

Rank Reduction of Correlation Matrices by Majorization¹

Raoul Pietersz², Patrick J. F. Groenen³

Econometric Institute Report EI 2004-11

18 March 2004

Abstract. In this paper a novel method is developed for the problem of finding a low-rank correlation matrix nearest to a given correlation matrix. The method is based on majorization and therefore it is globally convergent. The method is computationally efficient, is straightforward to implement, and can handle arbitrary weights on the entries of the correlation matrix. A simulation study suggests that majorization compares favourably with competing approaches in terms of the quality of the solution within a fixed computational time. The problem of rank reduction of correlation matrices occurs when pricing a derivative dependent on a large number of assets, where the asset prices are modelled as correlated log-normal processes.

Key words: rank, correlation matrix, majorization, lognormal price processes

JEL Classification: G13

¹We are grateful for the constructive comments of Antoon Pelsser that have helped to improve the paper.

²Erasmus Research Institute of Management, Erasmus University Rotterdam, P.O. Box 1738, 3000 DR Rotterdam, The Netherlands (e-mail: pietersz@few.eur.nl) and Market Risk – Modelling and Product Analysis (HQ1059), ABN AMRO Bank, P.O. Box 283, 1000 EA Amsterdam, The Netherlands

³Econometric Institute, Erasmus University Rotterdam, P.O. Box 1738, 3000 DR Rotterdam, The Netherlands (e-mail: groenen@few.eur.nl)

1 Introduction

In this paper, we study the problem of finding a low-rank correlation matrix nearest to a given (correlation) matrix. First we explain how this problem occurs in an interest rate derivatives pricing setting. We will focus on interest rate derivatives that depend on several rates such as the 1 year LIBOR deposit rate, the 2 year swap rate, etc. An example of such a derivative is a Bermudan swaption. A Bermudan swaption gives its holder the right to enter into a fixed maturity interest rate swap at certain exercise dates. At an exercise opportunity, the holder has to choose between exercising then or hold the option with the chance of entering into the swap later at more favourable interest rates. Evidently, the value depends not only on the current available swap rate but, amongst others, also on the forward swap rates corresponding to future exercise dates. In contrast, an example of a derivative that is dependent on a single interest rate is a caplet, which can be viewed as a call option on LIBOR. In this case, the value of the caplet depends only on a single forward LIBOR rate.

To reiterate, we will focus on derivatives depending on several rates. Our discussion can however be applied, if desired, to the situation of a derivative depending on several assets. To this extend a model is set up that specifies the behaviour of the asset prices. Each of the asset prices is modelled as a log-normal martingale under its respective forward measure. Additionally, the asset prices are correlated. Suppose we model n correlated log-normal price processes,

$$(1) \quad \frac{ds_i}{s_i} = \dots dt + \sigma_i d\tilde{w}_i, \quad \langle d\tilde{w}_i, d\tilde{w}_j \rangle = r_{ij},$$

under a single measure. Here s_i denotes the price of the i^{th} asset, σ_i its volatility and \tilde{w}_i denotes the associated driving Brownian motion. Brownian motions i and j are correlated with coefficient r_{ij} , the correlation coefficient between the returns on assets i and j . The matrix $\mathbf{R} = (r_{ij})_{ij}$ should be positive semidefinite and should have a unit diagonal. In other words R should be a true correlation matrix. The term $\dots dt$ denotes the drift term that stems from the change of measure under the non-arbitrage condition. The models that fit into the framework of (1) and which are most relevant to our discussion are the LIBOR and swap market models for valuation of interest rate derivatives. These models were developed by Brace, Gątarek & Musiela (1997), Jamshidian (1997) and Miltersen, Sandmann & Sondermann (1997). In this case, an asset price corresponds to a forward LIBOR or swap rate. For example, if we model a 30 year Bermudan swaption with annual call and payment dates, then our model would consist of 30 annual forward

LIBOR rates or 30 co-terminal forward swap rates. In the latter case, this means that we consider 30 forward starting annual-paying swaps, starting at each of the 30 exercise opportunities and all ending after 30 years. The discussion is however not restricted to the interest rates case and may, for example, also be applied to a derivative depending on a number of stocks, if desirable.

Given the model specified by (1) the price of any derivative depending on the assets can be calculated by non-arbitrage arguments. Because the number of assets is assumed to be high and the derivative is assumed complex in this exposition, the derivative value can be calculated only by Monte Carlo simulation. To implement scheme (1) by Monte Carlo we need a decomposition $\mathbf{R} = \mathbf{X}\mathbf{X}^T$, with \mathbf{X} an $n \times n$ matrix. In other words, if we denote the i^{th} row vector of \mathbf{X} by \mathbf{x}_i , then the decomposition reads $\langle \mathbf{x}_i, \mathbf{x}_j \rangle = r_{ij}$, where $\langle \cdot, \cdot \rangle$ denotes the scalar product. We then implement the scheme

$$(2) \quad \frac{ds_i}{s_i} = \sigma_i \{ x_{i1}dw_1 + \cdots + x_{in}dw_n \}, \quad \langle \mathbf{x}_i, \mathbf{x}_j \rangle = r_{ij},$$

where the w_i are now independent Brownian motions. Scheme (2) indeed corresponds to scheme (1) since both volatility and correlation are implemented correctly. The instantaneous variance is $\langle ds_i/s_i \rangle = \sigma_i^2 dt$ since $\|\mathbf{x}_i\| = r_{ii} = 1$ – and volatility is the square root of instantaneous variance divided by dt . Moreover, for the instantaneous correlation we have $\langle ds_i/s_i, ds_j/s_j \rangle = \langle \mathbf{x}_i, \mathbf{x}_j \rangle dt = r_{ij} dt$.

For large (financial) correlation matrices, usually almost all variance (say 99%) can be attributed to only 3–6 stochastic Brownian factors. Therefore, (2) contains a large number of almost redundant Brownian motions that cost expensive computational time to simulate. Instead of taking into account all Brownian motions, we would wish to simulate with a smaller number of factors $d < n$ say, typically between 2 and 6. The scheme then becomes

$$\frac{ds_i}{s_i} = \sigma_i \{ x_{i1}dw_1 + \cdots + x_{id}dw_d \}, \quad \langle \mathbf{x}_i, \mathbf{x}_j \rangle = r_{ij}.$$

The $n \times d$ matrix \mathbf{X} is a decomposition of \mathbf{R} . This approach immediately implies that the rank of \mathbf{R} be less than or equal to d . For financial correlation matrices, this rank restriction is generally not satisfied. It follows that an approximation be required. We could proceed in two possible ways. The first way involves approximating the covariance matrix $(\sigma_i \sigma_j r_{ij})_{ij}$. The second involves approximating the correlation matrix while maintaining an exact fit to the volatilities. In a derivatives pricing setting, usually the volatilities are well-known. These can be calculated via a Black-type formula from

the European option prices quoted in the market – or indeed mostly these volatilities are directly quoted in the market. The correlation is usually less known and can be obtained in two ways. First, it can be estimated from historical time series. Second, it can be implied from correlation sensitive options traded in the market such as spread options. A spread option is an option on the difference between two rates or asset prices. Such correlation sensitive products are not traded as liquidly as the European plain-vanilla options. Consequently, in both cases of historic or market-implied correlation, we are more confident of the volatilities. For that reason in a derivative pricing setting we approximate the correlation matrix rather than the covariance matrix.

We are led to solve the following problem:

$$(3) \quad \begin{aligned} & \text{Find } \mathbf{X} \in \mathbb{R}^{n \times d}, \\ & \text{to minimize } f(\mathbf{X}) := \frac{1}{c} \sum_{i < j} w_{ij} (r_{ij} - \langle \mathbf{x}_i, \mathbf{x}_j \rangle)^2, \\ & \text{subject to } \|\mathbf{x}_i\|_2 = 1, \quad i = 1, \dots, n. \end{aligned}$$

Here w_{ij} are nonnegative weights and $c := 4 \sum_{i < j} w_{ij}$. The objective value f is scaled by the constant c in order to make it independent of the problem dimension n . Because each term $r_{ij} - \langle \mathbf{x}_i, \mathbf{x}_j \rangle$ is always between 0 and 2, it follows for the choice of c that f is always between 0 and 1. The weights w_{ij} have been added for three reasons:

- For squared differences, a large difference will be weighted more than a small difference. The weights can then be appropriately changed to adjust for this.
- Financial reasons may sometimes compel us to assign higher weights to particular correlation pairs. For example, we could be more confident about the correlation between the 1 and 2 year swap rates than about the correlation between the 8 and 27 year swap rates.
- The objective function with weights has been considered before in the literature. See for example Rebonato (1999*b*, Section 10) or Rebonato (2002, Section 9), also for an excellent discussion of the pros and cons of using weights.

The simplest case of f is $f(\mathbf{X}) := c^{-1} \|\mathbf{R} - \mathbf{X}\mathbf{X}^T\|_F^2$, where $\|\cdot\|_F$ denotes the Frobenius norm, $\|\mathbf{Y}\|_F^2 := \text{tr}(\mathbf{Y}\mathbf{Y}^T)$ for matrices \mathbf{Y} . This objective function (which we shall also call ‘Frobenius norm’) fits in the framework of (3) – it corresponds to the case of all weights equal. The objective function in (3) will be referred to as ‘general weights’.

In the literature, there exist four methods for minimizing f defined in (3). These methods will be outlined in the next Section and will be shown to have several disadvantages, namely none of the methods is simultaneously

- (i) efficient,
- (ii) straightforward to implement,
- (iii) able to handle general weights and
- (iv) guaranteed to converge to a local minimum.

In this paper, we develop a novel method to minimize f that simultaneously has the four mentioned properties. The method is based on iterative majorization, which has the important property that the algorithm is guaranteed to converge to a stationary point. The algorithm is straightforward to implement. We show that the method can efficiently handle general weights. We investigate empirically the efficiency of majorization in comparison to other methods in the literature.

We end the introduction with an outline of the remainder of the paper. First, we provide an overview of the methods available in the literature. Second, the idea of majorization is introduced and the majorizing functions are derived for the particular case at hand. Third, an algorithm based on majorization is given along with reference to associated MATLAB code. It is also shown that the algorithm converges to a stationary point starting from any initial point. Fourth, empirical results are presented for certain numerical cases. The paper ends with conclusions.

2 Literature Review

Four methods available in the literature for minimizing f are described hereafter. It is indicated whether a method can handle general weights. If not, then the most general objective function it can handle stems from the weighted Frobenius norm $\|\cdot\|_{F,\Omega}$ with Ω a symmetric positive definite matrix. Here $\|\mathbf{X}\|_{F,\Omega}^2 := \text{tr}(\mathbf{X}\Omega\mathbf{X}^T\Omega)$. The objective function $f(\mathbf{X}) := c^{-1}\|\mathbf{R} - \mathbf{X}\mathbf{X}^T\|_{F,\Omega}$ will be referred to as ‘weighted Frobenius norm’ as well.

First, we mention the ‘modified principal component analysis (PCA)’ method. For ease of exposition we restrict to the case of the Frobenius norm, however the method can be applied to the weighted Frobenius norm as well though not for general weights. Modified PCA is based on an eigenvalue decomposition $\mathbf{R} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$, with \mathbf{Q} orthonormal and $\mathbf{\Lambda}$ the diagonal matrix with eigenvalues. If the eigenvalues are ordered descending then a low-rank

decomposition with associated approximated matrix close to the original matrix is found by

$$(4) \quad \{\mathbf{X}_{\text{PCA}}\}_i = \frac{\mathbf{z}}{\|\mathbf{z}\|_2}, \quad \mathbf{z} := \{\mathbf{Q}_d \mathbf{\Lambda}_d^{1/2}\}_i, \quad i = 1, \dots, n.$$

Here $\{\mathbf{Y}\}_i$ denotes the i^{th} row of a matrix \mathbf{Y} , \mathbf{Q}_d denotes the first d columns of \mathbf{Q} and $\mathbf{\Lambda}_d$ denotes the principal sub-matrix of $\mathbf{\Lambda}$ of degree d . Ordinary PCA stops with \mathbf{z} and it is the scaling that is the ‘modified’ part. It ensures that the resulting correlation matrices have unit diagonal. Modified PCA is popular among financial practitioners and implemented in numerous financial institutions. The modification of PCA in this way is believed to be due to Flury (1988). For a description in a finance related article, see for example Hull & White (2000). Modified PCA is easiest to implement, because almost all that is required is an eigenvalue decomposition. The calculation is almost instant, and the approximation is reasonably accurate. A strong drawback of modified PCA is its non-optimality: Generally one may find decompositions \mathbf{X} (even locally) for which the associated approximated correlation matrix is closer to the original matrix than the PCA-approximated correlation matrix. The modified PCA approximation becomes worse when the magnitude of the left out eigenvalues increases.

Second, we mention the geometric programming approach of Grubišić & Pietersz (2004). Here the constraint set is equipped with a differentiable structure. Subsequently geometric programming is applied, which can be seen as Newton-Rhapson or conjugate gradient over curved space. By formulating these algorithms entirely in terms of differential geometric means, a simple expression is obtained for the gradient. The latter allows for an efficient implementation. Until now the geometric programming approach has been shown empirically (Grubišić & Pietersz 2004, Section 6) to be the most efficient algorithm for finding the nearest low-rank correlation matrix. This was in a particular numerical setting with a large number of randomly generated correlation matrices. Another advantage of geometric programming is that it can handle general weights. The drawback of the geometric programming approach is however that it is rather involved and costly to implement.

Third, we mention the Lagrange multiplier method developed by Zhang & Wu (2003) and Wu (2003). This method theoretically has the potential of guaranteed convergence to the global minimum. The currently available associated algorithm however suffers from non-convergence. In practice, this algorithm performs not better and even worse than the geometric programming approach of Grubišić & Pietersz (2004). Another drawback is that only the weighted Frobenius norm can be handled and not general weights.

Fourth, we mention the ‘parametrization method’ of Rebonato (1999*a*), Rebonato (1999*b*, Section 10), Brigo (2002), Rapisarda, Mercurio & Brigo (2002) and Rebonato (2002, Section 9). In this method, each row vector of the $n \times d$ configuration matrix \mathbf{X} is parameterized by spherical coordinates. Subsequently, non-linear programming algorithms such as Newton-Rhaphson or conjugate gradient are applied on the ‘parameter’ or ‘angle’ space. In essence, this approach is the same as the geometric programming approach, bar the fundamental difference in the choice of coordinates. The parametrization by spherical coordinates implies that the objective function is given in terms of trigonometric sin and cos functions. In turn, these yield a computational burden when calculating the derivative, which hinders an efficient implementation. Grubišić & Pietersz (2004, Section 6) have shown empirically for a particular numerical setting with many randomly generated correlation matrices that the parametrization method is numerically less efficient than either the geometric programming approach or the Lagrange multiplier approach. The parametrization approach can handle general weights.

It should be noted that throughout this article we choose the starting point of any method (beyond modified PCA) to be the modified PCA solution.

3 Majorization

In this section, we briefly describe the idea of majorization and apply majorization to the objective function f of Problem (3). The idea of majorization has been described, amongst others, in De Leeuw & Heiser (1977), Kiers & Groenen (1996) and Kiers (2002). We follow here the lines of Borg & Groenen (1997, Section 8.4). The key to majorization is to, at a certain point \mathbf{y} , find a simpler function that has the same function value at \mathbf{y} and anywhere else is larger than or equal to the objective function to be minimized. Such a function is called a *majorization function*. By minimizing the majorization function – which is an easier task since this function is ‘simpler’ – we obtain the next point of the algorithm. This procedure guarantees that the function value never increases along points generated by the algorithm. Moreover, if the objective and majorization functions are once continuously differentiable (which turns out to hold in our case), then the properties above imply that the gradients should match at the point of outset \mathbf{y} . As a consequence, from any point where the gradient of the objective function is non-negligible, iterative majorization will be able to find a next point with a *strictly* smaller objective function value. This condition ensures the property of global convergence to a stationary point along the lines of Zangwill (1969).

We formalize the procedure somewhat more. Suppose given a function $f(\cdot)$. Let for each \mathbf{y} in the domain of f be given a majorization function $g(\cdot, \mathbf{y})$ such that

- (i) $f(\mathbf{x}) = g(\mathbf{x}, \mathbf{x})$,
- (ii) $f(\mathbf{x}) \leq g(\mathbf{x}, \mathbf{y})$ for all \mathbf{x} and
- (iii) the function $g(\cdot, \mathbf{y})$ is ‘simple’, that is, it is straightforward to calculate the minimum of $g(\cdot, \mathbf{y})$.

A majorization algorithm is then given by

- (i) Start at $\mathbf{x}^{(0)}$. Set $k := 0$.
- (ii) Set $\mathbf{x}^{(k+1)}$ equal to the minimum argument of the function $g(\cdot, \mathbf{x}^{(k)})$.
- (iii) If $f(\mathbf{x}^{(k)}) - f(\mathbf{x}^{(k+1)}) < \varepsilon$ then stop with $\mathbf{x} := \mathbf{x}^{(k+1)}$.
- (iv) Set $k := k + 1$ and repeat from (ii).

Two iterations of this algorithm have been illustrated in Figure 1.

Below we derive the majorizing function for $f(\cdot)$ in (3). The first step is to majorize $f(\mathbf{X})$ as a function of the i^{th} row only and then to repeat this for each row. To formalize the notion of ‘ $f(\mathbf{X})$ as a function of the i^{th} row only’ we introduce the notation $f_i(\mathbf{x}; \mathbf{X})$ to denote the function

$$f_i(\cdot, \mathbf{X}) : \mathbf{x} \mapsto f(\hat{\mathbf{X}}_i(\mathbf{x})),$$

for (column)vectors $\mathbf{x} \in \mathbb{R}^d$ with $\hat{\mathbf{X}}_i(\mathbf{x})$ denoting the matrix \mathbf{X} with the i^{th} row replaced by \mathbf{x}^T . Note that we interpret \mathbf{X} as $[\mathbf{x}_1^T \cdots \mathbf{x}_n^T]^T$. We find

$$\begin{aligned} f(\mathbf{X}) &= \frac{1}{c} \sum_{j_1 < j_2} w_{j_1 j_2} (r_{j_1 j_2} - \langle \mathbf{x}_{j_1}, \mathbf{x}_{j_2} \rangle)^2 \\ &= \frac{1}{c} \sum_{j_1 < j_2} w_{j_1 j_2} (r_{j_1 j_2}^2 + (\mathbf{x}_{j_1}^T \mathbf{x}_{j_2})^2 - 2\mathbf{x}_{j_1}^T \mathbf{x}_{j_2}) \\ (5) \quad &= (\text{const in } \mathbf{x}_i) + \frac{1}{c} \left\{ \underbrace{\mathbf{x}_i^T \left[\sum_{j:j \neq i} w_{ij} \mathbf{x}_j \mathbf{x}_j^T \right] \mathbf{x}_i}_{(I)} - 2\mathbf{x}_i^T \underbrace{\left[\sum_{j:j \neq i} w_{ij} r_{ij} \mathbf{x}_j \right]}_{(II)} \right\}. \end{aligned}$$

Part (I) is quadratic in \mathbf{x}_i whereas part (II) is linear in \mathbf{x}_i ; the remaining term is constant in \mathbf{x}_i . We only have to majorize Part (I), as follows. Define $\mathbf{B}_i(\mathbf{X}) := \sum_{j:j \neq i} w_{ij} \mathbf{x}_j \mathbf{x}_j^T$. For notational convenience, we shall denote $\mathbf{B}_i(\mathbf{X})$

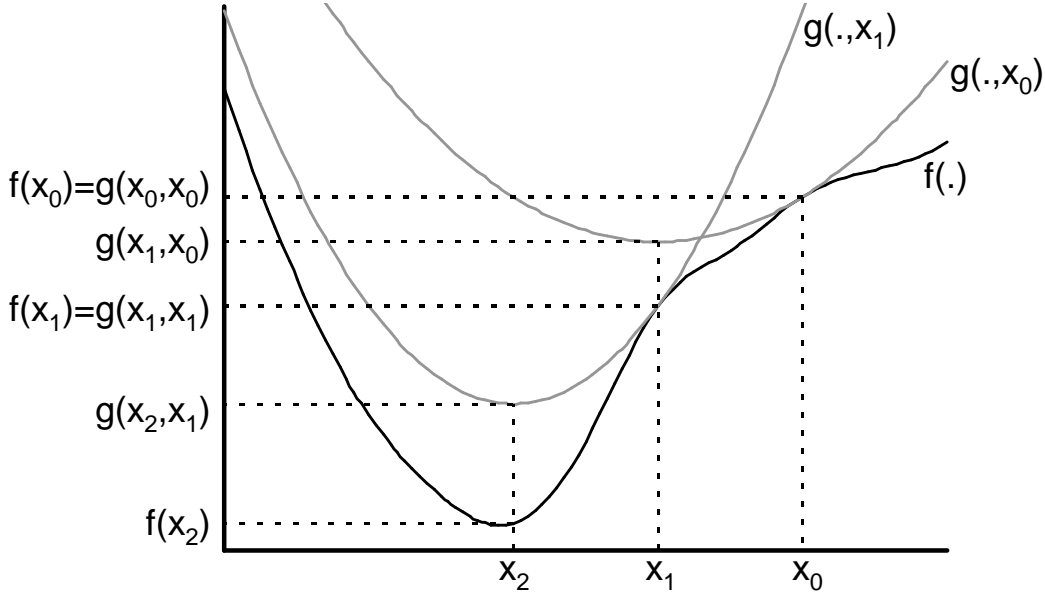


Figure 1: The idea of majorization. (Figure adopted from Borg & Groenen (1997, Figure 8.4).) The algorithm sets out at \mathbf{x}_0 . The majorization function $g(\cdot, \mathbf{x}_0)$ is fitted by matching the value and first derivative of $f(\cdot)$ at \mathbf{x}_0 . Subsequently the function $g(\cdot, \mathbf{x}_0)$ is minimized to find the next point \mathbf{x}_1 . This procedure is repeated to find the point \mathbf{x}_2 etc.

by \mathbf{B} , the running \mathbf{x}_i by \mathbf{x} , and the current \mathbf{x}_i , i.e. the current i^{th} row vector of \mathbf{X} , is denoted by \mathbf{y} . Let λ denote the largest eigenvalue of \mathbf{B} . Then, the matrix $\mathbf{B} - \lambda\mathbf{I}$ is negative semidefinite, so that the following inequality holds:

$$(\mathbf{x} - \mathbf{y})^T (\mathbf{B} - \lambda\mathbf{I})(\mathbf{x} - \mathbf{y}) \leq 0, \quad \forall \mathbf{x},$$

which gives after some manipulations

$$(6) \quad \mathbf{x}^T \mathbf{B} \mathbf{x} \leq 2\lambda - 2\mathbf{x}^T (\lambda \mathbf{y} - \mathbf{B} \mathbf{y}) - \mathbf{y}^T \mathbf{B} \mathbf{y}, \quad \forall \mathbf{x},$$

using the fact that $\mathbf{x}^T \mathbf{x} = \mathbf{y}^T \mathbf{y} = 1$.

Combining (5) and (6) we obtain the majorizing function of $f_i(\mathbf{x}; \mathbf{X})$, that is,

$$f_i(\mathbf{x}; \mathbf{X}) \leq -\frac{2}{c} \mathbf{x}^T (\lambda \mathbf{y} - \mathbf{B} \mathbf{y} + \sum_{j:i \neq j} w_{ij} r_{ij} \mathbf{x}_j) + (\text{const in } \mathbf{x}) = g_i(\mathbf{x}; \mathbf{X}), \quad \forall \mathbf{x}.$$

The big advantage of $g_i(\cdot; \mathbf{X})$ over $f_i(\cdot, \mathbf{X})$ is that it is linear in \mathbf{x} and that the minimization problem

$$(7) \quad \min \{ g_i(\mathbf{x}; \mathbf{X}) ; \|\mathbf{x}\|_2 = 1 \}$$

is readily solved by

$$\mathbf{x}^* := \mathbf{z} / \|\mathbf{z}\|_2, \quad \mathbf{z} := \lambda \mathbf{y} - \mathbf{B}\mathbf{y} + \sum_{j:j \neq i} w_{ij} r_{ij} \mathbf{x}_j.$$

If $\mathbf{z} = \mathbf{0}$ then any \mathbf{x}^* , $\|\mathbf{x}^*\|_2 = 1$, will solve Problem (7).

4 The Algorithm and Convergence Analysis

Majorization algorithms are known to converge to a point with negligible gradient. This property holds also for the current situation, as will be shown hereafter. As the convergence criterion is defined in terms of the gradient ∇f , an expression for ∇f is needed. We restrict to the case of all w_{ij} equal. As shown in Grubišić & Pietersz (2004), the gradient is then given by

$$\nabla f = 4c^{-1} \Psi \mathbf{X}, \quad \Psi := \mathbf{X}\mathbf{X}^T - \mathbf{R}.$$

An expression for the gradient for the objective function with general weights can be found by straightforward differentiation. The majorization algorithm has been displayed in Algorithm 1.

Algorithm 1 The majorization algorithm for finding a low-rank correlation matrix locally nearest to a given matrix. Here \mathbf{R} denotes the input matrix, \mathbf{W} denotes the weight matrix, n denotes its dimension, d denotes the desired rank, $\varepsilon_{\|\nabla f\|}$ is the convergence criterion for the norm of the gradient and ε_f is the convergence criterion on the improvement in the function value.

Input: \mathbf{R} , \mathbf{W} , n , d , $\varepsilon_{\|\nabla f\|}$, ε_f .

- 1: Find starting point \mathbf{X} by means of the modified PCA method (4).
- 2: **for** $k = 0, 1, 2, \dots$ **do**
- 3: **stop** if the norm of the gradient of f at $\mathbf{X}^{(k)} := \mathbf{X}$ is less than $\varepsilon_{\|\nabla f\|}$
 and the improvement in the function value $f_{k-1}/f_k - 1$ is less than ε_f .
- 4: **for** $i = 1, 2, \dots, n$ **do**
- 5: Set $\mathbf{B} := \sum_{j \neq i} w_{ij} \mathbf{x}_j \mathbf{x}_j^T$.
- 6: Calculate λ to be the largest eigenvalue of the $d \times d$ matrix \mathbf{B} .
- 7: Set $\mathbf{z} := \lambda \mathbf{x}_i - \mathbf{B}\mathbf{x}_i + \sum_{j \neq i} w_{ij} r_{ij} \mathbf{x}_j$.
- 8: If $\mathbf{z} \neq \mathbf{0}$, then set the i^{th} row \mathbf{x}_i of \mathbf{X} equal to $\mathbf{z} / \|\mathbf{z}\|_2$.
- 9: **end for**
- 10: **end for**

Output: The $n \times n$ matrix $\mathbf{X}\mathbf{X}^T$ is the rank- d approximation of \mathbf{R} satisfying the gradient-norm convergence constraint.

Zangwill (1969) developed generic sufficient conditions that guarantee convergence of an iterative algorithm. The result is repeated here in a form adapted to the case of majorization.

Let M be a compact set. Assume the specification of a subset $S \subset M$ called the *solution set*. A point $Y \in S$ is deemed a *solution*. An (*autonomous*) *iterative algorithm* is a map $A : M \rightarrow M \cup \{\text{stop}\}$ such that $A^{-1}(\{\text{stop}\}) = S$. The proof of the following theorem is adapted from the proof of theorem 1 in Zangwill (1969).

Theorem 1 (Global convergence) *Consider finding a local minimum of the objective function $f(\mathbf{X})$ by use of Algorithm 1. Suppose given a fixed tolerance level ε on the gradient of f . A point \mathbf{X} is called a solution if $\|\nabla f(\mathbf{X})\| < \varepsilon$. Then from any starting point $\mathbf{X}^{(0)}$, the algorithm either stops at a solution or produces an infinite sequence of points none of which are solutions, for which the limit of any convergent subsequence is a solution point.*

Proof: Without loss of generality we may assume that the procedure generates an infinite sequence of points $\{\mathbf{X}^{(k)}\}$ none of which are solutions. It remains to be proven that the limit of any convergent subsequence must be a solution.

First, note that the algorithm $A(\cdot)$ is continuous in \mathbf{X} . Second, note that if $\mathbf{X}^{(k)}$ is not a solution then

$$f(\mathbf{X}^{(k+1)}) = f(A(\mathbf{X}^{(k)})) < f(\mathbf{X}^{(k)}).$$

Namely if $\mathbf{X}^{(k)}$ is not a solution then its gradient is non-negligible. Since the objective and all majorization functions are differentiable, we necessarily have that the gradients agree at $\mathbf{X}^{(k)}$. Therefore, when minimizing the majorization functions $f_i(\cdot, \mathbf{X})$ there will be at least one i for which we find a strictly smaller objective value. Thus $\mathbf{X}^{(i+1)} := A(\mathbf{X}^{(i)})$ has a strictly smaller objective function value than $\mathbf{X}^{(k)}$. Third, note that the sequence $\{f(\mathbf{X}^{(k)})\}_{k=0}^{\infty}$ has a limit since it is monotonically decreasing and bounded from below by 0.

Let $\{\mathbf{X}^{(k_j)}\}_{j=1}^{\infty}$ be any subsequence that converges to \mathbf{X}^* , say. It must be shown that \mathbf{X}^* is a solution. Assume the contrary. By continuity of the iterative procedure $A(\mathbf{X}^{(k_j)}) \rightarrow A(\mathbf{X}^*)$. By the continuity of $f(\cdot)$, we then have

$$f(A(\mathbf{X}^{(k_j)})) \downarrow f(A(\mathbf{X}^*)) < f(\mathbf{X}^*),$$

which is in contradiction with $f(A(\mathbf{X}^{(k_j)})) \rightarrow f(\mathbf{X}^*)$. \square

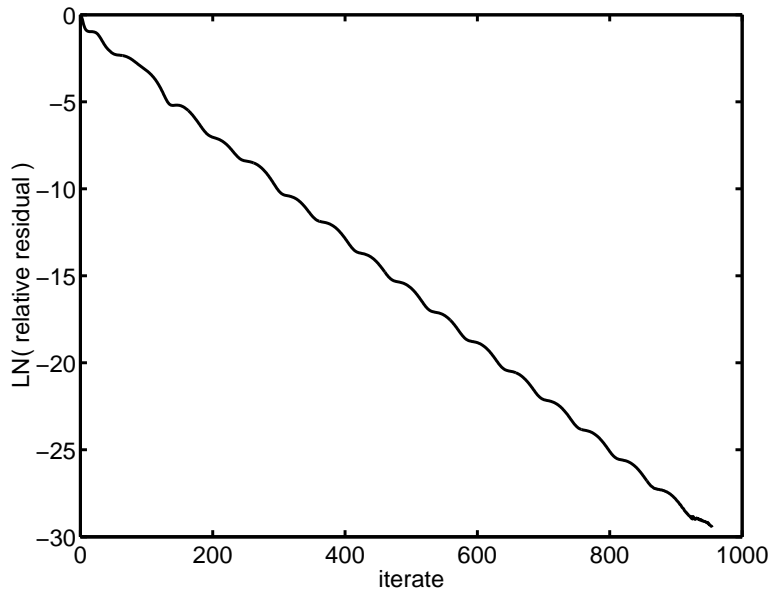


Figure 2: Convergence run for the matrix \mathbf{R} included in the `major` package, $n = 80$, $d = 3$. The relative residual is $\|\nabla f(\mathbf{X}^{(i)})\|_F / \|\nabla f(\mathbf{X}^{(0)})\|_F$.

5 Numerical Results

Algorithm 1 has been implemented in a MATLAB package called `major`. It can be downloaded from www.few.eur.nl/few/people/pietersz. The package consists of the following files: `clamp.m`, `dF.m`, `F.m`, `grad.m`, `guess.m`, `major.m`, `P_tangent.m` and `svdplus.m`. The package can be run by calling `[Xn,Fn]=major(R,d,ftol,gradtol)`. Here \mathbf{R} denotes the input correlation matrix, d the desired rank, \mathbf{X}_n the final configuration matrix, F_n denotes the final objective function value, `ftol` the convergence tolerance on the improvement of f , and `gradtol` the convergence tolerance on the norm of the gradient. The aforementioned web-page also contains a package `majorw` that implements non-constant weights for the objective function f . An example convergence run for `major` has been displayed in Figure 2.

The remainder of this Section is organized as follows. First, we numerically compare majorization with other methods in the literature. Second, we present an example with non-constant weights. Third, we explain and investigate the order effect. Fourth, we consider and study an alternative version of the majorization algorithm.

Table 1: Excerpt of Table 3 in De Jong et al. (2002).

	γ_1	γ_2	γ_3	γ_4
estimate	0.000	0.480	1.511	0.186
standard error	-	0.099	0.289	0.127

5.1 Numerical Comparison with Other Methods

The numerical performance of the majorization algorithm was compared to the performance of the Lagrange multiplier method, geometric programming⁴ and the parametrization method. The four algorithms were tested on randomly generated ‘interest rate’ correlation matrices.

The ‘interest rate’ correlation matrices were randomly generated as follows. A parametric form for correlation matrices is posed in De Jong, Driessen & Pelsser (2002, Equation (8)). We repeat here the parametric form for completeness.

$$\rho(t_i, t_j) = \exp \left\{ -\gamma_1 |t_i - t_j| - \frac{\gamma_2 |t_i - t_j|}{\max(t_i, t_j)^{\gamma_3}} - \gamma_4 |\sqrt{t_i} - \sqrt{t_j}| \right\},$$

with $\gamma_1, \gamma_2, \gamma_4 > 0$ and with t_i denoting the expiry time of rate i . (Our particular choice is $t_i = i$, $i = 1, 2, \dots$) This model was then subsequently estimated with USD historical interest rate data. In Table 3 of De Jong et al. (2002), the estimated γ parameters are listed, along with their standard errors. An excerpt of this table has been displayed in Table 1. The random DJDP matrix that we used is obtained by randomizing the γ -parameters. We assumed the γ -parameters distributed normally with mean and standard errors given by Table 1, with $\gamma_1, \gamma_2, \gamma_4$ capped at zero.

A number of 100 matrices were randomly generated, with n , d , and the computational time t varied. Subsequently the four algorithms were applied each with t seconds of computational time — and the computational time constraint was the only stopping criterion. A statistical description of the results has been displayed in Table 2. For each correlation matrix a ranking out of $\{1, 2, 3, 4\}$ was given to the algorithms, with 1 denoting the smallest obtained f and 4 denoting the largest obtained f . The abbreviations ‘avg’, ‘std dev’, ‘geom.prog’ and ‘paramet.’ denote ‘average’, ‘standard deviation’, ‘geometric programming’ and ‘parametrization’, respectively.

⁴For geometric programming we used the LRCM MIN package downloadable from www.few.eur.nl/few/people/pietersz. The Riemannian Newton-algorithm was applied.

Table 2: Comparison test. Statistical description of the results of four algorithms on 100 randomly generated DJDP matrices. Each algorithm was given t seconds of computational time.

	Lagrange	geom.prog	majorization	paramet.
<i>n</i> = 10, <i>d</i> = 2, <i>t</i> = 0.1 <i>s</i>				
avg ranking	3.2	2.2	1.9	2.8
std dev ranking	1.0	1.0	1.0	1.0
average <i>f</i>	2.65×10^{-03}	2.63×10^{-03}	3.36×10^{-03}	3.36×10^{-03}
std dev <i>f</i>	1.54×10^{-03}	1.56×10^{-03}	3.79×10^{-03}	3.79×10^{-03}
avg # iterations	12.3	5.0	199.0	400.0
std dev # iterations	2.1	0.1	10.0	0.0
<i>n</i> = 20, <i>d</i> = 4, <i>t</i> = 1 <i>s</i>				
avg ranking	2.4	2.9	1.7	3.1
std dev ranking	1.2	0.7	1.0	1.0
average <i>f</i>	5.61×10^{-04}	5.59×10^{-04}	6.98×10^{-04}	9.25×10^{-04}
std dev <i>f</i>	3.82×10^{-04}	3.85×10^{-04}	1.00×10^{-03}	1.25×10^{-03}
avg # iterations	60.3	26.2	821.4	2000.0
std dev # iterations	9.4	1.7	44.6	0.0
<i>n</i> = 80, <i>d</i> = 20, <i>t</i> = 2 <i>s</i>				
avg ranking	2.7	2.6	1.2	3.6
std dev ranking	0.8	0.8	0.6	0.8
average <i>f</i>	2.20×10^{-05}	2.20×10^{-05}	1.81×10^{-05}	6.15×10^{-05}
std dev <i>f</i>	1.69×10^{-05}	1.73×10^{-05}	4.13×10^{-05}	7.86×10^{-05}
avg # iterations	6.2	13.3	87.4	15.0
std dev # iterations	0.6	1.1	1.5	0.0

We consider the average ranking to be the most important measure for the efficiency of an algorithm. Table 2 suggests that within a fixed computational time the majorization algorithm is the most efficient algorithm in all cases considered.

To assess the quality of the solutions, we checked whether the matrices produced by the algorithms were converging to a global minimum. Here, we have the special case that we can check for a global minimum, which in other minimization problems may be difficult to assess. We allowed the algorithms 20 seconds of computational time. If a produced solution point satisfied a strict convergence criterion on the norm of the gradient, then we checked whether such stationary point is a global minimum by inspecting the Lagrange multipliers, see Zhang & Wu (2003), Wu (2003) and Grubišić & Pietersz (2004, Lemma 12). Namely if the produced configuration contains the largest d eigenvalues out of the matrix associated with the Lagrange multipliers, then we have a global minimum. The percentage of matrices that were deemed global minima was between 95% and 100% for both geometric programming and majorization, respectively, for the cases $n = 20, d = 4$ and $n = 10, d = 2$. The Lagrange multiplier and parametrization methods did not produce any stationary points within 20 seconds of computational time. The percentage of global minima is high since the eigenvalues of financial correlation matrices are rapidly decreasing. In effect, there are large differences between the first 4 or 5 consecutive eigenvalues. For the case $n = 80, d = 20$ it was more difficult to check the global minimum criterion since subsequent eigenvalues are smaller and closer to each other. In contrast, if we apply the methods for all cases to random correlation matrices of Davies & Higham (2000), for which the eigenvalues are all very similar, we find that a much lower percentage of produced stationary points were global minima.

5.2 Non-Constant Weights

We considered the example with non-constant weights described in Rebonato (2002, Section 9.3), in which a functional form for the correlation matrix is specified, repeated here for clarity:

$$r_{ij} = \text{LongCorr} + (1 - \text{LongCorr}) \exp \left\{ -\beta |t_i - t_j| \right\}, \quad i, j = 1, \dots, n.$$

The parameters are set to $n = 10$, $\text{LongCorr} = 0.6$, $\beta = 0.1$, $t_i = i$. Subsequently Rebonato presents the rank 2, 3, and 4 matrices found by the parametrization method for the case of equal weights. The majorization algorithm was also applied and its convergence criterion was set to machine precision for the norm of the gradient. Comparative results for the parametrization and majorization algorithms have been displayed in Table 3. Columns

Table 3: Comparative results of the parametrization and majorization algorithms for the example described in Rebonato (2002, Section 9.3.1).

d	$\ \nabla f\ _F$ major.	f major.	f Rebonato	I	II	CPU major.
2	2×10^{-17}	5.131×10^{-04}	5.137×10^{-04}	41×10^{-04}	0.02×10^{-04}	0.4s
3	2×10^{-17}	1.26307×10^{-04}	1.26311×10^{-04}	15×10^{-04}	0.01×10^{-04}	1.0s
4	2×10^{-17}	4.85×10^{-05}	4.86×10^{-05}	70×10^{-04}	0.01×10^{-04}	2.1s

I and II denote $\|\mathbf{R}_{\text{Reb}}^{\text{Approx}} - \mathbf{R}_{\text{major}}^{\text{Approx}}\|_F$ and $\|\mathbf{R}_{\text{major, rounded}}^{\text{Approx}} - \mathbf{R}_{\text{major}}^{\text{Approx}}\|_F$, respectively. Here ‘Approx’ stands for the rank-reduced matrix produced by the algorithm and ‘rounded’ stands for rounding the matrix after 6 digits, as this is the precision displayed in Rebonato (2002). Columns I and II show that the matrices displayed in Rebonato (2002) are not yet fully converged up to machine precision, since the round-off error from displaying only 6 digits is much smaller than the error in obtaining full convergence to the stationary point.

Rebonato proceeds by minimizing f for rank 3 with two different weights matrices. These weights matrices are chosen by financial arguments specific to a ratchet cap and a trigger swap, which are interest rate derivatives. The weights matrix $\mathbf{W}^{(R)}$ for the ratchet cap is a tridiagonal matrix

$$w_{ij}^{(R)} = 1 \text{ if } j = i - 1, i, i + 1, \quad w_{ij}^{(R)} = 0, \text{ otherwise}$$

and the weights matrix $\mathbf{W}^{(T)}$ for the trigger swap has ones on the first two rows and columns

$$w_{ij}^{(T)} = 1 \text{ if } i = 1, 2 \text{ or } j = 1, 2, \quad w_{ij}^{(T)} = 0, \text{ otherwise.}$$

Rebonato subsequently presents the solution matrices found by the parametrization method. These solutions exhibit a highly accurate yet non-perfect fit to the relevant portions of the correlation matrices. In contrast, majorization finds exact fits. The results have been displayed in Table 4.

5.3 The Order Effect

The majorization algorithm is based on sequentially looping over the rows of the matrix \mathbf{X} . In Algorithm 1 the row index runs from 1 to n . There is however no distinct reason to start with row 1, then 2, etc. It would be

Table 4: Results for the ratchet cap and trigger swap. Here ‘tar.’ denotes the target value, ‘maj.’ and ‘Reb.’ denote the resulting value obtained by the majorization algorithm and Rebonato (2002, Section 9.3), respectively.

Ratchet cap									
First principal sub-diagonal; CPU time major: 2.8s; obtained $f < 2 \times 10^{-30}$									
tar.	.961935	.961935	.961935	.961935	.961935	.961935	.961935	.961935	.961935
maj.	.961935	.961935	.961935	.961935	.961935	.961935	.961935	.961935	.961935
Reb.	.961928	.961880	.961977	.962015	.962044	.962098	.961961	.961867	.962074
Trigger swap									
First two rows (or equivalently first two columns); CPU time major: 2.4s; obtained $f < 2 \times 10^{-30}$									
Row 1 (without the unit entry (1,1))									
tar.	.961935	.927492	.896327	.868128	.842612	.819525	.798634	.779732	.762628
maj.	.961935	.927492	.896327	.868128	.842612	.819525	.798634	.779732	.762628
Reb.	.961944	.927513	.896355	.868097	.842637	.819532	.798549	.779730	.762638
Row 2 (without the unit entry (2,2))									
tar.	.961935	.961935	.927492	.896327	.868128	.842612	.819525	.798634	.779732
maj.	.961935	.961935	.927492	.896327	.868128	.842612	.819525	.798634	.779732
Reb.	.961944	.962004	.927565	.896285	.868147	.842650	.819534	.798669	.779705

equally reasonable to consider any permutation p of the numbers $\{1, \dots, n\}$ and then let the row index run as $p(1), p(2), \dots, p(n)$. A priori, there is nothing to guarantee or prevent that the resulting solution point produced with permutation p would differ from or be equal to the solution point produced by the default loop $1, \dots, n$. This dependency of the order is termed ‘the order effect’. We show empirically that the solutions produced by the algorithm can differ when using a different permutation, however we show that this is unlikely to happen for financial correlation matrices. The order effect can have two consequences. First, the produced solution correlation matrix can differ – this generally implies a different objective function value as well. Second, even when the produced solution correlation matrix is equal, the configuration \mathbf{X} can differ – in this case we have equal objective function values. Namely consider a $n \times d$ configuration matrix \mathbf{X} and assume given any orthonormal $d \times d$ matrix \mathbf{Q} , i.e. $\mathbf{Q}\mathbf{Q}^T = \mathbf{I}$. Then the configuration matrices \mathbf{X} and $\mathbf{X}\mathbf{Q}$ are associated with the same correlation matrices⁵: $\mathbf{X}\mathbf{Q}\mathbf{Q}^T\mathbf{X} = \mathbf{X}\mathbf{X}^T$.

We investigated the order effect for Algorithm 1 numerically, as follows.

⁵The indeterminacy of the result produced by the algorithm can easily be resolved by either considering only $\mathbf{X}\mathbf{X}^T$ or by rotation of \mathbf{X} into its principal axes. For the latter, let $\mathbf{X}^T\mathbf{X} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$ be an eigenvalue decomposition. Then the principal axes representation is given by $\mathbf{X}\mathbf{Q}$.

Table 5: The order effect. Here $n = 30$, $d = 2$ and 100 random permutations were applied. Four types of produced correlation matrices could be distinguished. The table displays the associated f and frequency.

type	I	II	III	IV
f	0.110423	0.110465	0.110630	0.110730
frequency	2%	88%	7%	3%

We generated either a random DJDP matrix, see Section 5.1, or a random correlation matrix in MATLAB by

```
rand('state',0);randn('state',0);n=30;R=gallery('randcorr',n);
```

The random correlation matrix generator `gallery('randcorr',n)` has been described in Davies & Higham (2000). Subsequently we generated 100 random permutations with `p=randperm(n);`. For each of the permutations, Algorithm 1 was applied with $d = 2$ and a high accuracy was demanded: $\varepsilon_{\|\nabla f\|} = \varepsilon_f = 10^{-16}$. The results for the two different correlation matrices are as follows.

(*DJPD random correlation matrix.*) Only one type of produced solution correlation matrix could be distinguished, which turned out to be a global minimum by inspection of the Lagrange multipliers. We also investigated the orthonormal transformation effect. For \mathbb{R}^2 , an orthonormal transformation can be characterized by the rotation of the two basis vectors and then by -1 or +1 denoting whether the second basis vector is reflected in the origin or not. All produced matrices \mathbf{X} were differently rotated, but no reflection occurred. The maximum rotation was equal to 0.8 degrees and the standard deviation of the rotation was 0.2 degrees.

(*Davies & Higham (2000) random correlation matrix.*) Essentially four types of produced solution correlation matrices could be distinguished, which we shall name I, II, III, and IV. The associated objective function values and the frequency at which the types occurred have been displayed in Table 5. We inspected the Lagrange multipliers to find that none of the four types was a global minimum. For type II, the most frequently produced low-rank correlation matrix, we also investigated the orthonormal transformation effect. Out of the 88 produced matrices \mathbf{X} that could be identified with type II, all were differently rotated, but no reflection occurred. The maximum rotation was equal to 38 degrees and the standard deviation of the rotation was 7 degrees.

From the above results we conclude that the order effect is not much of an issue for the case of interest rate correlation matrices, at least not for the numerical setting that we investigated.

5.4 Using an Estimate for the Largest Eigenvalue

In Algorithm 1, the largest eigenvalue of B is calculated by an eigenvalue decomposition. This decomposition may be relatively expensive to calculate. Instead of a full calculation, we could consider finding an easy-to-calculate upper bound on the largest eigenvalue of B . Such upper bound is readily determined as $n - 1$ due to the unit length restrictions on the $n - 1$ vectors \mathbf{x}_i . Replacing λ and its calculation by $n - 1$ in algorithm 1 will result in a reduction of computational time by not having to calculate the eigenvalue decomposition. A disadvantage is however that the resulting fitted majorizing function will be much steeper causing its minimum to be much closer to the point of outset. In other words, the steps taken by the majorization algorithm will be smaller. Whether to use $n - 1$ instead of λ is thus a trade-off between computational time for the decomposition and the step-size.

We tested replacing λ by $n - 1$ for 100 correlation matrices of dimension 80×80 . These matrices were randomly generated with the procedure of Davies & Higham (2000). We allowed both versions of the algorithm a computational time of less than 1 second. We investigated $d = 3$, $d = 6$, $d = 40$ and $d = 70$. For all 400 cases, without a single exception, the version of the algorithm with the full calculation of λ produced a matrix that had a lower value f than the version with $n - 1$. This seems to suggest that a complete calculation of the largest eigenvalue is most efficient, however we point out here that these results could be particular to our numerical setting. The ‘ $n - 1$ ’ version of the algorithm remains an interesting alternative and could potentially be beneficial in certain experimental setups.

6 Conclusions

We have developed a novel algorithm for finding a low-rank correlation matrix locally nearest to a given matrix. The algorithm is based on iterative majorization and this paper is the first to apply majorization to the area of derivatives pricing. We showed theoretically that the algorithm converges to a stationary point from any starting point. As an addition to the previously available methods in the literature, majorization was in our simulation setup more efficient than either geometric programming, the Lagrange multiplier technique or the parametrization method. Furthermore majorization is easier

to implement than any method other than modified PCA. The majorization method efficiently and straightforwardly allows for arbitrary weights.

References

- Borg, I. & Groenen, P. J. F. (1997), *Modern Multidimensional Scaling*, Springer-Verlag, Berlin.
- Brace, A., Gątarek, D. & Musiela, M. (1997), ‘The market model of interest rate dynamics’, *Mathematical Finance* **7**(2), 127–155.
- Brigo, D. (2002), A note on correlation and rank reduction, Downloadable from www.damianobrigo.it.
- Davies, P. I. & Higham, N. J. (2000), ‘Numerically stable generation of correlation matrices and their factors’, *BIT* **40**, 640–651.
- De Jong, F., Driessen, J. & Pelsser, A. A. J. (2002), On the information in the interest rate term structure and option prices, Working paper, Erasmus University Rotterdam.
- De Leeuw, J. & Heiser, W. (1977), Convergence of correction-matrix algorithms for multidimensional scaling, *in* J. C. Lingoes, E. E. Roskam & I. Borg, eds, ‘Geometric representations of relational data’, Mathesis Press, Ann Arbor, MI, pp. 735–752.
- Flury, B. (1988), *Common Principal Components and Related Multivariate Models*, J. Wiley & Sons, New York.
- Grubišić, I. & Pietersz, R. (2004), Efficient rank reduction of correlation matrices, Working paper, Utrecht University, Utrecht, Downloadable from www.few.eur.nl/few/people/pietersz.
- Hull, J. C. & White, A. (2000), ‘Forward rate volatilities, swap rate volatilities, and implementation of the LIBOR market model’, *Journal of Fixed Income* **3**, 46–62.
- Jamshidian, F. (1997), ‘Libor and swap market models and measures’, *Finance and Stochastics* **1**, 293–330.
- Kiers, H. A. L. (2002), ‘Setting up alternating least squares and iterative majorization algorithms for solving various matrix optimization problems’, *Computational Statistics and Data Analysis* **41**, 157–170.

- Kiers, H. A. L. & Groenen, P. J. F. (1996), ‘A monotonically convergent algorithm for orthogonal congruence rotation’, *Psychometrika* **61**, 375–389.
- Miltersen, K. R., Sandmann, K. & Sondermann, D. (1997), ‘Closed form solutions for term structure derivatives with log-normal interest rates’, *Journal of Finance* **52**(1), 409–430.
- Rapisarda, F., Mercurio, F. & Brigo, D. (2002), Parametrizing correlations: A geometric interpretation, Banca IMI Working Paper, Downloadable from www.fabiomercurio.it.
- Rebonato, R. (1999a), ‘Calibrating the BGM model’, pp. 74–79. Risk Magazine.
- Rebonato, R. (1999b), *Volatility and Correlation in the Pricing of Equity, FX and Interest-Rate Options*, J. Wiley & Sons, Chichester.
- Rebonato, R. (2002), *Modern Pricing of Interest-Rate Derivatives*, Princeton University Press, New Jersey.
- Wu, L. (2003), ‘Fast at-the-money calibration of the LIBOR market model using Lagrange multipliers’, *Journal of Computational Finance* **6**(2), 39–77.
- Zangwill, W. I. (1969), ‘Convergence conditions for nonlinear programming algorithms’, *Management Science (Theory Series)* **16**(1), 1–13.
- Zhang, Z. & Wu, L. (2003), ‘Optimal low-rank approximation to a correlation matrix’, *Linear Algebra and its Applications* **364**, 161–187.