draw our conclusions of the numerical tests in Chapter 6. Note that our changes do not affect the convergence analysis of Section 3.2 apart from taking the steepest descent direction when Armijo backtracking cannot be applied and the condition (5.5) (see below) cannot be satisfied. However, since by Lemma 4.7.2 this condition is satisfied for k sufficiently large this algorithm still converges to the solution of our problem quadratically. For the implementation of this algorithm see Section A.2 of Appendix A.

Algorithm 2. Given $G \in \mathbb{R}^{n\times n}$ this quadratically convergent algorithm computes the nearest correlation matrix X to G in the Frobenius norm. On termination $\|\nabla\theta(y_k)\|_2 \leq \mathit{error\_tol}$ (see (2.14) for the formula of $\nabla\theta(y)$) with *error_tol* the given error tolerance.

Step 1: Set the starting values: $y_0 \in \mathbb{R}^n$ with $y_0 := e - \mathrm{diag}(G)$, $\eta = 0.5$, $\varphi = 10^{-6}$, $\mu \in (0,1)$, ρ, $\sigma \in (0, 1/2)$ and $k := 0$.

Step 2: Set $G := \frac{G+G^T}{2}$ if G is nonsymmetric.

Step 3: Calculate $\nabla\theta(y_k)$. If $\|\nabla\theta(y_k)\|_2 \leq \mathit{error\_tol}$ compute X with (2.22) ($y_* :=$

y_k, $X := X_*$) and exit.

Step 4: Perform a spectral decomposition of $G + \mathrm{Diag}(y_k)$, compute the matrix W_{y_k} and the preconditioner M_k defined by (2.35) and (4.6), respectively.

Step 5: Determine the new direction d_k for the step k by applying an iterative method to the preconditioned linear system (using formula (4.5) to compute $V_k d_k$)

$$V_k d_k = -\nabla\theta(y_k) \tag{5.1}$$

such that the conditions

$$\|\nabla\theta(y_k) + V_k d_k\|_2 \leq \eta_k \|\nabla\theta(y_k)\|_2, \qquad \text{for } \eta_k = \min(\eta, \|\nabla\theta(y_k)\|_2) \tag{5.2}$$

and

$$-\frac{\nabla\theta(y_k)^T}{\|d_k\|_2} \cdot \frac{d_k}{\|d_k\|_2} \geq \min(\varphi, \|\nabla\theta(y_k)\|_2) \tag{5.3}$$

are satisfied. If either one of these conditions cannot be satisfied, let

$$d_k := -\nabla\theta(y_k). \tag{5.4}$$

Step 6: Test whether Armijo backtracking can be applied regarding the discussions in Section 4.7.1. If the test is positive goto Step 7.

If not, set $\alpha_k := 1$ and test whether step length 1 is sufficient to satisfy the global convergence, that is if

$$\frac{\|\nabla\theta(y_k + \alpha_k d_k)\|_2}{\|\nabla\theta(y_k)\|_2} \leq 1 - \mu. \tag{5.5}$$

If (5.5) is satisfied goto Step 8, otherwise set $d_k := -\nabla\theta(y_k)$ and goto Step 8 if d_k is not too small relative to y_k or else, exit.

Step 7: Choose an appropriate step length α_k by applying Armijo backtracking: find the smallest number $m_k \in \mathbb{N}_0$ (start testing whether (5.6) is satisfied with $m_k = 0$ and increment by 1 repeatedly) such that

$$\theta(y_k + \rho^{m_k} d_k) - \theta(y_k) \leq \sigma\rho^{m_k} \nabla\theta(y_k)^T d_k \tag{5.6}$$

is satisfied. During the Armijo backtracking check after every increment of m_k whether Armijo backtracking can be applied again. If yes, continue, otherwise proceed analogously to Step 6 with $\alpha_k := \rho^{m_k}$. If Armijo backtracking was successful, set $\alpha_k := \rho^{m_k}$.

Step 8: Set $y_{k+1} = y_k + \alpha_k d_k$ and $k \leftarrow k+1$. If the relative change of $\|\nabla\theta(y_{k+1})\|_2$ is sufficiently large go to Step 3, otherwise exit.

Chapter 6

Numerical Tests

In this chapter we test our implemented version of Algorithm 2 for efficiency and robustness. The first tests in Section 6.2, 6.3, 6.4 are performance tests where we investigate the discussions in Chapter 4 to reduce the computation time of the algorithm. In Section 6.5 we test the robustness of our algorithm by means of direct optimization methods and finally in Section 6.6 we compare Algorithm 2 with the Qi and Sun's algorithm (Algorithm 1) and Higham's alternating projections method.

6.1 Information about our Tests

6.1.1 Software and Hardware

Our numerical tests are performed in MATLAB using version 7.3.0.298 (R2006b) on 2.4 GHz AMD Athlon(tm). The unit roundoff for this MATLAB version is $u = 2^{-53} \approx 1.1 \cdot 10^{-16}$.

6.1.2 Applied Examples

We use six kinds of test examples.

Example 1

We generate randomly a correlation matrix $C \in \mathbb{R}^{n\times n}$ by using `gallery('randcorr', n)` of MATLAB which uses an algorithm described in [5]. Furthermore, we generate a symmetric matrix $M \in \mathbb{R}^{n\times n}$ by means of the function `rand` of the same MATLAB version with elements from a uniform distribution in the range $[-\beta, \beta]$ where β is given. Subsequently, we add these two matrices and obtain our test matrix $G = C + M$.

Example 2

Here, we generate randomly a symmetric matrix $G \in \mathbb{R}^{n\times n}$ also by using the MATLAB function `rand` which is uniformly distributed and has its values $G_{ij} \in [-1, 1]$. Additionally, we set $G_{ii} = 1$ for all $i = 1, \ldots, n$.

Example 3

Here we generate randomly a correlation matrix $A \in \mathbb{R}^{n\times n}$ again by means of the MATLAB function `gallery('randcorr',n)` and then we select randomly $i, j \in \{1, \ldots, n\}$ ($i = j$ is also allowed) and change the entry A_{ij} of the matrix A to a randomly generated value in the range of $[-1, 1]$. We repeat this procedure $n/10 - 1$ times (if the value $n/10$ is not an integer we will round the value up). In both cases the determined random values are uniformly distributed, obtained from the MATLAB function `rand`. In order to preserve the symmetry, we set $A_{ji} := A_{ij}$ after every modification.

Example 4

In this example, we generate randomly two correlation matrices A and B by means of the function `gallery('randcorr',n)` whereas the first correlation matrix A has size n and the second B size $n/2$. Assume here that $n/2$ is an integer: if not we round this value up to the next largest integer by means of the MATLAB function `ceil`. Now we substitute the leading part $A(1\colon n/2, 1\colon n/2)$ by the second generated

matrix B. If the resulting matrix has only nonnegative eigenvalues we will repeat the procedure until at least one eigenvalue is negative.

The idea of this example is deduced from the assumptions of Finger's algorithm in Section 1.5 where we assumed that the leading part C_{11} contains all correlations of random variables which are desired to be changed. That means it is desired to substitute the matrix C_{11} by a matrix $\widetilde{C}_{11}$ which is potentially a correlation matrix.

Example 5

This example consists of two symmetric matrices from stock data provided by a fund management company which are approximate correlation matrices with ones on the diagonal. The first one has a dimension of $n = 1399$, is highly rank-deficient and its entries are in the interval $[-0.9644, 1.1574]$ on the off-diagonals. The second one is larger with a dimension of $n = 3120$ and has full rank. Its values on the off diagonal are in the interval of $[-0.6250, 1.0751]$.

Example 6

The last example contains matrices from the RiskMetrics database which we obtained from RiskMetrics website [1]. We get the following information about the content of the provided data sets from there.

> "The data sets contain consistently calculated volatilities and correlation forecasts for use in estimating market risk. The asset classes covered are government bonds, money markets, swaps, foreign exchange and equity indices (where applicable) for 31 currencies, and commodities."

We obtained two matrices for a one day and a one month horizon assigned to July 15, 2006 which have the size of $n = 387$ and a smallest eigenvalue of $-7.92 \cdot 10^{-6}$ and $-4.91 \cdot 10^{-6}$, respectively.

6.1.3 Stopping Criterion

We stop our algorithm if

$$\|\nabla\theta(y_k)\|_2 \leq error\_tol$$

where $error\_tol = 10^{-7} \cdot n$ with n the size of our given symmetric matrix G.

6.1.4 Other Considerations

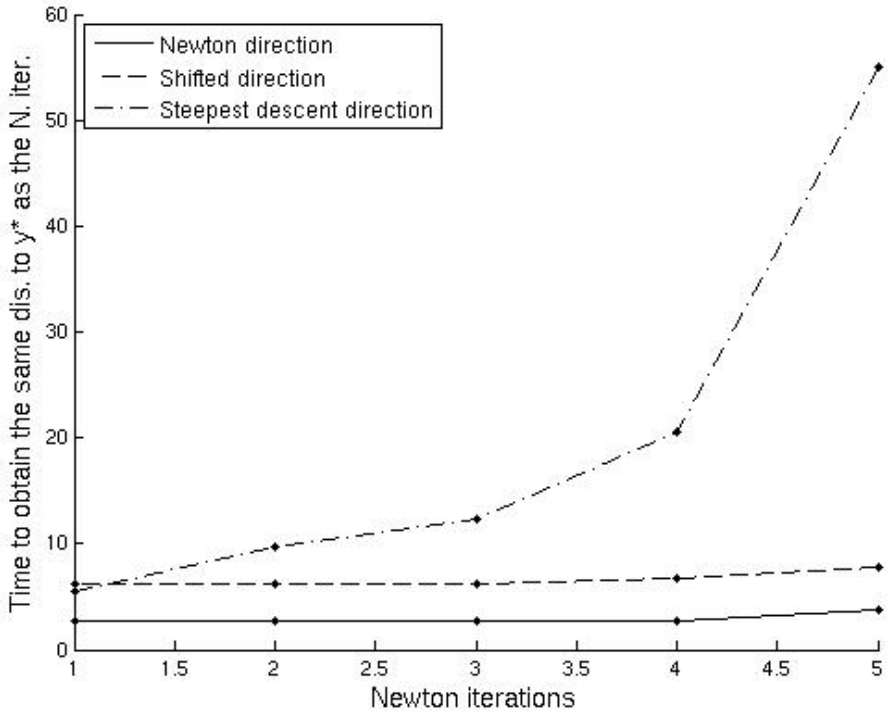
Unless otherwise stated, we use MINRES as our method for solving (5.1) and apply the diagonal preconditioner suggested in Section 4.4.2 for all tests. Furthermore, for the required eigenvalue decomposition we access the NAG Fortran Library, Mark 21 by means of the MATLAB-NAG Toolbox and the use routine `f08fd`. What is more, $\varphi = 10^{-6}$ in (5.3), $\eta_k = \min(0.5, \|\nabla\theta(y)\|_2)$ in (5.2), $\mu = 0.9$ in (5.5), $\sigma = 10^{-4}$ and $\rho = 0.5$ in (5.6). According to the discussions in Section 4.7.2, we do not change the determined matrix after its computation for our tests.

6.2 Comparison of Different Step Directions

Here, we want to demonstrate the difference of taking the negative gradient direction, the step direction as in Section 4.6 suggested and the Newton direction. We proceed in the following way. We first compute the minimizer of our problem by starting our algorithm choosing always the Newton direction. Afterwards, we start our algorithm again where we compute our iterate y_k several times, i.e. we take the Newton direction first, compute the iterate y_k and save the difference in the Frobenius norm to our minimizer. Now we set $y_k := y_{k-1}$ and take for example the negative gradient direction until the difference of the minimizer and the iterate y_{k+l} is less than or equal to the difference produced by taking the Newton direction. We repeat this procedure for all step directions and start then again with the last iterate y_{k+1} computed by the Newton method. We proceed in this way until we satisfy our stopping criterion. When the stopping criterion is satisfied we compare the time in seconds which has been taken by the particular step direction at every iteration referring to the Newton

iterate.

For this purposes we take Example 1 with $\beta = 0.5$ and a size of $n = 700$ and we use an error tolerance of $error\_tol = n \cdot 10^{-12}$. We report our result in Figure 6.1 where we use the two abbreviations dis. and N. iter. which stand for distance and Newton iteration, respectively.



(6.1)

Figure 6.1: Comparison of different step directions

We observe from Figure 6.1 that computing the Newton direction requires approximately the same amount of time at every iteration whereas when taking the steepest descent direction the computation time increases in every step. Furthermore, more time is always required when taking the steepest descent direction in comparison to the Newton direction. The choice of determining the direction as described in Section 4.6 is not as good as the Newton direction, but gives similar results to the Newton direction and is clearly better than the negative gradient direction.

This result is reasonable since in order to compute both the Newton direction

and the shifted direction a more accurate approximation of the function $\theta(y)$ is used which finally guarantees quadratic convergence for y_0 close to the solution as discussed in Chapter 2 and in Section 4.6. Furthermore, for computing the steepest descent direction also an eigenvalue decomposition is required (see (2.14)) and as we will see later computing the eigenvalue decomposition is an expensive part in our algorithm. Hence, taking the negative gradient direction is not tremendously less expensive than taking the Newton direction.

Since the additional cost of computing the Newton direction is obviously also worth for the first steps (the other examples of Section 6.1.2 also confirm these results in our tests which we omitted here) we conclude that the Newton direction should be taken in every step. The choice of the shifted direction is obviously advisable if we cannot solve the equation (3.1) but using it instead of the Newton-direction is not recommended according to the behaviour of our algorithm for our tested examples. That is why we abandon the approach of choosing the shifted direction and do not change our Algorithm 2.

6.3 Testing Iterative Methods plus Preconditioner

In this section, we test all the methods which we introduced in Section 4.3 using Example 1 and Example 5. The size of the matrix G is always $n = 700$ when it can be chosen and the value of β is stated in the Table 6.1 where we report our numerical results.

We access for the methods CG, SYMMLQ, RGMRES (restarting value is 3), TFQMR and BI-CGTAB (value $l = 1$, we denote this method as BS in Table 6.1) the NAG Fortran Library, Mark 21 by means of the MATLAB-NAG Toolbox where we call the routines `f11bd`, `f11be` and `f11bf`. For the MINRES method we use an implementation in MATLAB obtained from the authors of [7]. In the Table 6.1 Time Tot., Time Mvp., Time Eig., Iter., Calls Mvp. and Time Pre., respectively, stand for the total time used to run the algorithm, the time spent to compute all the matrix-vector products $V_k h$ (see (2.33)) in the algorithm which is essentially the same time

to solve the equation (5.1) summed over all iterations, the time used for computing all spectral decompositions, the iteration number in the outer loop, the number of computed matrix-vector products $V_k h$ and the time to compute all preconditioners. The time is measured in seconds.

Table 6.1: Comparison of different iterative methods

	CG	**SYMMLQ**	**RGMRES**	**TFQMR**	**MINRES**	**BS**
Example 1, $\beta = 0.01$, *No Preconditioning*						
Time Tot.	9.25	8.27	5.3	4.81	7.31	4.81
Time Mvp.	4.33	3.37	1.93	1.50	2.40	1.50
Time Eig.	4.33	4.3	2.87	2.87	4.3	2.87
Iter.	3	3	2	2	3	2
Calls Mvp.	9	7	4	3	5	3
With Preconditioning						
Time Tot.	9.71	8.75	5.53	5.06	7.78	5.05
Time Mvp.	4.33	3.36	1.92	1.51	2.40	1.51
Time Eig.	4.30	4.31	2.87	2.87	4.3	2.87
Iter.	3	3	2	2	3	2
Calls Mvp.	9	7	4	3	5	3
Time Pre.	0.48	0.48	0.24	0.24	0.48	0.24
Example 1, $\beta = 0.1$, *No Preconditioning*						
Time Tot.	12.7	11.26	12.23	10.84	9.81	10.76
Time Mvp.	6.25	4.81	5.79	4.34	3.36	4.33
Time Eig.	5.74	5.77	5.76	5.83	5.73	5.73
Iter.	4	4	4	4	4	4
Calls Mvp.	13	10	12	9	7	9
With Preconditioning						
Time Tot.	13.54	11.99	12.96	11.54	10.53	11.49
Time Mvp.	6.43	4.81	5.80	4.39	3.36	4.33
Time Eig.	5.74	5.77	5.76	5.75	5.73	5.73

	CG	**SYMMLQ**	**RGMRES**	**TFQMR**	**MINRES**	**BS**
Iter.	4	4	4	4	4	4
Calls Mvp.	13	10	12	9	7	9
Time Pre.	0.75	0.72	0.72	0.72	0.72	0.72
Example 1, $\beta = 1$, *No Preconditioning*						
Time Tot.	24.79	21.75	21.09	22.64	18.39	17.92
Time Mvp.	14.04	11.03	11.55	11.58	7.70	8.18
Time Eig.	10.25	10.17	8.64	10.0	10.09	8.66
Iter.	7	7	6	7	7	6
Calls Mvp.	28	22	24	24	16	17
With Preconditioning						
Time Tot.	26.53	23.15	20.39	18.93	19.36	18.94
Time Mvp.	15.02	12.27	9.63	8.20	7.25	8.52
Time Eig.	10.3	10.1	8.64	8.64	10.08	8.64
Iter.	7	7	6	6	7	6
Calls Mvp.	28	22	20	17	15	17
Time Pre.	1.45	1.46	1.21	1.21	1.45	1.21
cor1399, *No Preconditioning*						
Time Tot.	344.4	322.4	266.13	249.3	314.6	245.06
Time Mvp.	150.6	128.4	99.92	82.64	137.8	85.63
Time Eig.	178.7	178.7	153.1	153.1	178.6	153.1
Iter.	7	7	6	6	7	6
Calls Mvp.	42	36	28	23	30	36
With Preconditioning						
Time Tot.	253.12	235.0	245.7	235.3	229.14	245.3
Time Mvp.	78.6	60.6	72.1	61.7	47.4	60.4
Time Eig.	153.2	153.2	153.1	153.1	153.1	153.1
Iter.	6	6	6	6	6	6
Calls Mvp.	22	17	20	17	13	17

	CG	**SYMMLQ**	**RGMRES**	**TFQMR**	**MINRES**	**BS**
Time Pre.	8.83	8.7	8.76	8.76	8.9	8.7
cor3120, *No Preconditioning*						
Time Tot.	4384	4154	3423	3764	3208	3731
Time Mvp.	2224	1990	1880	1914	1332	1845
Time Eig.	2141	2139	1531	1835	1837	1835
Iter.	7	7	5	6	6	6
Calls Mvp.	57	51	48	49	31	47
With Preconditioning						
Time Tot.	2284	2137	2247	2095	2000	2142
Time Mvp.	664.7	506.8	624.6	470.3	352.8	470.2
Time Eig.	1530	1527	1527	1530	1531	1528
Iter.	5	5	5	5	5	5
Calls Mvp.	17	13	16	12	9	12
Time Pre.	72.9	73.0	72.8	72.84	72.89	73.12

We conclude from Table 6.1 that first the time for computing our preconditioner is as expected in Section 4.4.2 small relative to the total time. Furthermore, if we use a preconditioner the calls for the matrix-vector products are reduced in most cases which indicates that our iterate in the iterative methods converges faster and thus the total time is also reduced as can be seen particularly for the two examples from stock data. However, we observe that the total time is not always reduced and the number of matrix-vector products is the same with the preconditioner, so obviously how well the preconditioner improves the convergence behaviour depends on our input matrix G. However, in such a case the total time using our preconditioner is only slightly larger than the total time using no preconditioner.

If we compare the methods using our preconditioner, then the MINRES routine gives us the best results overall, in particular, if we consider the iterative methods for symmetric matrices (CG, SYMMLQ, MINRES). This is expected because MINRES reduces the residual in every step and is rather suitable for nearly singular matrices.

Moreover, methods like RGMRES which are for general matrices cannot perform better in general than methods for symmetric matrices. Thus we also have to consider that we used for the MINRES routine a MATLAB implementation so that the performance of that routine can be improved potentially (so for example the performance of while and for loops) when transforming it into another language like *Fortran*.

We observe further that a large part of the total time to run the algorithm is spent in computing the eigenvalue decomposition. Considering the larger matrix from stock data in the case of preconditioning approximately 70% of the total time is required for the spectral decomposition. This fact motivates the test in the next section where different methods are tested to compute the eigenvalue decomposition.

6.4 Testing Methods for the Eigendecomposition

Here we compare the three methods that we briefly introduced in Section 4.8 for the eigenvalue decomposition of $G+\text{Diag}(y_k)$ in Algorithm 2. We test the methods `f08fa`, `f08fc`, `f08fd` of the MATLAB-NAG Toolbox, i.e. we access the NAG Fortran Library, Mark 21 in MATLAB again and test actually `F08FAF` (QR-algorithm), `F08FCF` (divide and conquer algorithm) and `F08FDF` (dqds algorithm) of that Fortran Library. We use the Example 1, 2, 3 and 4 in Section 6.1.2 for our testing purposes with different sizes of the matrix $n = 500, 1000$ and choose the value of β as stated in Table 6.2 if it is required. Furthermore, we also test Example 5 where the particular matrix is indicated by the size. We compare the total time in seconds that is used to run the Algorithm 2 with the different solvers. We report our results in Table 6.2.

Table 6.2: Different methods for the eigenvalue decomposition

	f08fa	**f08fc**	**f08fd**
Example 1	**Used Time**		
$n = 500$			
$\beta = 0.01$	4.54	1.68	1.84

	f08fa	**f08fc**	**f08fd**
$\beta = 1$	16.8	7.42	7.98
$\beta = 10$	26.92	12.76	13.59
$n = 1000$			
$\beta = 0.01$	51.97	20.4	20.83
$\beta = 1$	127.56	51.75	52.94
$\beta = 10$	221.28	94.4	96.36
Example 2			
$n = 500$	17.11	7.48	8.08
$n = 1000$	122.87	50.31	51.47
Example 3			
$n = 500$	8.32	4.27	4.55
$n = 1000$	46.43	18.83	19.35
Example 4			
$n = 500$	4.33	1.67	1.83
$n = 1000$	52.13	20.29	20.76
Example 5			
$n = 1399$	238.02	111.51	210.01
$n = 3120$	2122.6	907.79	1974.1

In Table 6.2 we observe clearly that the method `f08fc` (divide and conquer algorithm) performs best in all examples in terms of the used time. Furthermore, we notice that the difference of the used time between the method can be enormous, consider Example 5 using method `f08fc` leads to a reduction of more 50% of the used time in comparison to using the method `f08fa` (QR-algorithm).

6.5 Robustness Testing

6.5.1 Direct Optimization Methods

In this section, we test the robustness and reliability of our algorithm. For that purpose we use direct search methods. These methods search for a local maximum of a function $f : \mathbb{R}^m \mapsto \mathbb{R}$ heuristically, that is

$$\max_{x \in R^m} f(x) \tag{6.2}$$

and involve only function values in their computation; derivatives of f are not computed and also not approximated. Various direct search methods have been developed as described in [15], however, we restrict our investigation by means of only two direct search methods which were treated by Higham also in [15]. These two methods are the alternating directions (AD) method and the multidirectional search (MDS) method. The alternating directions method attempts to maximize the function $f(x)$ by a line search over each coordinate, starting from a given value x, for all $i = 1, \ldots, m$ we attempt to find an α_i such that

$$\alpha_i = \text{argmax}_{\alpha \in \mathbb{R}} f(x + \alpha e_i) \tag{6.3}$$

and update our iterate $x := x + \alpha_i e_i$ subsequently. This procedure is repeated until the iterate x converges. The stopping criterion is a small relative increase in f between one iteration and the next. To find the α_i Higham used in [15] a crude scheme. He starts with a small value for α then he doubles the value of α if $f(x + \alpha e_i) > f(x)$, otherwise he changes the search direction (he reverses the sign of α). Now α is doubled as long as $f(x + \alpha e_i) > f(x)$ and at most 25 times. After this procedure α_i is set to α.

In the multidirectional search method of Dennis and Torczon [33], [34] we maximize over more than one direction. Starting from a simplex $\{v_i\}_{i=0}^m$ with $v_i \in \mathbb{R}^m$ and $f(v_0) = \max_i f(v_i)$ the method attempts to find along the lines joining v_i and v_0 a new vertex $v_0 + \alpha(v_j - v_0)$ for $j \in \{1, \ldots, m\}$ at which the function value is greater than $f(v_0)$ so that a new simplex can be formed from v_0 and the new vertices

$v_0 + \alpha(v_i - v_0)$. In the next iteration $v_j = \text{argmax}_{v_i} f(v_i)$ is taken as the point of intersection of the lines joining v_i and v_j. This procedure is repeated until the iterate v_j converges.

α is determined by a similar method to AD. First we do a reflection step, that means we set $\alpha := -1$. If f at one of the new vertices $v_0 + \alpha(v_i - v_0)$ is greater than $f(v_0)$ a new simplex is formed by v_0 and the new vertices, then α is doubled and the same test is done for the doubled size. If the test is successful the simplex is renewed by the new vertices with the doubled size of α and our iteration is complete. If the reflection step is unsuccessful, that is $f(v_0) \geq \max_i f(v_0 + \alpha(v_i - v_0))$ we reverse the search direction. That means we set $\alpha = 1/2$ and perform our test again. If we can form a new simplex with the desired properties the current iteration is complete, otherwise we jump back to the reflection step and work with half of length of α again. For further details of the MDS method, see [15], [33], [34] and the references therein.

6.5.2 Using these Methods for our Algorithm

Now we test our algorithm by means of the two methods described in the last section. Thereby we use the implementations of the author in [15] which are two m-files `adsmax` for AD and `mdsmax` for MDS; these implementations are available from the Matrix Computation Toolbox [13].

Our testing purposes are finding a matrix where our algorithm fails. Our algorithm may fail by arriving at an infinite loop, not returning a reasonable matrix to the input matrix or returning an error in spite of reasonable input parameters.

In order to detect such a behaviour we form a function according to (6.2) which is to be maximized. We try to increase

- The iterations of the algorithm.
- The function evaluations of the algorithm.

Therefore, we write a function in MATLAB which has a vector x as input parameter and the particular item which we want to increase as output parameter. The vector

$x \in \mathbb{R}^{n(n+1)/2}$ comprises the upper triangular part and the diagonal of the matrix for which we want to run our Algorithm 2 strung out into one long vector. Internally, this function reforms the vector into the corresponding symmetric matrix and runs our algorithm for that matrix. Now we have formed our function f for which we start our direct optimization methods.

We choose Example 1 as starting value with $\beta = 0.1$, 10 and Example 3 comprised as a vector, our size is $n = 15$. The chosen size of the matrix is small because, otherwise, the number of free parameters would be too large for the direct optimization methods to complete in reasonable time. To run our algorithm we use the machine epsilon determined by the MATLAB function `eps` as error tolerance in order to test whether the algorithm terminates properly even for too small tolerances. In other words, we expect to obtain a warning that the machine precision is limiting the convergence.

Now we run our direct optimization methods starting with `adsmax` using the starting value described above and subsequently `mdsmax` where we take the output vector of `adsmax` as the new starting vector for `mdsmax`. We terminate `adsmax` (respectively `mdsmax`) if the relative increase in f (respectively the relative size of the simplex) is less than 10^{-8}.

Once a vector has been returned by `mdsmax`, we reshape this vector to the corresponding matrix and run Algorithm 2 on that matrix and on our original starting matrix with the same parameters displaying intermediate outputs in order to notice the difference in the behaviour of the algorithm.

In Table 6.3 we report the worst case out of 30 runs of the direct optimization methods for both when we maximize the number of iterations and the number of function evaluations. In other words, we show for both cases the behaviour of Algorithm 2 for the starting matrix and the matrix returned by the direct optimization methods. In Table 6.3 we indicate for every matrix in order, the number of iterations, function evaluations and inner iterations summed over all outer iterations which the algorithm requires, the smallest and the largest eigenvalue, how often the Newton direction is taken, how often the negative gradient is taken and the distance $\|G - X\|_F$. Note

that we also indicate when taking the Newton direction whether we have activated or deactivated the Armijo backtracking rule.

In order to notice whether our algorithm returns a reasonable matrix to the matrix which is determined by the direct optimization methods we also run the alternating projections method of Higham [17] for that matrix and compare then the distances of the input matrix to the returned matrix in the Frobenius norm. For that purposes we include a further row for every example where we indicate the distance $\|G - X\|_F$ with X computed by the alternating projections method.

In the column "↑Iter." we show the described data for the two matrices when we maximize the number of iterations and in the column "↑F_eval." when we maximize the function evaluations. Each column is divided into two further columns where every first column "St. M." indicates the data for the starting matrix and every second column "Re. M." for the returned matrix.

For simplicity we use the abbreviations St., Re., M., Iter., F_eval., Sm., La., eig., Num., tak. neg. grad. d., tak. New. d. and AP in Table 6.3 which stand for starting, returned, matrix, iterations, function evaluations, smallest, largest, eigenvalue, number, taking negative gradient direction, taking Newton direction and alternating projections, respectively.

Table 6.3: Robustness testing

	↑Iter.		**↑F_eval.**	
	St. M.	**Re. M.**	**St. M.**	**Re. M.**
Example 1, $\beta = 0.1$				
Iter.	1	10	5	9
F_eval.	3	15	7	15
Inner Iterations	0	0	0	0
Sm. eig.	0.077	-0.17	-0.156	-0.272
La. eig.	1.83	2.09	1.71	2.08
Num. tak. New. d.	0	6	4	4

	↑Iter.		↑F_eval.	
	St. M.	**Re. M.**	**St. M.**	**Re. M.**
Armijo activated	0	3	2	3
Armijo deactivated	0	3	2	1
Num. tak. neg. grad. d.	1	4	1	5
$\|G - X\|_F$ Newton method	0.2275	0.3397	0.34511	0.40479
$\|G - X\|_F$ AP method	0.2275	0.3397	0.34511	0.40479
Example 1, $\beta = 10$				
Iter.	9	15	9	16
F_eval.	11	18	11	47
Inner Iterations	0	1	0	29
Sm. eig.	-40.93	-46.8	-36.6	-53.4
La. eig.	40.53	42.9	40.6	68.3
Num. tak. New. d.	9	15	9	16
Armijo activated	8	9	7	10
Armijo deactivated	1	6	2	6
Num. tak. neg. grad. d.	0	0	0	0
$\|G - X\|_F$ Newton method	86.59	92.54	77.37	116.8
$\|G - X\|_F$ AP method	86.59	92.54	77.37	116.8
Example 3				
Iter.	5	10	5	10
F_eval.	7	15	7	16
Inner Iterations	0	0	0	0
Sm. eig.	-0.025	-0.025	-0.203	-0.21
La. eig.	2.388	2.388	2.65	2.62
Num. tak. New. d.	4	6	4	5
Armijo activated	2	2	3	3
Armijo deactivated	2	4	1	2
Num. tak. neg. grad. d.	1	4	1	5

	↑Iter.		↑F_eval.	
	St. M.	**Re. M.**	**St. M.**	**Re. M.**
$\|G - X\|_F$ Newton method	0.02785	0.02788	0.2339	0.2707
$\|G - X\|_F$ AP method	0.02785	0.02788	0.2339	0.2707

As Table 6.3 shows the direct optimization methods find examples which increase the used number of iterations up to 15 likewise the number of function evaluations which is increased up to 47. Nevertheless, we observe that the algorithm returns reasonable matrices for every example if we compare the distance $\|G - X\|_F$ with that computed by the alternating projections method. Furthermore, the algorithm always terminates with the warning that the precision is limiting the convergence which is forced by the small tolerance and thus, does not arrive in an infinite loop. That means the algorithm terminates properly.

Apart from the Example 1 with $\beta = 10$, we notice that the results of the second column and the fourth are similar, when maximizing the function evaluations we do not obtain a matrix which requires more inner iterations to satisfy the Armijo backtracking rule, we rather obtain a matrix which requires more Newton iterations. For the Example 1 with $\beta = 10$ the distance between the input matrix and the nearest correlation matrix is large in comparison to the size of the matrix, in particular, for the matrix when we require 47 inner iterations. Furthermore, the smallest eigenvalues of these matrices are large in magnitude so that step lengths that are extremely long occur. Nevertheless, the algorithm does not arrive in an infinite loop ($\theta(y_{k+1})$ is reduced in every backtracking step), terminates properly and returns the nearest correlation matrix.

All examples show that when we have to deactivate the Armijo rule taking the Newton direction with step length 1 is a good alternative since this choice has often been taken. Furthermore, when taking the negative gradient direction (Num. tak. neg. grad. d.) then we could not use the Armijo backtracking rule, thus neither the angles condition was violated nor solving (5.1) failed.

For these robustness tests we conclude that the algorithm behaves properly and

that the suggestions made in Section 4.7.1 help to overcome the problem with the Armijo rule in our examples.

6.6 Comparison with other Methods

Finally, we compare the performance of the Algorithm 2 with Qi and Sun's algorithm and Higham's alternating projections method in [17]. Since Higham's alternating projections method provides full accuracy we will perform the latter comparison with different error tolerances. We use the default value $10^{-7} \cdot n$ and `eps`$\cdot n$. We will obtain a result which accuracy of the nearest correlation matrix computed by our algorithm is achieved for a certain error tolerance and how it improves when we decrease the tolerance. This is also a test of the correctness of our returned matrix.

In the former comparison we are only interested in the efficiency so we use the default value of our error tolerance. Note that according to the results in Section 6.4, we use here the method `f08fc` (divide and conquer algorithm) for the eigenvalue decompositions in Algorithm 2 whereas for the alternating projections method we call LAPACK's `DSYEVR` (via a MEX interface).

6.6.1 Comparison with Qi and Sun's Algorithm

We now test the efficiency of Algorithm 2 in comparison to Qi and Sun's algorithm. Recall that because of the modifications of the Algorithm 2 due to the tests in Section 6.3 and Section 6.4 the main differences between these algorithms are the different routine for the eigenvalue decomposition (the MATLAB function `eig` in Algorithm 1 and `f08fc` in Algorithm 2), the different method to compute the step direction (CG in Algorithm 1 and MINRES in Algorithm 2) and the use of a preconditioner (no preconditioner is applied in Algorithm 1 whereas in Algorithm 2 the Jacobi preconditioner is used).

Our results are reported in Table 6.4 where we state the used time in seconds, the required iteration number of the particular algorithm and the ratio of the used time of Qi and Sun's algorithm and of the Algorithm 2 in the columns denoted by Time,

Iter. and Ratio, respectively.

First, we test Example 1 with different sizes $n = 300, 500, 1000$ for our given matrix G and different values for $\beta = 0.01, 0.1, 1.0$. We use the same sizes for testing the Example 2, 3 and 4, we also compare our efficiency for the Example 5 where the particular matrix is stated by the size in the Table 6.5 and for the Example 6 where the particular matrix is indicated by the words "daily" and "monthly".

Table 6.4: Comparison with Qi and Sun's algorithm

	Algorithm 1 (Qi and Sun)		**Algorithm 2 (Our Modification)**		**Ratio**
	Time	**Iter.**	**Time**	**Iter.**	
Example 1					
$n = 300$					
$\beta = 0.01$	0.86	1	0.76	1	1.1
$\beta = 0.1$	2.17	3	1.14	3	1.9
$\beta = 1$	3.65	5	2.06	6	1.8
$n = 500$					
$\beta = 0.01$	4.49	1	1.77	1	2.5
$\beta = 0.1$	10.41	3	4.05	3	2.6
$\beta = 1$	17.51	5	7.88	6	2.2
$n = 1000$					
$\beta = 0.01$	32.8	1	20.4	2	1.6
$\beta = 0.1$	72.8	3	27.2	3	2.7
$\beta = 1$	126	5	51.9	6	2.4
Example 2					
$n = 300$	3.58	5	2.12	6	1.7
$n = 500$	17.2	5	7.8	6	2.2
$n = 1000$	126	5	52.0	6	2.4
Example 3					

	Algorithm 1		**Algorithm 2**		**Ratio**
	Time	**Iter.**	**Time**	**Iter.**	
$n = 300$	2.26	3	1.21	3	1.9
$n = 500$	8.08	2	4.64	3	1.7
$n = 1000$	61.04	2	20.5	2	3.0
Example 4					
$n = 300$	0.84	1	0.79	2	1.1
$n = 500$	4.45	1	2.97	2	1.5
$n = 1000$	32.3	1	11.6	1	2.8
Example 5					
$n = 1399$	445	5	111	5	4.0
$n = 3120$	5893	4	906	4	6.5
Example 6					
daily	0.61	0	0.37	0	1.6
monthly	0.68	0	0.34	0	2.0

Table 6.4 shows that we achieve an enormous improvement of the efficiency in terms of the computation time with our modifications. We reduce the time in all examples and up to a factor 6.5 for the larger matrix of Example 5.

For the matrices of Example 6 we use zero iterations that means the computed norm of the gradient was already smaller than the tolerance, even though the smallest eigenvalues of both examples are less than zero. We will look at this example again when we decrease the error tolerance in the next Section 6.6.2.

Furthermore, from Table 6.4 we notice that Algorithm 2 sometimes requires more iterations than Qi and Sun's algorithm (Algorithm 1). This is caused by the modification of inequality 5.2 in our algorithm, this inequality is less strict at the beginning and thus sometimes more iterations are required but therefore less iterations in the iterative method potentially.

6.6.2 Comparison with Higham's Algorithm

Now, we compare our algorithm with Higham's alternating projections method. We start with the larger tolerance and report our numerical results in Table 6.5 where we, unlike the previous Table 6.4, include a new column $\|G - X\|_F$ for both methods which shows the difference of the given matrix G to the computed matrix X in the Frobenius norm. Apart from that, we label Table 6.5 like Table 6.4 and use the same examples as in the previous Section 6.6.1. Note that we indicate the ratio between the used time of Higham's alternating projections method and the used time of the Newton algorithm (Algorithm 2) in the column "Ratio" now.

Table 6.5: Comparison with Higham's algorithm

	Newton Algorithm			**Higham's Algorithm**			**Ratio**
	Time	$\|G - X\|_F$	**Iter.**	**Time**	$\|G - X\|_F$	**Iter.**	
Example 1							
$n = 300$							
$\beta = 0.01$	0.49	0.107233	1	0.81	0.107233	4	1.7
$\beta = 0.1$	0.96	5.805298	3	1.57	5.805271	10	1.6
$\beta = 1$	1.85	149.905994	6	8.22	149.905974	73	4.4
$n = 500$							
$\beta = 0.01$	1.66	0.142109	1	2.91	0.142103	3	1.8
$\beta = 0.1$	3.84	12.013733	3	6.92	12.013573	11	1.8
$\beta = 1$	7.36	257.190084	6	38.44	257.190061	90	5.2
$n = 1000$							
$\beta = 0.01$	19.75	0.312575	2	25.73	0.312572	4	1.3
$\beta = 0.1$	26.64	30.077392	3	51.88	30.076585	13	2.0
$\beta = 1$	50.16	530.396314	6	284.84	530.396240	104	5.6
Example 2							
$n = 300$	1.85	149.499553	6	8.28	149.499537	74	4.5
$n = 500$	7.37	256.828853	6	35.87	256.828821	84	4.9

	Newton Algorithm			**Higham's Algorithm**			**Ratio**
	Time	$\|G-X\|_F$	**Iter.**	**Time**	$\|G-X\|_F$	**Iter.**	
$n = 1000$	50.22	530.654053	6	285.48	530.653980	104	5.7
Example 3							
$n = 300$	1.11	0.441387	3	1.42	0.441199	8	1.3
$n = 500$	4.17	0.807703	3	5.88	0.807319	8	1.4
$n = 1000$	18.81	0.944036	2	36.48	0.943383	7	1.9
Example 4							
$n = 300$	0.72	0.097560	2	0.8	0.097559	4	1.1
$n = 500$	2.84	0.109107	2	2.93	0.109077	3	1.0
$n = 1000$	19.71	0.292461	2	21.21	0.292293	3	1.1
Example 5							
$n = 1399$	114.41	21.033907	5	901.4	21.033693	174	7.8
$n = 3120$	913.81	5.44375	4	51615	5.443524	161	56.5
Example 6							
daily	0.37	3.9655e-05	0	0.98	4.1705e-05	2	2.6
monthly	0.33	1.6297e-05	0	1.17	2.8258e-05	2	3.5

We observe that our method takes far less time for all examples in Section 6.1.2. Furthermore, the convergence is reached within 0 to 6 iterations. Whereas, the alternating projections method requires between 2 and 174 iterations.

In Section 4.7.1 we mentioned that the gradient method ($y_{k+1} = y_k - \nabla\theta(y_k)$) is mathematically equivalent to the alternating projections method and as we have seen in Section 6.2, the Newton direction was always preferable to the negative gradient direction in our tests. Moreover, Algorithm 2 claims quadratic convergence if the iterate is close enough to the solution whereas Higham's alternating projections method converges at best linearly (see [17, Section 3.2]).

Another observation is that the iteration number of both methods depends obviously less on the size, more on the perturbation of the matrix if we consider the Example 1.

Now we perform the same tests again with a smaller tolerance. We set our parameter *error_tol* in Algorithm 2 from $n \cdot 10^{-7}$ to $n \cdot$ `eps`. We obtain the results in Table 6.6.

Table 6.6: Comparison with Higham's algorithm using a smaller error tolerance

	Newton Algorithm			**Higham's Algorithm**			**Ratio**
	Time	$\|G - X\|_F$	**Iter.**	**Time**	$\|G - X\|_F$	**Iter.**	
Example 1							
$n = 300$							
$\beta = 0.01$	1.2	0.107234	3	1.92	0.107234	9	1.6
$\beta = 0.1$	1.87	5.805298	5	4.56	5.805298	26	2.4
$\beta = 1$	2.38	149.905994	7	29.37	149.905994	228	12.3
$n = 500$							
$\beta = 0.01$	4.38	0.142108	3	6.92	0.142108	8	1.6
$\beta = 0.1$	7.48	12.013734	5	19.63	12.013734	31	2.6
$\beta = 1$	9.47	257.190084	7	136.84	257.190084	293	14.4
$n = 1000$							
$\beta = 0.01$	30.69	0.312575	3	51.01	0.312575	9	1.7
$\beta = 0.1$	49.71	30.077394	5	150.64	30.077394	40	3.0
$\beta = 1$	81.07	530.396315	8	1058	530.396315	371	13.1
Example 2							
$n = 300$	2.39	149.499553	7	29.83	149.499553	233	12.5
$n = 500$	9.47	256.828853	7	129.43	256.828853	278	13.7
$n = 1000$	79.89	530.654054	8	1052.6	530.654054	369	13.2
Example 3							
$n = 300$	1.58	0.441387	4	6.04	0.441387	30	3.8
$n = 500$	6.35	0.807703	4	21.96	0.807703	29	3.5
$n = 1000$	44.77	0.944048	4	132.49	0.944048	27	2.9

	Newton Algorithm			Higham's Algorithm			Ratio
	Time	$\|G-X\|_F$	**Iter.**	**Time**	$\|G-X\|_F$	**Iter.**	
Example 4							
$n = 300$	1.1	0.097560	3	1.93	0.097560	9	1.8
$n = 500$	4.39	0.109107	3	7.69	0.109107	9	1.8
$n = 1000$	30.7	0.292461	3	50.92	0.292461	9	1.7
Example 5							
$n = 1399$	193.49	21.033911	7	3317	21.033911	614	17.14
$n = 3120$	1666.5	5.443751	6	-	-	-	-
Example 6							
daily	5.65	4.751866e-05	5	26.57	4.751866e-05	55	4.7
monthly	5.09	1.814086e-05	4	17.24	1.814086e-05	27	3.4

First, we notice that the warning that the precision is limiting the convergence has not occurred. Furthermore, we have not tested the Example 5 with the larger matrix for the alternating projections method since the expected value is more than a few days if we consider the difference between the used time for the smaller matrix of Example 5 in Table 6.5 and the used time in Table 6.6. Beyond that we observe in Table 6.6 the same convergence behaviour as in Table 6.5. The Newton method requires at the most 8 iterations, whereas the alternating projections method takes between 8 and 614 iterations.

Considering the distance $\|G - X\|_F$, we see that both methods return the same value of the computed difference of G and X now which clearly shows the increased accuracy and that our algorithm returns the correct solution.

For the matrices of Example 6 our algorithm uses zero iterations and returns the same input matrices as output matrices in our tests in Table 6.4 and Table 6.5 since the stopping criterion is satisfied in spite of the smallest eigenvalues of the input matrices are negative. In Table 6.6 we see that more iterations are taken for the matrices of Example 6 and the smallest eigenvalues of the output matrices are -7.7e-14 (Newton method, -3.1e-12 alternating projections method) for the day matrix and -8.9e-15

(Newton method, -1.93e-12 alternating projections method) for the month matrix now. This shows that after decreasing the error tolerance the computed matrices have become more accurate and satisfy the condition semidefiniteness numerically.

We conclude that our method can compute the nearest correlation matrix accurately and is clearly preferable to the alternating projections method in terms of the computation time especially for such matrices from stock data which we obtained and matrices which have a relatively large distance to their nearest correlation matrix.

6.7 Conclusion

After performing numerical tests on our modified version of Qi and Sun's algorithm (Algorithm 2) for robustness and efficiency we first notice that choosing the MINRES routine gives us the best results overall for our examples. We therefore conclude that the choice of using MINRES is preferable to using CG which is applied in Qi and Sun's algorithm. A further conclusion is that we should use the Jacobi preconditioner computed as suggested in Section 4.4.2 in our iterative method since this choice is inexpensive and can clearly improve the convergence behaviour and thus the performance as we saw in particularly for the matrices from stock data. Furthermore, from the testing results in Section 6.4 we conclude that the method `f08fc` which uses a divide and conquer algorithm is the best choice for forming the eigenvalue decomposition.

Considering our robustness tests, in Section 4.7.1 we identified that the Armijo backtracking rule cannot be applied when the difference of the function values $\theta(y_k)$ and $\theta(y_{k+1})$ is small which occurs when we use a small error tolerance for $\|\nabla\theta(y_k)\|_2$. We made suggestions in Section 4.7.1 to overcome this problem and the robustness tests in Section 6.5 and 6.6 have shown that we have avoided the problem. Moreover, in our test examples in Section 6.6 we used an error tolerance of $n \cdot \texttt{eps}$ for $\|\nabla\theta(y_k)\|_2$ and the Algorithm 2 has always been terminated at step 3 thus the algorithm has not been aborted and the final norm of $\nabla\theta(y)$ was smaller than the error tolerance, hence we can provide almost full accuracy. Furthermore, the results in Section 6.5 have

shown that the algorithm is reliable since the direct optimization methods could not find an example, in spite of 30 runs, where our algorithm shows a failing behaviour.

Chapter 7

The Alternating Projections Method

As mentioned in Section 1.5, Higham [17] proposes an alternating projections method which also solves the problem of finding the nearest correlation matrix. In the next Section 7.1 we explain the idea of the general alternating projections method: what the method is and how it works. In Section 7.2 we explain how we can use this method for our problem (1.14), we describe Higham's method [17]. Then we discuss changing the method by introducing a further projection in Section 7.3 and we perform some tests in Section 7.4. Finally, we draw our conclusions in Section 7.4.4.

7.1 The General Alternating Projections Method

Here we describe the general alternating projections method. We start with the definition of the best approximation from a subset of a Hilbert space H.

Definition 7.1.1. Let $C \subset H$ and $C \neq \emptyset$ and let $x \in C$ then x is called the *best approximation* from C to an element $h \in H$ if

$$\|h - x\| = \inf_{y \in C} \|h - y\| \tag{7.1}$$

where $\|\cdot\|$ is the norm corresponding to the Hilbert space H.

If C is a closed convex set then the best approximation from C to an element in H exists and is unique. Compare with Section 1.4 or see [18, Section 3.1]. Hence, the next definition of the projection operator onto closed convex sets is well defined.

Definition 7.1.2. Let C be a closed convex set in a Hilbertspace H, then the *projection operator* $P_C : H \mapsto C$ onto the set C is defined as

$$P_C(y) := \text{argmin}_{x \in C} \|y - x\| . \tag{7.2}$$

Now let $K_1, \ldots, K_m$ be closed convex sets in H. We are interested in the problem of finding the best approximation to an element $g \in H$ from $K := \cap_{i=1}^m K_i$, that is

$$\text{argmin}_{x \in K} \|g - x\| . \tag{7.3}$$

Since K is also a closed convex set in H the solution of (7.3) exists and is unique. Moreover, if the projection operator P_K projecting onto K is known then by definition applying this operator yields the solution of (7.3). However, P_K is often not easily computable. The method of alternating projections can remedy this problem. It reduces the problem (7.3) to finding the best approximations from the individual sets K_i.

The idea of the method is to project g onto the sets K_i one after the other and repeat iteratively, that is forming a sequence $\{a_k\}$ with

$$a_{k+1} := P_{K_m} \ldots P_{K_2} P_{K_1}(a_k) \tag{7.4}$$

and $a_0 := g$. Here, P_{K_i} is the projection operator onto K_i for all $i = 1, \ldots, m$.

The question arises whether a_k converges to the best approximation to g from the set K. By the subsequent Theorem 7.1.3 we obtain that the iterate a_k converges indeed to the solution of (7.3) if the set K_i are closed subspaces and $\cap_{i=1}^m K_i$ is not empty. This result was obtained by von Neumann for two subspaces and also studied in [6] to which we refer. See [6, Corollary 9.28] and [6, Theorem 9.27] for a proof or also for the case of two subspaces [6, Von Neumann Theorem 9.3].

Theorem 7.1.3. *Let $M_1, \ldots, M_m$ be closed convex subspaces of the Hilbert space H. Then with $M := \cap_{i=1}^{m} M_i$ and $M \neq \emptyset$*

$$\lim_{k\to\infty} (P_{M_n} \ldots P_{M_2} P_{M_1})^k(x) = P_M(x) \tag{7.5}$$

for each $x \in H$ where P_{M_i} and P_M are the projection operators onto M_i and M, respectively.

Fortunately, we can extend this result to closed convex sets, but we have to incorporate a correction to each projection due to Dykstra. Let $[k]$ denote k modulo m. We change our iterative procedure of (7.4) to Dykstra's algorithm and obtain the two new sequence $\{a_k\}$, $\{e_k\}$ with

$$\begin{aligned} &a_0 := g, \quad e_{-(m-1)} = \cdots = e_{-1} = e_0 = 0, \\ &a_k := P_{K_{[k]}}(a_{k-1} + e_{k-m}), \\ &e_k := a_{k-1} + e_{k-m} - a_k \end{aligned} \tag{7.6}$$

and e_k the correction at each iteration. By [6, Boyle-Dykstra Theorem 9.24] we have the following convergence result for that algorithm.

Theorem 7.1.4. *Let $K_1, \ldots, K_m$ be closed convex subsets of the Hilbert space H. For each $g \in H$, define the sequence $\{a_k\}$ as in (7.6). Then with $K := \cap_{i=1}^{m} K_i$ and $K \neq \emptyset$*

$$\lim_{k\to\infty} \|a_k - P_K(g)\| = 0 \tag{7.7}$$

where P_{K_i} and P_K are the projection operators onto K_i and K, respectively.

Now we have also obtained a convergence result when all K_i are closed convex sets only but we have not clarified yet whether we can omit the correction of a particular projection if the corresponding set K_j is one of the closed convex sets which is also a closed subspace for $j \in \{1, \ldots, m\}$. Fortunately, we can omit the particular correction and moreover not only if the particular K_j is a closed convex subspace but also if it is a closed affine set (translate of a subspace). That means the second term in (7.6) becomes

$$a_k := P_{K_{[k]}}(a_{k-1}) \tag{7.8}$$

if $K_{[k]}$ is a closed affine set. See [6, Section 9.26] for a proof. Note also that by [6, Theorem 9.33] the convergence rate of the algorithm in (7.6) is linear if all K_i are subspaces and the constant depends on the angles between one subspace K_j and the intersection of all other subspaces K_i with $i \neq j$.

Summarizing all this, by the algorithm in (7.6) we can find the best approximation from K to an element in H if all K_i are closed convex sets. If a particular K_j is a subspace or translate of a subspace then we can omit the correction.

The only problem which remains is to find for a closed convex set, from which the best approximation is desired and for which the projection operator is not easily computable, a finite number of individual closed convex sets which intersection is the actual set and which correspond to easily computable projection operators.

7.2 Higham's Alternating Projections Method

In this section, we show how we can solve the problem of finding the nearest correlation matrix (see (1.14)) by means of the alternating projections method as proposed by Higham in [17]. We define the following two closed convex sets first which are

$$S := \mathcal{S}^n_+ \tag{7.9}$$

and

$$U := \{Y \in \mathcal{S}^n : Y_{ii} = 1, i = 1, \ldots, n\}. \tag{7.10}$$

Clearly, the solution of our problem is an element of $U \cap S$, which is not empty since the identity I_n is an element of both sets S and U. We are interested in finding the best approximation to a matrix G with $G = G^T$ as in (1.14) from the set $U \cap S$.

In order to apply the alternating projections method we need to consider how to project onto the set S and U individually. Note that our Hilbert space is the set $\mathcal{S}^n$ with the inner product defined in (1.6) now. As regarded in Section 2.1.1 our projection onto the set S is the operator $(\cdot)_+$. See (2.2) for its definition. Thus, it remains the projection operator onto the set U which we deduce from the later stated Theorem 7.3.2 with $\mathcal{N} := \{(i,i) : i = 1, \ldots, n\}$ (see also [17, Theorem 3.1])

and obtain that

$$P_U(G) = G - \text{Diag}(\text{diag}(G - I_n)) \tag{7.11}$$

with I_n the identity in $\mathcal{S}^n$. Now we satisfy all conditions of Theorem 7.1.4 so that the algorithm in (7.6) with $m = 2$, $K_1 := S$, $K_2 := U$, $P_{K_1} = P_S$ and $P_{K_2} = P_U$ converges to our solution of the problem (1.14). Note that U is a translate of a subspace so that the correction can be omitted:

$$U := I_n + \{Y \in \mathcal{S}^n : Y_{ii} = 0 \quad \forall i = 1, \ldots, n\}. \tag{7.12}$$

7.3 Adding further Constraints

7.3.1 Example Requiring further Constraints

Now we address the question of changing our problem (1.14) due to variations. From Section 7.1 we saw that we can apply the alternating projections method as long as our variations or additional constraints can be expressed as intersection of closed convex sets under the assumption that the intersection is not empty and all projections onto the individual sets are known. Our variation which we consider here is to add additional constraints so that certain correlations of the input matrix are preserved during the computation.

Therefore we consider an example first where preserving correlations is required. Let $\{\xi_1, \ldots, \xi_m, \xi_{m+1}, \ldots, \xi_n\}$ be n assets of a portfolio and let $C \in \mathcal{S}^n$ be the corresponding correlation matrix with the block structure

$$C := \begin{bmatrix} C_{11} & C_{12} \\ C_{12}^T & C_{22} \end{bmatrix} \tag{7.13}$$

where $C_{11} \in \mathcal{S}^m$ and $C_{22} \in \mathcal{S}^{n-m}$ contain the correlations of $\xi_1, \ldots, \xi_m$ and $\xi_{m+1}, \ldots, \xi_n$, respectively. Now suppose that it is desired to change the correlations contained in C_{22} for stress testing purposes and that the matrix C does not satisfy the conditions of a correlation matrix after its modification. Let $\widehat{C} \in \mathcal{S}^n$ denote the modified matrix

with

$$\widehat{C} := \begin{pmatrix} C_{11} & C_{12} \\ C_{12}^T & \widehat{C}_{22} \end{pmatrix}. \tag{7.14}$$

That means the nearest correlation matrix to $\widehat{C}$ is desired but in general without changing the values of C_{11} since our modification does not affect the assets $\xi_1, \ldots, \xi_m$ and thus, also not the correlations in C_{11}. Consider also Finger's algorithm in Section 1.5, the matrix C_{22} in (1.27) is not changed; see also [8].

However, fixing elements of our input matrix G is not covered in our problem (1.14) so that all values of G are allowed to be changed. See therefore the concrete example of [8] in Section 7.4.3. In order to tackle the problem described here, we consider in the next sections our convex optimization problem (1.14) with additional constraints and subsequently a variation of the alternating projections method.

7.3.2 The Problem

Before we change our alternating projections method so that the problem described in Section 7.3.1 can be solved by this method we express our new problem mathematically as

$$\begin{aligned} &\min \frac{1}{2}\|G - X\|_F^2, \quad G \in \mathcal{S}^n \text{ given} \\ &\text{s.t.} \quad \operatorname{diag}(X) = e,\ X \in \mathcal{S}_+^n \text{ and } X_{ij} = G_{ij} \quad \forall i,j \in \mathcal{N} \end{aligned} \tag{7.15}$$

where $\mathcal{N}$ is the index set which specifies all locations of the entries of G which are to be preserved. Just as (1.14), this is also a convex optimization problem which has a unique solution if the feasible set is not empty, $\mathcal{N}$ must be chosen so that a solution of problem (7.15) exists.

7.3.3 Solving the Problem by Adding a further Projection

We clarify here what we have to change in Higham's alternating projections method in order to solve the problem (7.15). We define a new set F as

$$F := \{Y \in \mathcal{S}^n : Y_{ij} = G_{ij} \quad \forall (i,j) \in \mathcal{N}\}. \tag{7.16}$$

Clearly, the solution of the problem (7.15) is the best approximation to g from the set $F \cap U \cap S$ in the Hilbert space $\mathcal{S}^n$ with the inner product as defined in (1.6). Furthermore, since F is a translate of a subspace (compare with U) and thus a closed convex set our theory in Section 7.1 applies and the projection operator P_F exists. Hence the algorithm in (7.6) can also be applied for solving the problem (7.15) with $m = 3$ and $K = F \cap U \cap S$. That means we extend Higham's alternating projections method only by the projection P_F. Note that the correction is still only required for the projection onto the set S.

It remains to specify the projection operator onto the set F. Therefore, we characterise general projections onto closed convex set with the subsequent Lemma 7.3.1 [18, Theorem 3.1.1] first and show then by means of this lemma how we can compute the projection P_F.

Lemma 7.3.1. *Let H be a Hilbert space and C be a closed convex subset of H. Let x be an arbitrary element in H. Then an element $y_p \in C$ is the projection $P_C(x)$ if and only if*

$$\langle x - y_p, y - y_p \rangle \leq 0 \quad \forall y \in C. \tag{7.17}$$

with $\langle \cdot, \cdot \rangle$ the inner product in H.

Proof. "$\Rightarrow$": Let $y_p = P_C(x)$ and let y be an arbitrary element in C. Since C is convex we have that $y_p + \alpha(y - y_p) \in C$ for any $\alpha \in (0, 1)$ so that

$$\|x - y_p\| = \min_{z \in C} \|x - z\| \leq \|x - (y_p + \alpha(y - y_p))\|. \tag{7.18}$$

From (7.18) we obtain that

$$\begin{aligned} \|x - y_p\|^2 &\leq \|x - y_p - \alpha(y - y_p)\|^2 \\ &= \|x - y_p\|^2 - 2\alpha\langle x - y_p, y - y_p \rangle + \alpha^2 \|y - y_p\|^2. \end{aligned} \tag{7.19}$$

Since $\alpha > 0$ we divide the expression in (7.19) by α and simplify it to

$$\langle x - y_p, y - y_p \rangle \leq \frac{\alpha}{2} \|y - y_p\|^2. \tag{7.20}$$

Now we let $\alpha \searrow 0$ and we obtain one direction of the proof.

"$\Leftarrow$": Let y_p be satisfying the inequality (7.17). If $y_p = x$ then obviously $y_p = P_C(x)$. If not, it holds for arbitrary $y \in C$ that

$$\begin{aligned} 0 &\geq \langle x - y_p, y - y_p \rangle \\ &= \langle x - y_p, y - x + x - y_p \rangle \\ &= \|x - y_p\|^2 + \langle x - y_p, y - x \rangle \\ &\geq \|x - y_p\|^2 - \|x - y_p\| \, \|y - x\| . \end{aligned} \tag{7.21}$$

The latter inequality holds because of the Cauchy-Schwarz inequality ($\langle x, y \rangle \leq \|x\| \, \|y\|$); see Lemma 1.3.3. Since $\|x - y_p\| > 0$ it follows that $y_p = P_C(x)$. □

Now we can formulate our theorem which specifies the projection P_F.

Theorem 7.3.2. *The projection P_F onto F is the following operator $P_{\widetilde{F}} : \mathcal{S}^n \mapsto F$ with*

$$P_{\widetilde{F}}(X) := \begin{cases} G_{ij} & \text{if } \ (i,j) \in \mathcal{N} \\ X_{ij} & \text{otherwise} \end{cases} \tag{7.22}$$

for each X in $\mathcal{S}^n$.

Proof. Let X be an arbitrary element in $\mathcal{S}^n$. Furthermore, for simplicity let $T_1 := X - P_{\widetilde{F}}(X)$ and $T_2 := Y - P_{\widetilde{F}}(X)$ with $Y \in F$. Then by Lemma 7.3.1 it is enough to show that $\langle T_1, T_2 \rangle \leq 0$ for all $Y \in F$. From (7.22) we obtain for T_1 that

$$T_1 = \begin{cases} X_{ij} - G_{ij} & \text{if } \ (i,j) \in \mathcal{N} \\ 0 & \text{otherwise} \end{cases} \tag{7.23}$$

and equally for T_2 that

$$T_2 = \begin{cases} 0 & \text{if } \ (i,j) \in \mathcal{N} \\ Y_{ij} - X_{ij} & \text{otherwise} . \end{cases} \tag{7.24}$$

By the definition (1.6) of the inner product $\langle \cdot, \cdot \rangle$ in $\mathcal{S}^n$ we have that $\langle T_1, T_2 \rangle = \sum_{i,j=1}^n (T_1)_{ij}(T_2)_{ij} = 0$ for all $Y \in \mathcal{S}^n$ which implies our assertion. □

7.3.4 The Algorithm

Here we formulate the modified version of Higham's algorithm [17, Algorithm 3.3], we include an additional projection which is the projection onto F defined in the Theorem 7.3.2. This allows us to hold fix specified elements of the input matrix during our computation. Note that these specified elements must be in the real interval $[-1, 1]$ and must be chosen such that a nearest correlation matrix with these entries exists. For the implementation of this algorithm see Section A.3 of Appendix A.

Algorithm 3. Given a symmetric matrix $G \in \mathbb{R}^{n\times n}$ this algorithm computes the nearest correlation matrix to G in the Frobenius norm subject to certain designated elements of G remaining fixed. Convergence is tested using a given error tolerance *error_tol* .

Step 1: Set the starting values: $\Delta E_0 = 0$, $Z_0 = A$ and $k := 1$.

Step 2: Add Dykstra's correction ΔE_{k-1}:

$R_k := Z_{k-1} + \Delta E_{k-1}$

Compute the projection onto S:

$X_k := P_S(R_k)$

Compute the new correction:

$\Delta E_k := R_k - X_k$

Step 3: Compute the projection onto U:

$Y_k := P_U(X_k)$

Step 4: Compute the projection onto F:

$Z_k := P_F(Y_k)$

Step 6: Compute the following relative differences

$X_{diff} := \frac{\|X_k - X_{k-1}\|_F}{\|X_k\|_F}$, $Y_{diff} := \frac{\|Y_k - Y_{k-1}\|_F}{\|Y_k\|_F}$

$Z_{diff} := \frac{\|Z_k - Z_{k-1}\|_F}{\|Z_k\|_F}$, $XY_{diff} := \frac{\|Y_k - X_{k-1}\|_F}{\|Y_k\|_F}$

$ZX_{diff} := \frac{\|Z_k - X_{k-1}\|_F}{\|Z_k\|_F}$

If $\max\{X_{diff}, Y_{diff}, Z_{diff}, XY_{diff}, ZX_{diff}\} \leq error\_tol$ exit, otherwise goto Step 2.

Note that we choose the stopping criterion according to Higham's convergence test for his algorithm in [17, Algorithm 3.3].

7.4 Numerical Tests

In this section, we perform some tests referring to how the required iteration number changes when we fix certain correlations. Therefore, we use the same software and hardware as in Section 6.1.1 and set our tolerance to $10^{-7} \cdot n$ unless otherwise stated where n is the size of the given symmetric matrix G. The examples used are described in the particular section. Furthermore, we use the MATLAB function `eig` for our eigenvalue decomposition in the alternating projections method.

7.4.1 Fixing Blocks of Correlations

In this section, we test how the convergence behaviour in terms of the iteration number varies when we perform a local correlation stress testing and a band correlation stress testing. Our tests are similar to those in [28]. Therefore, let C be a correlation matrix generated by the MATLAB function `gallery('randcorr',n)`. We divide C into 4 parts, that is

$$C := \begin{bmatrix} C_1 & C_2 \\ C_3 & C_4 \end{bmatrix} \tag{7.25}$$

with $C_1 \in \mathcal{S}^m, C_2 \in \mathbb{R}^{m\times(n-m)}, C_3 \in \mathbb{R}^{(n-m)\times m}, C_4 \in \mathcal{S}^{n-m}$ and $C_3 = C_2^T$.

Band Correlation Stress Testing

In our first experiment (Experiment 1) we perform the band correlation stress testing. That means we fix all correlations of the part C_1 only and stress the remaining parts by forming a convex combination of C and Z with Z as a matrix of Example 1 (see

Section 6.1.2) with $\beta = 0.2$. That is defining our set $\mathcal{N}$ as

$$\mathcal{N} := \{(i,j) : i,j = 1,\ldots,m\} \tag{7.26}$$

and generating our example matrix $\widehat{C}$ with

$$\widehat{C} := \begin{cases} C_{ij} & \text{if } (i,j) \in \mathcal{N} \\ \lambda C_{ij} + (1-\lambda) Z_{ij} & \text{otherwise} \end{cases} \tag{7.27}$$

and $\lambda = 1/2$.

For our testing purposes we run Higham's algorithm on this test example with different values of $n = 300, 500, 1000$ and $m = \frac{1}{5}n, \frac{2}{5}n, \frac{3}{5}n, \frac{4}{5}n$ first, we do not fix the correlations. Once we have run Higham's method, we also run our modified version on these examples and fix the correlations. If m is not an integer value we round the value up to the next largest integer. Note that we use the same generated matrix Z for all values of m.

We report our results in Table 7.1, where Higham's Algorithm, Algorithm 3, Time, $\|G - X\|_F$ and Iter. stand for Higham's alternating projections method (see Section 7.2), our modified algorithm where we add the projection onto F, the time in seconds to run the particular algorithm, the distance between the computed matrix and the input matrix in the Frobenius norm and the required iteration number, respectively. In the column "Ratio" we state the ratio between the number of the used iterations for not fixing and fixing correlations. Furthermore, our test examples are distinguished by the size of m and n in the Table 7.1.

Table 7.1: Band correlation stress testing, Experiment 1

	Higham's Algorithm			**Algorithm 3**			**Ratio**
	Time	$\|G - X\|_F$	**Iter.**	**Time**	$\|G - X\|_F$	**Iter.**	
$\boldsymbol{n = 300}$							
$m = \frac{1}{5}n$	2.85	4.86	9	3.59	4.93	11	1.2
$m = \frac{2}{5}n$	2.42	4.38	8	5.79	4.59	17	2.1
$m = \frac{3}{5}n$	2.57	3.59	8	7.9	3.97	23	2.9

	Higham's Algorithm			**Algorithm 3**			**Ratio**
	Time	$\|G-X\|_F$	**Iter.**	**Time**	$\|G-X\|_F$	**Iter.**	
$m=\frac{4}{5}n$	2.63	2.40	8	15.73	2.89	46	5.8
$\boldsymbol{n=500}$							
$m=\frac{1}{5}n$	17.48	10.72	10	23.32	10.8	13	1.3
$m=\frac{2}{5}n$	17.49	9.70	10	36.94	10.1	20	2.0
$m=\frac{3}{5}n$	17.13	8.06	10	56.33	8.82	30	3.0
$m=\frac{4}{5}n$	14.6	5.63	9	74.25	6.62	41	4.6
$\boldsymbol{n=1000}$							
$m=\frac{1}{5}n$	162.9	27.9	12	238.3	28.2	17	1.4
$m=\frac{2}{5}n$	161.7	25.4	12	320.6	26.4	23	1.9
$m=\frac{3}{5}n$	154.8	21.3	12	424.9	23.1	31	2.6
$m=\frac{4}{5}n$	152.9	15.3	12	495.7	17.6	37	3.1

From Table 7.1 we observe that the iteration number of Higham's algorithm decreases only slightly, although the distance $\|G-X\|_F$ clearly becomes smaller when we increase m, but the iteration number of Algorithm 3 increases more intensively with m so that the ratio of the iteration numbers also increases when m becomes larger. Furthermore, the ratio of the iteration number seems to be constant when we increase n apart from the case when $m=\frac{4}{5}n$.

The closer m is to n the more correlations are fixed, hence the less correlations are allowed to be changed and the problem becomes more difficult to solve which explains why more iterations are required in Algorithm 3. Note also that if we increase m then we change our convex sets and so presumably also the convergence rate.

If the matrix becomes larger (n becomes larger) the time which is used to run the algorithms increases since e.g. performing the eigenvalue decomposition is more expensive. However, the ratio between m and n is the same so that obviously the ratio between the iteration numbers also remains similar.

It is interesting that the alternating projections method requires almost always the same number of iterations if we increase m. Although less entries are changed if

we increase m the problem only becomes slightly easier to solve.

Now we perform the second experiment (Experiment 2) in which we try to determine how the algorithm behaves when we vary the number of modifications in the parts C_2, C_3, C_4. Therefore, we repeat the first experiment but restrict m to $\frac{3}{5}n$ and change randomly x times an entry of the parts C_2, C_3, C_4 in the same manner to Example 3 in Section 6.1.2 instead of forming the convex combination. We start with $x = \frac{1}{5}k$ and increase it by $\frac{1}{5}k$ 4 times where k is number of all entries in C_2, C_3, C_4. Note that we use only one generated correlation matrix for all values of x and if we increase x then we leave the modifications from the previous test example (smaller x) and change entries only remainingly times.

We report our results in Table 7.2 labelled similarly to Table 7.1. The only difference is that we use other test examples now which are specified by the value of x indicating how often we modified a randomly chosen entry of the starting correlation matrix in the appropriate blocks. The number in per cent indicates how much per cent of all entries was changed.

Table 7.2: Band correlation stress testing, Experiment 2

	Higham's Algorithm			**Algorithm 3**			**Ratio**
	Time	$\|G - X\|_F$	**Iter.**	**Time**	$\|G - X\|_F$	**Iter.**	
$\boldsymbol{n = 300}$							
$x = \frac{1}{5}k$ (21.2%)	14.34	61.4	42	25.78	64.8	73	1.7
$x = \frac{2}{5}k$ (35.3%)	16.83	84.1	52	27.9	88.5	85	1.6
$x = \frac{3}{5}k$ (44.8%)	18.11	96.3	59	31.3	101.1	100	1.7
$x = \frac{4}{5}k$ (51.2%)	19.19	103.3	64	32.0	108.2	99	1.5
$\boldsymbol{n = 500}$							
$x = \frac{1}{5}k$ (21.1%)	84.88	106.6	49	137.4	112.2	79	1.6
$x = \frac{2}{5}k$ (35.3%)	107.8	144.4	64	163.3	151.1	96	1.5
$x = \frac{3}{5}k$ (44.7%)	119.9	165.9	72	181.8	173.0	107	1.5
$x = \frac{4}{5}k$ (51.0%)	127.7	178.5	77	180.5	186.0	111	1.4

	Higham's Algorithm			Algorithm 3			Ratio
	Time	$\|G-X\|_F$	**Iter.**	**Time**	$\|G-X\|_F$	**Iter.**	
$\boldsymbol{n = 1000}$							
$x = \frac{1}{5}k$ (21.1%)	855.9	225.5	64	1199	235.4	92	1.4
$x = \frac{2}{5}k$ (35.2%)	1089	301.5	83	1408	312.9	116	1.4
$x = \frac{3}{5}k$ (44.7%)	1222	344.0	94	1444	356.1	122	1.3
$x = \frac{4}{5}k$ (51.1%)	1296	370.5	100	1515	382.9	128	1.3

In Table 7.2 we observe that when we increase x the iteration number of both algorithms increases gradually and the ratio between the iteration numbers decreases slightly. We see the same behaviour when we increase n. The larger the value of x, the further the distance $\|G-X\|_F$ and hence the more iterations are required.

Local Correlation Stress Testing

In our next experiments we stress correlations locally, we perform the same experiments as above but we change correlations in the part C_4 only and fix all remaining parts. Therefore, we redefine our set $\mathcal{N}$ as

$$\mathcal{N} := \{(i,j) : i = 1,\dots,n;\ j = 1,\dots,m\} \cup \{(i,j) : i = 1,\dots,m;\ j = 1,\dots,n\}.$$

We report the results in Table 7.3 and in Table 7.4. In the former one we perform Experiment 1 for local correlation stress testing and in the latter one Experiment 2.

Table 7.3: Local correlation stress testing, Experiment 1

	Higham's Algorithm			Algorithm 3			Ratio
	Time	$\|G-X\|_F$	**Iter.**	**Time**	$\|G-X\|_F$	**Iter.**	
$\boldsymbol{n = 300}$							
$m = \frac{1}{5}n$	2.34	3.67	8	9.56	3.97	29	3.6
$m = \frac{2}{5}n$	1.99	2.41	7	19.1	6.32	55	7.9
$m = \frac{3}{5}n$	1.67	1.34	6	35.1	1.86	101	16.8
$m = \frac{4}{5}n$	1.32	0.56	5	65.3	0.83	181	36.2

	Higham's Algorithm			**Algorithm 3**			**Ratio**
	Time	$\|G-X\|_F$	**Iter.**	**Time**	$\|G-X\|_F$	**Iter.**	
$\boldsymbol{n=500}$							
$m=\frac{1}{5}n$	14.57	8.22	9	59.5	8.70	33	3.7
$m=\frac{2}{5}n$	12.52	5.47	8	106.2	2.89	59	7.3
$m=\frac{3}{5}n$	10.59	3.0	7	198.0	3.96	107	15.3
$m=\frac{4}{5}n$	9.24	1.10	6	535.5	1.88	296	49.3
$\boldsymbol{n=1000}$							
$m=\frac{1}{5}n$	141.4	21.5	11	610.66	22.48	44	4.0
$m=\frac{2}{5}n$	123.51	14.7	10	986.7	16.36	72	7.2
$m=\frac{3}{5}n$	109.2	8.44	9	1844	10.35	130	14.4
$m=\frac{4}{5}n$	70.9	3.06	6	4196	4.71	296	49.3

In Table 7.3 we obtain similar results as in Table 7.1. We notice a difference in the ratio between the iteration numbers which is clearly larger and in the distance between the input matrix and the solution which is smaller throughout all test examples. Furthermore, we observe more clearly now that the iteration number decreases with m when we run Higham's algorithm but increases when we run the Algorithm 3 where we fix the correlations.

These results are not surprising because the number of correlations which are allowed to be changed is the same as in the band correlation stress test when we do not fix our correlations but is far less when fixing the correlations, thus more iterations are required if we run Algorithm 3 and approximately the same if we run Higham's algorithm. This yields the increased ratio between the iteration numbers. Moreover, since we only change the C_4 block in the local stress test, less correlations need to be changed in order to obtain to the solution than in the band stress test which explains that the iteration number in Higham's algorithm decreases more clearly when m increases and the smaller distance between the input matrix and the solution.

Table 7.4: Local correlation stress testing, Experiment 2

	Higham's Algorithm			**Algorithm 3**			**Ratio**
	Time	$\|G-X\|_F$	**Iter.**	**Time**	$\|G-X\|_F$	**Iter.**	
$\boldsymbol{n=300}$							
$x=\frac{1}{5}k$ (5.3%)	8.06	28.72	27	275.02	31.13	849	31.4
$x=\frac{2}{5}k$ (8.9%)	8.85	39.63	30	333.49	42.32	1098	36.6
$x=\frac{3}{5}k$ (11.2%)	9.64	45.7	33	356.29	48.4	1168	35.4
$x=\frac{4}{5}k$ (12.8%)	10.53	49.4	36	381.74	52.2	1287	35.8
$\boldsymbol{n=500}$							
$x=\frac{1}{5}k$ (5.3%)	44.02	50.97	28	1362.9	53.19	853	30.5
$x=\frac{2}{5}k$ (8.8%)	51.42	68.36	34	1558.2	71.95	1137	33.4
$x=\frac{3}{5}k$ (11.1%)	52.14	79.2	36	1816	82.53	1336	37.1
$x=\frac{4}{5}k$ (12.7%)	56.24	85.48	39	2012	89.10	1419	36.4
$\boldsymbol{n=1000}$							
$x=\frac{1}{5}k$ (5.2%)	356.6	108.3	31	12498	113.6	1111	35.8
$x=\frac{2}{5}k$ (8.8%)	417.5	145.9	39	15515	150.3	1451	37.2
$x=\frac{3}{5}k$ (11.2%)	430.4	166.7	42	17087	172.5	1679	40.0
$x=\frac{4}{5}k$ (12.7%)	554.6	180.4	44	18627	186.0	1805	41.0

Table 7.4 provides a similar insight in the behaviour of our algorithm. Again, the iteration number increases if we increase x and the ratio of the iteration numbers does not change intensively. Furthermore, in comparison to the band correlation stress testing (experiment 2) a clearly larger number of the ratio of iteration numbers is obtained which is obviously caused by stressing correlations locally only.

7.4.2 Fixing Arbitrarily Chosen Correlations

In the experiment in this section, we generate a correlation matrix A using the same MATLAB function as in Section 7.4.1 for different size $n = 300, 500, 1000$. Then we choose similarly to Example 3 randomly (uniformly) $\frac{1}{2}n^2$ times a pair (i, j) with $i, j \in$

$\{1, \ldots, n\}$ (recurrences are allowed) and leave both entries A_{ij} and A_{ji} unchanged during our subsequent perturbation. Moreover, after the perturbation we look for the matrix which is nearest to our perturbed matrix and has the entries A_{ij} (respectively A_{ji}) for (i, j) in the set of the first x (x is to specify) chosen pairs (i, j).

We perturb our matrix in similar manner to Example 1 for all remaining pairs (i, j) only so that a solution always exits. We generate a symmetric matrix $M \in \mathbb{R}^{n\times n}$ with elements from a uniform distribution in the range $[-\beta, \beta]$ where β is given and add M_{ij} to A_{ij} for all pairs (i, j) which are not chosen.

We run our algorithm with these test examples where we use different value of $\beta = 0.01, 0, 05, 0.1, 0.5$ and different values of x. We start with $x = 0$ for every test example and increase it by $n^2/10$ 5 times. If the value $n^2/10$ is not an integer then we round the value up by means of the MATLAB function `ceil`.

We report our results in Table 7.5 where we show how many iterations in our algorithm are used for all example matrices with all different values of x and β. The example matrices are specified by the size n. Note that we only generate a correlation matrix and choose the pairs (i, j) when we change the size n.

Table 7.5: Fixing arbitrarily chosen correlations

	Iteration Number					
	$\boldsymbol{x = 0}$	$\boldsymbol{x = \frac{1}{10}n^2}$	$\boldsymbol{x = \frac{2}{10}n^2}$	$\boldsymbol{x = \frac{3}{10}n^2}$	$\boldsymbol{x = \frac{4}{10}n^2}$	$\boldsymbol{x = \frac{5}{10}n^2}$
$\boldsymbol{n = 300}$						
$\beta = 0.01$	3	4	4	4	4	4
$\beta = 0.05$	5	7	9	11	14	18
$\beta = 0.1$	7	10	13	18	23	29
$\beta = 0.5$	21	44	72	110	156	214
$\boldsymbol{n = 500}$						
$\beta = 0.01$	3	4	5	6	7	8
$\beta = 0.05$	5	7	10	13	16	20
$\beta = 0.1$	8	11	16	21	28	36

	Iteration Number					
	$\boldsymbol{x = 0}$	$\boldsymbol{x = \frac{1}{10}n^2}$	$\boldsymbol{x = \frac{2}{10}n^2}$	$\boldsymbol{x = \frac{3}{10}n^2}$	$\boldsymbol{x = \frac{4}{10}n^2}$	$\boldsymbol{x = \frac{5}{10}n^2}$
$\beta = 0.5$	23	56	93	142	201	280
$\boldsymbol{n = 1000}$						
$\beta = 0.01$	3	4	5	6	8	9
$\beta = 0.05$	6	8	11	14	18	23
$\beta = 0.1$	9	14	19	26	36	48
$\beta = 0.5$	27	74	127	192	270	365

Note that between 36% and 37% of all entries of our test examples were changed by our perturbation. Considering the results in Table 7.5, we observe that the iteration number becomes larger when we increase x and more intensively when β is also increased. We obtain for example an increase of the iteration number by a factor 13.5 if we compare the iteration number between $x = 0$ and $x = \frac{5}{10}n^2$ for $\beta = 0.5$ and $n = 1000$. A further observation is that the iteration number also increases if we increase only the value of β or the size of the matrix n.

The larger the values of β and n the further the distance from our test example to the original correlation matrix and also potentially to the nearest correlation matrix in the Frobenius norm. Thus more iterations are required. The less values of the test examples are allowed to be changed (value of x) the more values are changed back when projecting onto F after projecting onto S and hence, more iterations are required. Moreover, many more iterations are required the larger β since more values need then to be changed to find the nearest correlation matrix because of the more intensive modification of the original correlation matrix. This explains our observations in Table 7.5.

7.4.3 Example of Finger

For our last experiment, we choose a concrete example of four Asian and three non-Asian currencies from [8] which is actually RiskMetrics data of June 13, 1997. The

correlations are contained in the matrix

$$\widehat{G} := \begin{bmatrix} 1.0 & 0.18 & -0.13 & -0.26 & 0.19 & -0.25 & -0.12 \\ 0.18 & 1.0 & 0.22 & -0.14 & 0.31 & 0.16 & 0.09 \\ -0.13 & 0.22 & 1.0 & 0.060 & -0.080 & 0.040 & 0.04 \\ -0.26 & -0.14 & 0.06 & 1.0 & -0.21 & 0.14 & -0.15 \\ 0.19 & 0.31 & -0.08 & -0.21 & 1.0 & 0.22 & 0.10 \\ -0.25 & 0.16 & 0.04 & 0.14 & 0.22 & 1.0 & 0.07 \\ -0.12 & 0.09 & 0.04 & -0.15 & 0.10 & 0.07 & 1.0 \end{bmatrix}$$

where $\widehat{G}(1:3, 1:3)$ contains only the correlation between the non-Asian currencies and $\widehat{G}(4:7, 4:7)$ between the Asian currencies. This matrix is a correlation matrix since the smallest eigenvalue is 0.518. Now we suppose that a user does not agree with the correlation between the Asian currencies and sets all these correlations to a blanket level of 0.85 which yields the matrix G with

$$G := \begin{bmatrix} 1.0 & 0.18 & -0.13 & -0.26 & 0.19 & -0.25 & -0.12 \\ 0.18 & 1.0 & 0.22 & -0.14 & 0.31 & 0.16 & 0.09 \\ -0.13 & 0.22 & 1.0 & 0.06 & -0.08 & 0.040 & 0.04 \\ -0.26 & -0.14 & 0.06 & 1.0 & \mathbf{0.85} & \mathbf{0.85} & \mathbf{0.85} \\ 0.19 & 0.31 & -0.08 & \mathbf{0.85} & 1.0 & \mathbf{0.85} & \mathbf{0.85} \\ -0.25 & 0.16 & 0.04 & \mathbf{0.85} & \mathbf{0.85} & 1.0 & \mathbf{0.85} \\ -0.12 & 0.09 & 0.04 & \mathbf{0.85} & \mathbf{0.85} & \mathbf{0.85} & 1.0 \end{bmatrix}.$$

This obtained matrix G is not a correlation matrix since the smallest eigenvalue is -0.038. That is why we apply Higham's alternating projections method in order to compute the nearest correlation matrix and obtain the solution X after 35 iterations with

$$X := \begin{bmatrix} 1.0 & 0.1838 & -0.1318 & -0.2514 & 0.1784 & -0.2479 & -0.1191 \\ 0.1838 & 1.0 & 0.2182 & -0.1316 & 0.2986 & 0.1620 & 0.09092 \\ -0.1318 & 0.2182 & 1.0 & 0.05607 & -0.07469 & 0.03905 & 0.03957 \\ -0.2514 & -0.1316 & 0.05607 & 1.0 & 0.8245 & 0.8545 & 0.8521 \\ 0.1784 & 0.2986 & -0.07469 & 0.8245 & 1.0 & 0.8439 & 0.8472 \\ -0.2479 & 0.1620 & 0.03905 & 0.8545 & 0.8439 & 1.0 & 0.8505 \\ -0.1191 & 0.09092 & 0.03957 & 0.8521 & 0.8472 & 0.8505 & 1.0 \end{bmatrix}$$

which has a distance $\|G - X\|_F = 0.0491$ to the matrix G. We realize that the part $X(1:3, 1:3)$ containing the correlation between the non-Asian currencies has changed. However, it is not desired to change this part since these correlations are assumed to be correct so that we apply our Algorithm 3. We set $\mathcal{N} := \{(i,j) : i, j = 1, \ldots, 3\}$ so that the leading part which contains the correlations between the non-Asian currencies remains fixed. We obtain the solution X_P after 45 iterations with

$$X_P := \begin{bmatrix} 1.0 & 0.18 & -0.13 & -0.25 & 0.18 & -0.25 & -0.12 \\ 0.18 & 1.0 & 0.22 & -0.13 & 0.30 & 0.16 & 0.090 \\ -0.13 & 0.22 & 1.0 & 0.060 & -0.070 & 0.040 & 0.040 \\ -0.25 & -0.13 & 0.060 & 1.0 & 0.82 & 0.85 & 0.85 \\ 0.18 & 0.30 & -0.070 & 0.82 & 1.0 & 0.84 & 0.85 \\ -0.25 & 0.16 & 0.040 & 0.85 & 0.84 & 1.0 & 0.85 \\ -0.12 & 0.090 & 0.040 & 0.85 & 0.85 & 0.85 & 1.0 \end{bmatrix}$$

where $\|G - X_P\|_F = 0.0547$. We compare with Finger's obtained matrix. He applied his algorithm described in Section 1.5 with the following adjusting value $\nu = 0.7831$ and obtained as solution the following matrix X_F [8] with

$$X_F := \begin{bmatrix} 1.0 & 0.18 & -0.13 & -0.27 & -0.08 & -0.24 & -0.20 \\ 0.18 & 1.0 & 0.22 & -0.10 & 0.27 & 0.20 & 0.19 \\ -0.13 & 0.22 & 1.0 & 0.04 & -0.01 & 0.03 & 0.04 \\ -0.27 & -0.10 & 0.04 & 1.0 & 0.81 & 0.87 & 0.81 \\ -0.08 & 0.27 & -0.01 & 0.81 & 1.0 & 0.89 & 0.86 \\ -0.24 & 0.20 & 0.03 & 0.87 & 0.89 & 1.0 & 0.86 \\ -0.20 & 0.19 & 0.04 & 0.81 & 0.86 & 0.86 & 1.0 \end{bmatrix}$$

which has a distance $\|G - X_F\|_F = 0.2978$ to the matrix G. Note that we rounded the matrix X_F to the second post decimal position by means of the MATLAB function `round` in order to compare it with the matrix of Finger. The matrix X is rounded to the fourth post decimal position so that the change in the values of the block $X(1:3, 1:3)$ can be recognized.

First we notice that the difference between $\|G - X_P\|_F$ and $\|G - X\|_F$ is only 0.0056 even though we rounded the matrix X_P. Second, we observe that our matrix X_P computed by means of the Algorithm 3 is clearly nearer to the input matrix G than the matrix X_F determined by Finger. This shows that our algorithm returns accurate results. We expect these results because our computed matrix is the numerical solution of our posed optimization problem differing from the true solution only because of rounding errors. In contrast to X_P, the matrix X_F is obtained by an approach which actually finds a matrix being in the feasible set only and does not underlie a minimization strategy.

7.4.4 Conclusion

From the numerical tests our first conclusions are as we expected that the returned matrix has indeed the correlations which are desired to be preserved. Furthermore, we reason that the iteration number increases if we include the further projection onto F and that the intensity of the increase depends on how many correlations are desired to be preserved and how far the distance between the matrix to which the

nearest correlation matrix is desired and the nearest correlation itself is, in other words, how intensively the original correlation matrix was modified. We noticed a large difference in the ratio between the iteration numbers of Higham's alternating projections method and Algorithm 3 when we stressed the correlations locally towards the band correlation stress.

Clearly, a further analysis is needed to understand the behaviour of the alternating projections method in more detail when we include a further projection onto convex sets, in particular onto the set F. It is certainly advisable to consider alternative algorithms with a better convergence rate like the recently proposed method of Qi and Sun in [28] for preserving correlations, in particular, for local correlation stressing and a large size of the matrix.

We also remark here that most of time (about 95%) in all test examples is spent on computing the projection onto S, the main part of the computational time is the eigenvalue decomposition. The time to compute the projection onto U or F was negligible.

Chapter 8

Concluding Remarks

Throughout the first chapters, we have investigated solving the problem of finding the nearest correlation matrix. Based on the algorithm of Qi and Sun which converges quadratically to the solution of this problem, we dealt with the question how we can improve the efficiency and the reliability of this algorithm and suggested a modified version. These modifications concern the iterative method, which we changed from the CG method to the MINRES routine and using a preconditioner in the iterative method. We found an inexpensive way to compute the Jacobi preconditioner. We also changed the method for eigenvalue decomposition from the MATLAB function `eig` to `f08fc` which uses a divide and conquer algorithm (accessing the MATLAB-NAG Toolbox). Furthermore, we detected a problem with the Armijo rule which we tried to avoid by implementing the suggestions in Section 4.7.1. According to our numerical tests in Chapter 6, we conclude that we could improve our efficiency and that our suggested algorithm is reliable and robust. We avoided the problem with the Armijo rule and made sure that the algorithm is competitive with the alternating projections method in terms of the accuracy.

In the second instance, we have investigated the alternating projections method, in particular Higham's method, which solves also the problem of finding the nearest correlation matrix, and variations of it. We added further constraints to the problem, more precisely, involved the option of preserving correlations during the computation and investigated how the iteration number varies towards solving the original problem

with Higham's method. According to the tests in Section 7.4 we conclude that the convergence depends significantly on how many correlations are fixed and how far the distance to the nearest correlation matrix is. Furthermore, we also confirm our remark in Section 1.5 that the alternating projections method of Higham is quite flexible to variations since it was fairly easy to implement the further projection and thus to consider further constraints.

For future work, the Algorithm 2 could also be extended to solving the problem in (1.14) with further constraints, in particular, for fixing correlations. So for example Qi and Sun [28] also propose an algorithm for finding the nearest correlation matrix which involves the option of preserving correlations. They use in their method the quadratically convergent algorithm (Algorithm 1) introduced here. This extended algorithm could be an enormous improvement in efficiency towards the alternating projections method when the nearest correlation matrix is to be computed and certain correlations are to be preserved.

Bibliography

[1] Educational datasets. `http://www.riskmetrics.com/stddownload_edu.html`.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorenson. *LAPACK Users' Guide, 2nd Edition.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1995.

[3] M. F. Anjos, N. J. Higham, M. Takouda, and H. Wolkowicz. A semidefinite programming approach for the nearest correlation matrix problem. Preliminary research report, University of Waterloo, Waterloo, Ontario, 2003.

[4] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, MA, 1995. 2nd edition 1999.

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

[6] Frank Deutsch. *Best Approximation in Inner Product Spaces.* CMS Books in Mathematics/Ouvrages de Mathématiques de la SMC, 7. Springer-Verlag, New York, 2001.

[7] Howard Elman, David Silvester, and Andy Wathen. *Finite Elements and Fast Iterative Solvers with Applications in Incompressible Fluid Dynamics.* Numerical Mathematics and Scientific Computation. Oxford University Press, Oxford, New York, 2005.

[8] C. Finger. A methodology to stress correlations. *RiskMetrics Monitor*, pages 3–11, 1997.

[9] R. W. Freund and N. M. Nachtigal. QMR: A quasi-minimal residual method for non-Hermitian linear systems. *Numerische Mathematik*, 60:315–340, 1991.

[10] Roland W. Freund. A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. *SIAM J. Sci. Comput.*, 14(2):470–482, 1993.

[11] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, 1981.

[12] Gene H. Golub and Charles F. Van Loan. *Matrix Computation*. John Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, Maryland, third edition, 1996.

[13] N. J. Higham. The matrix computation toolbox. `http://www.ma.man.ac.uk/~higham/mctoolbox`.

[14] N. J. Higham. Computing a nearest symmetric positive semidefinite matrix. *Linear Algebra Appl.*, 103:103–118, 1988.

[15] N. J. Higham. Optimization by direct search in matrix computations. *SIAM J. Matrix Anal. Appl.*, 14(2):317–333, April 1993.

[16] N. J. Higham. *Accuracy and Stability of Numerical Algorithms, Second Edition*. SIAM, 2002.

[17] N. J. Higham. Computing the nearest correlation matrix—A problem from finance. *IMA J. of Numerical Analysis*, 22(3):329–343, 2002.

[18] Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Convex Analysis and Minimization Algorithms*. Springer-Verlag, Berlin, 1993.

[19] R. A. Horn and C. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, Cambridge, 1991.

[20] J. E. Dennis Jr. and Robert. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, 1983.

[21] A. León, J. E. Peris, J. Silva, and B. Subiza. A note on adjusting correlation matrices. *Applied Mathematical Finance*, pages 61–67, 2002.

[22] Craig Lucas. Computing nearest covariance and correlation matrices. Master's thesis, University of Manchester, October 2001.

[23] J. Malick. A dual approach to semidefinite least-squares problems. *SIAM J. Matrix Anal. Appl.*, 26(1):272–284, 2004.

[24] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag, New York, 1999.

[25] Numerical Algorithms Group. *NAG Fortran Library Manual, Mark 21*. NAG, Oxford, UK, 2006.

[26] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12:617–629, 1975.

[27] H. Qi and D. Sun. A quadratically convergent Newton method for computing the nearest correlation matrix. *SIAM J. Matrix Anal. Appl.*, 28(2):360–385, 2006.

[28] H. Qi and D. Sun. Correlation stress testing for value-at-risk: An unconstrained convex optimization approach, March 2007.

[29] Riccardo Rebonato and Peter Jäckel. The most general methodology to create a valid correlation matrix for risk management and option pricing purposes, March 20 1999.

[30] R. Tyrrell Rockafellar and Roger J.-B. Wets. *Variational analysis*, volume 317 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 1998.

[31] Y. Saad and M. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 7:856–869, 1986.

[32] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, second edition, 2003.

[33] Virginia J. Torczon. *Multi-Directional Search: A Direct Search Algorithm for Parallel Machines*. PhD thesis, Rice University, Houston, TX, USA, May 1989.

[34] Virginia J. Torczon. On the convergence of the multidirectional search algorithm. *SIAM J. Optim.*, 1(1):123–145, 1991.

[35] Henk A. van der Vorst. *Iterative Krylov Methods for Large Linear Systems*, volume 13 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, UK, 2003.

Appendix A

Implementations

A.1 Qi and Sun's Algorithm

```
%%%%%%% This code is designed to solve %%%%%%%%%%%%%
%%%%%%% min 0.5*<X-G, X-G>
%%%%%%% s.t. X_ii =1, i=1,2,...,n
%%%%%%% X>=0 (symmetric and positive semi-definite) %%%%%%%%%%%%%%%
%%%%%%%
%%%%%%% based on the algorithm  in %%%%%
%%%%%%% ``A Quadratically Convergent Newton Method for %%%
%%%%%%%   Computing the Nearest Correlation Matrix %%%%%
%%%%%%%   By Houduo Qi and Defeng Sun  %%%%%%%%%%%%
%%%%%%%   SIAM J. Matrix Anal. Appl. 28 (2006)360--385.
%%%%%%%
%%%%%%% Last modified date:  August 17, 2006
%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%% The  input argument is the given symmetric G   %%%%%%%%
%%%%%%% The outputt are the optimal primal and dual solutions
%%%%%%%%
%%%%%%%
%%%%%%% Send your comments and suggestions to    %%%%%%
%%%%%%% hdqi@soton.ac.uk  or matsundf@nus.edu.sg %%%%%%
%%%%%%%                                 %%%%%%%%%%%%%%%
%%%%%%% Warning: Accuracy may not be guaranteed!!!!! %%%%%%%%
```

```
%%%%%% Note that this is a Modification of Qi and Sun's algorithm!

function [X,y] = cornewton_qi_sun(G)
disp(' ---Newton method starts--- ')
t0=cputime;
[n,m] = size(G);

global  b0

b0 = ones(n,1);

Res_b = zeros(300,1);

y=zeros(n,1);
y=b0-diag(G);                %Initial point

Fy=zeros(n,1);

k=0;
f_eval = 0;

Iter_Whole = 100;
Iter_inner = 40; % Maximum number of Line Search in Newton method
maxit = 200; %Maximum number of iterations in PCG

Inner = 0;
tol = 1.0e-6;

error_tol = 1.0e-6; % termination tolerance
sigma_1 = 1.0e-4; %tolerance in the line search of the Newton method


x0 = y;
M = eye(n,n); % Preconditioner
```

```
d = zeros(n,1);

C = G + diag(y);
[P,D] = eig(C);
lambda = diag(D);
[f0,Fy] = gradient(y,lambda,P,b0,n);
f_eval = f_eval + 1;
b =b0 - Fy;
norm_b = norm(b);

fprintf('Newton: Norm of Gradient %d \n',norm_b)
Omega = omega_matrix(lambda,n);
x0 = y;

while (norm_b>error_tol & k< Iter_Whole)

  [d,flag,relres,iterk] = pre_cg(b,tol,maxit,M,Omega,P,n);

  %fprintf('Newton: Number of CG Iterations %d \n', iterk)

  if (flag ~= 0); % if CG is unsuccessful, use the negative gradient
   direction
    d = b0 - Fy;
  end %else was stated in the original file

  slope = (Fy-b0)'*d; %%% nabla f d
  y = x0+d; %temporary x0+d

  C = G + diag(y);
  [P,D] = eig(C); % Eig-decomposition: C =P*D*P^T

  lambda = diag(D);

  [f,Fy] = gradient(y,lambda,P,b0,n);
  k_inner = 0;
  while(k_inner <= Iter_inner & f > f0 + sigma_1*0.5^k_inner*slope)
```

```
91      k_inner = k_inner + 1;
        y = x0 + 0.5^k_inner*d; % backtracking
        C = G + diag(y);
        [P,D] = eig(C); % Eig-decomposition: C =P*D*P^T
        lambda = diag(D);
96      [f,Fy] = gradient(y,lambda,P,b0,n);
    end % loop for while
    f_eval = f_eval+k_inner+1;
    x0 = y;
    k = k + 1;
101 f0 = f; %not included in the original file
    b = b0-Fy;
    norm_b = norm(b);
    %fprintf('Newton: Norm of Gradient %d \n',norm_b)

106 Res_b(k) = norm_b;
    Omega = omega_matrix(lambda,n);
    % here end statement in the original file
  end %end loop for while
  fprintf('Newton: function value %d \n',f0)
111 fprintf('Newton: Norm of Gradient %d \n',norm_b)
  fprintf('Newton: Number of Iterations %d \n', k)
  fprintf('Newton: Number of Function Evaluations %d \n', f_eval)
  i = 1;
  %sum(max(lambda,0).^2)
116 C = P';
  while (i < n+1)
    C(i,:) = max(0,lambda(i)) * C(i,:);
    i = i + 1;
  end
121 X = P*C; % Optimal solution X*
  %norm(X,'fro')^2
  time_used = cputime - t0

  %%% end of the main program
126
```

```
%%%%%%
%%%%%% To generate F(y) %%%%%%%
%%%%%%%

function [f,Fy] = gradient(y,lambda,P,b0,n)
%global P omega
%[n,n]=size(P);
f = 0.0;
Fy = zeros(n,1);
%Im =find(lambda<0);
%Ip =find(lamba>=0);
%lambdap=max(0,lambda);
%H =diag(lambdap); %% H =P^T* diag(x) *P
%  H =H*P'; %%% Assign H*P' to H
H = P';
i = 1;
while (i <= n)
  H(i,:) = max(lambda(i),0)*H(i,:);
  i = i+1;
end
i = 1;
while (i <= n)
  Fy(i) = P(i,:) * H(:,i);
  i = i+1;
end
%compute frobenius - norm
i = 1;
while (i <= n)
  f = f+(max(lambda(i),0))^2;
  i = i+1;
end

f = 0.5*f - b0'*y;

return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% end of gradient.m %%%%%%

%%%%%%%%%%%%%%%         %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% To generate the first -order difference of lambda
%%%%%%%
function omega = omega_matrix(lambda,n)
omega = ones(n,n);
%Im =find(lambda<0);
%Ip =find(lamba>=0);
i = 1;
while (i <= n)
  j = 1;
  while (j <= n)
    if abs(lambda(i) - lambda(j))>1.0e-10
      omega(i,j) = (max(0,lambda(i)) - max(0,lambda(j)))/(lambda(i)-
      lambda(j));
    elseif max(lambda(i),lambda(j)) <= 1.0e-15
      omega(i,j) = 0;
    end
    j = j+1;
  end
  i = i+1;
end

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% end of omega_matrix.m %%%%%%%%%%

%%%%%% PCG method %%%%%%%
%%%%%%% This is exactly the algorithm by  Hestenes and Stiefel
(1952)
%%%%%An iterative method to solve A(x) =b
%%%%%The symmetric positive definite matrix M is a
```

```
%%%%%%%% preconditioner for A.
%%%%%%  See Pages 527 and 534 of Golub and va Loan (1996)

function [p,flag,relres,iterk] = pre_cg(b,tol,maxit,M,Omega,P,n);
% Initializations
r = b;  %We take the initial guess x0=0 to save time in calculating
A(x0)
n2b = norm(b);     % norm of b
tolb = tol * n2b;  % relative tolerance
p = zeros(n,1);
flag = 1;
iterk = 0;
relres = 1000; %%% To give a big value on relres
% Precondition
z = r;  %%%%% z = M\r; if M is not the identity matrix
rz1 = r'*z;
rz2 = 1;
d = z;
% CG iteration
for k = 1:maxit
  if k > 1
    beta = rz1/rz2;
    d = z + beta*d;
  end
  w = Jacobian_matrix(d,Omega,P,n); %w = A(d);
  denom = d'*w;
  iterk = k;
  relres = norm(z)/n2b;              %relative residue =norm(z) /
  norm(b)
  if denom <= 0
    sssss = 0
    p = d/norm(d); % d is not a descent direction
    break % exit
  else
    alpha = rz1/denom;
    p = p + alpha*d;
```

```
231     r = r - alpha*w;
      end
      z = r; %  z = M\r; if M is not the identity matrix ;
      if norm(z) <= tolb % Exit if Hp=b solved within the relative
      tolerance, condition 35
        iterk = k;
236     relres = norm(z)/n2b;           %relative residue =norm(z) / norm
        (b)
        flag = 0;
        break
      end
      rz2 = rz1;
241   rz1 = r'*z;
    end

    return

246 %%%%%%%% %%%%%%%%%%%%%
    %%% end of pre_cg.m%%%%%%%%%%%



251 %%%%%% To generate the Jacobian product with x: F'(y)(x) %%%%%%%
    %%%%%%%

    function Ax = Jacobian_matrix(x,Omega,P,n)

256 Ax = zeros(n,1);
    %Im =find(lambda<0);
    %Ip =find(lamba>=0);
    %H =diag(x);
    H = P;
261 i = 1;
    while (i <= n)
      H(i,:) = x(i)*H(i,:); % H=diag(x)*P
      i = i + 1;
```

```
end
H = P'*H; %% H =P^T* diag(x) *P
i = 1;
while (i <= n)
  %j=1;
  %while (j<=n)
  %     H(i,j)=Omega(i,j)*H(i,j);
  %     j=j+1;
  %end
  H(i,:) = Omega(i,:).*H(i,:);
  i = i+1;
end
H = H*P'; %%% Assign H*P' to H= Omega o P^T*diag(x)*P)*P^T
i = 1;
while (i <= n)
  Ax(i) = P(i,:)*H(:,i);
  i = i + 1;
end

return

%%%%%%%%%%%%%%
%end of Jacobian_matrix.m%%%
```

A.2 Implementation of Algorithm 2

```
%PROGRAM FOR FINDING THE NEAREST CORRELATION MATRIX
%Modified version of Qi and Sun's algorithm

function [X,iter,f_eval,normgrad] = cornewton(G,error_tol,flag2,
diaones,maxitmeth,maxit,pre,prnt)
% [X,iter,f_eval,normgrad] = cornewton(G,error_tol,flag2,diaones,
maxitmeth,maxit,pre,prnt)
% finds the nearest correlation matrix to the symmetric matrix G.
% error_tol is a termination tolerance
% (default is length(G)*10^(-7)).
```

```
% flag2 = 0: using for full eigendecomposition MATLAB function eig
% (LAPACK-function DSYEV).
% flag2 = 1: using for full eigendecomposition MATLAB-NAG Toolbox
% function f08fa
% flag2 = 2: using for full eigendecomposition MATLAB-NAG Toolbox
% function f08fc (default)
% flag2 = 3: using for full eigendecomposition MATLAB-NAG Toolbox
% function f08fd
% diaones = 0: the matrix X is not changed after
% the computation (default)
% diaones = 1: the diagonal is set to 1 after computing X
% diaones = 2: the diagonal is rescaled to 1 and then set to 1
% maxitmeth: is the number of the maximal iterations in the
% iterative method,  (default is 200)
% maxit: is the number of maximal iterations in the Newton-method,
% (default is 200)
% pre = 1: using a preconditioner in the iterative method
% (default is 1)
% prnt = 1 for display of intermediate output. (default is 0)

%Test input values


%Test the matrix
if isempty(G), error_function(5), end
[n,m] = size(G);
if (m ~= n), error_function(6); end;
if (~isreal(G)), error_function(7); end;
if ~isequal(G,G')
  %matrix not symmetric
  G = (G + G')/2;
end;
%Test error_tol
if nargin < 2 || isempty(error_tol)
  error_tol = length(G) * 10^(-7);
else
  if error_tol <= 0, error_function(8), end
```

```
  end
  %Test flag2
  if nargin < 3 || isempty(flag2)
48   flag2 = 2;
  end
  %Test diaones
  if nargin < 4 || isempty(diaones), diaones = 0; end
  %Test maxitmeth
53 if nargin < 5 || isempty(maxitmeth)
    maxitmeth = 200;
  else
    if maxitmeth <= 0, error_function(9), end
  end
58 %Test maxit
  if nargin < 6 || isempty(maxit)
    maxit = 200;
  else
    if maxit <= 0, error_function(10), end
63 end
  %Test pre
  if nargin < 7 || isempty(pre), pre = 1; end

  %Test prnt
68 if nargin < 8 || isempty(prnt), prnt = 0; end

  print_function(prnt,1); %print Newton-method starts

  t0 = cputime;
73
  % Determination of constants
  Iter_inner = 40; % Maximum number of Line Search in Newton method
  tol = 1.0e-6;   %constant for the angle condition
  sigma_1 = 1.0e-4; %tolerance in the line search of the Newton method
78 b0 = ones(n,1);

  method = 'pminres';
```

```
%method is a string which determinates the iterative method
% method = 'CG' - Conjugate gradient method (NAG-routine)
% method = 'SYMMLQ' - SYMMLQ (NAG-routine)
% method = 'pminres' - MINRES (implemetation of David Silvester)
% (default)
% method = 'RGMRES'  - restarted generalized minimal residual
% method (NAG-routine)
% method = 'TFQMR' - transpose free quasi minimal residual method
% (NAG-routine)
% method = 'BIGCSTAB' - bi-conjugate gradient stabilized method
% (NAG-routine)


%Initial values
y = b0 - diag(G); %Initial point
x0 = y;

f_eval = 0;

[P,lambda] = eigdecomp(G,y,flag2,n);
%compute grad(theta(y))
[f0,Fy] = gradient(y,lambda,P,b0,n);
f_eval = f_eval + 1;
b = b0 - Fy;
grad = -b;
normgrad = norm(b);
Omega = omega_matrix(lambda,n);

print_function(prnt,2,normgrad); %print the norm of the gradient

k = 0;
% NEWTON-ITERATION STARTS
while (normgrad > error_tol && k < maxit)
  % COMPUTE V_k *d = -grad
  if(pre)
    precond = precond_matrix(Omega,P);
```

```
    [d,flag,iterk] = solver(@(x)Jacobian_matrix(x,Omega,P,n,0),n,b,
    min(0.5,normgrad),maxitmeth,method,precond);
  else
    [d,flag,iterk] = solver(@(x)Jacobian_matrix(x,Omega,P,n,0),n,b,
    min(0.5,normgrad),maxitmeth,method,[]);
  end;

  %TEST CONDITION 2 (angle condition, descent conditon)
  if (flag == 0)
    norm_d = norm(d);
    if (abs(norm_d) <= eps)
      %X = computeX(P,lambda,n,diaones);
      error_function(12);
    end;
    %Test for descent direction and angle-condition
    if (-(grad/norm_d)'*(d/norm_d) < min(tol,normgrad))
      flag = 1;
    else
      slope = (grad)'*d;
    end;
  end;
  % if iterative method was unsuccessful or the angle condition
  % failed, use the negative gradient direction
  if (flag ~= 0)
    d = -grad;
    slope = -normgrad^2;
  end

  print_function(prnt,3,flag,k); % print which direction we use
  print_function(prnt,4,k,iterk);

  %Temporary update
  y = x0 + d;
  [P,lambda] = eigdecomp(G,y,flag2,n);
  [f,Fy] = gradient(y,lambda,P,b0,n); %compute F(y)
  norm_x=norm(x0);
```

```
    f_eval = f_eval + 1;

153 %ARMIJO-CONDITION
    % test whether accuracy problem can occur
    if (abs((f - f0)/(1 + abs(f) + abs(f0)))/100 < eps) && (abs(slope
    /(1 + abs(f0) + abs(f))/100) < eps)
      %TEST convergence condition
      print_function(prnt,13);
158   if (norm(Fy - b0)/normgrad > 0.9)
        %Take negative gradient direction
        d = -grad;
        if (norm(d)/(1 + norm_x)/10 < eps)
          error_function(1); %warning d too small
163       break;
        end;
        print_function(prnt,15);
        y = x0 + d;
        [P,lambda] = eigdecomp(G,y,flag2,n);
168     [f,Fy] = gradient(y,lambda,P,b0,n);
        f_eval = f_eval + 1;
      else
        print_function(prnt,14);
      end
173 else
      %ARMIJO-BACK-TRACKING
      k_inner = 0;
      while (k_inner <= Iter_inner && f > f0 + sigma_1*0.5^k_inner*
      slope)
        k_inner = k_inner + 1;
178     y = x0 + 0.5^k_inner * d; % backtracking
        [P,lambda] = eigdecomp(G,y,flag2,n);
        [f,Fy] = gradient(y,lambda,P,b0,n);% compute F(y)
        if (abs((f - f0)/(1 + abs(f) + abs(f0)))/100 < eps) && (abs(
        slope/(1 + abs(f0) + abs(f))/100) < eps)
          %TEST convergence condition
183       print_function(prnt,13);
```

```
        if (norm(Fy - b0)/normgrad > 0.9)
          d=0; %next if-condition is satisfied
          break;
        else
          break;
        end
      end;
    end; % loop for while

    %Check whether step is sufficiently large
    if (k_inner > 0) && (norm(0.5^k_inner*d)/(1 + norm_x)/10 < eps)
      %Take negative gradient direction
      d = -grad;
      if (norm(d)/(1 + norm_x)/10 < eps)
        error_function(1); %warning d too small
        break;
      end;
      print_function(prnt,15);
      y = x0 + d;
      [P,lambda] = eigdecomp(G,y,flag2,n);
      [f,Fy] = gradient(y,lambda,P,b0,n);
    end;
    print_function(prnt,5,k,k_inner); %print the iterations number
    f_eval = f_eval + k_inner;
  end;

  %UPDATE
  f0 = f;
  x0 = y;
  k = k + 1;
  b = b0 - Fy;
  norm_b = norm(b);
  %TEST whether norm of the gradient has become smaller
  if (abs(norm_b - normgrad)/(1 + norm_b + normgrad)/10 < eps)
    error_function(1); %warning minimal value achieved
    break;
```

```
    end;
    normgrad = norm_b;
    grad = -b;
    Omega = omega_matrix(lambda,n);
  end %end loop for while
  if (k > maxit)
    error_function(2); %warning Maximal iteration number achieved
  end;

  iter = k;
  %compute X
  if (iter == 0) && (min(lambda) >= 0)
    %Setting the diagonal to 1 by our chosen
    %starting value was
    %sufficient to satisfy the stopping criterion
    %and all eigenvalues are nonnegative
    X = G - diag(diag(G)) + eye(n);
  else
    X = computeX(P,lambda,n,diaones);
  end;
  %PRINT outputs
  print_function(prnt,7,f0);
  print_function(prnt,8,normgrad);
  print_function(prnt,9,k);
  print_function(prnt,10,f_eval);
  print_function(prnt,12,norm(G - X,'fro'));
  print_function(prnt,11,cputime - t0); %print time


  %%% end of the main program


  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %function for computing the solution X
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  function [X] = computeX(P,lambda,n,diaones)
```

```
C = P';
for i = 1:n
  C(i,:) = max(0,lambda(i)) * C(i,:);
end
X = P*C; % Optimal solution X*
switch diaones
  case 1 %set diagonal to 1
    X = X - diag(diag(X)) + eye(n);
  case 2 %rescale diagonal to 1 and set diagonal to 1
    scale = diag(X);
    scale(find(scale <= 1.0e-6)) = 1.0e-6;
    scale = scale.^(-1/2);
    X = diag(scale) * X * diag(scale);
    X = X - diag(diag(X)) + eye(n);
end;
X = (X + X')/2; %Ensuring symmetry


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%function for computing the eigenvalue decomposition of G+Diag(y)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [P,lambda] = eigdecomp(G,y,flag2,n)
C = G + diag(y);
%eigendecomposion
switch flag2
  case 0
    [P,D] = eig(C);
    lambda = diag(D);
    info = 0;
  case 1
    [P,lambda,info] = f08fa('V','U',C);
  case 4
    [a, m, lambda, P, jfail, info] = f08fb('V', 'A', 'U', C,0,0,
    int32(1),int32(n),0);
  case 3
```

```
      [a,m,lambda,P,is,info] = f08fd('V','A','U',C,0,0,int32(1),int32(
      n),0);
    otherwise
      [P, lambda, info] = f08fc('V', 'U', C);
end;
if ((flag2 > 1) && (info ~= 0))
  error_function(3);
  [P,lambda,info] = f08fa('V','U',C);
end;
if (info ~= 0)
  error_function(4);
end;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%function for printing warnings and errors on the screen
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function error_function(pos)
switch pos
  case 1
    warning('Machine precision is limiting convergence.');
  case 2
    warning('Maximal iteration number achieved.');
  case 3
    warning('Error occured during computing an eigenvalue
    decomposition, trying another method.');
  case 4
    error('Error occured during computing an eigenvalue
    decomposition');
  case 5
    error('No matrix.');
  case 6
    error('Matrix is not square.');
  case 7
    error('Matrix is not real.');
  case 8
```

```
323     error('error_tol must be greater than zero');
    case 9
      error('maxitmeth must be greater than zero');
    case 10
      error('maxit must be greater than zero');
328 case 12
      error('Norm of the step direction too small.');
  end;


333 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %function for printing comments on the screen
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  function print_function(prnt,pos,para1,para2)
  if prnt
338   switch pos
      case 1
        disp(' ---Newton method starts--- ');
      case 2
        fprintf('Newton: Norm of Gradient %d \n',para1);
343   case 3
        if (para1 ~= 0)
          fprintf('%2.0f:  we use linesearch method\n', para2);
        else
          fprintf('%2.0f:  we use Newton method\n', para2);
348     end
      case 4
        fprintf('%2.0f:  we use %4.0f iteration in CG\n', para1,para2)
        ;
      case 5
        fprintf('%2.0f:  inner-iterations %2.0f\n', para1,para2);
353   case 7
        fprintf('Newton: function value %d \n',para1);
      case 8
        fprintf('Newton: Norm of Gradient %d \n',para1);
      case 9
```

```
358         fprintf('Newton: Number of Iterations %d \n', para1);
        case 10
          fprintf('Newton: Number of Function Evaluations %d \n', para1)
          ;
        case 11
          fprintf('Time used: %d \n',para1);
363     case 12
          fprintf('The distance ||G-X||_F is: %d \n',para1);
        case 13
          fprintf('Armijo deactivated\n');
        case 14
368       fprintf('Taking Newton-direction with step length 1\n');
        case 15
          fprintf('Taking negative gradient direction\n');
      end;
    end;
373

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %function for generating J(y)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
378
    function [f,Fy] = gradient(y,lambda,P,b0,n)
    H = P;
    for i = 1:n
      H(:,i) = max(lambda(i),0) * H(:,i);
383 end
    Fy = sum(P.*H,2);

    %compute Frobenius - norm squared
    f = sum(max(lambda,0).^2);
388 f = 0.5*f - b0'*y;



    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% function for generating the constructed matrix M_y
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function omega = omega_matrix(lambda,n)
omega = ones(n,n);
for i = 1:n
  for j = 1:n
    if abs(lambda(i) - lambda(j))>1.0e-10
      omega(i,j) = (max(0,lambda(i)) - max(0,lambda(j)))/(lambda(i)
      - lambda(j));
    elseif max(lambda(i),lambda(j)) <= 1.0e-15
      omega(i,j) = 0;
    end
  end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function for generating the preconditioner
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function c = precond_matrix(Omega,P)
PS = P.^2;
c = sum(PS.*(PS * Omega),2); %L_k = W_k*Q_k, v_ii = q_i^l_i
c(find(c <= 1.0e-6)) = 1.0e-6;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function for solving our linear system V_y d = -grad
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [d,ifail,iterk] = solver(fun,n,b,tol,maxitmeth,method,
precond)
i = 0;
if (strcmp(method,'CG'))
  %initialize the solver
  if isempty(precond)
    [lwreq,work,ifail] = f11gd(method,'N',int32(n),tol,int32(
    maxitmeth),eps,0,int32(min(10,4)),int32(maxitmeth),'norm_p','2',
    'weight','N','iterm',int32(1));
```

```
    else
      [lwreq,work,ifail] = f11gd(method,'P',int32(n),tol,int32(
      maxitmeth),eps,0,int32(min(10,4)),int32(maxitmeth),'norm_p','2',
      'weight','N','iterm',int32(1));
    end;
    irevcm = int32(0); d = zeros(n,1); v = b;

    while (((irevcm == 0) || (irevcm == 1) || (irevcm == 2)) && (ifail
     == 0))
      [irevcm, d, v, work, ifail] = f11ge(irevcm, d, v, zeros(n,1),
      work, 'lwork', int32(120+5*n));
      i = i + 1;
      if ((irevcm == 1) && (ifail == 0))
        v = fun(d);
      end;
      if ((irevcm == 2) && (ifail == 0))
        v = d./precond;
      end;
    end;
    [iterk] = f11gf(work, 'lwork', int32(120+5*n));
  end

  if (strcmp(method,'SYMMLQ'))
    %initialize the solver
    if isempty(precond)
      [lwreq,work,ifail] = f11gd(method,'N',int32(n),tol,int32(
      maxitmeth),eps,0,int32(min(10,4)),int32(maxitmeth),'norm_p','2',
      'weight','N','iterm',int32(1));
    else
      [lwreq,work,ifail] = f11gd(method,'P',int32(n),tol,int32(
      maxitmeth),eps,0,int32(min(10,4)),int32(maxitmeth),'norm_p','2',
      'weight','N','iterm',int32(1));
    end;
    irevcm = int32(0); d = zeros(n,1); v = b;
```

```
    while (((irevcm == 0) || (irevcm == 1) || (irevcm == 2)) && (ifail
     == 0))
453   [irevcm, d, v, work, ifail] = f11ge(irevcm, d, v, zeros(n,1),
      work, 'lwork', int32(120+6*n));
      i = i + 1;
      if (irevcm == 1) && (ifail == 0)
        v = fun(d);
      end;
458   if (irevcm == 2) && (ifail == 0)
        v = d./precond;
      end;
    end;
    [iterk] = f11gf(work, 'lwork', int32(120 + 6*n));
463
  end

  if (strcmp(method,'RGMRES'))
    %initialize the solver
468 m = 3;
    lwork = int32(101 + n*(m+3) + m*(m+5));
    if (isempty(precond))
      [lwreq, work, ifail] = f11bd(method, 'N', int32(n), int32(m),
      tol, int32(maxitmeth), eps, 0, int32(maxitmeth),lwork,'norm_p','
      2', 'weight','N','iterm', int32(1));
    else
473   [lwreq, work, ifail] = f11bd(method, 'P', int32(n), int32(m),
      tol, int32(maxitmeth), eps, 0, int32(maxitmeth),lwork,'norm_p','
      2', 'weight','N','iterm', int32(1));
    end;

    irevcm = int32(0); d = zeros(n,1); v = b;

478 while (((irevcm == 0) || (irevcm == 1) || (irevcm == 2)) && (ifail
     == 0))
      [irevcm, d, v, work, ifail] = f11be(irevcm, d, v, zeros(n,1),
      work, 'lwork', lwork);
```

```
      i = i + 1;
      if (irevcm == 1) && (ifail == 0)
        v = fun(d);
483   end;
      if (irevcm == 2) && (ifail == 0)
        v = d./precond;
      end;
    end;
488 [iterk] = f11bf(work, 'lwork', lwork);

  end
  if (strcmp(method,'BICGSTAB'))
    %initialize the solver
493 m = 1;
    lwork = int32(100 + (2*n + m) * (m + 2) + n);
    if (isempty(precond))
      [lwreq, work, ifail] = f11bd(method, 'N', int32(n), int32(m),
      tol, int32(maxitmeth), eps, 0, int32(maxitmeth),lwork, 'norm_p',
      '2','weight','N','iterm', int32(1));
    else
498   [lwreq, work, ifail] = f11bd(method, 'P', int32(n), int32(m),
      tol, int32(maxitmeth), eps, 0, int32(maxitmeth),lwork, 'norm_p',
      '2','weight','N','iterm', int32(1));
    end;
    irevcm = int32(0); d = zeros(n,1); v = b;
    %min(precond)
    while (((irevcm == 0) || (irevcm == 1) || (irevcm == 2)) && (ifail
    ==0))
503   [irevcm, d, v, work, ifail] = f11be(irevcm, d, v, zeros(n,1),
      work,'lwork', lwork);
      i = i + 1;
      if (irevcm == 1) && (ifail == 0)
        v = fun(d);
      end;
508   if (irevcm == 2) && (ifail == 0)
        v = d./precond;
```

```
    end;
  end;
  [iterk] = f11bf(work, 'lwork', lwork);
end
if (strcmp(method,'TFQMR'))
  %initialize the solver
  lwork = int32(100 + 10*n);
  if(isempty(precond))
    [lwreq, work, ifail] = f11bd(method, 'N', int32(n), int32(n),
    tol, int32(maxitmeth), eps, 0, int32(maxitmeth),lwork,'norm_p','
    2','weight','N','iterm', int32(1));
  else
    [lwreq, work, ifail] = f11bd(method, 'P', int32(n), int32(n),
    tol, int32(maxitmeth), eps, 0, int32(maxitmeth),lwork,'norm_p','
    2','weight','N','iterm', int32(1));
  end;
  irevcm = int32(0); d = zeros(n,1); v = b;

  while (((irevcm == 0) || (irevcm == 1) || (irevcm == 2)) && (ifail
   == 0))
    [irevcm, d, v, work, ifail] = f11be(irevcm, d, v, zeros(n,1),
    work, 'lwork', lwork);
    i = i + 1;
    if (irevcm == 1) && (ifail == 0)
      v = fun(d);
    end;
    if (irevcm == 2) && (ifail == 0)
      v = d./precond;
    end;
  end;
  [iterk] = f11bf(work, 'lwork', lwork);
end;
if (strcmp(method,'pminres'))
  if (isempty(precond))
    [d,ifail,iterk] = pminres(@(x)fun(x),@(x)x,b,tol,maxitmeth,zeros
    (n,1));
```

```
    else
      [d,ifail,iterk] = pminres(@(x)fun(x),@(x)(x./precond),b,tol,
      maxitmeth,zeros(n,1));
    end;
  end;
543
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %function for generating the Jacobian product with d: V_y d
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
548 function Ax = Jacobian_matrix(x,Omega,P,n,ep)
  H = P;
  for i = 1:n
    H(i,:) = x(i) * H(i,:); % H = diag(x)*P
  end
553 H = Omega .* (H' * P); % H = M o P^T* diag(x) *P
  H = P * H; %%% Assign H*P' to H = (M o P^T*diag(x)*P)*P^T

  %Computing the diagonal elements
  Ax = sum(P .* H,2) + ep*x;
```

A.3 Implementation of Algorithm 3

```
  function [X,iter] = APfix(A,tol,flag,maxits,n_pos_eig,w,fun,prnt)
  %NEAR4     Nearest correlation matrix.
3 %          X = NEAR4(A,TOL,FLAG,MAXITS,N_POS_EIG,W,PRNT)
  %          finds the nearest correlation matrix to the symmetric
  matrix A.
  %          TOL is a convergence tolerance, which defaults to 16*EPS.
  %          If using FLAG == 1, TOL must be a 2-vector, with first
  component
  %          the convergence tolerance and second component a tolerance
8 %          for defining "sufficiently positive" eigenvalues.
  %          FLAG = 0: solve using full eigendecomposition (EIG).
  %          FLAG = 1: treat as "highly non-positive definite A" and
  solve
```

```
%                   using partial eigendecomposition (EIGS).
%         FLAG = 2: same as FLAG ==1 except uses LAPACK's bisection
code.
%         N_POS_EIG (optional) is the known number of positive
eigenvalues of A.
%         W is a vector defining a diagonal weight matrix diag(W).
%         FUN(Z) is a function changing entries of Z to those
entries which are
%         desired to be preserved in the output matrix X (default
%         FUN=@(Z)Z)
%         PRNT = 1 for display of intermediate output.
%
%         By N. J. Higham, 13/6/01.
%         Reference:  N. J. Higham, Computing the nearest
correlation
%         matrix---A problem from finance. IMA J. Numer. Anal.,
%         22(3):329-343, 2002.

if ~isequal(A,A'), error('A must by symmetric.'), end
if nargin < 2 || isempty(tol), tol = length(A)*eps*[1 1]; end
if nargin < 3, flag = 0; end
if nargin < 4, maxits = 100; end
if nargin < 6 || isempty(w), w = ones(length(A),1); end
if nargin < 7 || isempty(fun), fun=@(x)x; end;
if nargin < 8, prnt = 0; end %1

n = length(A);
if flag >= 1
  if isempty(n_pos_eig)
    [V,D] = eig(A); d = diag(D);
    n_pos_eig = sum(d >= tol(2)*d(n));
  end
  if prnt, fprintf('n = %g, n_pos_eig = %g\n', n, n_pos_eig), end
end

X = A; Y = A; Z=A; b=ones(length(A),1);
```

```
iter = 1;
rel_diffX = inf; rel_diffY = inf; rel_diffXY = inf; grad=inf;
rel_diffXZ = inf;
rel_diffZ = inf;
dS = zeros(size(A));

w = w(:); Whalf = sqrt(w*w');
%rel_diffX rel_diffY rel_diffXY
while max([rel_diffX rel_diffY rel_diffXY rel_diffXZ rel_diffZ]) >
tol(1)
  Xold = X;
  R = X - dS;
  R_wtd = Whalf.*R;
  if flag == 0
    X = proj_spd(R_wtd);
  elseif flag == 1
    [X,np] = proj_spd_eigs(R_wtd,n_pos_eig,tol(2));
  elseif flag == 2
    [X,np] = proj_spd_bisect(R_wtd,n_pos_eig,tol);
  end
  X =  X ./ Whalf;
  grad=norm(diag(X)-b,inf);
  dS = X - R;

  Zold = Z;
  Z = proj_values(Z,fun);
  Yold = Y;
  Y = proj_unitdiag(Z);
  rel_diffZ = norm(Z-Zold,'fro')/norm(Z,'fro');
  rel_diffXZ = norm(Z-X,'fro')/norm(Z,'fro');

  rel_diffX = norm(X-Xold,'fro')/norm(X,'fro');
  rel_diffY = norm(Y-Yold,'fro')/norm(Y,'fro');
  rel_diffXY = norm(Y-X,'fro')/norm(Y,'fro');

  if prnt
```

```
    fprintf('%2.0f:  %9.2e  %9.2e  %9.2e', ...
78      iter, rel_diffX, rel_diffY, rel_diffXY)
    if flag >= 1, fprintf('  np = %g\n',np), else fprintf('\n'), end
  end

  iter = iter + 1;
83  if iter > maxits, error(['Stopped after ' num2str(maxits) ' its.'
  ]), end
  X = Y;

  %   X, pause
end
88


%%%%%%%%%%%%%%%%%%%%%%%%
function A = proj_spd(A)
%PROJ_SPD
93
if ~isequal(A,A'), error('Not symmetric!'), end
[V,D] = eig(A);
A = V*diag(max(diag(D),0))*V';
A = (A+A')/2; % Ensure symmetry.
98
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [A,n_pos_eig_found] = proj_spd_eigs(A,n_pos_eig,tol)
%PROJ_SPD_EIGS

103 if ~isequal(A,A'), error('Not symmetric!'), end
k = n_pos_eig + 10; % 10 is safety factor.
if k > length(A), k = n_pos_eig; end
opts.disp = 0;
[V,D] = eigs(A,k,'LA',opts); d = diag(D);
108 j = (d > tol*max(d));
n_pos_eig_found = sum(j);
% [n_pos_eig sum(j)]
```

```
A = V(:,j)*D(j,j)*V(:,j)';  % Build using only the selected
eigenpairs.
A = (A+A')/2; % Ensure symmetry.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [A,n_pos_eig_found] = proj_spd_bisect(A,n_pos_eig,tol)
%PROJ_SPD_BISECT

if ~isequal(A,A'), error('Not symmetric!'), end
k = n_pos_eig + 10; % 10 is safety factor.
if k > length(A), k = n_pos_eig; end
%[V,d] = eig_big2(A,k,tol(1)/100);  % Call mex file.  100 = safety
factor.
[V,d] = eig_mex(A,k,tol(1)/100);
d = d(1:k);
j = (d > tol(2)*max(d)); n_pos_eig_found = sum(j);
if n_pos_eig_found > n_pos_eig fprintf('>>> '), [n_pos_eig
n_pos_eig_found], end
A = V(:,j)*diag(d(j))*V(:,j)';  % Build using only the selected
eigenpairs.
A = (A+A')/2; % Ensure symmetry.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function A = proj_unitdiag(A)
%PROJ_UNIT

n = length(A);
A(1:n+1:n^2) = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function A = proj_values(Z,fun)
%PROJ_VALUES
A = fun(Z); %(X,F,A);
```