Correcting Non Positive Definite Correlation Matrices

Stef Marée
BSc Thesis Applied Mathematics, TU Delft
s.c.maree@student.tudelft.nl

June 15, 2012

To my little sisters.

Contents

1 Introduction

1.1 Motivation

This report is written as a part of my bachelor internship in the Derivative, Research and Validation team of Rabobank International.

In finance, the movements of assets, like stocks or interest rates are often modelled by the Wiener process, see for example [18]. In the case of a financial derivative on two or more assets, the correlation between the underlying is an important driver for this process and should be taken into account in the modeling. By writing the correlations in matrix form, Cholesky decomposition is applied to this matrix to correlate the Wiener process linearly.

However, Cholesky decomposition prerequisites that the correlation matrix is positive definite, see Theorem 4. Unfortunately, this is not always the case since correlations are not directly available in the market, but have to estimated from for example market data. If there is not enough market data available, the estimated correlation matrix could be non positive definite. In this report, we consider different algorithms of *correcting* these *invalid* correlation matrices.

By *correcting invalid* correlation matrices, we mean we are looking for the *nearest* valid correlation matrix given an *invalid* correlation matrix. As a measure of nearness we use the Frobenius norm.

At Rabobank International, for a specific application, time series of seven years of weekly market data have to be analyzed. This means that 364 correlation matrices are processed, of which all of them have to be checked for positive definiteness. Subsequently, we have to *correct* all non positive definite correlation matrices. Since a lot of these checks for positive definiteness have to be done, we look for a fast and reliable method to do this.

The target of this report is to derive an algorithm that computes the nearest *valid* correlation matrix as *fast* as possible. However, it is also important that the resulting valid matrix is *close* to the original invalid matrix.

An invalid correlation matrix contains estimated correlations. While estimating these correlations, one has more confidence in the value of some correlation than is has in others. Therefore, we also consider a *weighted* distance measure.

All of the methods we discuss in this report are implemented in Matlab, and the programs are included in the end of the report. All tests are performed in Matlab 7.11.0 (R2010b) on a Windows 7 64-Bit pc with a 3.07 GHz processor and 4GB of memory.

1.2 Literature

A lot of research is done on the subject of adjusting invalid correlation matrices. See for example [16, 20, 10]. This is most likely due to the fact that there is no analytic or exact solution for finding the nearest correlation matrix [10].

In 1999 Rebonato and Jäckel published the article titled: "The most general methodology to create a valid correlation matrix for risk management and option pricing purposes." [16], which mentions two algorithms of correcting correlation matrices. The first one is the Hypersphere Decomposition method. This method uses a parameterization in terms of hypersphere coordinates, which guarantee that the resulting matrix will be positive definiteness with unit diagonal. The other method Rebonato and Jäckel introduce is the Spectral Decomposition method. This method is based on the property that all the eigenvalues of a positive definite matrix are positive. After decomposing the correlation matrix in its eigenvalues and eigenvectors, they replace the negative eigenvalues by a *small* positive value and recompose it back together. However, this destroys the unit diagonal. The clever solution proposed by Rebonato and Jäckel is to scale the

matrix back in a way it has unit diagonal again. Improvements on the methods of Rebonato and Jäckel are made in [17, 15].

Two years later, Higham published the Alternating Projections method [10]. Given the subset of all positive definite matrices and the subset of all matrices with unit diagonal, the Alternating Projections method iterates over both subsets until it reaches the intersection of it.

Zhang and Yin proposed a gradient updating scheme for finding the nearest correlation matrix [20]. Also Qi and Sun succeeded in calculating the dual of the nearness problem, and this resulted in a quadratically convergent quasi-Newton algorithm [14].

Kurowicka and Cooke [12] used partial correlations and vines to find the nearest correlation matrix by only changing those correlations that *destroy* the positive definiteness of the matrix.

1.3 Background Knowledge

In this section the necessary background knowledge is given. Consider a portfolio containing n assets, denoted by the random variables $X_1, \ldots, X_n$. Let $i, j \in \{1, 2, \ldots, n\}$, then denote the expected value of X_i by $\mathbb{E}(X_i) = \mu_i$ and the variance of X_i by $\mathrm{Var}(X_i) = \sigma_i^2$. We now define the correlation between a pair X_i and X_j.

Definition 1 (Pearson Correlation)**.** *The correlation between two random variables X_i and X_j is given by*

$$\rho_{i,j} = \frac{\mathbb{E}\left[(X_i - \mu_i)(X_j - \mu_j)\right]}{\sigma_i \sigma_j},$$

and is defined only if both σ_i and σ_j are finite and nonzero.

Bounds for the correlations can be found using the Cauchy-Schwarz inequality [3, p. 30] Note that $\sigma_i^2 = \mathbb{E}[(X_i - \mu_i)^2]$, then

$$|\rho_{i,j}|^2 = \frac{|\mathbb{E}[(X_i - \mu_i)(X_j - \mu_j)]|^2}{\sigma_i^2 \sigma_j^2} \leq \frac{\mathbb{E}[(X_i - \mu_i)^2]\mathbb{E}[(X_j - \mu_j)^2]}{\sigma_i^2 \sigma_j^2} = 1.$$

From the above equation it follows that correlations take values in the interval $[-1, 1]$. It is clear from the definition that correlations are independent of order, $\rho_{i,j} = \rho_{j,i}$, and that the correlation between a random variable and itself is equal to one.

For convenience, we arrange the correlations into matrix form. This matrix is what we call a *correlation matrix* and is precisely defined in the next definition.

Definition 2 (Correlation Matrix)**.** *The correlation matrix of n random variables $X_1, \ldots, X_n$ is the $n \times n$ matrix whose $(i, j)^{th}$ entry is $\rho_{i,j}$.*

From the properties of correlation we derived previously, we see that a correlation matrix is symmetric ($\rho_{i,j} = \rho_{j,i}$), has unit diagonal ($\rho_{i,i} = 1$) and all of its entries are contained in the interval $[-1, 1]$.

Now consider again the portfolio containing the n assets $X_1, \ldots, X_n$. Let $a_i \in \mathbb{R}$ denote the amount of X_i in the portfolio. *Note that a_i can be negative due to for example short selling.* Then the variance of the portfolio is given by

$$0 < \mathrm{Var}\left(\sum_{i=1}^{n} a_i X_i\right) = \sum_{i,j=1}^{n} a_i a_j \sigma_i \sigma_j \rho_{i,j}. \tag{1.1}$$

We can write this down as a vector matrix product by letting $\mathbf{v}$ be the column vector whose i^{th} entry is $\sigma_i a_i$ and C the correlation matrix of $X_1, \ldots, X_n$. Then we can reformulate Equation (1.1) to

$$0 < \sum_{i,j=1}^{n} a_i a_j \sigma_i \sigma_j \rho_{i,j} = \mathbf{v}^T C \mathbf{v}.$$

A matrix which satisfies the above inequality for every vector $\mathbf{v}$ unequal to zero is called positive definite.

Definition 3 (Positive Definite Matrix)**.** *A matrix $A \in \mathbb{R}^{n \times n}$ is called positive definite if $\boldsymbol{v}^T A \boldsymbol{v} > 0$ for all vectors $\boldsymbol{v} \in \mathbb{R}^n \backslash \{0\}$.*

We distinguish correlation matrices which are *valid* and *invalid* in the following definition.

Definition 4 (Valid and Invalid Correlation Matrices)**.** *A **valid** correlation matrix is a symmetric positive definite matrix with unit diagonal and entries in [-1,1]. An **invalid** correlation matrix is a symmetric matrix with unit diagonal and entries in [-1,1], but which is non positive definite.*

1.4 Problem Statement

Now we have recalled the necessary background knowledge, we can formulate our research question.

Problem 1 (Nearness Problem)**.** Given an *invalid* correlation matrix C, we want to find a *valid* correlation matrix $\hat{C}$ such that the distance $||C - \hat{C}||$ is minimal in a certain matrix norm.

In this report we look for algorithms that solve Problem 1. This algorithm has to be fast and the resulting matrix $\hat{C}$ has to be *close* to C. The speed of an algorithm is measured by the computational time, i.e. the time it takes to find the matrix $\hat{C}$. The *accuracy* or *distance* of an algorithm is expressed by the distance between the resulting matrix $\hat{C}$ and C. We discuss in the next section which matrix norm is most appropriate for measuring this distance.

2 Measure Of Distance

As in Problem 1, let C be an *invalid* $n \times n$ correlation matrix, and $\hat{C}$ a *valid* correlation matrix. Then we want the distance

$$||C - \hat{C}||,$$

to be as small as possible in a certain matrix norm. Different norms can be used, as discussed in [9]. We discuss the most common norms and mention their advantages and disadvantages.

Consider p-norms, which are defined for a real number $1 \leq p \leq \infty$ and a vector $\mathbf{x} \in \mathbb{R}^n$ by

$$||A||_p = \max_{|\mathbf{x}|_p = 1} ||A\mathbf{x}||_p,$$

where $|\mathbf{x}|_p$ is the vector norm given by

$$|\mathbf{x}|_p = \left(\sum_{i=1}^{n} x_i^p \right)^{1/p}.$$

In this report, we consider the length of a vector in the $p = 2$ vector norm.

The task of computing a matrix p-norm is difficult for $p > 1$ since it is a non linear optimization problem constrained by $|\mathbf{x}|_p = 1$. So we look for norms which are easier to evaluate.

The matrix p-norm with $p = 1$ is given by

$$||A||_1 = \max_j \sum_{i=1}^{n} |a_{i,j}|.$$

The p-norm for $p = \infty$ is defined similar but per row, $||A||_\infty = \max_i \sum_{j=1}^{n} |a_{i,j}|$.

The p-norms for $p = 1$ and $p = \infty$ use a *maximum* function and are therefore non-continuous functions. Also, for these norms only absolute difference of the row that is most changed is measured, but not the difference of all the matrix entries.

The Frobenius norm, sometimes also called the Euclidean norm, of a matrix A with entries $a_{i,j}$ for $i, j = 1, 2, \ldots, n$, is defined by

$$||A||_F = \sqrt{\sum_{i,j=1}^{n} |a_{i,j}|^2}. \tag{2.1}$$

The Frobenius norm is numerically easy to compute and therefore of interest to be used as measure of distance in Problem 1. Also, the Frobenius norm measures the distance per matrix entry for all of the entries. This is exactly what we want to accomplish in Problem 1, since we want to make as few adjustments to the estimated correlations as possible.

In Matlab, the Frobenius norm from Equation (2.1) is given by the command `norm(A,'fro')`.

3 Checking for Positive Definiteness

Definition 3 states whether a matrix is positive definite or not. It is however cumbersome to implement this definition numerically, since it is impossible to verify the positive definite property for *all* vectors $\mathbf{v} \in \mathbb{R}^n$ (in finite time). Therefore, we look for equivalences of Definition 3.

Determining whether a symmetric matrix is positive definite can be done by looking at its eigenvalues. We prove that a real symmetric matrix is positive definite if and only if all of its eigenvalues are positive.

3.1 Eigenvalue Criteria

Let us recall the definition of eigenvalues.

Definition 5 (Eigenvalues)**.** *Let A be an $n \times n$ matrix. Then, every scalar λ_i which solves*

$$A\boldsymbol{s}_i = \lambda_i \boldsymbol{s}_i, \tag{3.1}$$

is called an eigenvalue of A corresponding to the eigenvector $\boldsymbol{s}_i \in \mathbb{R}^n \backslash \{0\}$. The eigenspace of A is given by de span (i.e. all linear combinations) of the eigenvectors $\boldsymbol{s}_i$.

Eigenvalues λ of matrix A can be found by solving the equation[1]

$$\det(A - \lambda I) = 0,$$

where I is the $n \times n$ identity matrix, containing ones on the diagonal and zeros everywhere else. The eigenvectors can be found by substituting the eigenvalues λ in Equation (3.1).

[1] This is derived from the fact that $A\mathbf{s} = \lambda\mathbf{s}$ can be written as $(A - I\lambda)\mathbf{s} = 0$ for $\mathbf{s} \neq 0$. So $(A - I\lambda)$ is singular, and has therefore a determinant equal to zero.

The next theorem is a general result from the linear algebra, and guarantees that the eigenvalues λ exist for (*invalid*) correlation matrices.

Theorem 1 (Spectral Theorem)**.** *Any real symmetric $n \times n$ matrix A can be written as*

$$A = SLS^T,$$

where L is a diagonal matrix containing the eigenvalues $\lambda_i \in \mathbb{R}$ for $i = 1, 2, \ldots, n$ of A and the columns of S are the eigenvectors of A, in the same order as the corresponding eigenvalues in such way that S is orthogonal[2]*. This is called the Spectral decomposition of A.*

Proof. This is a well known result in the linear algebra. The proof and more background reading on this topic can be found in [3, p. 354] or any linear algebra book. □

In Matlab, the spectral decomposition of a matrix A in the orthogonal matrix S and the matrix L containing the eigenvalues on the diagonal such that $A = SLS^T$ is given by `[S,L] = eig(A)`. We illustrate Theorem 1 with an example.

Example 3.1 (Spectral Decomposition)**.** Let

$$C = \begin{pmatrix} 1 & 0.9 & 0.7 \\ 0.9 & 1 & 0.3 \\ 0.7 & 0.3 & 1 \end{pmatrix}, \tag{3.2}$$

then the eigenvalues of C are $\lambda_1 = -0.007$, $\lambda_2 = 0.711$ and $\lambda_3 = 2.297$. The corresponding eigenvectors are

$$\mathbf{s}_1 = \begin{pmatrix} 0.748 \\ -0.563 \\ -0.352 \end{pmatrix}, \mathbf{s}_2 = \begin{pmatrix} 0.074 \\ 0.597 \\ -0.799 \end{pmatrix} \text{ and } \mathbf{s}_3 = \begin{pmatrix} 0.660 \\ 0.571 \\ 0.488 \end{pmatrix}.$$

Note that length of all eigenvectors is one and $\mathbf{s}_i^T \mathbf{s}_j = 0$ for $i \neq j$. We set the eigenvectors as columns of S and retrieve the decomposition $C = SLS^T$ given by

$$S = \begin{pmatrix} 0.748 & 0.074 & 0.660 \\ -0.563 & 0.597 & 0.571 \\ -0.352 & -0.799 & 0.488 \end{pmatrix} \text{ and } L = \begin{pmatrix} -0.007 & 0 & 0 \\ 0 & 0.711 & 0 \\ 0 & 0 & 2.297 \end{pmatrix}. \tag{3.3}$$

△

Let C be an (*invalid*) correlation matrix. Since it is symmetric, we can decompose it in its eigenvalues and eigenvectors

$$C = SLS^T,$$

using Spectral Theorem 1. Then L is a diagonal matrix with the eigenvalues λ_i for $i = 1, 2, \ldots, n$ on the diagonal and S the orthogonal matrix with the corresponding eigenvectors $\mathbf{s}_i$ as columns.

[2]Matrix A is orthogonal if $AA^T = I$. This is equal to saying for all columns $\mathbf{v}_i$ of A that $\mathbf{v}_i^T \mathbf{v}_i = 1$ and $\mathbf{v}_i^T \mathbf{v}_j = 0$ for $i \neq j$.

Since the span of all eigenvectors $\mathbf{s}_i$ is $\mathbb{R}^n$, i.e. we can write any vector $\mathbf{x} \in \mathbb{R}^n$ as linear combination as

$$\mathbf{x} = \sum_{i=1}^{n} c_i s_i,$$

with c_i a scalar. Then according to the definition of eigenvalues, Equation (3.1), we get

$$C\mathbf{x} = C\left(\sum_{i=1}^{n} c_i \mathbf{s}_i\right) = \sum_{i=1}^{n} c_i C\mathbf{s}_i = \sum_{i=1}^{n} c_i \lambda_i \mathbf{s}_i,$$

and if we multiply this on the left side with $\mathbf{x}^T$, we get

$$\mathbf{x}^T C\mathbf{x} = \mathbf{x}^T \sum_{i=1}^{n} c_i \lambda_i \mathbf{s}_i = \sum_{i=1}^{n} c_i^2 \lambda_i \mathbf{s}_i^T \mathbf{s}_i.$$

Where the last equality follows from the fact that S is orthogonal, which implies $\mathbf{s}_i^T \mathbf{s}_j = 0$ for $i \neq j$. Also, from the orthogonality of S follows that $\mathbf{s}_i^T \mathbf{s}_i = 1$, so

$$\mathbf{x}^T C\mathbf{x} = \sum_{i=1}^{n} c_i^2 \lambda_i. \tag{3.4}$$

Now note that c_i^2 is always positive since it it squared. Therefore the expression on the right hand side of Equation (3.4) is positive for any choice of c_i if and only if all the λ_i are positive. Let us summarize this result into a theorem.

Theorem 2 (Eigenvalue criteria)**.** *A symmetric matrix is positive definite if and only if all of its eigenvalues are positive.*

Proof. The proof follows from the above reasoning. More background reading on the relation between eigenvalues and positive definiteness can be done in for example [3, Chapter 8]. □

The calculation of eigenvalues for an $n \times n$ matrix requires that an n-dimensional polynomial is solved. This cannot be done exactly, so the eigenvalues are estimated. This results however into numerical errors. Moreover, if an *invalid* correlation matrix has t non positive eigenvalues, then the nearest correlation matrix will have at least t zero eigenvalues [10, Theorem 2.5]. If numerical errors such as round-off errors occur when calculating an eigenvalue *close* to zero, we could conclude incorrectly that a matrix is non positive definite.

3.2 Cholesky Decomposition

The reason we need the estimated correlation matrix to be positive definite, is that the Cholesky decomposition is applied to it. This Cholesky decomposition fails when the matrix is non positive definite, which we prove in Theorem 4. We first formulate another criterion for determining whether a symmetric matrix is positive definite. Then we define the Cholesky decomposition of a symmetric matrix and use it to determine if a matrix is positive definite.

Theorem 3. *Let A be an $n \times n$ symmetric matrix. Then A is positive definite if and only if there exists a matrix $B \in \mathbb{R}^{n \times n} \backslash \{0\}$ such that*

$$A = BB^T.$$

Proof. Let A be a symmetric positive definite matrix. Then, according to Theorem 1 it can be decomposed as

$$A = SLS^T, \tag{3.5}$$

where L is a diagonal matrix containing the eigenvalues λ_i for $i = 1, 2, \ldots, n$ of A, and S an orthogonal matrix with as columns the eigenvectors $\mathbf{s}_i$ of A. A is assumed to be positive definite, so it follows from Theorem 2 that all $\lambda_i > 0$. Therefore we can take the square roots of λ_i. Let $\sqrt{L}$ be the diagonal matrix with $\sqrt{\lambda_i}$ on the diagonal. Note that $L = \sqrt{L}\sqrt{L}$. Then we can write Equation (3.5) as

$$A = S\sqrt{L}\sqrt{L}S^T = S\sqrt{L}\left(S\sqrt{L}\right)^T.$$

Let $B = S\sqrt{L}$ and the result follows immediately.

To prove the sufficiency, let $B \in \mathbb{R}^{n\times n}$ and $\mathbf{v} \in \mathbb{R}^n \backslash \{0\}$. Let $A = BB^T$, then

$$\mathbf{v}^T A\mathbf{v} = \mathbf{v}^T BB^T\mathbf{v} = (B^T\mathbf{v})^T(B^T\mathbf{v}) = |B^T\mathbf{v}|^2 > 0. \tag{3.6}$$

□

Now, consider again an estimated correlation matrix C. If we can find a matrix B such that $C = BB^T$, it is positive definite by Theorem 3. In the proof above is shown that if C is positive definite, this matrix B can be found using Spectral Theorem 1. Since it is rather slow to compute all eigenvalues and eigenvectors of a matrix, we use the Cholesky decomposition to find such matrix.

Theorem 4 (Cholesky Decomposition)**.** *If A is a symmetric positive definite matrix, then A can be decomposed as*

$$A = MM^T,$$

where M is a unique lower triangular matrix with strictly positive diagonal entries. We call M the Cholesky factor of A.

We derive an algorithm to find the lower diagonal matrix M from Theorem 4, such that $A = MM^T$ for a positive definite $n \times n$ matrix A. Partition A and M as the block matrices

$$A = \left(\begin{array}{c|c} a_{1,1} & \mathbf{a}_1^T \\ \hline \mathbf{a}_1 & A_1 \end{array}\right) \text{ and } M = \left(\begin{array}{c|c} m_{1,1} & 0 \\ \hline \mathbf{m}_1 & M_1 \end{array}\right), \tag{3.7}$$

where $a_{1,1}$ is the upper left entry of A, $\mathbf{a}_1 \in \mathbb{R}^{(n-1)}$ and A_1 an $(n-1) \times (n-1)$ matrix, and analogue definitions for matrix M. Then, we substitute this partitioning into the multiplication $A = MM^T$ to get the equations

$$\begin{aligned} a_{1,1} &= m_{1,1}^2, \\ \mathbf{a}_1 &= m_{1,1}\mathbf{m}_1, \\ A_1 &= \mathbf{m}_1\mathbf{m}_1^T + M_1M_1^T. \end{aligned} \tag{3.8}$$

Where we define the multiplication $\mathbf{v}\mathbf{v}^T$of a vector $\mathbf{v} \in \mathbb{R}^{1\times n}$ with entries v_i as an $n \times n$ matrix with its $(i,j)^{th}$ entry given by v_iv_j. We want to derive expressions for the entries of M, so first we need to be sure that $a_{1,1} > 0$. This is proven in the next theorem.

Theorem 5. *The diagonal entries of a symmetric positive definite matrix A are positive.*

Proof. Given a symmetric positive definite $n \times n$ matrix A. According to Theorem 3, there exists a matrix $B \in \mathbb{R}^{n\times n}$ such that $A = BB^T$. Let $\tilde{\mathbf{b}}_i$ be the i^{th} row of B, then the diagonal entries $a_{i,i}$ of A are given by $\tilde{\mathbf{b}}_i\tilde{\mathbf{b}}_i^T = |\tilde{\mathbf{b}}_i|^2 > 0$. □

We can write $m_{1,1} = \sqrt{a_{1,1}}$, since $a_{1,1}$ is a diagonal entry of A. The second equation of (3.8) does not cause any trouble and gives

$$\mathbf{m}_1 = \mathbf{a}_1/\sqrt{a_{1,1}}.$$

The last equation of (3.8) can be formulated as

$$M_1 M_1^T = A_1 - \mathbf{m}_1\mathbf{m}_1^T.$$

To find M_1, we need to apply the Cholesky decomposition of $A_1 - \mathbf{m}_1\mathbf{m}_1^T$. This procedure thus results into an iterative algorithm for finding M. For the second step, set $A = A_1$ and repeat the above. This we can repeat until A_1 is a single entry. We then have found the Cholesky factor M. The remaining step is that we need to prove that if A is positive definite, than $A_1 - \mathbf{m}_1\mathbf{m}_1^T$ is also positive definite. This is formulated in the next theorem.

Theorem 6. *Let A be a symmetric positive definite $n \times n$ matrix and $\boldsymbol{m}_1 = \boldsymbol{a}_1/\sqrt{a_{1,1}}$, where $a_{1,1}$, $\boldsymbol{a}_1$ and A_1 are as in Equation (3.7). Then $A_1 - \boldsymbol{m}_1\boldsymbol{m}_1^T$ is symmetric positive definite.*

Proof. Since A is assumed to be symmetric so are A_1 and $A_1 - \mathbf{m}_1\mathbf{m}_1^T$. Now let $\mathbf{v}_1 \in \mathbb{R}^{n-1}\backslash\{0\}$. Define $\mathbf{v} = \begin{pmatrix} v_0 \\ \hline \mathbf{v}_1 \end{pmatrix}$ where $v_0 = -\mathbf{a}_1^T\mathbf{v}_1/a_{1,1}$. Then since $\mathbf{v} \neq 0$ we have

$$\begin{aligned}
0 < \mathbf{v}^T A\mathbf{v} &= \left(\begin{array}{c} v_0 \\ \hline \mathbf{v}_1 \end{array}\right)^T \left(\begin{array}{c|c} a_{1,1} & \mathbf{a}_1^T \\ \hline \mathbf{a}_1 & A_1 \end{array}\right) \left(\begin{array}{c} v_0 \\ \hline \mathbf{v}_1 \end{array}\right) \\
&= \left(\begin{array}{c} v_0 \\ \hline \mathbf{v}_1 \end{array}\right)^T \left(\begin{array}{c} a_{1,1}v_0 + \mathbf{a}_1^T\mathbf{v}_1 \\ \hline \mathbf{a}_1 v_0 + A_1\mathbf{v}_1 \end{array}\right) \\
&= a_{1,1}v_0^2 + v_0\mathbf{a}_1^T\mathbf{v}_1 + \mathbf{v}_1^T\mathbf{a}_1 v_0 + \mathbf{v}_1^T A_1\mathbf{v}_1 \\
&= a_{1,1}\tfrac{\mathbf{a}_1^T\mathbf{v}_1}{a_{1,1}}\tfrac{\mathbf{a}_1^T\mathbf{v}_1}{a_{1,1}} - \tfrac{\mathbf{a}_1^T\mathbf{v}_1}{a_{1,1}}\mathbf{a}_1^T\mathbf{v}_1 - \mathbf{v}_1^T\mathbf{a}_1\tfrac{\mathbf{a}_1^T\mathbf{v}_1}{a_{1,1}} + \mathbf{v}_1^T A_1\mathbf{v}_1 \\
&= \mathbf{v}_1^T\left(A_1 - \tfrac{\mathbf{a}_1\mathbf{a}_1^T}{a_{1,1}}\right)\mathbf{v}_1 \\
&= \mathbf{v}_1^T\left(A_1 - \mathbf{m}_1\mathbf{m}_1^T\right)\mathbf{v}_1.
\end{aligned}$$

We conclude that $A_1 - \mathbf{m}_1\mathbf{m}_1^T$ is symmetric positive definite. □

Now we can state the proof of the Cholesky decomposition.

Proof of Theorem 4. Let A be an $n \times n$ symmetric positive definite matrix. Then for $n = 1$, $A = [a_{1,1}]$ is a real positive number. The Cholesky matrix M is equal to $\sqrt{a_{1,1}}$.

Now assume that A is positive definite, then from Theorem 6, $A_1 - \mathbf{m}_1\mathbf{m}_1^T$ is positive definite, where A_1 is an $(n-1) \times (n-1)$ matrix. Now by induction it follows that Theorem 4 is true for all n.

□

Cholesky decomposition is used to solve the linear system $A\mathbf{x} = \mathbf{y}$ when A is a symmetric positive definite matrix. Substituting the Cholesky factors into the equation yields $MM^T\mathbf{x} = \mathbf{y}$. By Letting $\mathbf{z} = M^T\mathbf{x}$ we get

$$A\mathbf{x} = M\underbrace{(M^T\mathbf{x})}_{\mathbf{z}} = M\mathbf{z} = \mathbf{y}.$$

Thus, $\mathbf{z}$ can be found by solving the triangular system of equations $M\mathbf{z} = \mathbf{y}$ and subsequently the solution $\mathbf{x}$ can be found by solving the triangular system $M^T\mathbf{x} = \mathbf{z}$. More information regarding the Cholesky decomposition can be found in [13, 11].

Since the application of the Cholesky decomposition has to be applied in the process of analyzing time series , it saves time if we use it to determine if a matrix is positive definite. Then, if the matrix is positive definite, we have already computed the Cholesky factor M, and have not lost any computational time.

A simple implementation of the Cholesky Decomposition for checking whether a symmetric matrix A is positive definite is given below. The implementation of this algorithm in Matlab is given in Section 11.3. Matlab also has a build-in function for calculating the Cholesky factor of matrix A, `chol(A)`.

Algorithm 1 (Cholesky Decomposition)**.** Let A be a symmetric matrix. Let M be an $n \times n$ matrix and $\tilde{\mathbf{m}}_i$ be the i^{th} row of M. Then

1. For $i = 1, 2, \ldots, n$ repeat
   (a) For $j = 1, 2, \ldots, i-1$ repeat $m_{i,j} = a_{i,j} - \tilde{\mathbf{m}}_i\tilde{\mathbf{m}}_j^T$
   (b) If $a_{i,i} - \tilde{\mathbf{m}}_i\tilde{\mathbf{m}}_i^T$ is positive, then set $m_{i,i} = \sqrt{a_{i,i} - \tilde{\mathbf{m}}_i\tilde{\mathbf{m}}_i^T}$. Else, stop. A is non positive definite.

We compare the speed of checking for positive definiteness by eigenvalues and Cholesky decomposition. The results are shown in Table 3.1. Here, the build-in Matlab functions `eig(A)` and `chol(A)` are used. As discussed in [5], there are more efficient eigenvalue solvers around, but they are by no means more efficient than using the Cholesky decomposition to check whether a matrix is positive definite.

	Cholesky	Eigenvalues
$n = 10$	0.02	0.04
$n = 25$	0.02	0.09
$n = 50$	0.02	0.33
$n = 100$	0.06	1.29
$n = 250$	0.18	11.14
$n = 500$	1.88	48.04
$n = 1000$	7.69	403.02

Table 3.1: CPU Time (ms) for checking if a matrix of size n is positive definite in Matlab.

The main reason we need the estimated correlation matrix to be positive definite is so that we can apply the Cholesky decomposition to it. Therefore, if an estimated correlation matrix is positive definite according to the Cholesky decomposition, we are sure the Cholesky factor M exists, regardless of the impact of round-off errors. This is not the case if we check the matrix by eigenvalue decomposition since round-off errors may occur during this decomposition.

To check if a matrix C is a valid correlation matrix we use the function `test_cor(M)`, given in Section 11.1.

4 Correcting Non Positive Definite Matrices

In this section we provide a number of methods to adjust non-positive definite correlation matrices in such a way they become positive definite. We look in to the behavior of the methods in terms of distance from the original matrix and computational time.

4.1 Iterative Spectral Method

The Iterative Spectral method is based upon the spectral decomposition as in Theorem 1. It is a simplified version of the Alternating Projections method proposed by Higham in [10]. We first discuss the simplified version and after that we discuss Higham's Alternating Projections method. The algorithm for the Iterative Spectral method is given in Algorithm 2.

Algorithm 2 (Iterative Spectral Algorithm)**.** Let C be an (invalid) correlation matrix. Set $C_1 = C$ and $k = 1$. Then

1. Determine L_k and S_k such that $C_k = S_k L_k S_k^T$ as in Theorem 1.
2. If all eigenvalues of C_k are positive, then stop, C_k is positive definite. Else, continue:
3. Let L_k^+ be L_k with negative eigenvalues replaced by a *small* positive value a.
4. Set $C_{k+1} = S_k L_k^+ S_k^T$.
5. Set the diagonal entries of C_{k+1} to 1.
6. Set $k = k + 1$ and go to Step 1.

The implementation of the Iterative Spectral method in Matlab for an invalid correlation matrix C is given in Section 11.4. We explain what happens in Algorithm 2.

First, we apply Spectral decomposition to matrix C_k. This is always possible as was shown in Theorem 1. Therefrom, we determine whether C_k is positive definite by checking the diagonal entries of L_k. If all the diagonal entries are positive, we may stop; C_k is positive definite. Else, we replace all negative eigenvalues by a *small* positive value a and compose it back together as C_{k+1}. However, this increases the diagonal entries of C_{k+1}. We can see this by noting that the diagonal entries $c_{i,i}^k$ of C_k, are given by

$$c_{i,i}^k = \lambda_1 v_{i,1}^2 + \cdots + \lambda_n v_{i,n}^2. \tag{4.1}$$

So by replacing all negative eigenvalues λ_i with $a \geq \lambda_i$, we get $c_{i,i}^{k+1} \geq 1$. We correct the diagonal of C_{k+1} by setting it equal to one. Note that the sum of the diagonal entries of a matrix is equal to the sum of its eigenvalues [3, p.302]. We lower the diagonal entries and therefore also the eigenvalues. This results most likely into a non positive definite matrix C_{k+1}. As we show in Example 4.1, the eigenvalues are less negative. Therefore, we repeat this procedure until C_k is positive definite and has unit diagonal.

Convergence of this algorithm to a point where C_k is both positive definite and has unit diagonal is proven by Higham in [10, p. 337]. He proved that this algorithm converges always to a valid correlation matrix. However, this matrix may not be the nearest valid correlation matrix. Dykstra's correction can be applied to guarantee convergence to the nearest correlation matrix. This is discussed in Section 4.1.1.

We illustrate the Iterative Spectral method in the following example.

Example 4.1 (The Iterative Spectral Algorithm)**.** We continue with the result of Example 3.1, Equation (3.3). We note C_1 has a negative eigenvalue, which we replace by $a = 0.001$. So we get

$$S_1 = \begin{pmatrix} 0.748 & 0.074 & 0.660 \\ -0.563 & 0.597 & 0.571 \\ -0.352 & -0.799 & 0.488 \end{pmatrix} \text{ and } L_1^+ = \begin{pmatrix} 0.001 & 0 & 0 \\ 0 & 0.711 & 0 \\ 0 & 0 & 2.297 \end{pmatrix}, \tag{4.2}$$

and the new matrix C_2 is given by

$$C_2 = S_1 L_1^+ S_1^T = \begin{pmatrix} 1.005 & 0.897 & 0.698 \\ 0.897 & 1.003 & 0.302 \\ 0.698 & 0.302 & 1.001 \end{pmatrix}. \tag{4.3}$$

We see the diagonal entries are no longer equal to one, so we change them back to ones and recompute the eigenvalues and eigenvectors. The eigenvalues are shown in Table 4.1. We see the negative eigenvalue has increased, but is still negative.

	C_1	C_2	C_3	C_4
λ_1	2.297	2.294	2.293	2.292
λ_2	0.711	0.709	0.708	0.708
λ_3	-0.007	-0.002	-0.0005	0.0003

Table 4.1: Eigenvalues of the example matrix of Equation (3.2) after two iterations of the Iterative Spectral method

We repeat these steps until we obtain the positive definite correlation matrix

$$C_4 = \begin{pmatrix} 1.0000 & 0.8943 & 0.6965 \\ 0.8943 & 1.0000 & 0.3027 \\ 0.6965 & 0.3027 & 1.0000 \end{pmatrix}, \tag{4.4}$$

with eigenvalues as shown in Table 4.1. The distance in terms of the Frobenius norm between C and C_4 is 0.0102. △

The speed of convergence of the Iterative Spectral method depends on the matrix size n and the value of $a > 0$ to which we set the negative eigenvalues in Step 3 of Algorithm 2. The influence of those parameters on the speed of convergence is shown in Figure 4.1.

For this figure, we generated random invalid correlation matrices. A random invalid correlation matrix is generated by sampling its entries uniform from [-1,1]. Then we take into account that the matrix has to be symmetric and that its diagonal entries are equal to 1. In the end, we check whether it is non positive definite. The Matlab implementation of this random matrix generation is given in Section 11.2. Then in each step we have averaged the results of 10 simulations.

In the left sub figure of Figure 4.1, we see the number of iterations increases as a decreases for all matrix sizes n. Since we cannot alter n, we look at more detail to the impact of a on the distance of the resulting matrix. We want a to be as high as possible to reduce the number of iterations. The impact of a on the distance is shown in more detail in Figure 4.2. Picking $a < 10^{-3}$ does not affect the distance within three decimals. Therefore, we use $a = 10^{-3}$ from now on.

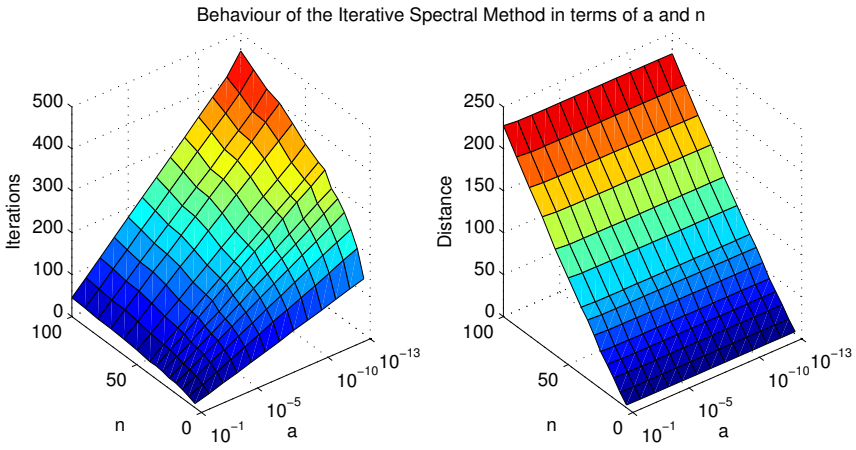

Figure 4.1: Iterations (left) and Distance (right) for the spectral method per matrix dimension n and parameter a.

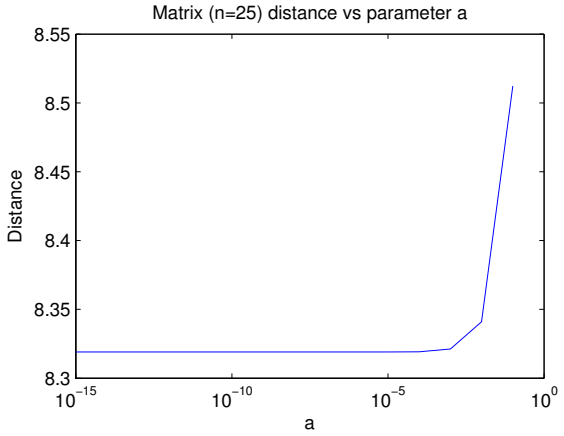

Figure 4.2: Distance versus different a in the Iterative Spectral method

4.1.1 Dykstra's Correction

In [10], Higham talks in detail about the convergence of the Iterative Spectral algorithm. He introduces two sets:

$$\begin{aligned} S &= \left\{A = A^T \in \mathbb{R}^{n\times n} : A \geq 0\right\}, \\ U &= \left\{A = A^T \in \mathbb{R}^{n\times n} : a_{i,i} = 1, i = 1, 2, \ldots, n\right\}. \end{aligned} \tag{4.5}$$

where S is the set of all symmetric positive definite matrices and U the set of symmetric matrices with unit diagonal. Then, we can call the projections $P_U(A)$ and $P_S(A)$ the projections of a matrix A on the sets U and S respectively, and see Algorithm 2 as iterating for $k = 1, 2, \ldots$ over the projections

$$A_{k+1} \leftarrow P_U(P_S(A_k)).$$

The idea of iterating over projections of subspaces was analyze by von Neumann, who proved convergence to the nearest intersection of both subspaces. See [6] for a survey on the von Neumann method. The sets S and U are no subspaces, so this von Neumann convergence does not apply. When the subspaces are replaced by closed convex sets, the iteration can converge to non optimal points [8]. Therefore Higham applies a modified iteration due to Dykstra [7], which introduces a correction to each projection which can be seen as a normal vector to the corresponding convex set. Applying the Dykstra's correction to the Iterative Spectral method results in Algorithm 3.

Algorithm 3 (Alternating Projections method)**.** Let C be an (invalid) correlation matrix. Set $C_1 = C$, $\Delta S_1 = 0$ and $k = 1$. Then

1. Dykstra's Corrections: Let $R_k = C_k - \Delta S_k$.
2. Determine L_k and S_k such that $R_k = S_k L_k S_k^T$.
3. P_S projection: Let L_k^+ be L with the negative eigenvalues replaced by a small positive value a.
4. Set $C_{k+1} = S_k L_k^+ S_k^T$.
5. Set $\Delta S_{k+1} = C_{k+1} - R_k$.
6. P_U projection: Set all diagonal elements of C_{k+1} to 1.
7. Check of C_{k+1} is positive definite. Then stop. Else set $k = k + 1$ and go to Step 1.

The implementation of Algorithm 3 is given in Section 11.5.

In the Iterative Spectral method, we apply a spectral decomposition to C_k each iteration. We use this to determine whether C_k is positive definite, so we can stop the algorithm.

The downside of the Alternating Projections method is that we do not decompose C_k spectrally, but R_k. So at the end of each iteration, an additional check has to be done to determine whether C_{k+1} is positive definite. Therefore, an iteration of the Alternating Projections method is more time consuming than an iteration of the Iterative Spectral method.

We stop Algorithm 3 as soon as matrix C_{k+1} is positive definite. We check this using the Cholesky decomposition of Section 3.2. The Iterative Spectral method (IS) and the Alternating Projections method (AP) are very similar, so we compare them in the next example.

Example 4.2 (Alternating Projections Method)**.** We continue to use the same invalid correlation matrix C as defined in Example 3.1. Applying the above described algorithm, using $a = 0.001$, results in

$$C_4 = \begin{pmatrix} 1.0000 & 0.8943 & 0.6965 \\ 0.8943 & 1.0000 & 0.3027 \\ 0.6965 & 0.3027 & 1.0000 \end{pmatrix}.$$

For different values of a, the results are shown in Table 4.2. We see that the behavior of the Alternating Projections method (AP) and Iterative Spectral method (IS) is the same for this small example matrix.

Since the results do not differ for this 3×3 matrix, we also compare different sizes of matrices. The results are shown in Table 4.3. It is clear Dykstra's corrections slows down the convergence. The Alternating Projections method is slower than the Iterative Spectral method, and the distance is almost the same.

△

a	AP iterations	AP distance	IS iterations	IS distance
$a = 10^{-1}$	1	0.0812	1	0.0812
$a = 10^{-2}$	1	0.0131	1	0.0131
$a = 10^{-3}$	3	0.0102	3	0.0102
$a = 10^{-4}$	6	0.0098	6	0.0098

Table 4.2: Iterations and distance for matrix C of Example 3.1 using the Alternating Projections method (AP) and the Iterative Spectral method (IS).

	IS	AP
$n = 25$	8.80 / 7.1	8.79 / 7.1
$n = 50$	20.36 / 24.0	20.35 / 39.7
$n = 100$	44.87 / 196.5	44.85 / 321.9

Table 4.3: *Distance / CPU time (ms)* Results the Alternating Projections method (AP) and the Iterative Spectral method (IS).

From our point of view, we note that Dykstra's correction is not worth adding to the Iterative Spectral method, since it slows down the convergence and the profit in accuracy is negligible.

4.2 Scaled Spectral Method

In the Iterative Spectral method from Section 4.1 we replace the negative eigenvalues of the invalid correlation matrix. This changes the diagonal entries of the resulting matrix. By changing the diagonal values manually back to 1, we destroy the positive definiteness of the matrix. In the Scaled Spectral method, we scale the resulting matrix in a such a way that the diagonal becomes unit. How to create such a scaling matrix and how to make sure the resulting matrix is symmetric and positive definite was presented in [16].

Consider an invalid correlation matrix C. We apply the Spectral decomposition from Theorem 1 to find the matrices S and L such that $C = SLS^T$. L is the matrix with the eigenvalues of C on the diagonal. Since we assumed C to be non positive definite, L has at least one negative diagonal entry. We replace these negative diagonal entries in L with a *small* positive value a and call it L^+. Then, we calculate $C_2 = SL^+S^T$. Now C_2 is positive definite, but has no longer unit diagonal.

The diagonal entries $c_{i,i}$ of C_2 are given by $c_{i,i} = \sum_{j=1}^{n} s_{i,j}^2 \lambda_j^+$ where $s_{i,j}$ is the $(i,j)^{th}$ entry of S and λ_j^+ the $(j,j)^{th}$ entry of L^+.

To scale the diagonal entries of C_2 back to 1, we have to multiply each diagonal entry with its inverse. Also we have to make sure C_2 stays positive definite. Define scaling matrix T as a diagonal matrix with diagonal entries $t_{i,i} = [c_{i,i}]^{-1}$, where $c_{i,i}$ is the i^{th} diagonal entry of C_2.

Now, scale C_2 as $\sqrt{T}C\sqrt{T}$. *Note that the square root of a diagonal matrix is equal to taking the square root of its diagonal entries.* Then we see

$$\sqrt{T}C_2\sqrt{T} = (\sqrt{T}S\sqrt{L^+})(\sqrt{L^+}S^T\sqrt{T}) = (\sqrt{T}S\sqrt{L^+})(\sqrt{T}S\sqrt{L^+})^T,$$

and therefore $\sqrt{T}C_2\sqrt{T}$ is positive definite according to Theorem 3. Also, the diagonal entries

of the scaled matrix are given by are given by

$$\sqrt{t_{i,i}}c_{i,i}\sqrt{t_{i,i}} = \left[\sqrt{\sum_{j=1}^{n} s_{i,j}^2\lambda_j^+}\right]^{-1}\left(\sum_{j=1}^{n} s_{i,j}^2\lambda_j^+\right)\left[\sqrt{\sum_{j=1}^{n} s_{i,j}^2\lambda_j^+}\right]^{-1} = 1.$$

Now, $\sqrt{T}C_2\sqrt{T}$ is a valid correlation matrix. We illustrate these steps in the following example.

Example 4.3 (Scaled Spectral decomposition)**.** We use the same matrix from Example 3.1. Let

$$C = \begin{pmatrix} 1 & 0.9 & 0.7 \\ 0.9 & 1 & 0.3 \\ 0.7 & 0.3 & 1 \end{pmatrix}, \tag{4.6}$$

then, since C is symmetric, we can apply Spectral decomposition as in Theorem 1. This results in diagonal eigenvalue matrix L and eigenvector matrix S given by

$$S = \begin{pmatrix} 0.748 & 0.074 & 0.660 \\ -0.563 & 0.597 & 0.571 \\ -0.352 & -0.799 & 0.488 \end{pmatrix} \text{ and } L = \begin{pmatrix} -0.007 & 0 & 0 \\ 0 & 0.711 & 0 \\ 0 & 0 & 2.297 \end{pmatrix}. \tag{4.7}$$

Since the first eigenvalue is negative, we replace it by $a = 10^{-13}$. Now, we can compute C_2,

$$C_2 = SL^+S^T = \begin{pmatrix} 1.004 & 0.897 & 0.698 \\ 0.897 & 1.002 & 0.302 \\ 0.698 & 0.302 & 1.001 \end{pmatrix},$$

and note the diagonal elements indeed increased. To fix this, we create the scaling matrix T, given by

$$T = \begin{pmatrix} \frac{1}{1.004} & 0 & 0 \\ 0 & \frac{1}{1.002} & 0 \\ 0 & 0 & \frac{1}{1.001} \end{pmatrix}.$$

This results in

$$\sqrt{T}C_2\sqrt{T} = \begin{pmatrix} 1.000 & 0.894 & 0.696 \\ 0.894 & 1.000 & 0.301 \\ 0.696 & 0.301 & 1.000 \end{pmatrix}, \tag{4.8}$$

which is positive definite with eigenvalues $\lambda_1 = 1 \cdot 10^{-13}$, $\lambda_2 = 2.2904$ and $\lambda_3 = 0.7096$. The distance in terms of the Frobenius norm with the original matrix is 0.0100, which is slightly better than the distance of the Iterative Spectral method (0.0102). △

Algorithm 4 (Scaled Spectral method)**.** Let C be an $n \times n$ *invalid* correlation matrix.

1. Determine S and L such that $C = SLS^T$ as in Theorem 1.
2. Let L^+ be L with negative eigenvalues replaced by a *small* positive value a.

3. Denote the entries of S by $s_{i,j}$ and the entries of L^+ by λ_j^+. Then calculate the scaling matrix T with entries $t_{i,j}$ given by

$$T = \begin{cases} t_{i,j} = \left[\sum_j s_{i,j}^2 \lambda_j^+\right]^{-1}, & i = j \\ t_{i,j} = 0. & i \neq j \end{cases} \tag{4.9}$$

4. Set $C_2 = \sqrt{T} S L^+ S^T \sqrt{T}$.

The Matlab implementation of this algorithm can be found in Section 11.6.

The Scaled Spectral method is non iterative. Therefore it applies only one eigenvalue decomposition. This makes the Scaled Spectral method faster than the Iterative Spectral method, since this method has to perform several eigenvalue decompositions. Also, the choice of parameter a has no effect on the number of iterations, but does affect the accuracy. This is shown in Figure 4.3.

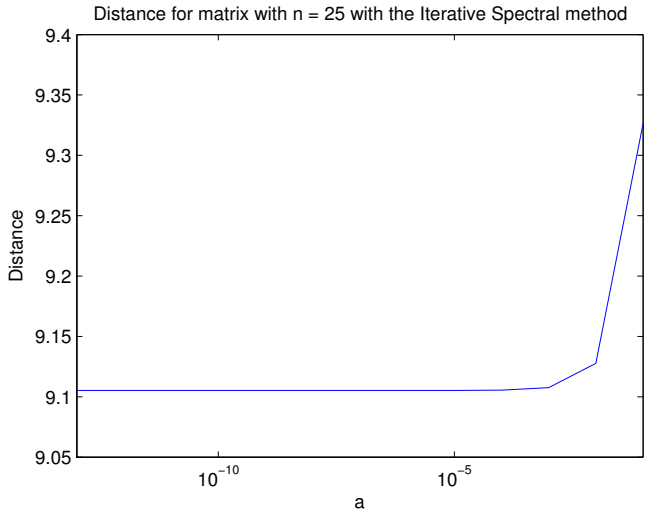

Figure 4.3: The influence of the parameter a for the Iterative Spectral method.

This implies that a should be chosen as small as possible. However, one needs to account for round-off errors. If the eigenvalues of the resulting matrix are too close to zero, the matrix could become non positive definite due to round-off errors.

Numerical experiments show that for a machine precision of 2.2^{-16}, it is safe to pick $a = 10^{-13}$.

4.3 Gradient Updating Method

Y. Zhang introduces another approach to find the solution of Problem 1 in [20]. In [20], finding the nearest correlation matrix is formulated as a minimization problem with the objective function given by

$$f(A,B) = \frac{1}{2}||BA - \hat{C}||_F^2.$$

If we set $A = B^T$ then by Theorem 3, BA is positive definite. Also, we want the diagonal of BA to be unit.

Denote the j^{th} column of A by $\mathbf{a}_j$, $j = 1, 2, \ldots, n$, and $a_{i,j}$ as the i^{th} element of the j^{th} column of A, also let $\tilde{\mathbf{a}}_i$ be the i^{th} row of A. Then the objective function can be written as

$$\begin{aligned} f(A,B) &= \tfrac{1}{2}||C - BA||_F^2 \\ &= \tfrac{1}{2}\sum_{j=1}^{n} ||\mathbf{c}_j - B\mathbf{a}_j||^2 \\ &= \tfrac{1}{2}\sum_{j=1}^{m} (\mathbf{c}_j - B\mathbf{a}_j)^T(\mathbf{c}_j - B\mathbf{a}_j) \\ &= \tfrac{1}{2}\sum_{i,j=1}^{n} (c_{i,j} - \tilde{\mathbf{b}}_i\mathbf{a}_j)^2. \end{aligned}$$

By differentiating the above equation to $a_{u,v}$ with $u, v \in \{1, 2, \ldots, n\}$ fixed, we get

$$\begin{aligned} \tfrac{\partial}{\partial a_{u,v}} f(A,B) &= \tfrac{1}{2}\sum_{i,j=1}^{n} 2\left(c_{i,j} - \tilde{\mathbf{b}}_i\mathbf{a}_j\right)\tfrac{\partial}{\partial a_{u,v}}\left(-\tilde{\mathbf{b}}_i\mathbf{a}_j\right) \\ &= \sum_{i,j=1}^{n}\left(\tilde{\mathbf{b}}_i\mathbf{a}_j - c_{i,j}\right)\tfrac{\partial}{\partial a_{u,v}}\left(\tilde{\mathbf{b}}_i\mathbf{a}_j\right), \end{aligned}$$

by the chain rule. Since

$$\frac{\partial}{\partial a_{u,v}}\left(\tilde{\mathbf{b}}_i\mathbf{a}_j\right) = \begin{cases} 0, & \text{if } j \neq v, \\ b_{i,u}, & \text{if } j = v, \end{cases}$$

the derivative of f with respect to $a_{u,v}$ becomes

$$\frac{\partial}{\partial a_{u,v}} f(A,B) = \sum_{i=1}^{n}\left(\tilde{\mathbf{b}}_i\mathbf{a}_v - c_{i,v}\right) b_{i,u} = \mathbf{b}_u^T\left(B\mathbf{a}_v - \mathbf{c}_v\right).$$

Now we have an expression for the derivative of f to a fixed $a_{u,v}$, we can rewrite it in the matrix form

$$\frac{\partial f(A,B)}{\partial A} = B^T\left(BA - C\right).$$

For the gradient of $f(A,B)$ with respect to B, the computations are analog, which result in

$$\frac{\partial f(A,B)}{\partial B} = (BA - C)A^T.$$

From the multivariate analysis, we know a multivariate function decreases fastest in the direction of the negative gradient. This topic is extensively discussed in [1, §12.7]. We use an updating scheme for the matrices A and B with the above calculated gradients. Therefore, let ϕ_k and ψ_k be positive scalars, denoting the step length for respectivily A and B. Then, the gradient updating scheme is given by

$$A_{k+1} = A_k - \phi_k \frac{\partial f(A_k, B_k)}{\partial A}, \tag{4.10}$$

$$B_{k+1} = B_k - \psi_k \frac{\partial f(A_k, B_k)}{\partial B},$$

We use the updating schemes above to iterate to the nearest valid correlation matrix. To ensure positive definiteness of this resulting matrix we need A_k to be equal to B_k^T. In that way we have $B_k A_k = B_k B_k^T$, which is positive definite by Theorem 3. So we average each iteration

$$B_{k+1} \equiv (B_{k+1} + A_{k+1}^T)/2,$$

and set

$$A_{k+1} \equiv B_{k+1}^T.$$

To guarantee unit diagonal, we create a scaling matrix T in the same way as in Equation (4.9). Therefore, we define the operation diag($\mathbf{v}$).

Definition 6 (The Diag Operator)**.** *The operation diag($\boldsymbol{v}$) for a vector $\boldsymbol{v} \in \mathbb{R}^n$ maps that vector to the diagonal of an $n \times n$ matrix, with all off-diagonal entries equal to zero. The operation diag(A) for a matrix A is the vector of diagonal entries of A. This definition is the same as the Matlab function* `diag()`*.*

Using Definition 6 we define $T = \text{diag}(\text{diag}(B_{k+1}A_{k+1})^{-1})$. and scale A_{k+1} and B_{k+1} as

$$B_{k+1} = \sqrt{T} B_{k+1} \text{ and } A_{k+1} = A_{k+1}\sqrt{T}.$$

Next, we discuss how to choose the steplength parameters ϕ_k and ψ_k. Therefore we use the *Armijo rule*, which guarantees that we reduce the objective function each iteration of the gradient updating scheme.

Definition 7 (Armijo Rule)**.** *Given the gradient updating scheme*

$$x_{k+1} = x_k - a_k \frac{\partial f(x_k)}{\partial x},$$

we define the Armijo inequality as

$$f(x_{k+1}) - f(x_k) \leq \sigma \left(\frac{\partial f(x_k)}{\partial x} \right)^T (x_{k+1} - x_k),$$

with $\sigma \in (0, 1)$. If the inequality is satisfied, the step length a_k is 'sufficient', which means the objective function decreased enough.[2, 19]

In the definition of the Armijo rule, the updated x_k is defined as a vector. In this case, we apply gradient updating to matrices. Therefore, we need to define the dot product analogue for matrices. We illustrate this in a very simple example.

Example 4.4. Given two real-valued vectors

$$\mathbf{v} = \begin{pmatrix} a & b & c & d \end{pmatrix}^T \text{ and } \mathbf{w} = \begin{pmatrix} e & f & g & h \end{pmatrix}^T,$$

their dot product is given by

$$\langle \mathbf{v}, \mathbf{w} \rangle = \mathbf{v}^T \mathbf{w} = ae + bf + cg + dh.$$

Now in matrix format,

$$V = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \text{ and } W = \begin{pmatrix} e & f \\ g & h \end{pmatrix},$$

we define $\langle V, W \rangle = \text{trace}(V^T W)$, where the trace of a matrix is the sum of its diagonal elements. Thus,

$$\langle V, W \rangle = \text{trace}(V^T W) = ae + cg + bf + dh.$$

△

We use the Armijo rule for the updating scheme of Equation (4.10). With this rule we determine an upper bound for ϕ_k. From Definition 7 follows

$$f(A_{k+1}, B_k) - f(A_k, B_k) \leq \sigma \left\langle \frac{\partial f(A_k, B_k)}{\partial A}, (A_{k_1} - A_k) \right\rangle,$$

To predict $f(A_{k+1}, B_k)$ we use a Taylor expansion. Therefore, we calculate the second order gradient of f with respect to A.

$$\begin{aligned} \frac{\partial^2 f(A,B)}{\partial a_{u,v}^2} &= \frac{\partial}{\partial a_{u,v}} \left[\sum_{i=1}^n b_{i,u} \tilde{b}_i a_v \right] \\ &= \sum_{i=1}^n b_{i,u} \frac{\partial}{\partial a_{u,v}} \left[\tilde{b}_i a_v \right] \\ &= \sum_{i=1}^n b_{i,u} b_{i,u}, \end{aligned}$$

which, in matrix form is written as

$$\frac{\partial^2 f(A, B)}{\partial A^2} = B^T B.$$

We thus get the Taylor expansion

$$\begin{aligned} f(A_{k+1}, B) &\approx f(A_k, B_k) + \left\langle A_{k+1} - A_k, \tfrac{\partial}{\partial A} f(A_k, B_k) \right\rangle \\ &\quad + \tfrac{1}{2} \left\langle A_{k+1} - A_k, \tfrac{\partial^2}{\partial A^2} f(A_k, B_k)(A_{k+1} - A_k) \right\rangle. \end{aligned} \tag{4.11}$$

This expansions is exact since the second order gradient is a constant function and therefore the next term vanishes. Combining these results and substituting them in the Armijo rule gives

$$0 \geq 2(1-\sigma) \left\langle \frac{\partial f}{\partial A}(A_k, B_k), A_{k+1} - A_k \right\rangle + \left\langle A_{k+1} - A_k, (B_k^T B_k)(A_{k_1} - A_k) \right\rangle.$$

Using the gradient updating scheme of Equation (4.10), we can rewrite $A_{k+1} - A_k$ and get

$$\begin{aligned} 0 &\geq -\phi_k 2(1-\sigma) \left\langle \tfrac{\partial f}{\partial A}(A_k, B_k), \tfrac{\partial f}{\partial A}(A_k, B_k) \right\rangle \\ &\quad + \phi_k^2 \left\langle \tfrac{\partial f}{\partial A}(A_k, B_k), (B_k^T B_k) \tfrac{\partial f}{\partial A}(A_k, B_k) \right\rangle, \end{aligned}$$

Since we have chosen the step length ϕ_k to be positive, we can divide by it without changing the direction of the unequal sign. Also note that $\text{trace}(A^T A) = ||A||_F^2$. By dividing the above inequality by $-\phi_k$ we get

$$0 \leq 2(1-\sigma) \left|\left| \frac{\partial f(A_k, B_k)}{\partial A} \right|\right|_F^2 - \phi_k \left|\left| B_k \frac{\partial f(A_k, B_k)}{\partial A} \right|\right|_F^2.$$

We can repeat the above computations analogues for ψ_k, which results in the follow bounds on the step length.

$$\phi_k \leq \frac{2(1-\sigma)\left|\left|\frac{\partial f(A_k,B_k)}{\partial A}\right|\right|_F^2}{\left|\left|B\frac{\partial f(A_k,B_k)}{\partial A}|\right|\right|_F^2} \text{ and } \psi_k \leq \frac{2(1-\sigma)\left|\left|\frac{\partial f(A_k,B_k)}{\partial B}\right|\right|_F^2}{\left|\left|\frac{\partial f(A_k,B_k)}{\partial B}A\right|\right|_F^2}. \tag{4.12}$$

We want the step lengths ϕ_k and ψ_k to be as large as possible, within the above derived bounds, such that the distance decreases most each iteration. So we take ϕ_k and ψ_k be equal to their upper bounds. Combining the above results we obtain the following algorithm.

Algorithm 5. Choose starting matrices A_1 and B_1 such that A_1B_1 is close to C. Also choose parameter σ and set $k = 1$. Then repeat the following steps until you are satisfied.

1. Compute the gradient $\text{grad}(A_k) = B_k^T(B_kA_k - C)$,
2. Choose step length $\phi_k = \frac{2(1-\sigma)||\text{grad}(A_k)||_F^2}{||B_k\text{grad}(A_k)||_F^2}$
3. Set $A_{k+1} = A_k - \phi_k\text{grad}(A_k)$.
4. Compute the gradient $\text{grad}(B_k) = (B_kA_{k+1} - C)A_{k+1}^T$,
5. Choose step length $\psi_k = \frac{2(1-\sigma)||\text{grad}(B_k)||_F^2}{||\text{grad}(B_k)A_k||_F^2}$
6. Set $B_{k+1} = B_k - \psi_k\text{grad}(B_k)$.
7. Set $B_{k+1} = (B_{k+1} + A_{k+1}^T)/2$,
8. Compute $T = \text{diag}(\text{diag}(B_{k+1}^TB_{k+1})^{-1})$,
9. Scale $B_{k+1} = \sqrt{T}B_{k+1}$ and $A = B_{k+1}^T\sqrt{T}$
10. Set $k = k + 1$ and go to Step 1. Stop when satisfied.

To use the above algorithm, we need to determine a stopping criteria and certain input matrices A_1 and B_1. For this, we compare different input possibilities. We want the initial matrix B_1A_1 to be as close as possible to the target matrix C, i.e.

$$||B_1A_1 - C||_F^2,$$

has to be low. Since B_1A_1 is positive definite by Theorem 3, we look for forms of positive definite matrices close to the target matrix.

4.3.1 Input Matrix: Cholesky Form

We look for a positive definite matrix $\tilde{C}$ close to an *invalid* $n \times n$ correlation matrix C. Let $\bar{c}$ be the average of all off-diagonal entries of C, which is

$$\bar{c} = \frac{1}{n^2 - n}\left(\sum_{i,j} c_{i,j} - n\right).$$

Intuitively, the matrix

$$\tilde{C} = \begin{cases} \tilde{c}_{i,j} = \bar{c}, & \text{if } i \neq j \\ \tilde{c}_{i,j} = 1, & \text{if } i = j \end{cases}$$

would be close to C. However, we need to write it in the form $\tilde{C} = B_1A_1$. Therefore we apply the Cholesky decomposition to it. In order to do so, $\tilde{C}$ has to be positive definite. We show for which value of the off-diagonal entries the matrix $\tilde{C}$ is positive definite.

Theorem 7. *Let C be an $n \times n$ matrix with unit diagonal and off diagonal entries equal to $\bar{c} \in \mathbb{R}$. Then C is positive definite if $\bar{c} \in \left(\frac{-1}{n-1}, 1\right)$.*

Proof. Let the entries of C be given by $c_{i,j} = 1$ for $i = j$ and $c_{i,j} = \bar{c}$ for $i, j \in \{1, 2, \ldots, n\}$. Then the eigenvalues of C are given by

$$\lambda_{1,\ldots,n-1} = 1 - \bar{c} \text{ and } \lambda_n = (n-1)\bar{c} + 1,$$

and therefore, C is positive definite if $\bar{c} \in \left(\frac{-1}{n-1}, 1\right)$. □

We see from the above theorem that $\tilde{C}$ is positive definite if $\bar{c} \in \left(\frac{-1}{n-1}, 1\right)$, which is a prerequisite to apply Cholesky decomposition. So, let $\tilde{C_1}$ be given by

$$\tilde{C}_1 = \begin{cases} \tilde{c}_{i,j} = \max\left(\frac{-1}{n-1}, \bar{c}\right) & \text{if } i \neq j \\ \tilde{c}_{i,j} = 1 & \text{if } i = j \end{cases}.$$

Now apply the Cholesky decomposition to $\tilde{C}$ to get the initial matrices

$$B_1 = \text{chol}(\tilde{C}) \text{ and } A_1 = B_1^T. \tag{4.13}$$

4.3.2 Input Matrix: Scaled Form

The second form of input matrices we consider is based on the scaling in Step 8 and 9 of Algorithm 5. Since we do not account for the scaling in the gradient updating, it might be an advantage to scale the input matrices A_1 and B_1 in advance.

We set $B_1 = C$ with diagonal entries replaced by α and $A_1 = B_1^T$. Let $\tilde{c}_{i,j}$ be the $(i,j)^{th}$ entry of $\tilde{C} = B_1 A_1$ and $\tilde{\mathbf{b}}_i$ the i^{th} row of B_1. Then the diagonal entries $\tilde{c}_{i,i} = \tilde{\mathbf{b}}_i^T \tilde{\mathbf{b}}_i$ of $\tilde{C}$ will be unequal to one in general.

We create the scaling matrix T with diagonal entries $t_{i,i} = \left(\tilde{\mathbf{b}}_i^T \tilde{\mathbf{b}}_i\right)^{-1}$. Therefore the initial guess $\tilde{C} = \sqrt{T} B_1 A_1 \sqrt{T}$ will have unit diagonal. However, we want $\tilde{C}$ to be close to C, so each entry $\tilde{c}_{i,j} = \tilde{\mathbf{b}}_i^T \tilde{\mathbf{b}}_j$ of $\tilde{C}$, has to be close to $c_{i,j}$. We determine which α to choose such that $\tilde{C}$ is close to C. Assume all entries of C are constant and with value c. We then solve

$$\tilde{c}_{i,j} \approx c = \frac{2\alpha c + (n-2)c^2}{\alpha^2 + (n-1)c^2} \approx \tilde{\mathbf{b}}_i^T \tilde{\mathbf{b}}_j = \tilde{c}_{i,j},$$

for α. We get as solution

$$\alpha = 1 + \sqrt{1 - (c^2(n-1) + c(2-n))}. \tag{4.14}$$

We could set c equal to the average of all off-diagonal entries of C. However, taking the average of C costs about 0.35 milliseconds for a matrix of $n = 100$. This is about 20% of the time the algorithm needs to perform one iteration, as we see in Section 5. Also, after one iteration the choice of c is negligible. We illustrate this in Table 4.4. *We have to take in account the bounds $\frac{-1}{n-1} < c < 1$ to make sure the square root in Equation (4.14) is defined properly.*

Therefore, we assume $c = 0.5$, which gives $\alpha = 1 + \frac{\sqrt{n+1}}{2}$. Numerical results show that taking values of α in the neighborhood of $1 + \frac{\sqrt{n+1}}{2}$ gives more accurate results. We therefore pick $\alpha = \frac{\sqrt{n}}{2}$.

	Initital distance	Distance after 1 iteration
$c = 0$	52.41	47.13
$c = 0.25$	48.18	46.11
$c = 0.5$	48.11	46.14
$c = 0.75$	48.20	46.11

Table 4.4: Comparing different c to determine α for an 100×100 matrix with an average off-diagonal entry value of 0 for the Gradient Updating method.

Summarizing the above derivation gives us the initial matrices

$$B_1 = \begin{cases} b_{i,j} = c_{i,j}, & \text{for } i \neq j \\ b_{i,j} = \frac{\sqrt{n}}{2}, & \text{for } i = j. \end{cases}, \text{ and } A_1 = B_1^T.$$

Since we have made a lot of assumptions, these initial matrices are not very accurate. However they converge very fast in the first iteration as we saw in Table 4.4. This is most likely due to the fact the initial matrix has unit diagonal and is scaled already. We show this after we have determined a stopping criterion.

4.3.3 Input Matrix: Scaled Spectral Form

We could first apply the scaled spectral method to the invalid correlation matrix C. The resulting matrix $\hat{C}$ is close to C in terms of the Frobenius norm. Since $\hat{C}$ is positive definite, we can decompose it in B_1 and A_1 such that $\hat{C} = B_1 A_1$, where $A_1 = B_1^T$.

However, the scaled spectral method uses an eigenvalue decomposition which is rather time consuming. Using the same 100×100 matrix as we have used to generate the results in Table 4.4, we see the distance of the resulting matrix from the Scaled Spectral method to the target matrix C is 47.04. This distance is about the same as the results in Table 4.4. However, the spectral method takes about 7.9 milliseconds to calculate this input matrix, while the calculations in Table 4.4 for 1 iteration took 4.5 milliseconds.

We conclude that the other forms of input matrices are faster, and the resulting distance is similar. So we do not use the Scaled Spectral method as input for the Gradient Updating method.

4.3.4 Stopping Criteria

Let $r_k = ||A_k B_k - C||_F$ be the distance at iteration k, then we define the stopping criterion as

$$\frac{|r_k - r_{k-1}|}{r_k} \leq 10^{-2}$$

Of course, this value can be adjusted to gain or lose accuracy, at the expense of duration.

Warning: Lowering this tolerance too much can result in numerical errors due to, for example, the machine precision. How small this tolerance value may be depends highly upon the matrix dimension.

4.3.5 Numerical Calibration

We have derived two different input matrices and a stopping criterion for the Gradient Updating method. The only variable left to determine is σ.

We calibrate the Gradient Updating method (GU) for both the Cholesky based input matrix from Section 4.3.1 (Cholesky) and the scaled input matrix from Section 4.3.2 (Scaled). For different values of σ we show the behavior for the Gradient Updating method in Figure 4.4. This figure is generated by generating 50 invalid correlation matrices with $n = 25$ and entries uniformly distributed over $(-1, 1)$, see the Matlab file from Section 11.2. The results are shown in Figure 4.4.

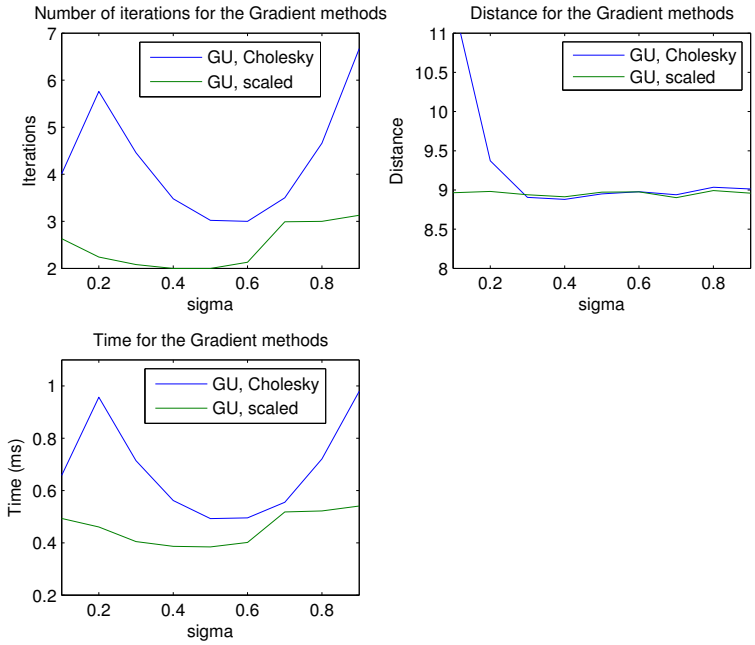

Figure 4.4: Results of the Gradient method with the two starting matrices for different σ.

As we see in Figure 4.4 sub figure 2, the distance of both initial matrices is almost constant for $\sigma \geq 0.3$. For small σ, the performance of the Cholesky decomposition is unsatisfactory. From Figure 4.4 sub figure 1, it follows that we should pick a $\sigma \in [0.5, 0.6]$. This results in the fasted convergence.

However, by choosing the upper bounds for the step length ϕ_k and ψ_k, we do not take in account that the matrices A_k and B_k will be averaged and scaled every iteration. Therefore, we cannot guarantee that the bounds will hold. This becomes visible when we sample an invalid correlation matrix C with off diagonal entries $c_{i,j} \in (-1, 0)$. This is shown in Figure 4.5.

We see clearly in Figure 4.5 that the Gradient Updating method is not converging. Since the convergence fails, we check whether this is caused by a wrong step length.

In Figure 4.6, the results of the Gradient Updating methods are compared for different values of σ. As we see for matrices with negative entries, σ should be picked $\sigma > 0.6$

To force convergence of the algorithm, we check each iteration if the distance of the calculated matrix to the original matrix C has decreased. If it did not, divide the step length by two and

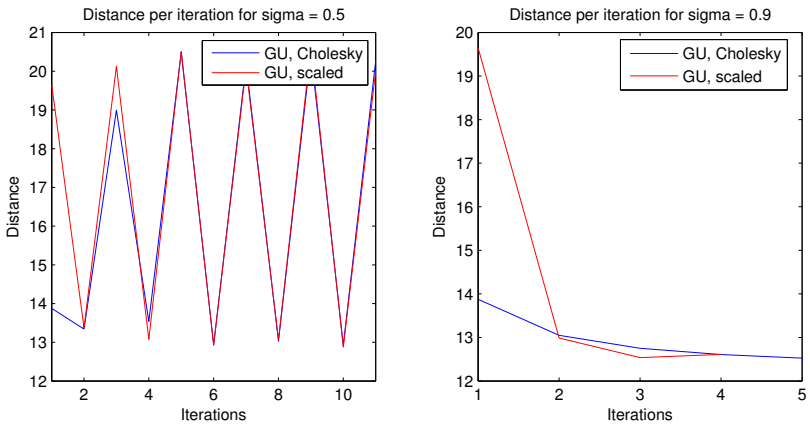

Figure 4.5: The convergence of the Gradient Updating methods with $\sigma = 0.5$ (left) and $\sigma = 0.9$ (right) and tol $= 10^{-2}$ for matrix $n = 25$ and entries $c_{i,j} \in (-1, 0)$

try again. This is called *Armijo backtracing*.

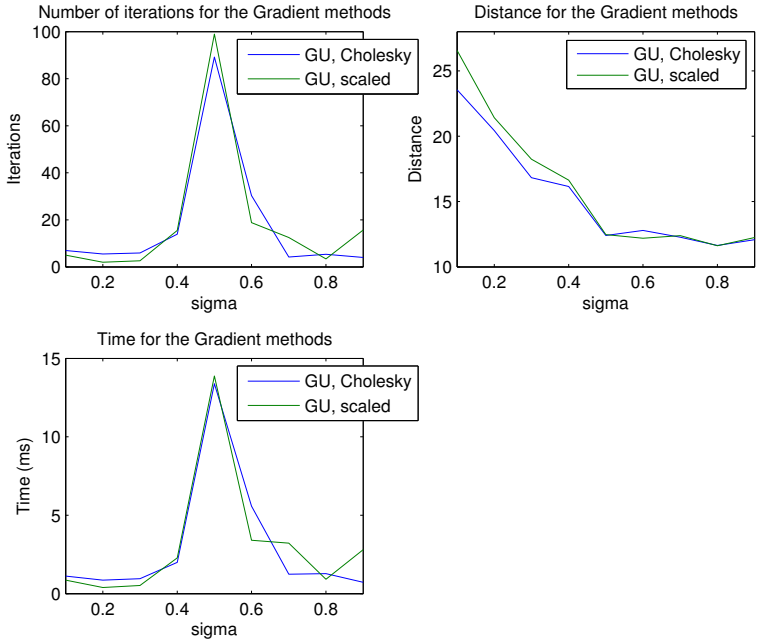

Figure 4.6: Results of the Gradient method with the two starting matrices for different σ for matrices with $c_{i,j} \in (-1, 0)$.

Let $m = 0, 1, 2, \ldots$ and let $1 - \sigma = \tau^m$. Then calculate ϕ_k and ψ_k by Equation (4.12). If the distance is smaller, remember the value of m and use for the next iteration. Combining this with Algorithm 5 leads to the the Gradient Updating Algorithm.

Algorithm 6 (Gradient Updating Algorithm)**.** Choose starting matrices A_1 and B_1 such that A_1B_1 is close to C. Also choose the parameter τ and set $m = 1$ and $k = 1$. Then

1. Compute the gradient $\text{grad}(A_k) = B_k^T(B_kA_k - C)$,
2. Determine step length $\phi_k = \frac{2(\tau^m)||\text{grad}(A_k)||_F^2}{||B_k\text{grad}(A_k)||_F^2}$
3. Set $A_{k+1} = A_k - \phi_k\text{grad}(A_k)$.
4. Compute the gradient $\text{grad}(B_k) = (B_kA_{k+1} - C)A_{k+1}^T$,
5. Determine step length $\psi_k = \frac{2(\tau^m)||\text{grad}(B_k)||_F^2}{||\text{grad}(B_k)A_k||_F^2}$
6. Set $B_{k+1} = B_k - \psi_k\text{grad}(B_k)$.
7. Set $B_{k+1} = (B_{k+1} + A_{k+1}^T)/2$,
8. Compute $T = \text{diag}(\text{diag}(B_{k+1}^TB_{k+1})^{-1})$,
9. Scale $B_{k+1} = \sqrt{T}B_{k+1}$ and $A = B_{k+1}^T\sqrt{T}$.
10. Calculate the distance $r_k = ||B_{k+1}A_{k+1} - C||_F$.
11. If $r_{k+1} - r_k < 0$ then continue. Else set $m = m + 1$ and go to Step 2.
12. If $\frac{|r_k - r_{k-1}|}{r_k} \leq 10^{-2}$ then stop; we are satisfied. Else go to Step 1.

For this new algorithm, we calibrate for τ using the same sampled invalid correlation matrices as in Figure 4.6. We concluded above that $\sigma \in [0.5, 0.6]$ was optimal matrices with entries in $(-1, 1)$. Since $\tau^m = 1 - \sigma$ and $m = 1$, we should look at the interval $\tau \in [0.4, 0.5]$. Different values of τ are compared in Figure 4.7. From sub figure 2, we see we should pick $\tau = 0.5$ since it results into a smaller distance of the resulting matrix to the original one. The implementation of this method is given in Section 11.7 for the Scaled initial matrix and in Section 11.8 for the Cholesky initial matrix.

This algorithm was derived by Zhang in [20]. However, it was proposed without the check if the algorithm is converging and the reduction of the steplength if it is not converging.

4.4 Adjusted Gradient Updating Method

In the previous section, the objective function was written as a function of $f(A, B)$. However, since $A^T = B$ we can reformulate this as a function of B only. Let

$$g(B) = f(B^T, B) = \frac{1}{2}||BB^T - C||_F^2,$$

then we can reduce the number of steps in the Gradient Updating algorithm, since we only have to iterate over one matrix B instead of over B and its transposed A.

As before, we can calculate the derivative of this function. Let b_i be the i^{th} row of B and $\tilde{b}_j$ its j^{th} column, then

$$g(B) = \frac{1}{2}\sum_{i,j=1}^{n}\left(c_{i,j} - \tilde{b}_i\tilde{b}_j^T\right)^2.$$

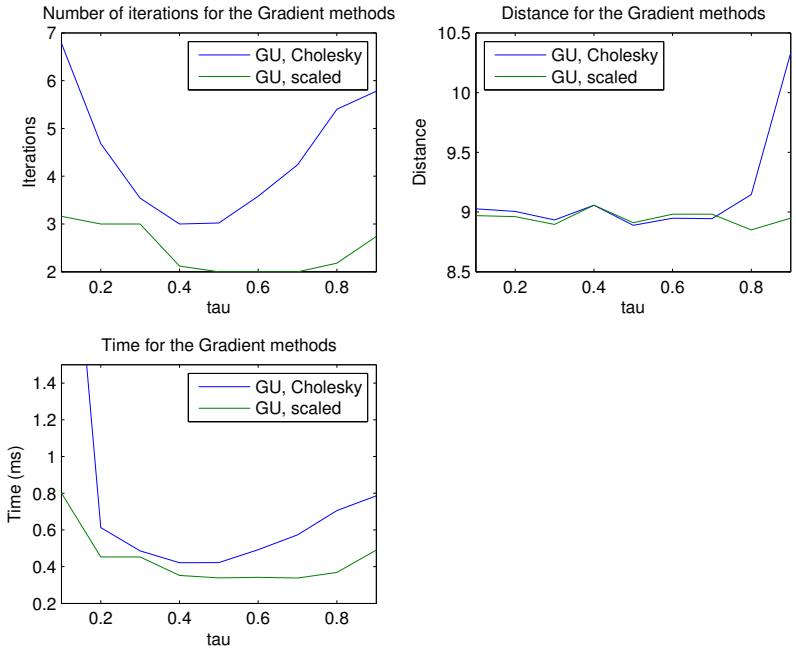

Figure 4.7: Results of the Gradient Updating method with the two initial matrices for different τ for matrices with $c_{i,j} \in (-1, 0)$.

By picking a fixed entry $b_{u,v}$ with $u, v \in \{1, 2, \ldots, n\}$ we can calculate the derivative of $g(B)$ with respect to $b_{u,v}$,

$$\frac{\partial}{\partial b_{u,v}} g(B) = \sum_{i,j=1}^{n} \left(\tilde{b}_i \tilde{b}_j - c_{i,j} \right) \left(\frac{\partial}{\partial b_{u,v}} \tilde{b}_i \tilde{b}_j^T \right).$$

By evaluating the right hand side we see

$$\frac{\partial}{\partial b_{u,v}} \tilde{b}_i \tilde{b}_j^T = \begin{cases} 0 & \text{if } i, j \neq u, \\ b_{j,v} & \text{if } i = u, j \neq u, \\ b_{i,v} & \text{if } j = u, i \neq u, \\ 2b_{u,v} & \text{if } i = j = u. \end{cases}$$

Combining these results and using the symmetry of C and BB^T, we get

$$\begin{aligned} \tfrac{\partial}{\partial b_{u,v}} g(B) &= \textstyle\sum_{j=1, i \neq i}^{n} \left(\tilde{b}_u \tilde{b}_j^T - c_{u,j} \right) b_{j,v} \\ &\quad + \textstyle\sum_{i=1, i \neq j}^{n} \left(\tilde{b}_i \tilde{b}_i^T - c_{i,u} \right) b_{i,v} + 2 \left(\tilde{b}_u \tilde{b}_u^T - c_{u,u} \right) b_{u,v} \\ &= 2 \textstyle\sum_{j=1}^{n} \left(\tilde{b}_u \tilde{b}_j^T - c_{u,j} \right) b_{j,v} \\ &= 2 \left(\tilde{b}_u B^T - \tilde{c}_u \right) b_v, \end{aligned}$$

which, in matrix form, gives us the gradient of g with respect to B,

$$\frac{\partial g}{\partial B} = 2(BB^T - C)B.$$

Again, we use the updating scheme based on the fact that the function $g(B)$ decreases most in the direction of the negative gradient. This gives

$$B_{k+1} = B_k - \theta_k \frac{\partial g(B_k)}{\partial A},$$

where we have to determine the positive step length θ_k. But since the gradient function is a non linear function on B, we cannot use a Taylor expansion to exactly estimate $g(B_{k+1})$ in two terms. There is simple formula to determine θ_k. We use the bounds on ϕ_k and ϕ_k to determine the bound on θ_k.

The bound on ϕ_k results into a converging iteration for the Gradient Updating method, disregarding the scaling. Since

$$2\frac{\partial f(B^T, B)}{\partial B} = \frac{\partial g(B)}{\partial B},$$

we let $2\theta_k = \phi_k$, so the first steps of the Gradient Updating algorithm and the adjusted method are the same. The Gradient Updating method performs iterations, on first A_k and than on B_k. Since we do these iterations in 1 step, we need to halve the step length ϕ_k. This results in

$$\theta_k = \frac{1}{4}\phi_k = \frac{(\tau^m)\left|\left|\frac{\partial g(B)}{\partial B}\right|\right|_F^2}{2\left|\left|\frac{\partial g(B)}{\partial B}^T\right|\right|_F^2}.$$

Numerical results confirm that this is a correct estimation of θ_k. The resulting algorithm is

Algorithm 7 (Adjusted Gradient Updating)**.** Choose starting matrix B_1 such that C is close to $B_1 B_1^T$. Also choose parameter $\tau = 0.5$ and set $m = 1$ and $k = 1$. Then

1. Compute the gradient $\text{grad}_k = 2(B_k B_k^T - C)B_k$,
2. Compute parameter $\theta_k = \frac{(\tau^m)||\text{grad}_k||_F^2}{2||\text{grad}_k B^T||_F^2}$,
3. Calculate the next matrix $B_{k+1} = B_k - \theta_k \text{grad}_k$.
4. Compute $T = \text{diag}(\text{diag}(B_{k+1}B_{k+1}^T)^{-1})$,
5. Scale $B_{k+1} = \sqrt{T}B_{k+1}$
6. Calculate the distance $r_{k+1} = ||B_{k+1}B_{k+1}^T - C||_F$
7. If $r_{k+1} - r_k < 0$ then continue. Else set $m = m + 1$ and go to Step 2.
8. If $\frac{|r_k - r_{k-1}|}{r_k} \leq 10^{-2}$ then stop. Else set $k = k + 1$ and go to Step 1.

The implementation in Matlab for the Adjusted Gradient Updating method is given in Section 11.9 for the Scaled initial matrix and in Section 11.10 for the Cholesky initial matrix

5 Comparing Methods

In this section, we compare the methods we have discussed in Section 4. The results we discuss are shown in Table 5.1. For each matrix size we have generated 20 invalid correlation matrices by picking its entries uniform for the specified interval, using the algorithm in Section 11.2. Then we applied the methods of Section 4 to it, and averaged the resulting distance and time.

First, consider the results for the Iterative Spectral method. We see this method is always the most accurate method, since its distance is the lowest for all matrices. However, this method applies an eigenvalue decomposition each iteration and is therefore extremely slow for large matrix sizes. It takes a second to process a matrix of size 250 and 4 to 7 seconds to process an $n = 500$ matrix. This is extremely slow, since we see other methods are at least 50 times faster for matrices of the same size.

The Scaled Spectral method is non iterative, and therefore the CPU time is constant. It is a lot faster than the Iterative Spectral method, however it is less accurate. This is most clear for the matrices with $c_{i,j} \in (0, 1)$. It is very accurate for matrices with negative matrix entries, there the distance differs less than 1 % from the Iterative Spectral method.

Looking at the Adjusted Gradient Updating methods, we see that for both of the initial matrices the performance is good. For larger matrices, the performance is good, it is faster and more accurate than the Scaled Spectral method. Only for $c_{i,j} \in (-1, 0)$, the Scaled Spectral method is slightly more accurate, however, the Adjusted Gradient Updating methods are almost twice as accurate on matrices $c_{i,j} \in (0.25, 0.75)$ and $c_{i,j} \in (0, 1)$.

As initial matrix for the Adjusted Gradient Updating methods, we see the Scaled input matrix performs better in general.

The Gradient Updating methods perform very similar to the Adjusted Gradient updating methods, but they are slower. Only for $c_{i,j} \in (-1, 0)$, it is faster, but also it is less accurate.

Some large differences appear for the Adjusted Gradient Updating methods and the Gradient Updating methods. This is due to the stopping criteria. For matrices of size 500×500, only two or three iterations are performed by the Gradient Updating methods and the Adjusted Gradient Updating methods, after which the decrease in distance is already less than a percent.

Results		Spectral Methods		Adjusted Gradient Updating		Gradient Updating	
		Iterative	Scaled	Cholesky	Scaled	Cholesky	Scaled
$c_{i,j} \in (-1,1)$	$n = 25$	8.91 / 4.16	9.23 / 0.30	8.95 / 0.44	8.98 / 0.36	9.01 / 0.60	9.03 / 0.48
	$n = 50$	20.32 / 14.93	21.09 / 0.83	20.36 / 1.40	20.46 / 0.53	20.48 / 1.06	20.55 / 0.77
	$n = 100$	45.00 / 115.03	46.56 / 3.62	45.06 / 2.59	45.24 / 1.52	45.22 / 4.49	45.38 / 2.48
	$n = 150$	70.66 / 370.15	72.93 / 9.00	70.76 / 6.74	70.99 / 3.72	70.94 / 11.21	71.18 / 6.02
	$n = 250$	122.77 / 1200.71	126.31 / 31.45	122.87 / 24.72	123.36 / 11.70	123.14 / 48.20	123.53 / 21.17
	$n = 500$	256.87 / 7440.98	263.04 / 198.23	257.52 / 156.78	258.16 / 72.36	257.90 / 328.81	258.26 / 143.27
$c_{i,j} \in (0,1)$	$n = 25$	4.21 / 2.77	5.16 / 0.22	4.38 / 0.77	4.39 / 0.41	4.67 / 1.05	4.39 / 0.52
	$n = 50$	9.90 / 13.69	13.31 / 0.81	10.37 / 1.56	10.34 / 0.79	11.24 / 2.48	10.42 / 1.27
	$n = 100$	22.17 / 100.31	32.32 / 3.52	23.40 / 4.39	23.39 / 1.79	23.77 / 9.57	23.27 / 4.06
	$n = 150$	34.99 / 329.88	53.33 / 8.90	37.48 / 8.94	36.87 / 3.77	39.15 / 11.08	37.22 / 5.63
	$n = 250$	61.00 / 1082.39	97.70 / 30.97	65.71 / 23.24	64.44 / 7.78	67.41 / 30.00	64.67 / 12.18
	$n = 500$	128.00 / 6681.93	216.82 / 196.26	143.40 / 45.99	133.94 / 44.35	141.09 / 124.37	134.02 / 126.99
$c_{i,j} \in (-1,0)$	$n = 25$	11.84 / 2.31	11.94 / 0.25	11.98 / 0.47	11.89 / 0.57	12.16 / 0.73	11.95 / 0.82
	$n = 50$	24.92 / 9.15	25.11 / 0.83	25.21 / 1.16	25.58 / 1.12	25.04 / 1.83	25.12 / 1.86
	$n = 100$	52.07 / 70.08	52.48 / 3.54	52.65 / 4.70	52.69 / 3.84	52.40 / 5.74	52.49 / 7.12
	$n = 150$	79.33 / 225.77	79.95 / 9.07	80.28 / 11.98	80.22 / 10.28	80.23 / 11.94	80.06 / 17.10
	$n = 250$	134.76 / 723.03	135.77 / 31.71	139.34 / 24.43	136.81 / 24.59	138.23 / 12.35	139.54 / 11.98
	$n = 500$	274.35 / 4589.42	276.28 / 200.42	283.26 / 122.29	278.93 / 149.96	286.06 / 82.83	282.58 / 92.20
$c_{i,j} \in (0.25, 0.75)$	$n = 25$	1.28 / 1.59	1.67 / 0.21	1.41 / 1.31	1.41 / 1.16	1.45 / 1.71	1.47 / 0.75
	$n = 50$	3.57 / 7.74	5.53 / 0.81	3.96 / 2.67	4.19 / 1.90	4.47 / 3.45	4.10 / 1.85
	$n = 100$	8.84 / 59.94	16.11 / 3.46	9.90 / 6.42	9.98 / 5.31	10.95 / 8.52	10.22 / 6.90
	$n = 150$	14.58 / 194.96	29.00 / 8.79	16.69 / 11.49	16.57 / 11.11	16.73 / 25.72	16.97 / 15.56
	$n = 250$	26.43 / 645.11	58.29 / 30.95	31.11 / 24.78	30.75 / 20.52	31.21 / 46.99	30.07 / 50.21
	$n = 500$	57.72 / 4088.31	143.30 / 195.13	71.70 / 47.35	65.55 / 73.50	68.53 / 191.24	64.82 / 142.54

Table 5.1: *Distance / CPU time (ms)* per method and entry interval.

6 Weighting Correlations

By estimating a correlation matrix, one may have much faith in some specific correlations while others do not matter at all. Therefore, we like to add some weighting to the correlations. We illustrate this for the Iterative Spectral method of Section 4.1.

6.1 Weighting the Iterative Spectral Method

Let W be a positive definite diagonal matrix, i.e. it has positive diagonal elements, where the higher $w_{i,i}$ is, the more faith we have in the i^{th} asset. This is extensively explained by Higham in [10].

Let C be an invalid correlation matrix and $\hat{C}$ be a positive definite correlation matrix. Then denote the weighted objective function by

$$f(C) = ||W^{1/2}(C - \hat{C})W^{1/2}||_F, \tag{6.1}$$

then, we get the following algorithm.

Algorithm 8 (Weighted Iterative Spectral Algorithm)**.** Let C be an invalid correlation matrix and W be a positive definite diagonal matrix. Set $C_1 = C$ and $k = 1$. Then

1. Determine L_k and S_k such that $W^{1/2}C_kW^{1/2} = S_kL_kS_k^T$ as in Theorem 1.
2. If all eigenvalues of C_k are positive, then stop, C_k is positive definite. Else, continue:
3. Let L^+ be L with negative eigenvalues replaced by a *small* positive value a.
4. Set $C_{k+1} = W^{-1/2}S_kL_k^+S_k^TW^{-1/2}$
5. Set the diagonal elements of C_{k+1} to 1.
6. Set $k = k + 1$ and go to Step 1.

The implementation of this algorithm is given in Section 11.11. We illustrate the algorithm with an example.

Example 6.1. Let W be the diagonal matrix with entries $c_{i,i} = 1$ for $i = 1, 2, 3$ and $c_{i,i} = 0.01$ for $i = 1, 2, \ldots, 25$. Applying this to a randomly generated 25×25 invalid correlation matrix with entries $c_{i,j} \in (0, 1)$ gives the result shown in Figure 6.1. This figure shows the absolute difference per matrix entry.

We have put extra weight on the first three assets, as is clearly visible in the figure. Some details about the image are shown in the Table 6.1. We see the weight distance, given in Equation (6.1) has decreased by 1/3, but the total distance of the matrix has doubled.

△

6.1.1 Choosing Weights

How to pick a weighting matrix W is an arbitrary process and depends on the origins of the estimates. We try to cover most of the questions concerning this by considering the following test case.

Given an invalid correlation matrix with $n = 25$ and positive entries $c_{i,j}$ and let the entries of the diagonal weighting matrix W be given by

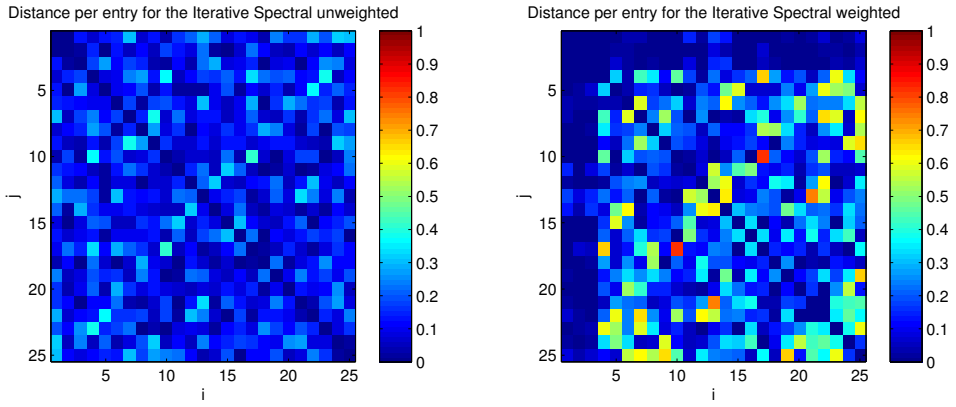

Figure 6.1: Distance of both the unweighted and weighted It. Spectral method per matrix entry.

	Unweighted It. Spectral	Weight It. Spectral
Iterations	13	102
Unweight Distance	3.88	6.49
Weight Distance	0.27	0.09
CPU time (ms)	5.8	31.1

Table 6.1: Results from the Weighted and Unweighted Iterative Spectral method

$$w_{i,} = \begin{cases} a, & \text{if } i = 1, 2, 3 \\ b, & \text{if } i = 4, 5, \ldots, 25 \end{cases}$$

Then we test whether W depends on the absolute weights or just on the difference between them. Therefore, let $a = 10b$ and consider $b = 0.001$, $b = 0.01$,$b = 0.1$, $b = 1$, $b = 10$. Define the *gain factor* as the outcome of the unweighted divided by the weighted results. This is shown in Table 6.2. As is clear, the number of iterations increases as b increases. So we should pick b as small as possible. However, as we see for $b = 0.001$, the weighted distance has not increased, but the total distance is worse. So the interval for the weighting entries is $w_{i,i} \in (0, 1]$, where we assign 1 to the entries which we have the most confidence in.

	$b = 0.001$	$b = 0.01$	$b = 0.1$	$b = 1$	$b = 10$
Unweight Distance	0.66	0.79	0.83	0.84	0.84
Weight Distance	1.00	1.20	1.26	1.26	1.26
Iterations	2	7	15	26	36

Table 6.2: Gain factors from the Weighted and Unweighted Iterative Spectral method with $a = 10b$

Remark: As we see, we can increase the speed of the Iterative Spectral method of Section 4.1 by using a weighting matrix W with all entries equal to a positive value $b < 1$. However, the

method is still slower than the Gradients Methods are and it is even less accurate, so we do not include this adjustment.

Now, let $a = 1$. Then the question raises what value to assign to b, the rows which we have less faith in. This is shown in Figure 6.2. We see that the gain factor on the weighted distance decreases as b increases, so therefore we want b to be as low as possible. However, the total distance and the number of iterations increases as b decreases.

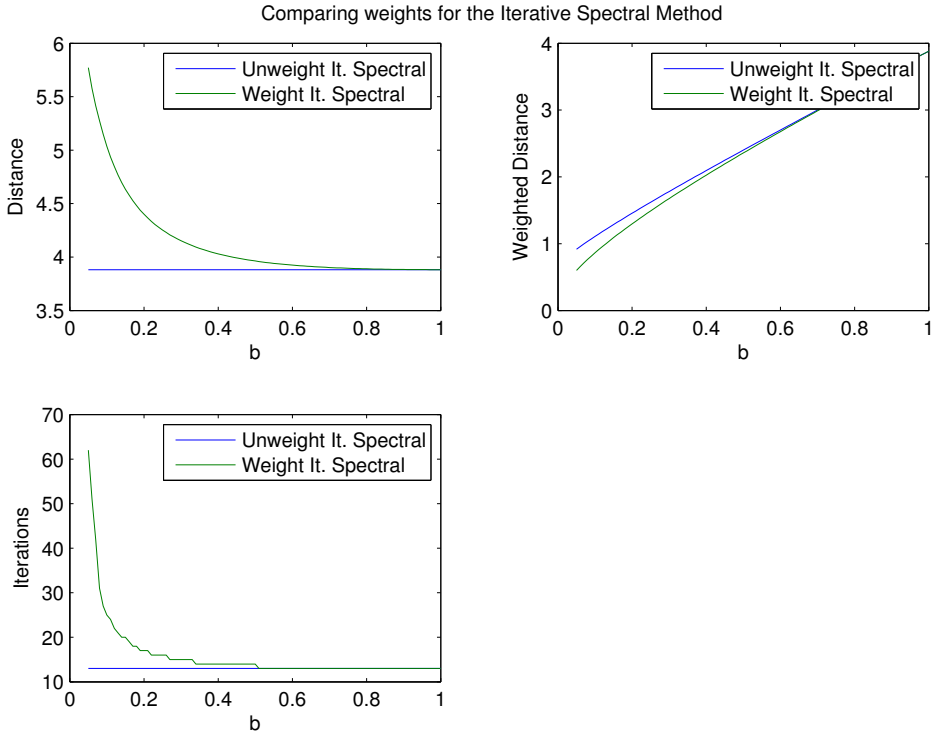

Figure 6.2: $n = 25$ invalid correlation matrix example. Assigned 1 to the first three weights, and b to the others.

6.2 Weighting the Scaled Spectral Method

We can apply the same weighting to the Scaled Spectral method. This is exactly analog with the Iterative Spectral method, so we only mention the algorithm.

Algorithm 9 (Weighted Scaled Spectral method)**.** Let C be an $n \times n$ (non-positive definite) correlation matrix and W be a positive definite diagonal matrix. Then

1. Spectral decompose $W^{1/2}CW^{1/2}$ as in Theorem 1, with L the diagonal matrix with eigenvalues and S the orthogonal matrix with eigenvectors as columns.

2. Let L^+ be L with negative eigenvalues changed to a *small* positive value a.

3. Denote the entries of S by $s_{i,j}$. Calculate the scaling matrix T with entries $t_{i,j}$ given by

$$T \equiv \begin{cases} t_{i,j} = \left[\sum_j s_{i,j}^2 \lambda_j\right]^{-1}, & i = j \\ t_{i,j} = 0, & i \neq j \end{cases}$$

4. Set $C_2 = W^{-1/2}\sqrt{T}SL^+S^T\sqrt{T}W^{-1/2}$.

The implementation of the Weighted Scaled Spectral method is given in Section 11.12. We illustrate the addition of weighting on the scaled method in the following example.

Example 6.2. As in Example 6.1, we add more weights to the first three rows and columns. This results in Figure 6.3. From this figure we see the overall distance has increased. This is indeed correct, the Unweight Scaled Spectral method has a distance of 4.8, the Weighted has a distance of 5.7. The computation time is almost the same for both methods, since it only differs by 4 times the multiplication of a diagonal matrix.

And the good part is, the weighted distance has decreased, but only slightly, from 0.28 to 0.17.

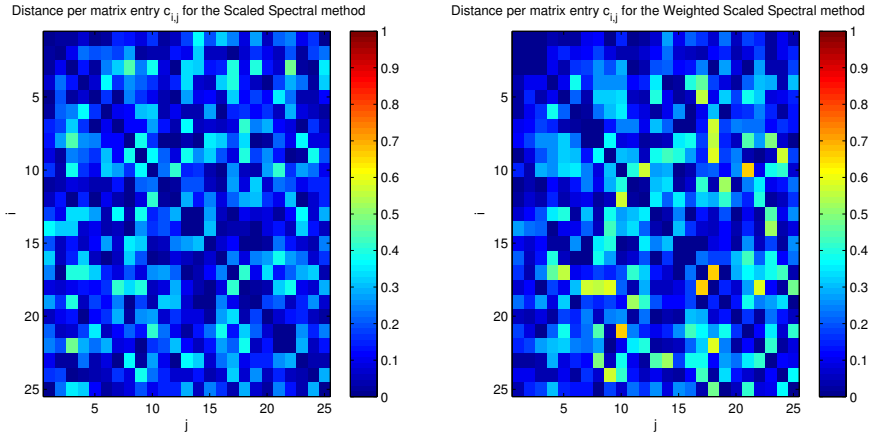

Figure 6.3: Distance of both the unweighted and weighted Scaled Spectral method per matrix entry.

△

7 Other Methods

There are a number of other methods around which are not discussed in the comparison. In this section those methods are shortly discussed and the advantages and disadvantages are listed.

7.1 Hypersphere Method

This method is first introduced in [15] and is based on spherical coordinates. We parameterize a matrix B such that BB^T has unit diagonal. This can by done by spherical coordinates with unit diagonal. These are given by

$$b_{i,j} = \begin{cases} \cos\theta_{i,j} \prod_{k=1}^{j-1} \sin\theta_{i,k}, & \text{for } j = 1..n-1 \\ \prod_{k=1}^{j-1} \sin\theta_{i,k}. & \text{for } j = n \end{cases} \tag{7.1}$$

Because all entries of B are products of cosine and sine functions, their values will be between -1 and 1 and so are all values of BB^T. We prove that BB^T has unit diagonal.

Theorem 8 (Hypersphere unit diagonal)**.** *View matrix B as a projection on a unit hypersphere, which coordinate system is given by Equation (7.1). Then the diagonal entries of BB^T are equal to 1.*

Proof. Let B be given by Equation (7.1). Then i^{th} diagonal element of BB^T is given by

$$\begin{aligned} \textstyle\sum_{j=1}^{n} b_{i,j}^2 &= \textstyle\sum_{j=1}^{n-1} \left(\cos\theta_{i,j} \cdot \prod_{k=1}^{j-1} \sin\theta_{i,k}\right)^2 + \left(\prod_{k=1}^{n-1} \sin\theta_{i,k}\right)^2, \\ &= \textstyle\sum_{j=1}^{n-1} \cos^2\theta_{i,j} \cdot \prod_{k=1}^{j-1} \sin^2\theta_{i,k} + \prod_{k=1}^{n-1} \sin^2\theta_{i,k}. \end{aligned}$$

Now we have an expression for the diagonal entries, we show by induction that it equals one by taking the last item from the sum, and compare it with the term b_{in}. We note we can write $\cos^2 + \sin^2 = 1$ out of the brackets, and reduce the summation by one step:

$$\begin{aligned} \textstyle\sum_{j=1}^{n} b_{i,j}^2 &= \textstyle\sum_{j=1}^{n-2} \cos^2\theta_{i,j} \cdot \prod_{k=1}^{j-1} \sin^2\theta_{i,} + \cos^2\theta_{i(n-1)} \cdot \prod_{k=1}^{n-2} \sin^2\theta_{i,} + \prod_{k=1}^{n-1} \sin^2\theta_{i,k}, \\ &= \textstyle\sum_{j=1}^{n-2} \cos^2\theta_{i,j} \cdot \prod_{k=1}^{j-1} \sin^2\theta_{i,k} + (\cos^2\theta_{i(n-1)} + \sin^2\theta_{i(n-1)}) \left(\prod_{k=1}^{n-2} \sin^2\theta_{i,}\right), \\ &= \textstyle\sum_{j=1}^{n-2} \cos^2\theta_{i,j} \cdot \prod_{k=1}^{j-1} \sin^2\theta_{i,k} + \prod_{k=1}^{n-2} \sin^2\theta_{i,k}. \end{aligned}$$

If we iterate over the above steps $n-1$ times, we get the result

$$\sum_{j=1}^{n} b_{i,j}^2 = \cos^2\theta_{i,1} + \sin^2\theta_{i,1} = 1.$$

□

We now have an unconstrained parameterization of a valid correlation matrix. However, it introduces n(n-1) parameters θ for only $n(n-1)/2$ matrix entries, since the diagonal is fixed and the matrix is symmetric. As is shown by Brigo en Mercurio in [15], the resulting matrix depends only on the difference between parameters. We show this in the next example.

Example 7.1 (Hypersphere decomposition for a 2×2 matrix)**.** Consider the 2×2 case hypersphere coordinates system, then B is given by

$$B = \begin{pmatrix} \cos\theta_{1,1} & \sin\theta_{1,1} \\ \cos\theta_{2,1} & \sin\theta_{2,1} \end{pmatrix} \text{ and } BB^T = \begin{pmatrix} 1 & \cos(\theta_{1,1} - \theta_{2,1}) \\ \cos(\theta_{1,1} - \theta_{2,1}) & 1 \end{pmatrix}.$$

We note we only have one missing value, which depends on the difference between two parameters $\theta_{1,1}$ and $\theta_{2,1}$.

△

Because all correlations depend on the difference between vectors, and not on the coordinates of a vector itself, we parameterize the difference between angles, as in [15]. Consider v_i as the i^{th} column of B^T. Take the first column v_1 as the reference column, so let it be the unit vector e_1[3].

Now, consider the second vector v_2 such that it forms an angle $\theta_{2,1} = \arccos(c_{1,2})$ with the first vector in the (x_1, x_2) plane, such that $v_1 v_2 = c_{1,2}$ follows. Note that we only allow $\theta_{2,1}$ to be in $[0, \pi]$. Each $\hat{\theta}_{2,1} = 2\pi - \theta_{2,1}$ also complies, but we only use the first solution, such that it is unique.

Then the third vector v_3 must satisfy $c_{1,3} = v_3 v_1$ and $c_{2,3} = v_2 v_1$. This can be fulfilled by rotating e_1 through two angles, $\theta_{3,1}$ in the (x_1, x_2) plane and $\theta_{3,2}$ in the (x_2, x_3) plane, as shown in Figure 7.1.

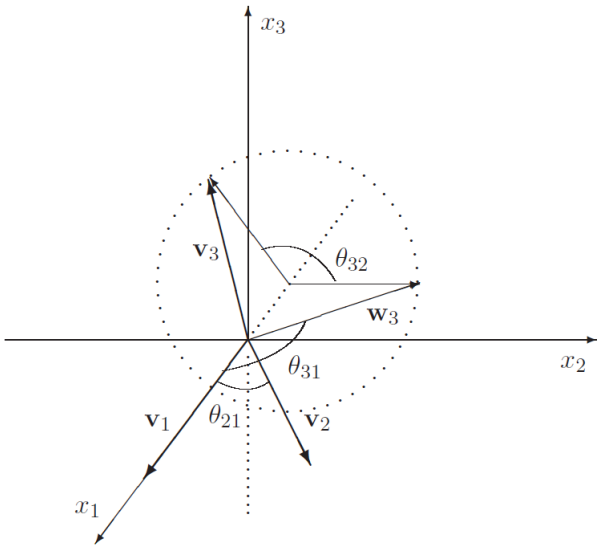

Figure 7.1: The first three columns of B^T. v_1 lies along the x_1 axis, v_2 rotated over $\theta_{2,1}$ radians in the (x_1, x_2) plane and v_3 is a rotation of $\theta_{3,1}$ radians in the (x_1, x_2) plane (w_3) and a rotation of $\theta_{3,2}$ in the (x_2, x_3) plane.

The mathematical interpretation of rotating vectors in different dimensions is given by Jacobi Rotations.

[3]The i^{th} unit vector is a vector consisting of all zeros but a one on the i^{th} position, denoted by e_i.

Definition 8 (Jacobi Rotations)**.** *Denote x_i the i^{th} coordinate in $\mathbb{R}^n$ The Jacobi rotation by clockwise angle θ of a vector $w \in \mathbb{R}^n$ in the (x_i, x_k) plane by applying the following $n \times n$ matrix G*[4]

$$G(i,k;\theta) = \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \cos\theta & \cdots & \sin\theta & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & -\sin\theta & \cdots & \cos\theta & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix}.$$

We show how to derive the mentioned rotations of the vectors v_i above by multiplying it by the Jacobi rotation matrix G in the following example.

Example 7.2 (Applying Jacobi Rotations)**.** Rotate e_1 over $\theta_{2,1}$ counterclockwise in the (x_1, x_2) plane.

$$G(1,2;-\theta_{2,1})e_1 = \begin{pmatrix} \cos\theta_{2,1} & -\sin\theta_{2,1} & 0 & \cdots & 0 \\ \sin\theta_{2,1} & \cos\theta_{2,1} & 0 & \cdots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & \ldots & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} \cos\theta_{2,1} \\ \sin\theta_{2,1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

△

Applying those rotations for each v_i as column of B^T gives us $\hat{B}$, with its entries $\hat{b}_{i,j}$ given by

$$\hat{b}_{i,j} = \begin{cases} \cos\theta_{i,j} \prod_{k=1}^{j-1} \sin\theta_{i,k}, & \text{for } j = 1..n-1, \\ \prod_{k=1}^{j-1} \sin\theta_{i,k}, & \text{for } j = n, \\ 0, & \text{for } i+1 \le j \le n, \end{cases} \tag{7.2}$$

with $\theta \in [0, \pi]$. As we note, this is the same as de hypersphere coordinates, but only the lower diagonal part. To get rid of the bounded region for θ, we introduce

$$\theta_{k,j} = \frac{\pi}{2} - \arctan x_{k,j}, \quad \text{for } j = 1, \ldots, k-1$$

which maps the function from $\theta \in [0, \pi]$ to $x \in [-\infty, \infty]$. This changes (7.2) with the fact that

$$\cos\left(\frac{\pi}{2} - \arctan x_{i,j}\right) = \frac{x_{i,j}}{\sqrt{x_{i,j}^2 + 1}} \text{ and } \sin\left(\frac{\pi}{2} - \arctan x_{i,j}\right) = \frac{1}{\sqrt{x_{i,j}^2 + 1}},$$

[4]Matrix G only differs from the identity matrix in the entries (i,k), (k,i), (i,i), (k,k).

and some simple algebra to

$$\hat{b}_{i,j} = \begin{cases} x_{i,j} \left(\prod_{k=1}^{j} (x_{i,k}^2 + 1) \right)^{-\frac{1}{2}}, & \text{for } j = 1..i-1 \\ \left(\prod_{k=1}^{j-1} (x_{i,k}^2 + 1) \right)^{-\frac{1}{2}}, & \text{for } j = i \\ 0, & \text{for } i+1 \leq j \leq n \end{cases} \tag{7.3}$$

with $\mathbf{x} \in \mathbb{R}^{n \times n-1}$, where $x_{i,j} = 0$ for $j \geq i$.

Now, we use this parameterization to iterate from a starting vector $\mathbf{x}_0$ to a vector $\hat{\mathbf{x}}$ which results in the nearest valid correlation matrix BB^T to the invalid correlation matrix C.

Therefore, we need to derive a starting vector $\mathbf{x}_0$. We show how to find this vector using the Scaled Spectral decomposition in the next example.

Example 7.3 (Inverse Triangular Angle Parameterization)**.** Let C be as in Example 3.1. After applying spectral decomposition (with $a = 10^{-13}$), we get a valid correlation matrix $\hat{C}$ as in Equation (4.8). Now we apply a Cholesky decomposition to get an upper triangular matrix B^T, which satisfies $\hat{C} = BB^T$.

$$B = \begin{pmatrix} 1.0000 & 0 & 0 \\ 0.8932 & 0.4496 & 0 \\ 0.6958 & -0.7126 & 0.0895 \end{pmatrix}$$

By inverting Equation (7.3), we get

$$x_{i,j} = \begin{cases} b_{i,j} \left(1 - b_{i,j}^2\right)^{-\frac{1}{2}}, & \text{for } j = 1 \\ b_{i,j} \left(\sum_{k=j+1}^{n} b_{i,k}^2 \right)^{-\frac{1}{2}}, & \text{for } j = 2..i-1 \\ 0. & \text{for } i+1 \leq j \leq n \end{cases} \tag{7.4}$$

Here x is a matrix containing zeros for $i + 1 \leq j \leq n$, To remove all unused entries, we redefine coordinate the vector x corresponding to a valid correlation matrix $\hat{C}$ as

$$x = (x_{2,1}, x_{3,1}, x_{3,2}, x_{4,1}, \ldots, x_{n,n-1}).$$

with $x_{i,j}$ defined as in equation 7.4. And use it as the starting vector x_0 for the algorithm. For the example, this means,

$$x_0 = (1.9866, 0.9688, -7.9643).$$

△

The Matlab functions `tap(x)`, which is an implementation in Matlab of Equation (7.3), and the function `arctap(B)`, which is the implementation of Equation (7.4) are included in Section 11.13.

7.1.1 Results for the Hypersphere method

We described how to parameterize B in a way BB^T always has unit diagonal. This is a property none of the other methods have. However, we have not found a fast way to implement the parameterization in Matlab, since it cannot be expressed easily as matrix multiplication.

Also, as soon as we have an input matrix, we need to minimize the distance. The build-in `matlab`-function `fminsearch` could do this, but it is very slow (> 30 seconds for a 50×50 matrix). An improvement is the function `fminunc`, which takes about 10 seconds for the same matrix. But this is still far to slow, since the Iterative Spectral method takes about 0.007 seconds to calculate the nearest valid correlation matrix to the same matrix.

We could speed up the process by calculating the gradient of the target function with respect to the input vector $\mathbf{x}$. But since we have no nice way of writing B as a function of $\mathbf{x}$ We did not succeed in calculating the gradient of B with respect to x, if this is even possible.

The Hypersphere Decomposition method could only be a fast method if it the gradient of the distance with respect to x is calculated and an efficient implementation of the parameterization is found.

7.2 Vines Method

The method of vines is discussed in [12] by Kurowicka and Cooke and is based on the partial correlation of random variables. We demonstrate how the method works, but do not cover the full theory behind it. More on vines in [4].

Definition 9 (Partial Correlation)**.** *Let $X_1, \ldots, X_n$ be random variables, and let $\{i, j, k\}$ be a set of distinct indices and let S be a set of indices disjoint from $\{i, j, k\}$. Then the partial correlation of X_i and X_j given $\{X_k, \bigcup\{X_h | h \in S\}\}$ is*

$$\rho_{ij;kS} = \frac{\rho_{ij;S} - \rho_{ik;S}\rho_{jk;S}}{\sqrt{1 - \rho^2_{ik;S}}\sqrt{1 - \rho^2_{jk;S}}}.$$

Where ρ_{ij} is the correlation between X_i and X_j. If $\rho^2_{ik;S} = 1$ or $\rho^2_{jk;S} = 1$, then the partial correlation is undefined.

We illustrate this by the use of an example.

Example 7.4. Since the example matrix of size 3×3 we used before in the examples is not very illustrative in this application, let

$$C = \begin{pmatrix} 1 & -0.6 & -0.8 & 0.5 & 0.9 \\ -0.6 & 1 & 0.6 & -0.4 & -0.4 \\ -0.8 & 0.6 & 1 & 0.1 & -0.5 \\ 0.5 & -0.4 & 0.1 & 1 & 0.7 \\ 0.9 & -0.4 & -0.5 & 0.7 & 1 \end{pmatrix}.$$

Then we calculate the partial correlation

$$\rho_{2,3;1} = \frac{\rho_{2,3} - \rho_{1,2}\rho_{1,3}}{\sqrt{1 - \rho^2_{1,2}}\sqrt{1 - \rho^2_{1,2}}} = \frac{0.6 - (-0.6)(-0.8)}{\sqrt{1 - (-0.6)^2}\sqrt{1 - (-0.8)^2}} = 0.25$$

In the above partial correlation, we call 1 the root-node, and since it is only one variable, X_1, we call this a partial correlation of order one.

By this formula we can calculate all partial correlations with root one, which are

$$\begin{pmatrix} \rho_{2,3;1} & \rho_{2,4;1} & \rho_{2,5;1} \\ \rho_{3,4;1} & \rho_{3,5;1} & \rho_{4,5;1} \end{pmatrix} = \begin{pmatrix} 0.25 & -0.14 & 0.40 \\ 0.96 & 0.84 & 0.66 \end{pmatrix}.$$

Using all partial correlations of order one, we calculate all second order partial correlations and also the third order partial correlation.

$$\begin{aligned} \begin{pmatrix} \rho_{3,4;12} & \rho_{3,5;12} & \rho_{45;12} \end{pmatrix} &= \begin{pmatrix} 1.04 & 0.84 & 0.79 \end{pmatrix}, \\ \begin{pmatrix} \rho_{4,5;123} \end{pmatrix} &= \begin{pmatrix} 0.47i \end{pmatrix}. \end{aligned}$$

△

The reason the last partial correlation $\rho_{4,5;123}$ is complex, is since $\rho_{3,4;12} \notin (-1,1)$. This is also the criterion for positive definiteness. A estimated correlation matrix C is positive definite if all partial correlations are contained in $(-1,1)$ [12, Thrm. 3.2]. This is implemented in the next algorithm.

Algorithm 10 (Checking Positive definiteness by Vines)**.** Let C_0 be an $n \times n$ estimated correlation matrix, then calculate all partial correlations by repeating for $k = 1, 2, \dots (n-2)$,

1. Write $C_k = \begin{pmatrix} X_k & Y_k \\ Y_k^T & Z_k \end{pmatrix}$, where X_k is a 1×1 matrix, Y_k a $1 \times (n-1)$ matrix and Z_k a $(n-1) \times (n-1)$ matrix,
2. Compute $C_{k+1} = Z_k - Y_k^T Y_k$.
3. Create a scaling matrix T, with entries $t_{i,j}$ given by
$$t_{i,j} = \begin{cases} c_{i,j}^{-1} & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$
4. Scale $C_{k+1} = \sqrt{T} C_{k+1} \sqrt{T}$.
5. if all entries of C_{k+1} are contained in $(-1,1)$, continue. If one entry is outside the interval, C is non positive definite.

The Matlab implementation of Algorithm 10 is given in Section 11.14. This algorithm is derived in [12, section 4]. Using this algorithm, we develop a method of correcting non positive definite correlation matrices.

If by applying Algorithm 10, we find a partial correlation not in $(-1,1)$, we replace it with a close value inside that interval. For example if a partial correlation is -1.4, we replace it by -0.9, if it was originally 1.4, we replace it by 0.9. The closer we stay to the original value, the less will the distance of the resulting matrix to the original one [12, Thrm 5.1]. We illustrate this by continuing the previous example.

Example 7.5. We continue with the results of Example 7.4. As we saw, $\rho_{3,4;12} > 1$, so we replace it by $V(\rho_{3,4;12}) = 0.9$, but now, we have to change $\rho_{3,4;1}$ and $\rho 3, 4$ by inverting the definition of partial correlations to

$$\rho_{3,4;1} = \rho_{3,4;12} \sqrt{1 - \rho_{2,3;1}^2} \sqrt{1 - \rho_{2,4;1}^2} + \rho_{2,3;1} \rho_{2,4;1} = 0.96,$$

and

$$\rho_{3,4} = \rho_{3,4;1}\sqrt{1-\rho_{1,3}^2}\sqrt{1-\rho_{1,4}^2} + \rho_{1,3}\rho_{1,4} = 0.03.$$

Now, we have only changed $\rho_{3,4}$ in the estimated correlation matrix and by recalculating all partial correlations with this changed value, we see C is positive definite now with

$$C = \begin{pmatrix} 1 & -0.6 & -0.8 & 0.5 & 0.9 \\ -0.6 & 1 & 0.6 & -0.4 & -0.4 \\ -0.8 & 0.6 & 1 & 0.029 & -0.5 \\ 0.5 & -0.4 & 0.029 & 1 & 0.7 \\ 0.9 & -0.4 & -0.5 & 0.7 & 1 \end{pmatrix}.$$

Note that only cell (3,4) is altered. This results into a distance of 0.1. For comparison, the Iterative Spectral method results into a distance of 0.015 which is about seven times more accurate. △

The implementation in Matlab of the method described in Example 7.5 is given in Section 11.15.

7.2.1 Results for the Vines Method

The Vines method only changes wrong values of an estimated correlation matrices, and therefore, it loses accuracy. So in general, it is not accurate and also, it is quite slow since for an $n \times n$ matrix, we need to calculate

$$\frac{(n-2)(n-1)n}{6} \leq \frac{n^3}{6},$$

partial correlations. And if one of the partial correlations has to be adjusted, we have to recalculate most of the previously done calculations.

Also, we ran into numerical roundoff errors, since we adjust only the wrong correlations, the matrices balances on the edge of non positive definiteness, and there is no solution yet to fix this.

So for the unweighted problem this method is not fast and accurate enough. But since the method changes only specific *wrong* correlations, we might use it for the weighting problem.

7.2.2 Weighting Correlations with the Vines Method

The vines method favors the first row of the correlation matrix. These entries are not changed. The changes are greater the further we go from the first row. Hence, we should rearrange variables to have to most reliable entries in the first row [12, Remark 5.2].

We create a diagonal matrix W with diagonal entries $w_{i,i}$ which denote the confidence we have in the estimated correlations of X_i. This confidence is a relative number, which is used to reorder the rows and columns of the correlation matrix.

For a randomly generated invalid correlation matrix with $n = 25$ and entries $c_{i,j} \in [-1, 1]$, we compare both the weighted and unweighted versions. The results are shown in Table 7.1 and Figure 7.2. We see the re-ordering allows us to reduce the weighted distance, but if we compare this to weighting the same matrix with the Iterative Spectral method, we see the distance is 9.4 and the weighted distance is 6.6, so the Vines method is also for applying weighted not accurate enough.

The implementation of the Weighted Vines method is given in Section 11.16.

	Unweighted Vines	Weight Vines
Iterations	299	299
Unweight Distance	17.7	18.6
Weight Distance	16.8	15.0
CPU time (ms)	26	22

Table 7.1: Results from the Weighted and Unweighted Vines method for an $n = 25$ matrix with entries $c_{i,j} \in [-1, 1]$.

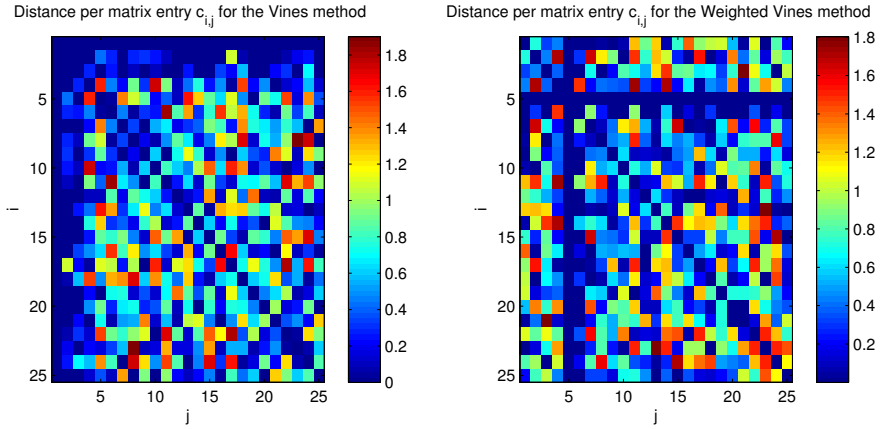

Figure 7.2: Comparing the Vines method with (left) and without (right) reordering the rows. We no confidence in the first three rows ($w_{i,i} = 0.01$) and full confidence in the other rows ($w_{i,i} = 1$).

7.3 Quadratically Convergent Newton Method

This method is introduced by Qi and Sun in [14], and is based on the dual of the problem. An improved version is derived by Borsdorf and Higham in [5]. The dual of the problem of finding the nearest correlation matrix is given by minimizing

$$\frac{1}{2}||(A + \text{diag}(y))_+||_F^2 - e^T y,$$

where the operator $(\cdot)_+$ is the projection of a matrix on the set of positive definite matrices. This projection is the same as the projection P_S described by Higham in Section 4.1.1. Spectral decompose the invalid correlation matrix and replace the negative eigenvalues by a *small* positive value a. Then put the matrix back together like in the Spectral methods.

However, the gradient is not explicitly available and needs to be approximated, but this results in a much slower algorithm than the Gradient Updating method.

8 Conclusion

In this report, we have discussed different methods of correcting non positive definite correlation matrices. As we have seen in Section 3, the fastest way to check if a matrix is positive definite is by applying the Cholesky decomposition. If this method fails to compute the Cholesky factor M of a matrix C, then C is non positive definite.

When we have computed a valid correlation matrix, we want to know how **close** it is to its original one. Therefore, we use the Frobenius norm to measure accuracy. The Frobenius norm for matrices sums over the absolute difference of all matrix entries.

The most accurate method is the Iterative Spectral method of Section 4.1 with Dykstra's corerction (Algorithm 3). For matrices up to 25×25 the Iterative Spectral method is about four times slower than the Adjusted Gradients method. Since the Iterative Spectral method uses an eigenvalue decomposition each iteration, it is extremely slow for large matrices. For a matrix of size 500×500, it is about 50 times slower than the Adjusted Gradients method.

To correct a non positive definite correlation matrix, the best method appears to be the Adjusted Gradients method with an Scaled initial matrix. This method is discussed in Section 4.4. It is nearly as accurate as the Iterative Spectral method, and is very fast. (Table 5.1). Also, since this method produces a valid correlation matrix every iteration, we can specify what distance is acceptable for us. Of course, by increasing the accuracy we also increase the computational time.

The Adjusted Gradient Updating method with Scaled initial matrix is flexible, fast and has with a good accuracy. Therefore, this method is the winning method.

We have also looked at weighing correlations. This is done in Section 6. It can be implemented for the Spectral methods, and has most effect for the Iterative Spectral method. We can add weights per matrix row (or column, since a correlation matrix is symmetric). We can decrease the weighed distance with a factor three, but then the unweighted distance doubles. Also, the number of iterations increases when adding weights. Since the Iterative Spectral method was was already slow, this weighting can only be applied to small matrices, up to about size 25×25.

Weighting can also be applied to the Scaled Spectral method, but its not very effective.

Two other methods which are not yet fast enough are the Quadratically convergent Newton method of Section 7.3 and the Vines method of Section 7.2. If the quadratically convergent Newton method can be implemented efficiently, it would be useful for large matrices of size 1000×1000 and higher.

The Vines method is a different method than all others since it only changes those correlations which it determines are *wrong*. However, is it not very fast and is affected by numerical round-off errors, for which no solution was found.

9 Recommendations and Discussion

As mentioned in the conclusion of Section 8, the winning method is the Adjusted Gradient Updating method with scaled initial matrices.

However, we have only measured it's accuracy in terms of the Frobenius norm. To use this method in a real life financial environment, it is advisable to first check the impact of the changes to asset prices, as we do not measure that.

All of the matrices are only tested in Matlab. Since Matlab if very efficient in making calculations with matrices, we have reformulated the algorithms in terms of matrix multiplications. However, when using another programming environment, results may differ.

For testing the methods, we have generated invalid correlation matrices of size $n \times n$ using the Matlab function `generate_cor(n)` of Section 11.2. These random invalid correlation matrices might not be a good representation of real financial correlation matrices, and therefore the converging speed and the resulting distance may differ from the results shown in this paper. However, the results for the randomly generated matrices are promising.

10 References

[1] R Adams. *Calculus : a complete course*. Pearson Addison Wesley, 7th edition, 2010.

[2] L Armijo. Minimization of functions having Lipschitz continious first partial derivatives. *Pacific Journal of Math.*, 16(1):1–3, 1966.

[3] F Beauregard. *Linear Algebra*. Addison-Wesley, 3th edition, 1995.

[4] T Bedford and R Cooke. Vines - a new graphical model for dependent random variables. *Ann. Statist.*, 30(4):1031–1068, 2002.

[5] R Borsdorf and N Higham. A preconditioned Newton algorithm for the nearest correlation matrix. *IMA J. Numer. Anal.*, 30(1):94–107, 2010.

[6] F Deutsch. The method of alternating orthogonal projections. In *Approximation theory, spline functions and applications*. Kluwer Acad. Publ., 105-121, 1992.

[7] R Dykstra. An algorithm for restriced least squares regression. *J. Amer. Stat. Assoc.*, 78(384):837–842, 1983.

[8] S Han. A successive projection method. *Math. Prog.*, 40:1–14, 1988.

[9] N Higham. *Accuracy and Stability of Numerical Algorithms*. Philadelphia: Soc. Industrial and Appl. Math., 1996.

[10] N Higham. Computing the nearest correlation matrx - a problem from finance. *Numer. Anal.*, 22:329–343, 2001.

[11] N Higham. Cholesky factorization. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(2):251–254, 2008.

[12] D Kurowicka and R Cooke. A parameterization of positive definite matrices in terms of partial correlation vines. *Lin. Alg. Appl.*, 372:225–251, 2003.

[13] D Lay. *Linear Algebra and its Applications*. Addison Wesley, New York, 1993.

[14] H Qui and D Sun. A quadratically convergent newton method for computing the nearest correlation matrix. *Siam J. Matrix Anal. Appl.*, 28(2):360–385, 2005.

[15] F Rapisarda, D Brigo, and F Mercurio. Parameterizing correlations: a geometric interpretation. *IMA J Management Math*, 1:55–73, 2007.

[16] R Rebonato and P Jäckel. The most general methodology to create a valid correlation matrix for risk management and option pricing purposes. *J. Risk*, 2(2):17–27, 1999.

[17] K Schöttle and R Werner. Improving "the most general methodology to create a valid correlation matrix". 2004.

[18] S Schreve. *Stochastic calculus for finance*. Springer Finance, 2004.

[19] P Wolfe. Convergence conditions for ascend methods. *SIAM Rev.*, 11(2):226–235, 1969.

[20] Y Zhang and J Yin. Modified alternative gradients algorithm for computing the nearest correlation matrix. *Internal paper of the Tongji University, Shanghai*.

11 Matlab-files

Here are the Matlab files listed used in this report.

11.1 Test Correlation

The function `test_cor(C)` is used to test is a matrix C is a valid correlation matrix.

```
function [ varargout ] = test_cor( C )
%TestCor Test if the matrix C is a valid correlation matrix
toll = 1e-14; % error tollerance
varargout{1} = false;

% test if M is square
if(size(C,1) ~= size(C,2))
    if(nargout == 0)
        disp('E: Matrix is not square')
    end
    return
end

% test for ones on the diagonal
if(find(abs(diag(C)-1)>toll,1))
    if(nargout == 0)
        disp('E: Not just ones on the diagonal')
    end
    return
end

% test for values between -1 and 1
if (find(C>1+toll))
    if(nargout == 0)
        disp('E: Matrix entries outside [-1,1]');
    end
    return
end
if (find(C<-1-toll))
    if(nargout == 0)
        disp('E: Matrix entries outside [-1,1]');
    end
    return
end

% test for symmetry
if(max(max(abs(C-C')))>=toll)
    if(nargout == 0)
        disp('E: Matrix is not symetric');
    end
    return
end

% test for positive definiteness
[~,p] = chol(C);
if(p>0)
    if(nargout == 0)
        disp('E: Matrix non positive definite!');
    end
    return
end

varargout{1} = true;
```

```
end
```

11.2 Generate Invalid Correlation Matrix

To generate a random invalid correlation matrix of size n we use

```
function [C] = generate_cor(varargin)
% This generates a random invalid correlation layout matrix
% Input matrix dimension, lower entry bound, upper entry bound
% Output C: invalid corr. matrix

    % Default parameters
    n = 25;
    lower = -1;
    upper = 1;

    % input parameters
    if(nargin > 0)      n = varargin{1};   end
    if(nargin > 1) lower = varargin{2};   end
    if(nargin > 2) upper = varargin{3};   end


    for i = 1:1000

        % sampleuniform from [lower,upper]
        C = lower + rand(n,n)*(upper-lower);

        % take only the upper triangular and
        % the diagonal 1's
        C = triu(C,1) + eye(n) + triu(C,1)';

        % repeat untill matrix is non positive definite
        [~,p] = chol(C);
        if(p ~= 0)
            return
        end
    end


end
```

11.3 Cholesky decomposition

Given a symmetric positive definite matrix A, we calculate the Cholesky factor M as in Section 3.2.

```
function [ M ] = cholesky( A )
%CHOLESKY computes the cholesky factor M of symmetric input matrix A
    n = length(A);
    M = zeros(n);

    % per row
    for i = 1:n

        % lower diagonal entries
        for j = 1:i-1
            M(i,j) = (A(i,j) - M(i,:)*M(j,:)')/M(j,j);
        end
```

```
        % diagonal entries
        M(i,i) = sqrt(A(i,i)-M(i,:)*M(i,:)');

        %check for positive definiteness
        if(imag(M(i,i)) ~= 0)
            M = false;
            return
        end
    end
end
```

11.4 Iterative Spectral Method

```
function [ output ] = posdef_iterative_spectral( varargin )
% Iterative Spectral Method, input invalid correlation matrix C

    %%%%%%%% Default Parameters
    a = 1e-3; % 1e-3 is in line with my report.

    %%%%%%%% Input parameters
    % Input at least a matrix M
    if(nargin <1)
        disp('Naive Method: Not enough input arguments.');
        disp('Input: C,[a]. Die now..');
        return
    end

    % This is our target matrix
    C = varargin{1};     % This is our target matrix
    n = length(C);       % matrix dimension
    Cold = C;            % save the old matrix for comparison

    %  accuary, set eigenvalues to this value.
    if(nargin >= 2 && varargin{2}>=0)
        a = varargin{2};
    end

    tic

    for i = 1:200

       % Step 5 - Set the diagonal elements of C to 1 and go to step 1
       C(eye(n)~=0) = 1;

       % Step 1 - Decompose C = SLS^T, where S is
       % the eigenspace corresponding to eigenvalue matrix L.
       [S,L] = eig(C);

       % Step 2 - If all eigenvalues are positive, then stop. Else, continue.
       if(diag(L)>0)
          break
       end

       % Step 3 - replace the negative eigenvalues of L with
       % small postive value a.
       L = diag(max(diag(L),a));

       % Step 4 - calculate C =  SLS^{T}$
       C = S*L*S';
```

```
    end

    % exeeded the number of iterations, quit.
    % disp('Exiting: Positive definiteness not reached');
    output.method = 'Naive';
    output.C = C;
    output.M = C;
    output.iterations = i-1;
    output.dist = norm(output.C-Cold,'fro');
    output.time = toc;
    output.a = a;
    output.valid = test_cor(output.C);
end
```

11.5 Alternating Projections Method

```
function [ output ] = posdef_alternating( varargin )
%POSDEF_ALTERNATE Alternating Projections by Higham

%%%%%%%% Default Parameters
    a = 1e-3; % 1e-3 is in line with my report.

    %%%%%%%% Input parameters
    % Input at least a matrix C
    if(nargin <1)
        disp('Scaled Spectral Method: Not enough input arguments.');
        disp('Input: C,[a]. Die now..');
        return
    end
    C = varargin{1};

    %  accuary, set eigenvalues to this value.
    if(nargin >= 2 && varargin{2}>=0)
        a = varargin{2};
    end

    tic

    n = length(C);
    deltaS = zeros(n);

    % starting values
    Cold = C;

    for k=1:200
        R = C - deltaS;

        % projection S
            [S,L] = eig(R);

            % set eigenvalues to positive ones
            L = diag(max(diag(L),a));

            % put C back together
            C = S*L*S';

        % calculate delta S
            deltaS = C - R;

        % projection U
            C(eye(n)~=0) = 1;

```

```
        % check if C is positive definite.
            [~,p] = chol(C);

        % break if posdef.
            if(p == 0)
                break;
            end
    end

    output.method = 'Alternating Projections';
    output.C = C;
    output.iterations = k;
    output.dist = norm(output.C-Cold,'fro');
    output.a = a;
    output.time = toc;
    output.valid = test_cor(output.C);
end
```

11.6 Scaled Spectral Decomposition

```
function [output] = posdef_scaled_spectral( varargin )
% Scaled Spectral Method

    %%%%%%%% Default parameters
    a = 1e-13;

    %%%%%%%% Input parameters
    % Input at least a matrix C
    if(nargin <1)
        disp('Scaled Spectral Method: Not enough input arguments.');
        disp('Input: C,[a]. Die now..');
        return
    end

    % This is our target matrix
    C = varargin{1};

    % starting position for eigenvalue value. a >= 0
    if(nargin >= 2 && varargin{2}>=0)
        a = varargin{2};
    end

    %%% Start Algoritme
    tic

    % Step 1
    [S,L] = eig(C);

    % Step 2
    L = diag(max(diag(L),a));

    % Step 3 ( including the right multiplication of S
    T = sqrt(diag(1./(S.^2*diag(L))))*S;

    % Output
    output.method = 'Scaled Spectral';

    % Step 4
    output.C = T*L*T';
    output.dist = norm(output.C-C,'fro');
    output.time = toc;
    output.a = a;
```

```
    output.valid = test_cor(output.C);

end
```

11.7 Gradient Updating Method, Starting Scaled

```
function [ output ] = posdef_gradient( varargin )
% Gradients Updating Algorithm

    %%%%%%%% Default parameters
    s = 0.5; % sigma, balance between speed and accuracy
    tol = 1e-2; % stopping criteria
    steps = 2:100; % number of iterations.

    %%%%%%%% Input parameters
    % Input at least a matrix C
    if(nargin <1)
        disp('Gradient Method: Not enough input arguments.');
        disp('Input: C,[s,tol,steps]. Die now..');
        return
    end

    % This is our target matrix
    C = varargin{1};

    %  speed vs. accuracy in gradient updating. s \in [0,1)
    if(nargin >= 2 && varargin{2}>=0 && varargin{2}<1)
        s = varargin{2};
    end

    % tolerance
    if(nargin >= 3 && varargin{3}>0)
        tol = varargin{3};
    end

    % maximum number of iterations. Start from 2!
    if(nargin >= 4 && varargin{4}>=1)
        steps = 2:(varargin{4}+1);
    end

    %%%%%%%% Algorithm %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%  this is where the algorithm starts
    tic

    n = length(C);

    % Choosing the input matrix B
    B = C;
    B(eye(n)~=0) = sqrt(n)/2;
    B = repmat(1./sqrt(sum(B.^2,2)),1,n).*B;
    A = B';

    % Matrix Distance.
    dist = zeros(length(steps),1);
    dist(1) = norm(B*A-C,'fro');


    mStart = 1;

    % Start Iteration
    for k = steps

```

```
        % step 1
        gradA = B'*(B*A-C);

        for m = mStart:5
            % step 2
            n1 = 2*(s^m)*norm(gradA,'fro')^2/(norm(B*gradA,'fro')^2);

            % step 3
            A2 = A - n1*gradA;

            % step 4
            gradB = (B*A2-C)*A2';

            % step 5
            n2 = 2*(s^m)*norm(gradB,'fro')^2/(norm(gradB*A2,'fro')^2);

            % step 6 (All transposed removed)
            B2 = B - n2*gradB;

            % step 7
            B2 = (B2+A2')/2;

            % step 8 & 9
            B2 = repmat(1./sqrt(sum(B2.^2,2)),1,n).*B2;

            % step 10
            dist(k) = norm(B2*B2'-C,'fro');

            % step 11
            if ( dist(k)-dist(k-1) < 0 )
                mStart = m;
                B = B2;
                A = B';
                break
            end
        end

        % step 12 - stopping criteria
        if(abs(dist(k)-dist(k-1))/dist(k-1) < tol )
            break
        end

    end

    % Output
    output.method = 'Gradient, Scaled';
    output.C = B*B';
    output.dist = norm(output.C-C,'fro');
    output.time = toc;
    output.iterations = k+1-steps(1);
    output.s = s;
    output.tol = tol;
    output.valid = test_cor(output.C);
end
```

11.8 Gradient Updating Method, Starting Cholesky

```
function [ output ] = posdef_gradient_chol( varargin )
% Gradients Updating Algorithm

    %%%%%%%% Default parameters
    s = 0.5; % sigma, balance between speed and accuracy
```

```
    tol = 1e-2; % stopping criteria
    steps = 2:100; % number of iterations.

    %%%%%%%% Input parameters
    % Input at least a matrix C
    if(nargin <1)
        disp('Gradient Method: Not enough input arguments.');
        disp('Input: C,[s,tol,steps]. Die now..');
        return
    end

    % This is our target matrix
    C = varargin{1};

    %  speed vs. accuracy in gradient updating. s \in [0,1)
    if(nargin >= 2 && varargin{2}>=0 && varargin{2}<1)
        s = varargin{2};
    end

    % tolerance
    if(nargin >= 3 && varargin{3}>0)
        tol = varargin{3};
    end

    % maximum number of iterations. Start from 2!
    if(nargin >= 4 && varargin{4}>=1)
        steps = 2:(varargin{4}+1);
    end

    %%%%%%%% Algorithm %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%  this is where the algorithm starts
    tic

    n = length(C);


    % Choleskly approach input matrix
    C = max(-0.99/(n-1),(sum(sum(M))-n)/(n^2-n))*ones(n,n);
    C(eye(n)~=0) = 1;
    C = chol(C);
    B = C';

    % Matrix Distance.
    dist = zeros(length(steps),1);
    dist(1) = norm(B*A-C,'fro');

    mStart = 1;

    % Start Iteration
    for k = steps

        % step 1
        gradA = B'*(B*A-C);

        for m = mStart:5
            % step 2
            n1 = 2*(s^m)*norm(gradA,'fro')^2/(norm(B*gradA,'fro')^2);

            % step 3
            A2 = A - n1*gradA;

            % step 4
```

```
            gradB = (B*A2-C)*A2';

            % step 5
            n2 = 2*(s^m)*norm(gradB,'fro')^2/(norm(gradB*A2,'fro')^2);

            % step 6 (All transposed removed)
            B2 = B - n2*gradB;

            % step 7
            B2 = (B2+A2')/2;

            % step 8 & 9
            B2 = repmat(1./sqrt(sum(B2.^2,2)),1,n).*B2;

            % step 10
            dist(k) = norm(B2*B2'-C,'fro');

            % step 11
            if ( dist(k)-dist(k-1) < 0 )
                mStart = m;
                B = B2;
                A = B';
                break
            end
        end

        % step 12 - stopping criteria
        if(abs(dist(k)-dist(k-1))/dist(k-1) < tol )
            break
        end

    end

    % Output
    output.method = 'Gradient, Cholesky';
    output.C = B*B';
    output.dist = norm(output.C-C,'fro');
    output.time = toc;
    output.iterations = k+1-steps(1);
    output.s = s;
    output.tol = tol;
    output.valid = test_cor(output.C);
end
```

11.9 Adjusted Gradient Updating Method, Starting Scaled

```
function [ output ] = posdef_adjusted_gradient(varargin)
% Adjusted Gradients Updating Algorithm

    %%%%%%%% Default parameters
    s = 0.5; % sigma, balance between speed and accuracy
    tol = 1e-2; % stopping criteria
    steps = 2:100; % number of iterations.

    %%%%%%%% Input parameters
    % Input at least a matrix C
    if(nargin <1)
        disp('Adjusted Gradient Method: Not enough input arguments.');
        disp('Input: C,[s,tol,steps]. Die now..');
        return
    end

```

```
% This is our target matrix
C = varargin{1};

%  speed vs. accuracy in gradient updating. s \in [0,1)
if(nargin >= 2 && varargin{2}>=0 && varargin{2}<1)
    s = varargin{2};
end

% tolerance
if(nargin >= 3 && varargin{3}>0)
    tol = varargin{3};
end

% maximum number of iterations. Start from 2!
if(nargin >= 4 && varargin{4}>=1)
    steps = 2:(varargin{4}+1);
end

%%%%%%%% Algorithm
% this is where the algorithm starts
tic

% also, define the non-zero elements of the diagonal scaling matrix T
% with respect to the eigen-system S.
n = length(C);

% Input matrix, Scaled
B = C;
B(eye(n)~=0) = sqrt(n)/2;
B = repmat(1./sqrt(sum(B.^2,2)),1,n).*B;
BB = B*B';

% r is the distance
r = zeros(length(steps),1);
r(1) = norm(BB-C,'fro');
mStart = 1;

% iterate
for k = steps

    % Step 1
    gradB = 2*(BB-C)*B;

    for m = mStart:10

        % Step 2
        theta = .5*(s^m)*(norm(gradB,'fro')/(norm(gradB*B','fro')))^2;

        % Step 3
        B2 = B - theta*gradB;

        % Step 4 and 5
        B2 = repmat(1./sqrt(sum(B2.^2,2)),1,n).*B2;
        BB = B2*B2';

        % Step 6.
        r(k) = norm(BB-C,'fro');

        % Step 7.
        if ( r(k)-r(k-1) < 0 )
            mStart = m;
            B = B2;
```

```
                break
            end
        end

        % Step 8.
        if(abs(r(k)-r(k-1))/r(k) < tol )
            break;
        end

    end

    output.method = 'Gradient Fast Scaled';
    output.C = B*B';
    output.dist = norm(output.C-C,'fro');
    output.time = toc;
    output.iterations = k+1-steps(1);
    output.s = s;
    output.tol = tol;
    output.valid = test_cor(output.C);
end
```

11.10 Adjusted Gradient Updating Method, Starting Cholesky

```
function [ output ] = posdef_adjusted_gradient_chol(varargin)
% Adjusted Gradients Updating Algorithm

    %%%%%%%% Default parameters
    s = 0.5; % sigma, balance between speed and accuracy
    tol = 1e-2; % stopping criteria
    steps = 2:100; % number of iterations.

    %%%%%%%% Input parameters
    % Input at least a matrix C
    if(nargin <1)
        disp('Adjusted Gradient Method: Not enough input arguments.');
        disp('Input: C,[s,tol,steps]. Die now..');
        return
    end

    % This is our target matrix
    C = varargin{1};

    %  speed vs. accuracy in gradient updating. s \in [0,1)
    if(nargin >= 2 && varargin{2}>=0 && varargin{2}<1)
        s = varargin{2};
    end

    % tolerance
    if(nargin >= 3 && varargin{3}>0)
        tol = varargin{3};
    end

    % maximum number of iterations. Start from 2!
    if(nargin >= 4 && varargin{4}>=1)
        steps = 2:(varargin{4}+1);
    end

    %%%%%%%% Algorithm
    % this is where the algorithm starts
    tic

    % also, define the non-zero elements of the diagonal scaling matrix T
```

```
    % with respect to the eigen-system S.
    n = length(C);

    % Input matrix, Cholesky
    B = max(-0.99/(n-1),(sum(sum(M))-n)/(n^2-n))*ones(n,n);
    B(eye(n)~=0) = 1;
    B = chol(B)';
    BB = B*B';

    % r is the distance
    r = zeros(length(steps),1);
    r(1) = norm(BB-C,'fro');
    mStart = 1;

    % iterate
    for k = steps

        % Step 1
        gradB = 2*(BB-C)*B;

        for m = mStart:10

            % Step 2
            theta = .5*(s^m)*(norm(gradB,'fro')/(norm(gradB*B','fro')))^2;

            % Step 3
            B2 = B - theta*gradB;

            % Step 4 and 5
            B2 = repmat(1./sqrt(sum(B2.^2,2)),1,n).*B2;
            BB = B2*B2';

            % Step 6.
            r(k) = norm(BB-C,'fro');

            % Step 7.
            if ( r(k)-r(k-1) < 0 )
                mStart = m;
                B = B2;
                break
            end
        end

        % Step 8.
        if(abs(r(k)-r(k-1))/r(k) < tol )
           break;
        end

    end

    output.method = 'Gradient Fast Cholesky';
    output.C = B*B';
    output.dist = norm(output.C-C,'fro');
    output.time = toc;
    output.iterations = k+1-steps(1);
    output.s = s;
    output.tol = tol;
    output.valid = test_cor(output.C);
end
```

11.11 Iterative Spectral Weighted

```
function [ output ] = posdef_iterative_spectral_weighted( varargin )
% The iterative spectral method, weighted version

    %%%%%%% Default Parameters
    a = 1e-3; % 1e-3 is in line with my report.

    %%%%%%% Input parameters
    % Input at least a matrix C
    if(nargin <1)
        disp('Naive Method: Not enough input arguments.');
        disp('Input: C,[a,W]. Die now..');
        return
    end

    % This is our target matrix
    C = varargin{1};    % This is our target matrix
    n = length(C);      % matrix dimension
    Cold = C;           % save the old matrix for comparison

    %  accuary, set eigenvalues to this value.
    if(nargin >= 2 && varargin{2}>=0)
        a = varargin{2};
    end

    % Weights
    W = eye(n);
    if(nargin >= 3 & size(varargin{3}) == size(M))
        W = varargin{3};
    end

    tic

    for i = 1:400

       % Step 5 - Set the diagonal elements of C to 1 and go to step 1
       C(eye(n)~=0) = 1;


       % Step 1 - Decompose C = SLS^T, where S is
       % the eigenspace corresponding to eigenvalue matrix L.
       [S,L] = eig(diag(sqrt(diag(W)))*C*diag(sqrt(diag(W))));

       % Step 2 - If all eigenvalues are positive, then stop. Else, continue.
       if(diag(L)>0)
          break
       end

       % Step 3 - replace the negative eigenvalues of L with
       % small postive value a.
       L = diag(max(diag(L),a));

       % Step 4 - calculate C =  SLS^{T}$
       C = diag(sqrt(1./diag(W)))*S*L*S'*diag(sqrt(1./diag(W)));

    end

    % exeeded the number of iterations, quit.
    % disp('Exiting: Positive definiteness not reached');
    output.method = 'Naive weights';
    output.C = C;
    output.iterations = i-1;
    output.dist = norm(output.C-Cold,'fro');
```

```
    output.weightsDist = norm(diag(sqrt(diag(W)))*(output.C-Cold)*diag(sqrt(diag(W
        ))),'fro');
    output.time = toc;
    output.a = a;
    output.W = W;
    output.valid = test_cor(output.C);
end
```

11.12 Scaled Spectral Weighted

```
function [output] = posdef_scaled_spectral_weighted( varargin )
% Scaled Spectral Method, Weighted

    %%%%%%% Default parameters
    a = 1e-13;

    %%%%%%% Input parameters
    % Input at least a matrix C
    if(nargin <1)
        disp('Scaled Spectral Method, Weighted: Not enough input arguments.');
        disp('Input: C,[a]. Die now..');
        return
    end

    % This is our target matrix
    C = varargin{1};

    % starting position for eigenvalue value. a >= 0
    if(nargin >= 2 && varargin{2}>=0)
        a = varargin{2};
    end

    % Weights
    W = eye(length(C));
    if(nargin >= 3 & size(varargin{3}) == size(C))
        W = varargin{3};
    end

    %%% Start Algoritme
    tic

    % Step 1
    [S,L] = eig(diag(sqrt(diag(W)))*C*diag(sqrt(diag(W))));

    % Step 2
    L = diag(max(diag(L),a));

    % Step 3
    output.C = diag(1./sqrt(diag(W)))*S*L*S'*diag(1./sqrt(diag(W)));
    T = sqrt(diag(1./(diag(output.C))));

    output.method = 'Scaled Spectral, Weighted';

    % Step 4
    output.C = T*(output.C)*T';
    output.dist = norm(output.C-C,'fro');
    output.weightsDist = norm(diag(sqrt(diag(W)))*(output.C-C)*diag(sqrt(diag(W)))
        ,'fro');
    output.time = toc;
    output.a = a;
    output.valid = test_cor(output.C);

```

```
end
```

11.13 Hypersphere Decomposition

To compose a matrix from a coordinate vector x, we use

```
function [ C ] = tap( x )
%TAP Compose matrix M from coordinate vector x.

    % matrix dimensions
    n = (sqrt(8*length(x)+1)+1)/2;

    % matrix B:
    B = zeros(n);

    % Loop over entries to calculate B.
    for i = 1:n
        for j = 1:(i-1)
            B(i,j) = x(((i-1)*(i-2)/2)+j)/sqrt(prod(x(((i-1)*(i-2)/2)+1:((i-1)*(i
                -2)/2)+j).^2+1));
        end
        j = i;
        B(i,j) = 1/sqrt(prod(x(((i-1)*(i-2)/2)+1:((i-1)*(i-2)/2)+j-1).^2+1));
    end
    C = B*B';
end
```

To decompose a positive definite matrix to a coordinate vector x, we use

```
function [ x ] = arctap( M )
%ARCTAP Compose coordinates x from matrix.

    b = chol(M)';

    % our parameter vector x:
    n = size(M,2);
    x = zeros(1,n*(n-1)/2);

    for i = 2:n

        j = 1;
        x((i-1)*(i-2)/2+j) = b(i,j)/sqrt(1-b(i,j)^2);

        for j = 2:i-1
            % x(i,j)  = b(i,j)/sqrt(sum(b(i,j+1:n).^2));
            x((i-1)*(i-2)/2+j) = b(i,j)/sqrt(sum(b(i,j+1:n).^2));
        end
    end
end
```

11.14 Positive Definite Check by Vines

```
function [varargout] = test_cor_vine(M)
% Test if a matrix is positive definite using vines

    % Use the vine method as described by Kurowicka and Cooke in
    % theorem 4.1 and 4.2
    toll = 1e-14; % error tollerance
    varargout{1} = false;

    % test if M is square
```

```
    if(size(M,1) ~= size(M,2))
        if(nargout == 0)
            disp('E: Matrix is not square')
        end
        return
    end

    % test for ones on the diagonal
    if(find(abs(diag(M)-1)>toll,1))
        if(nargout == 0)
            disp('E: Not just ones on the diagonal')
        end
        return
    end

    % test for values between -1 and 1
    if (find(M>1+toll))
        if(nargout == 0)
            disp('E: Matrix entries outside [-1,1]');
        end
        return
    end

    if (find(M<-1-toll))
        if(nargout == 0)
            disp('E: Matrix entries outside [-1,1]');
        end
        return
    end


    % test for symmetry
    if(max(max(abs(M-M')))>=toll)
        if(nargout == 0)
            disp('E: Matrix is not symetric')
        end
        return
    end

    % Test for positive definiteness
    % Create a cell array
    A = cell(1,length(M)-2);
    A{1} = M;

    % Define matrix parts X,Y,Z.
    for i = 1:(length(M)-2)

        % Actually, we call this A_k instead of M
        Y = A{i}(1,2:end);
        Z = A{i}(2:end,2:end);

        %then, this is tilde(A), which is equal to A after scaling.
        A{i+1} = Z-Y'*Y;
        D = diag(1./sqrt(diag(A{i+1})));
        A{i+1} = D*A{i+1}*D;

        if ~isempty(find([triu(A{i+1},1)>1 triu(A{i+1},1)<-1], 1))
            if(nargout == 0)
                disp('E: Matrix not positive definite!');
            end
            return
        end
```

```
    end

    varargout{1} = 1;
end
```

11.15 Vines Method

```
function [ output ] = posdef_vines( varargin )

    %%%%%%% Default parameters
    alpha = 0.9;

    %%%%%%% Input parameters
    % Input at least a matrix C
    if(nargin <1)
        disp('Vines Method: Not enough input arguments.');
        disp('Input: C,[alpha]. Die now..');
        return
    end

    % This is our target matrix
    C = varargin{1};

    %  speed vs. accuracy in gradient updating. s \in [0,1)
    if(nargin >= 2 && varargin{2}>=0)
       alpha = varargin{2};
    end

    %%% Begin Algorithm
    tic

    % Create a cell array
    n = length(C);
    A = cell(1,n-2);
    A{1} = C;

    % count the number of calculations of A;
    counter = 0;

    % walk from 1 to n-2
    % shift index by 1, so wlk from 2 to n-1
    for i = 2:(n-1)

        % Actually, we call this A_k instead of C
        Y = A{i-1}(1,2:end);
        Z = A{i-1}(2:end,2:end);

        %then, this is tilde(A), which is equal to A after scaling.
        counter = counter + 1;
        A{i} = Z-Y'*Y;
        D = diag(1./sqrt(diag(A{i})));
        A{i} = D*A{i}*D;

        % check for values larger then 1.
        % but do not check the diagonal
        V = find(triu(A{i},1)>0.90 | triu(A{i},1)<-0.90 | tril(A{i},-1)>0.90 |
            tril(A{i},-1)<-0.9);

        % if we have values outside (-1,1)
        % then
        if ~isempty(V)
```

```
            % change them to + or - 0.9;
            A{i}(V) = sign(A{i}(V))*alpha;

            % change the preceding values
            for j = i:-1:2

                counter = counter +1;
                D = sqrt(diag(diag(A{j-1}(2:end,2:end)-A{j-1}(1,2:end)'*A{j
                    -1}(1,2:end))));
                A{j-1}(2:end,2:end) = D*A{j}*D + A{j-1}(1,2:end)' * A{j-1}(1,2:end
                    );
            end

        end

    end

    output.method = 'Vines';
    output.C = A{1};
    output.p = A;
    output.time = toc;
    output.valid = test_cor(output.C);
    output.dist = norm(output.C-C,'fro');
    output.alpha = alpha;
    output.iterations = counter;
end
```

11.16 Vines Method, Weighted

```
function [ output ] = posdef_vines_weighted( varargin )

    %%%%%%%% Default parameters
    alpha = 0.9;

    %%%%%%%% Input parameters
    % Input at least a matrix C
    if(nargin <1)
        disp('Vines Method: Not enough input arguments.');
        disp('Input: C,[alpha]. Die now..');
        return
    end

    % This is our target matrix
    C = varargin{1};

    %  speed vs. accuracy in gradient updating. s \in [0,1)
    if(nargin >= 2 && varargin{2}>=0)
       alpha = varargin{2};
    end

    % Weights
    W = eye(length(C));
    if(nargin >= 3 & size(varargin{3}) == size(C))
        W = varargin{3};
    end

    %%% Begin Algorithm
    tic

    % Create a cell array
    n = length(C);
```

```
A = cell(1,n-2);

% Sort M
[~,Order] = sort(diag(W),'descend')
A{1} = C(:,Order);
A{1} = A{1}(Order,:);

% walk from 1 to n-2
% shift index by 1, so wlk from 2 to n-1

% count the number of calculations of A;
counter = 0;

for i = 2:(n-1)

    % Actually, we call this A_k instead of M
    Y = A{i-1}(1,2:end);
    Z = A{i-1}(2:end,2:end);

    %then, this is tilde(A), which is equal to A after scaling.

    counter = counter + 1;
    A{i} = Z-Y'*Y;
    D = diag(1./sqrt(diag(A{i})));
    A{i} = D*A{i}*D;

    % check for values larger then 1.
    % but do not check the diagonal
    V = find(triu(A{i},1)>0.90 | triu(A{i},1)<-0.90 | tril(A{i},-1)>0.90 |
        tril(A{i},-1)<-0.9);

    % if we have values outside (-1,1)
    % then
    if ~isempty(V)

        % change them to + or - 0.9;
        A{i}(V) = sign(A{i}(V))*alpha;

        % change the preceding values
        for j = i:-1:2

            counter = counter +1;
            D = sqrt(diag(diag(A{j-1}(2:end,2:end)-A{j-1}(1,2:end)'*A{j
                -1}(1,2:end))));
            A{j-1}(2:end,2:end) = D*A{j}*D + A{j-1}(1,2:end)' * A{j-1}(1,2:end
                );
        end

    end

end

output.method = 'Vines Weighted';
output.C = A{1};

% reorden M back
[~,Order] = sort(Order);
output.C = output.C(:,Order);
output.C = output.C(Order,:);

output.p = A;
```

```
    output.time = toc;
    output.valid = test_cor(output.C);
    output.dist = norm(output.C-C,'fro');
    output.weightsDist = norm(diag(sqrt(diag(W)))*(output.C-C)*diag(sqrt(diag(W)))
        ,'fro');
95  output.alpha = alpha;
    output.iterations = counter;
end
```