Computing Nearest Covariance and Correlation Matrices

A thesis submitted to the University of Manchester for the degree of Master of Science in the Faculty of Science and Engineering.

October 2001

Craig Lucas

Department of Mathematics

Contents

Abstract

We look at two matrix nearness problems posed by a finance company, where nearness is measured in the Frobenius norm. Correlation and covariance matrices are computed from sampled stock data with missing entries by a technique that produces matrices that are not positive semidefinite. In the first problem we find the nearest correlation matrix that is positive semidefinite and preserves any correlations known to be exact. In the second problem we investigate how the missing elements in the data should be chosen in order to generate the nearest covariance matrix to the indefinite matrix from the completed set of data. We show how the former problem can be solved using an alternating projections algorithm and how the latter problem can be investigated using a multi-directional search optimization method.

Declaration

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

Copyright and Intellectual Property Rights

Acknowledgements

I would like to thank my supervisor, Nick Higham, for his guidance and numerous helpful suggestions throughout the preparation of this thesis.

Thanks also go to Glenn for his companionship and reminding me there is more to my life than mathematics.

1 Introduction

1.1 Covariance and Correlation Matrices

In statistics, a *random variable*, say Y, is a function defined over a *sample space*, Ω, that has associated a real number, $Y(e) = y$, for each *outcome*, e, in Ω. We call y an *observation.* The sample space is the set of all possible outcomes of an *experiment*, the process of obtaining this outcome from some phenomenon. For example, a random variable could be 'The number of threes in an integer' if the sample space consists of all integers, and for the outcome 3003 we have $Y(3003) = 2$.

A *sample* is a subset of the sample space, and we say a random variable is *sampled* if we measure the values y from this subset only.

We can measure the spread or dispersion of the sampled random variable, and call this quantity the *sample variance.* Similarly we can measure how two sampled random variables vary relative to each other and call this the *sample covariance* of the two random variables. Another measure between two random variables is their *correlation coefficient.* The correlation coefficient is a measure of the linear relation between the two variables. It takes values between -1 and 1, where 1 indicates *perfect positive correlation* and -1 *perfect negative correlation.* Perfect correlation arises if x and y are vectors of observations for two sampled random variables and $x = ky, k \in \mathbb{R}$. We give formal definitions of the sample covariance and sample correlation coefficient later.

If we have a collection of sampled random variables we can construct *sample covariance matrices* and *sample correlation matrices.* The (i, j) element of a sample covariance matrix is the covariance of the ith and jth random variables. Similarly the (i, j) element of a sample correlation matrix is the correlation coefficient of the ith and jth random variables.

1.2 Application

Correlation matrices get their name from the fact they contain correlation coefficients, but they also arise in non-statistical applications in numerical linear algebra. They are used in error analysis of Cholesky factorisation, in a preconditioner for iteration methods for solving symmetric positive definite linear systems and in error analysis of Jacobi methods for finding the eigensystem of a symmetric matrix. See [3] for more details.

Our application, however, is a statistical one, where sample covariance and correlation matrices are generated from stock data. We look at two problems posed by a finance company who use the matrices for analysis and prediction purposes.

1.3 Properties

Covariance and correlation matrices are symmetric and positive semidefinite (see section 2.1.) Correlation matrices have a unit diagonal since a variable is clearly perfectly correlated with itself.

It is well known that for a positive semidefinite matrix A

$$|a_{ij}| \leq \sqrt{a_{ii}a_{jj}}.$$

Thus for correlation matrices we have

$$|a_{ij}| \leq 1$$

and the inequality is strict if variables i and j are not perfectly correlated.

1.4 Eigenvalues of a Correlation Matrix

Gershgorin's theorem states that the eigenvalues of $A \in \mathbb{C}^{n\times n}$ lie in the union of n disks in the complex plane. The ith disk is given by

$$D_i = \left\{ z \in \mathbb{C} : |z - a_{ii}| \leq \sum_{\substack{j=1\\ j\neq i}}^{n} |a_{ij}| \right\}, \qquad i = 1\colon n.$$

Since a correlation matrix, A, is symmetric all its eigenvalues are real so we have for an eigenvalue λ,

$$|\lambda - 1| \leq n - 1.$$

But also A is positive semidefinite, so its eigenvalues are nonnegative and we have finally

$$0 \leq \lambda_i \leq n, \qquad i\colon n.$$

Furthermore, since trace$(A) = \sum_i^n \ \lambda_i$,

$$\sum_i \lambda_i = n.$$

2 Calculation of Covariance and Correlation Matrices

2.1 Exact Sample Covariance and Correlation Matrices

There are several ways we can construct covariance and correlation matrices.

Consider a matrix $P \in \mathbb{R}^{m \times n}$ where each column represents m observations of a random variable and each row observations at a particular time. That is, p_{ij} is the ith observation of the jth random variable. Let S represent the *sample covariance matrix*, and R the *sample correlation matrix.* Sample covariance, for the ith and jth random variable, is defined as

$$s_{ij} = \frac{1}{m-1}(p_i - \bar{p}_i)^T(p_j - \bar{p}_j), \tag{2.1}$$

where the coefficient $(m-1)^{-1}$ is called the *normalisation.*

Here, p_i and p_j represent the ith and jth columns of P, and $\bar{p}_k \in \mathbb{R}$ the *sample mean* of random variable p_k,

$$\bar{p}_k = \frac{1}{m}\sum_{i=1}^{m} p_{ik}.$$

The sample correlation coefficient is defined as

$$r_{ij} = \frac{(p_i - \bar{p}_i)^T(p_j - \bar{p}_j)}{\|p_i - \bar{p}_i\|_2 \|p_j - \bar{p}_j\|_2}. \tag{2.2}$$

From (2.1) we can write

$$s_{ij} = \frac{1}{m-1}[(p_{1i} - \bar{p}_i)(p_{1j} - \bar{p}_j) + (p_{2i} - \bar{p}_i)(p_{2j} - \bar{p}_j) + \\ \cdots + (p_{mi} - \bar{p}_i)(p_{mj} - \bar{p}_j)],$$

and therefore

$$
\begin{aligned}
S &= \begin{bmatrix} p_{11}-\bar{p}_1 \\ p_{12}-\bar{p}_2 \\ \vdots \\ p_{1n}-\bar{p}_n \end{bmatrix} [p_{11}-\bar{p}_1, p_{12}-\bar{p}_2, \ldots, p_{1n}-\bar{p}_n] \\
&+ \begin{bmatrix} p_{21}-\bar{p}_1 \\ p_{22}-\bar{p}_2 \\ \vdots \\ p_{2n}-\bar{p}_n \end{bmatrix} [p_{21}-\bar{p}_1, p_{22}-\bar{p}_2, \ldots, p_{2n}-\bar{p}_n] \\
&+ \cdots + \\
&+ \begin{bmatrix} p_{m1}-\bar{p}_1 \\ p_{m2}-\bar{p}_2 \\ \vdots \\ p_{mn}-\bar{p}_n \end{bmatrix} [p_{m1}-\bar{p}_1, p_{m2}-\bar{p}_2, \ldots, p_{mn}-\bar{p}_n] .
\end{aligned}
$$

If we define the ith observation as $q_i = [p_{i1}, p_{i2}, \ldots, p_{in}]^T \in \mathbb{R}^n$ and $\bar{p} = [\bar{p}_1, \bar{p}_2, \ldots, \bar{p}_n] \in \mathbb{R}^n$ as the vector of sample means we have

$$S = \frac{1}{m-1}\sum_{i=1}^{m}(q_i-\bar{p})(q_i-\bar{p})^T \in \mathbb{R}^{n\times n}. \tag{2.3}$$

We can write (2.2) as

$$r_{ij} = \frac{(m-1)s_{ij}}{\sqrt{(m-1)s_{ii}}\sqrt{(m-1)s_{jj}}},$$

and defining

$$D_S^{1/2} = \text{ diag } (s_{11}^{-1/2}, s_{22}^{-1/2}, \ldots, s_{nn}^{-1/2})$$

we have that the sample correlation matrix is

$$R = D_S^{1/2} S D_S^{1/2} \in \mathbb{R}^{n\times n}. \tag{2.4}$$

We can write (2.3) as

$$\begin{aligned} S &= \frac{1}{m-1}(P^T - \bar{p}e^T)(P^T - \bar{p}e^T)^T \\ &= \frac{1}{m-1}(P^T - \bar{p}e^T)(P - e\bar{p}^T), \end{aligned}$$

where $e = [1, 1, \ldots, 1] \in \mathbb{R}^m$. Now, we can write

$$\bar{p} = m^{-1}P^T e,$$

so

$$\begin{aligned} S &= \frac{1}{m-1}(P^T - m^{-1}P^T ee^T)(P - m^{-1}ee^T P) \\ &= \frac{1}{m-1}P^T(I_m - m^{-1}ee^T)(I_m - m^{-1}ee^T)P, \end{aligned}$$

where I_m is the $m \times m$ identity matrix. Now, $I_m - \frac{1}{m}ee^T$ is idempotent so

$$S = \frac{1}{m-1}P^T(I_m - m^{-1}ee^T)P.$$

Now $m^{-1}ee^T$ is rank 1 with nonzero eigenvalue 1, so $I_m - m^{-1}ee^T$ has one zero eigenvalue, and the remainder are 1. Hence S is positive semidefinite with rank at most the rank of $I_m - m^{-1}ee^T$ which is $m-1$ (and certainly $\leq n$, as $S \in \mathbb{R}^{n \times n}$). For S to be positive definite we clearly need $m > n$, that is, more observations than variables.

It is worth noting the rank of S and R will be reduced if there is any linear dependence, either by two random variables being perfectly correlated or more generally if a column of P can be written as a linear combination of other columns. Also if one variable is actually a constant then it will have zero variance and all the covariances involving it will also be zero.

We define COV(P) and COR(P) to be the sample covariance and correlation matrices respectively, computed from the sample data matrix P, and refer to these as *exact*. (See Appendix A.1 for `gen_cov.m` which computes COV(P) and `gen_cor.m` which computes COR(P).)

2.2 Approximate Sample Covariance and Correlation Matrices

In the finance application not all elements of the sample data matrix P are known. That is, at a given moment in time it is not possible to record the value of all the stocks. Thus we need a method to compute an *approximate* covariance and correlation matrix. One such method is a *pairwise deletion* method.

We represent the missing data matrix elements by NaNs. We use (2.1) to compute each element of the covariance matrix, but we use only the data that is available at common times for both variables. For example if we have

$$p_i = \begin{bmatrix} p_{i1} \\ \mathrm{NaN} \\ p_{i3} \\ p_{i4} \\ p_{i5} \end{bmatrix}, \qquad p_j = \begin{bmatrix} p_{j1} \\ p_{j2} \\ p_{j3} \\ \mathrm{NaN} \\ p_{j5} \end{bmatrix},$$

then in the computation of s_{ij} we use only those components for which data is available in both vectors. Thus

$$\bar{p}_i = \frac{1}{3}\left[p_{i1} + p_{i3} + p_{i5}\right], \qquad \bar{p}_j = \frac{1}{3}\left[p_{j1} + p_{j3} + p_{j5}\right],$$

and the normalisation of $m - 1$ is replaced with the effective sample size minus one, giving

$$s_{ij} = \frac{1}{2}\begin{bmatrix} p_{i1} - \bar{p}_i & p_{i3} - \bar{p}_i & p_{i5} - \bar{p}_i \end{bmatrix}\begin{bmatrix} p_{j1} - \bar{p}_j \\ p_{j3} - \bar{p}_j \\ p_{j5} - \bar{p}_j \end{bmatrix}.$$

It is obvious that nothing in this method will force S to be positive semidefinite. We call this S an *approximate* covariance matrix.

The approximate correlation matrix R is calculated from (2.4). Note that calculating an approximate R from (2.2) in an analogous way to an approximate S above is not equivalent.

We define $\overline{\text{COV}}(P)$ and $\overline{\text{COR}}(P)$ to be the approximate sample covariance and correlation matrices respectively, computed from the data matrix P with missing elements. (See Appendix A.1 for `cov_bar.m` which computes $\overline{\text{COV}}(P)$ and `cor_bar.m` which computes $\overline{\text{COR}}(P)$.)

Indefinite covariance and correlation matrices are a common problem. It has been reported on the MathWorks web site [16] that indefinite covariance matrices can be generated due to round-off error when a small number of observations are supplied to functions `cov` and `ewstats` in MATLAB.

Users of Scientific Software International's statistical software LISTREL have also found the same problems due to, among other reasons, pairwise deletion methods for dealing with incomplete data sets [13].

Finally, Financial Engineering Associates Inc.'s MakeVC [6] software claims to recognise the problem and make covariance matrices positive definite if they are not already.

3 Testing

In this chapter we describe the test data we will use for our experiments.

3.1 Test Data

We use data that is the sale prices of the top 8 companies from the NASDAQ 100 on August 10th, 2001. The prices are for Aug 1st, 2001 and those at the first trading day of the previous nine months. See Table 3.1.

	Biogen	PACCAR	Electronic Arts Inc.	Amgen Inc.	eBay Inc.	Microsoft	Qualcomm Inc.	NVIDIA Corp.
1 Nov 00	59.875	42.734	47.938	60.359	54.016	69.625	61.500	62.125
1 Dec 00	53.188	49.000	39.500	64.813	34.750	56.625	83.000	44.500
2 Jan 01	55.750	50.000	38.938	62.875	30.188	43.375	70.875	29.938
1 Feb 01	65.500	51.063	45.563	69.313	48.250	62.375	85.250	46.875
1 Mar 01	69.938	47.000	52.313	71.016	37.500	59.359	61.188	48.219
2 Apr 01	61.500	44.188	53.438	57.000	35.313	55.813	51.500	62.188
1 May 01	59.230	48.210	62.190	61.390	54.310	70.170	61.750	91.080
1 Jun 01	61.230	48.700	60.300	68.580	61.250	70.340	61.590	90.350
2 Jul 01	52.900	52.690	54.230	61.670	68.170	70.600	57.870	88.640
1 Aug 01	57.370	59.040	59.870	62.090	61.620	66.470	65.370	85.840

Table 3.1: Sale Prices for 8 NASDAQ Companies

Some values are removed to simulate an incomplete data set, giving the following matrix, with a NaN representing the missing data:

$$P = \begin{bmatrix} 59.875 & 42.734 & 47.938 & 60.359 & 54.016 & 69.625 & 61.500 & 62.125 \\ 53.188 & 49.000 & 39.500 & \mathrm{NaN} & 34.750 & \mathrm{NaN} & 83.000 & 44.500 \\ 55.750 & 50.000 & 38.938 & \mathrm{NaN} & 30.188 & \mathrm{NaN} & 70.875 & 29.938 \\ 65.500 & 51.063 & 45.563 & 69.313 & 48.250 & 62.375 & 85.250 & \mathrm{NaN} \\ 69.938 & 47.000 & 52.313 & 71.016 & \mathrm{NaN} & 59.359 & 61.188 & 48.219 \\ 61.500 & 44.188 & 53.438 & 57.000 & 35.313 & 55.813 & 51.500 & 62.188 \\ 59.230 & 48.210 & 62.190 & 61.390 & 54.310 & 70.170 & 61.750 & 91.080 \\ 61.230 & 48.700 & 60.300 & 68.580 & 61.250 & 70.340 & \mathrm{NaN} & \mathrm{NaN} \\ 52.900 & 52.690 & 54.230 & \mathrm{NaN} & 68.170 & 70.600 & 57.870 & 88.640 \\ 57.370 & 59.040 & 59.870 & 62.090 & 61.620 & 66.470 & 65.370 & 85.840 \end{bmatrix}. \quad (3.1)$$

3.2 Test Machine

All computation was undertaken using MATLAB 6 on a 350 MHz Pentium II running Linux.

4 The Nearest Correlation Matrix Problem

4.1 The Problem

We look at the problem of finding

$$\min\left\{\|A - X\|_F : X \text{ is a correlation matrix with certain elements fixed}\right\} \tag{4.1}$$

where $A = A^T \in \mathbb{R}^{n\times n}$. Our interest is in the case when $A = \overline{\mathrm{COR}}(P)$ is an approximate correlation matrix and has some *exact* entries. $\|A\|_F^2 = \sum_{i,j} a_{ij}^2$ is the Frobenius norm.

We observe that all correlations involving a variable with missing entries will be approximate. From the computation of our approximate correlation matrix we can see that a missing element in P will affect a whole row and column of A. That is, a missing element for the ith random variable will cause the ith row and the ith column to be approximate in the computed correlation matrix.

Since the order of variables in the correlation matrix is arbitrary we can permute any two rows and corresponding columns. So we can arrange our approximate correlation matrix, A, for the data matrix, P, containing k columns of data with no missing entries as

$$\begin{bmatrix} E & B \\ B^T & C \end{bmatrix},$$

where $E = E^T \in \mathbb{R}^{k\times k}$ is the principal submatrix containing the exact correlations between the stocks $1\colon k$, $B \in \mathbb{R}^{k\times(n-k)}$ is approximate as it holds the correlations between stocks that have missing data and those which do not, and $C = C^T \in \mathbb{R}^{(n-k)\times(n-k)}$ contains the approximate correlations between the $n-k$ stocks that have data missing. Note that E will be positive semidefinite as it is an exact correlation matrix.

Thus we seek a nearest correlation matrix X to A such that

$$x_{ij} = e_{ij}, \qquad 1 \leq i, j \leq k.$$

.

In [10] a solution is found to the problem

$$\min \left\{ \|W^{1/2}(A - X)W^{1/2}\|_F : X \text{ is a correlation matrix} \right\}$$

by an *alternating projections* algorithm, where W is a symmetric positive definite matrix of weights. We follow the same approach and compare how using the weighted method to try and preserve exact correlations compares with our direct solution of the problem.

Also we consider an approach of applying *sequential quadratic programming* and ask whether a *matrix completion* method is suitable.

4.2 Alternating Projections

We define

$$\langle A, B \rangle = \text{trace}(A^T B),$$

which is an inner product on $\mathbb{R}^{n\times n}$ that induces the Frobenius norm.

We define also the sets

$$\begin{aligned} \Sigma &= \left\{ Y = Y^T \in \mathbb{R}^{n\times n} : Y \geq 0 \right\}, \\ \Sigma_E &= \left\{ Y = Y^T = \begin{bmatrix} E & B \\ B^T & C \end{bmatrix} \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{k\times(n-k)}, \right. \\ &\qquad\qquad\qquad\qquad \left. C \in \mathbb{R}^{(n-k)\times(n-k)}, y_{ii} \equiv 1 \right\}, \end{aligned}$$

where $Y \geq 0$ means that Y is positive semidefinite.

We seek the matrix in the intersection of Σ and Σ_E which is closest to A in the unweighted Frobenius norm, where E is the exact part of A as described above.

Since our sets are both closed and convex, so is there intersection, so from [14, p. 69], for example, it follows that the minimum in (4.1) is achieved and it is achieved at a unique matrix X.

Characterisation

The solution, X in (4.1), is characterised by the condition [14, p. 69]

$$\langle Z - X, A - X \rangle \leq 0, \qquad \text{for all } Z \in \Sigma \cap \Sigma_E. \tag{4.2}$$

Here, Z is of the form

$$Z = \begin{bmatrix} E & Z_1 \\ Z_1^T & Z_2 \end{bmatrix}. \tag{4.3}$$

The normal cone of a convex set $K \subset \mathbb{R}^{n \times n}$ at $B \in K$ is

$$\begin{aligned} \partial K(B) &= \left\{ Y = Y^T \in \mathbb{R}^{n \times n} : \langle Z - B, Y \rangle \leq 0 \text{ for all } Z \in K \right\} \\ &= \left\{ Y = Y^T \in \mathbb{R}^{n \times n} : \langle Y, B \rangle = \sup_{Z \in K} \langle Y, Z \rangle \right\}. \end{aligned} \tag{4.4}$$

The condition (4.2) can be rewritten as $A - X \in \partial(\Sigma \cap \Sigma_E)(X)$, the normal cone to $\Sigma \cap \Sigma_E$ at X.

For two convex sets K_1 and $K_2, \partial(K_1 \cap K_2)(B) = \partial K_1(B) + \partial K_2(B)$ if the relative interiors of the two set have a point in common [17, Cor. 23.8.1]. Thus we have

$$A - X \in \partial\Sigma(X) + \partial\Sigma_E(X) \tag{4.5}$$

since any matrix of the form

$$\begin{bmatrix} E & B \\ B^T & I \end{bmatrix}$$

which is positive definite (where I is the $(n-k) \times (n-k)$ identity matrix) is in the relative interiors of Σ and Σ_E.

So we assume that E is positive definite, which implies that we must have more observations than stocks with complete data sets, since, as we saw in section 2.1, the rank of E is at most $\min(m-1, k)$. Thus we only consider the case that $m \geq k+1$ and E is positive definite.

From [10] with $W = I$ we have

$$\partial\Sigma(A) = \big\{Y = -VDV^T, \textit{where } V \in \mathbb{R}^{n\times p} \textit{ has orthonormal columns spanning null}(A) \textit{ and } D = \operatorname{diag}(d_i) \geq 0\big\}. \quad (4.6)$$

Lemma 4.1 *For* $A \in \Sigma_E(A)$,

$$\partial\Sigma_E(A) = \left\{Y = Y^T = \begin{bmatrix} F & 0 \\ 0 & H \end{bmatrix} : F \in \mathbb{R}^{k\times k} \textit{ arbitrary}, H = \operatorname{diag}(h_{ii}) \textit{ arbitrary}\right\}.$$

Proof. Any $Z \in \Sigma_E(A)$ is of the form (4.3) with $\operatorname{diag}(Z_2) = I$. Let

$$Y = \begin{bmatrix} F & G \\ G^T & H \end{bmatrix} \in \partial\Sigma_E(A).$$

If $G \neq 0$ or $H \neq \operatorname{diag}(h_{ii})$ we can choose $(Z_1)_{ij}$ or $(Z_2)_{ij}$ in (4.3) arbitrarily large and the same sign as G_{ij} and $H_{ij} \neq 0$, respectively, and violate the sup condition (4.4). Therefore $G = 0$ and $H = \operatorname{diag}(h_{ii})$ and any such Y satisfies the sup condition. □

Write

$$VDV^T = \begin{bmatrix} (VDV^T)_{11} & (VDV^T)_{12} \\ (VDV^T)_{21} & (VDV^T)_{22} \end{bmatrix},$$

where $(VDV^T)_{11} \in \mathbb{R}^{k\times k}$.

Theorem 4.1 *The correlation matrix X solves* (4.1) *if and only if*

$$X = A + VDV^T + \begin{bmatrix} F & 0 \\ 0 & H \end{bmatrix}$$

where $V \in \mathbb{R}^{n \times p}$ has orthonormal columns spanning $\mathrm{null}(X)$*, $D = \mathrm{diag}(d_i) \geq 0$ and $F = -(VDV^T)_{11}$ and $H = \mathrm{diag}(h_{ii})$ is arbitrary.*

Proof The result follows from condition (4.5) on applying (4.6) and Lemma 4.1 and noting that F is completely determined by the need to preserve E. □

Now, if $a_{ii} \geq 1$, which is true in the finance application, we also have the following theorem which generalises [10, Thm. 2.5]

Theorem 4.2 *If A has diagonal elements $a_{ii} \geq 1$ and t nonpositive eigenvalues then the nearest correlation matrix that preserves the exact part, E, has at least t zero eigenvalues.*

Proof. From Theorem 4.1 we have

$$X = A + VDV^T + \begin{bmatrix} F & 0 \\ 0 & H \end{bmatrix},$$

where VDV^T is positive semidefinite, and hence $F = -(VDV)^T_{11}$ and the diagonal matrix H are negative semidefinite (since E is preserved in the former case and since $a_{ii} \geq 1$ in the latter case.) So if A has t nonpositive eigenvalues then

$$A + \begin{bmatrix} F & 0 \\ 0 & H \end{bmatrix} \tag{4.7}$$

has at least t nonpositive eigenvalues, from a standard result for symmetric matrices [11, Thm. 4.3.1]. Now the perturbation VDV^T of rank at most p to (4.7) produces nonnegative eigenvalues, so from a standard result for low rank perturbations [11, Thm. 4.3.6] we must have $p \geq t$. Now p is the dimension of the null space of X, by Theorem 4.1, and hence the result follows. □

Alternating Projections

The idea of *alternating projections* is to find in the intersection of a finite number of sets, $\{S\}_i^n$, a point nearest to some starting point, by repeating the operation

$$A \leftarrow (P_n \dots (P_2(P_1(A))))$$

where P_i is the projection on to the set S_i. The idea was first analysed by von Neumann [20] who showed that if we have two sets that are closed subspaces of a Hilbert space then this iteration converges to the point nearest the starting point.

If we have closed convex sets instead of subspaces it has been shown that the convergence result does not hold, and instead the convergence can be to a non-optimal point [8]. In this case we can use a correction, due to Dykstra [5], for each projection as follows: for n sets and a starting point A,

```
$\Delta_0^i = 0, X_0^i = A, \qquad i = 1\colon n$
for $k = 1, 2, \dots$
    for $i = 1\colon n$
        $\Gamma_k^i = X_{k-1}^{(i+1 \mod n)} - \Delta_{k-1}^i$
        $X_k^i = P_i(\Gamma_k^i)$
        $\Delta_k^i = X_k^i - \Gamma_k^i$
    end
end
```

Applying this algorithm the $X_k^i, i = 1\colon n$, all converge to the desired nearest point [2].

Finally, if a set is the translate of a subspace then the corresponding correction can be omitted [2].

Now Σ and Σ_E are both closed convex sets so we apply an alternating projections algorithm with a correction only for Σ, since Σ_E is a translate of a subspace.

From [10], with $W = I$, the projection onto Σ is

$$P_\Sigma(A) = Q\mathrm{diag}(\max(\lambda_i, 0))Q^T,$$

where $A = Q\Lambda Q^T$ is a spectral decomposition, with Q orthogonal and $\Lambda = \mathrm{diag}(\lambda_i)$.

The projection onto Σ_E is, in view of Lemma 4.1,

$$P_{\Sigma_E}(A) = (p_{ij}), \qquad p_{ij} = \begin{cases} e_{ij}, & 1 \leq i, j \leq k, \\ 1, & i = j > k, \\ a_{ij}, & \text{otherwise.} \end{cases}$$

Algorithm 4.1 *Given the matrix $A = A^T \in \mathbb{R}^{n\times n}$ with exact elements $e_{ij} = a_{ij}, 1 \leq i, j \leq k$, and with $E = (e_{ij})$ positive definite, this algorithm computes the nearest correlation matrix in the Frobenius norm that preserves E.*

```
$\Delta_0 = 0, Y_0 = A$
for $k = 1, 2, \ldots$
    $\Gamma_k = Y_{k-1} - \Delta_{k-1}$        % $\Delta_{k-1}$ is Dykstra's correction
    $X_k = P_\Sigma(\Gamma_k)$
    $\Delta_k = X_k - \Gamma_k$
    $Y_k = P_{\Sigma_E}(X_k)$
end
```

Note if E is not positive semidefinite the alternating projections algorithm will not converge. Every principle sub-matrix of a positive semidefinite matrix is itself positive semidefinite. Thus if E is not then no matrix containing it will be either, thus there is no intersection of the sets Σ and Σ_E and no convergence of the algorithm.

Weighted Norm

We now consider a weighted Frobenius norm from [10]

$$\min\left\{\|W^{1/2}(A - X)W^{1/2}\|_F : X \text{ is a correlation matrix}\right\}, \tag{4.8}$$

for the case where $W^{1/2}$ is diagonal, and with

$$A = \begin{bmatrix} E & B \\ B^T & C \end{bmatrix},$$

with E fixed. We try to encourage E to be preserved by the following weighting

$$W^{1/2} = (w_{ij}) = \begin{cases} f, & i = j \leq k, \\ 1, & i = j > k, \\ 0, & \text{otherwise,} \end{cases}$$

where $f \gg 1$ is chosen to try to force $x_{ij} \approx e_{ij}, 1 \leq i, j \leq k$. A diagonal weighting means, elementwise, for $W^{1/2} = (w_{ii})$ we seek

$$\min \|(w_{ii}(a_{ij} - x_{ij})w_{jj})\|_F.$$

so although E is heavily weighted and C is unweighted we are undesirably weighting B also.

We apply the alternating projection algorithm of [10] which solves (4.8), with no elements in A fixed.

4.3 Experiments

Using our test data (3.1) we generated the approximate correlation matrix $R = \overline{\mathrm{COR}}(P)$

$$R = \begin{bmatrix}
1.0000 & -0.3250 & 0.1881 & 0.5760 & 0.0064 & -0.6111 & -0.0724 & -0.1589 \\
-0.3250 & 1.0000 & 0.2048 & 0.2436 & 0.4058 & 0.2730 & 0.2869 & 0.4241 \\
0.1881 & 0.2048 & 1.0000 & -0.1325 & 0.7658 & 0.2765 & -0.6172 & 0.9006 \\
0.5760 & 0.2436 & -0.1325 & 1.0000 & 0.3041 & 0.0126 & 0.6452 & -0.3210 \\
0.0064 & 0.4058 & 0.7658 & 0.3041 & 1.0000 & 0.6652 & -0.3293 & 0.9939 \\
-0.6111 & 0.2730 & 0.2765 & 0.0126 & 0.6652 & 1.0000 & 0.0492 & 0.5964 \\
-0.0724 & 0.2869 & -0.6172 & 0.6452 & -0.3293 & 0.0492 & 1.0000 & -0.3983 \\
-0.1589 & 0.4241 & 0.9006 & -0.3210 & 0.9939 & 0.5964 & -0.3983 & 1.0000
\end{bmatrix},$$

which has eigenvalues

$$\lambda_R = [\ -0.2498 \quad -0.0160 \quad 0.0895 \quad 0.2192 \quad 0.7072 \quad 1.7534 \quad 1.9611 \quad 3.5355\ ]^T.$$

We first computed the nearest correlation matrix, with E empty, using an unweighted version of the algorithm in [10] (see `near_cor.m` in Appendix A.2), using default tolerances for convergence of the algorithm, namely

$$\frac{\|Y_k - X_k\|_\infty}{\|Y_K\|_\infty} \leq 1.0\text{e}{-5},$$

1.0e-5 for convergence of the eigenvalues found in the MEX routine and 1.0e-4 for defining the positivity of the eigenvalues.

All the MATLAB M-files use a MEX interface for the eigendecomposition instead of using a MATLAB built-in function. This was done to increase the efficiency of the algorithms and details are given in Section 6.

Using `near_cor.m` gave

$$R_N = \begin{bmatrix} 1.0000 & -0.3112 & 0.1889 & 0.5396 & 0.0268 & -0.5925 & -0.0621 & -0.1921 \\ -0.3112 & 1.0000 & 0.2050 & 0.2265 & 0.4148 & 0.2822 & 0.2915 & 0.4088 \\ 0.1889 & 0.2050 & 1.0000 & -0.1468 & 0.7880 & 0.2727 & -0.6085 & 0.8802 \\ 0.5396 & 0.2265 & -0.1468 & 1.0000 & 0.2137 & 0.0015 & 0.6069 & -0.2208 \\ 0.0268 & 0.4148 & 0.7880 & 0.2137 & 1.0000 & 0.6580 & -0.2812 & 0.8762 \\ -0.5925 & 0.2822 & 0.2727 & 0.0015 & 0.6580 & 1.0000 & 0.0479 & 0.5932 \\ -0.0621 & 0.2915 & -0.6085 & 0.6069 & -0.2812 & 0.0479 & 1.0000 & -0.4470 \\ -0.1921 & 0.4088 & 0.8802 & -0.2208 & 0.8762 & 0.5932 & -0.4470 & 1.0000 \end{bmatrix},$$

which has eigenvalues

$$\lambda_{R_N} = [\ 3.3233\text{e}{-17} \quad -2.8662\text{e}{-16} \quad 0.0381 \quad 0.1731 \quad 0.6894$$
$$1.9217 \quad 1.7117 \quad 3.4661\ ]^T.$$

So R_N is a correlation matrix as required. The algorithm converged in 10 iterations in less than half a second, and

$$\|R - R_N\|_F = 0.2960.$$

We now apply Algorithm 4.1 knowing that part of R is exact, namely the upper left corner

$$E = \begin{bmatrix} 1.0000 & -0.3250 & 0.1881 \\ -0.3250 & 1.0000 & 0.2048 \\ 0.1881 & 0.2048 & 1.0000 \end{bmatrix}.$$

This is implemented by `cor_exact.m` (see Appendix A.2). With $k = 3$ and the default tolerances, we obtain

$$R_E = \begin{bmatrix} 1.0000 & -0.3250 & 0.1881 & 0.5375 & 0.0258 & -0.5899 & -0.0625 & -0.1927 \\ -0.3250 & 1.0000 & 0.2048 & 0.2251 & 0.4145 & 0.2838 & 0.2914 & 0.4081 \\ 0.1881 & 0.2048 & 1.0000 & -0.1462 & 0.7882 & 0.2720 & -0.6084 & 0.8805 \\ 0.5375 & 0.2251 & -0.1462 & 1.0000 & 0.2141 & 0.0001 & 0.6071 & -0.2203 \\ 0.0258 & 0.4145 & 0.7882 & 0.2141 & 1.0000 & 0.6570 & -0.2810 & 0.8762 \\ -0.5899 & 0.2838 & 0.2720 & 0.0001 & 0.6570 & 1.0000 & 0.0475 & 0.5929 \\ -0.0625 & 0.2914 & -0.6084 & 0.6071 & -0.2810 & 0.0475 & 1.0000 & -0.4469 \\ -0.1927 & 0.4081 & 0.8805 & -0.2203 & 0.8762 & 0.5929 & -0.4469 & 1.0000 \end{bmatrix}, \tag{4.9}$$

which has eigenvalues

$$\lambda_{R_N} = [\ 1.0359\text{e}{-17} \quad 6.3707\text{e}{-17} \quad 0.0379 \quad 0.1736 \quad 0.6885 \quad 1.9226 \quad 1.7111 \quad 3.4664\ ]^T,$$

which illustrates Theorem 4.2, and

$$\|R - R_E\|_F = 0.2967.$$

The algorithm converged in 10 iterations and again in less than half a second. This matrix is not as near to R as R_N, as expected since R_N is the nearest correlation matrix to R.

We now apply the weighted algorithm (see `cor_weight.m` in Appendix A.2) with default tolerance to try to force E to be preserved.

If we let

$$W = \operatorname{diag}([\ 4.0 \quad 4.0 \quad 4.0 \quad 1.0 \quad 1.0 \quad 1.0 \quad 1.0 \quad 1.0\ ])$$

then we have after 12 iterations

$$R_W = \begin{bmatrix} 1.0000 & -0.3247 & 0.1880 & 0.5667 & 0.0083 & -0.6046 & -0.0711 & -0.1639 \\ -0.3247 & 1.0000 & 0.2048 & 0.2389 & 0.4070 & 0.2762 & 0.2876 & 0.4214 \\ 0.1880 & 0.2048 & 1.0000 & -0.1322 & 0.7680 & 0.2744 & -0.6163 & 0.8989 \\ 0.5667 & 0.2389 & -0.1322 & 1.0000 & 0.2127 & 0.0622 & 0.5974 & -0.1849 \\ 0.0083 & 0.4070 & 0.7680 & 0.2127 & 1.0000 & 0.6585 & -0.2799 & 0.8756 \\ -0.6046 & 0.2762 & 0.2744 & 0.0622 & 0.6585 & 1.0000 & 0.0506 & 0.5740 \\ -0.0711 & 0.2876 & -0.6163 & 0.5974 & -0.2799 & 0.0506 & 1.0000 & -0.4553 \\ -0.1639 & 0.4214 & 0.8989 & -0.1849 & 0.8756 & 0.5740 & -0.4553 & 1.0000 \end{bmatrix}.$$

However,

$$\|R - R_W\|_F = 0.3323.$$

Furthermore, we can see that the forcing in insufficient to preserve E.

By empirical testing we find that with

$$W = \text{ diag } (\begin{bmatrix} 6.8 & 6.8 & 6.8 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix})$$

we have, after 15 iterations

$$R_W = \begin{bmatrix} 1.0000 & -0.3250 & 0.1881 & 0.5720 & 0.0071 & -0.6083 & -0.0719 & -0.1608 \\ -0.3250 & 1.0000 & 0.2048 & 0.2416 & 0.4063 & 0.2744 & 0.2872 & 0.4231 \\ 0.1881 & 0.2048 & 1.0000 & -0.1320 & 0.7665 & 0.2755 & -0.6169 & 0.9001 \\ 0.5720 & 0.2416 & -0.1320 & 1.0000 & 0.2101 & 0.0797 & 0.5960 & -0.1817 \\ 0.0071 & 0.4063 & 0.7665 & 0.2101 & 1.0000 & 0.6568 & -0.2788 & 0.8761 \\ -0.6083 & 0.2744 & 0.2755 & 0.0797 & 0.6568 & 1.0000 & 0.0498 & 0.5738 \\ -0.0719 & 0.2872 & -0.6169 & 0.5960 & -0.2788 & 0.0498 & 1.0000 & -0.4550 \\ -0.1608 & 0.4231 & 0.9001 & -0.1817 & 0.8761 & 0.5738 & -0.4550 & 1.0000 \end{bmatrix}.$$

Note that E is preserved to the figures shown. However now

$$\|R - R_W\|_F = 0.3448,$$

which is significantly bigger than $\|R - R_E\|_F$. This result is not surprising given the undesired weighting of B.

4.4 Sequential Quadratic Programming

We now examine the same problem using Sequential Quadratic Programming (SQP). SQP uses a quasi-Newton method at each iteration, solves a quadratic approximation sub-problem and generates a line search. See [7], for example, for an overview.

From Theorem 4.2 we can write

$$X = x_1 x_1^T + x_2 x_2^T + \cdots + x_{n-t} x_{n-t}^T,$$

and form the constrained optimization problem

$$\min_x \|A - x_1 x_1^T - x_2 x_2^T - \cdots - x_{n-t} x_{n-t}^T\|_F,$$

subject to maintaining the unit diagonal

$$\sum_{k=1}^{n-t} x_{ki}^2 = 1, \qquad i = 1\colon n,$$

and preserving E, which gives the additional $k^2 - k$ constraints for the off-diagonal elements, which reduces to $(k^2 - k)/2$ by symmetry:

$$\sum_{k=1}^{n-t} x_{ki} x_{kj} = e_{ij}, \qquad i = 1\colon k-1, \quad j = i+1\colon k.$$

where x_{ki} is the ith element of x_k, and

$$x = \left[x_1^T, x_2^T, \ldots, x_{n-t}^T\right]^T.$$

We solve this nonlinear equality constrained optimization problem with SQP, using MATLAB's `fmincon` which implements an SQP algorithm, part of its Optimization Toolbox; see [15] for details.

4.5 Experiments

Our test data (3.1) was used once more, with $t = 2$. (See Appendix A.3 for the calling script file `sqp_run.m` and the function `fun.m` and constraint `con.m`.)

The default tolerances for `fmincon` were used for terminating the algorithm. These were all 1.0e-6, for changes in the function value, the constraints and the vector x.

Empirical tests showed that `fmincon` converged to a matrix of ones if x_0, the starting vector for the optimization, was a constant vector, including zero. Also if the any of the x_i is a multiple of another this same non-optimal solution was often found.

It was found that random values of x gave convergence to the desired optimal solution (4.9).

The solution was

$$R_O = \begin{bmatrix} 1.0000 & -0.3250 & 0.1881 & 0.5375 & 0.0257 & -0.5898 & -0.0625 & -0.1928 \\ -0.3250 & 1.0000 & 0.2048 & 0.2251 & 0.4144 & 0.2838 & 0.2914 & 0.4083 \\ 0.1881 & 0.2048 & 1.0000 & -0.1462 & 0.7883 & 0.2722 & -0.6083 & 0.8804 \\ 0.5375 & 0.2251 & -0.1462 & 1.0000 & 0.2142 & 0.0002 & 0.6071 & -0.2202 \\ 0.0257 & 0.4144 & 0.7883 & 0.2142 & 1.0000 & 0.6571 & -0.2808 & 0.8764 \\ -0.5898 & 0.2838 & 0.2722 & 0.0002 & 0.6571 & 1.0000 & 0.0475 & 0.5932 \\ -0.0625 & 0.2914 & -0.6083 & 0.6071 & -0.2808 & 0.0475 & 1.0000 & -0.4470 \\ -0.1928 & 0.4083 & 0.8804 & -0.2202 & 0.8764 & 0.5932 & -0.4470 & 1.0000 \end{bmatrix}$$

where each value is within 3e-04 of those in (4.9) and E is preserved, to the figures shown, as required.

The speed of convergence obviously varies due to the random starting vector x_0, but typically convergence was achieved in 12 seconds, in around 50 iterations with 2500 function calls.

This implies that an SQP method will be much slower for larger matrices, which our finance application involves. So we then tested both the alternating projections and SQP algorithms on an approximate correlation matrix, generated from a data matrix $P \in \mathbb{R}^{60 \times 80}$ of NASDAQ stock with 80 missing elements, exact part $E \in \mathbb{R}^{20 \times 20}$ and $t = 29$. The alternating projection algorithm, with convergence tolerance set to equal that for the SQP algorithm, converged in 1.7 seconds but the SQP algorithm took 2 hours and 59 minutes.

4.6 Matrix Completions

Here we consider whether a *matrix completion* approach is suitable to solve (4.1). Methods are discussed in [12] to *complete* a matrix to be positive (semi)definite from a *partial* (semi)definite matrix. A partial matrix is a matrix where only some elements are *known*, and a partial (semi)definite matrix has all its principal submatrices, comprising of these known entries, individually positive (semi)definite, a necessary condition for the *full* matrix to be so. The theory looks at principal submatrices of an $n \times n$ matrix of size $r \times r, r < n$ of the form

$$\begin{bmatrix} y & b^T & x \\ b & A & c \\ x & c^T & z \end{bmatrix}, \tag{4.10}$$

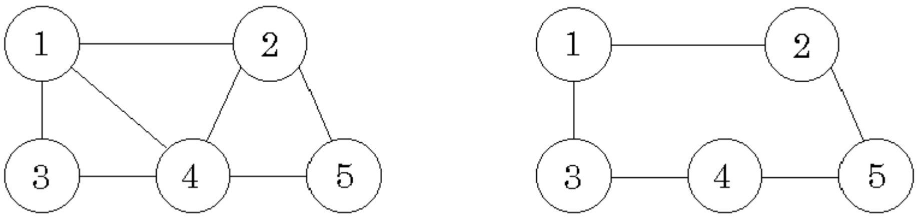
where $A \in \mathbb{R}^{(r-2)\times(r-2)}$ and x is the unknown entry.

For a completion to be possible there is the condition that the *undirected graph* made of the known entries is *chordal*. That is we form a graph with n nodes, joining the ith and jth nodes if the (i, j) element is a known one, and call this line an *edge*; we omit the loops at each node representing the (i, i) element. Now, we define a *simple circuit* as a collection of nodes joined in a loop with no other intersections across that loop. Finally if our graph contains no simple circuits of length four or more than the graph is said to be chordal.

To demonstrate, consider the following two symmetric matrices, with known entries marked X, and unknown entries marked ?:

$$A = \begin{bmatrix} \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & ? \\ \mathrm{X} & \mathrm{X} & ? & \mathrm{X} & \mathrm{X} \\ \mathrm{X} & ? & \mathrm{X} & \mathrm{X} & ? \\ \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ ? & \mathrm{X} & ? & \mathrm{X} & \mathrm{X} \end{bmatrix}, \qquad B = \begin{bmatrix} \mathrm{X} & \mathrm{X} & \mathrm{X} & ? & ? \\ \mathrm{X} & \mathrm{X} & ? & ? & \mathrm{X} \\ \mathrm{X} & ? & \mathrm{X} & \mathrm{X} & ? \\ ? & ? & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ ? & \mathrm{X} & ? & \mathrm{X} & \mathrm{X} \end{bmatrix}.$$

Then we have the corresponding graphs:



A's graph is chordal, but B's is not, as there is a simple circuit of length 5.

It is obvious that the chordal condition is met in our case. All our known elements form the full matrix E, and a full matrix clearly gives a chordal graph, since every node is connected to every other node.

One approach to solve (4.1) could be to form submatrices like (4.10) taking x to be each of the approximate entries in turn. In [12] formula are given to calculate an interval for x, so we can use this to set each approximate entry to one that is closest to its original value within this interval. However, any approximate entry will be in a row and column of unknown entries. Thus, even for $r = 3$, $c \in \mathbb{R}$ is unknown and we must take z as the diagonal entry. So we are forced to trust an approximate entry to give c with no knowledge what the cumulative effect of this would be. Also, we do not have a strategy for ordering the approximate values, noting that subsequent submatrices will use previously adjusted approximate values. Thus completion methods are of no use for solving our problem.

4.7 Conclusions

Using an unweighted version of the alternating projections algorithm in [10] is clearly inappropriate as it fails to preserve the correlations that are known to be exact. The weighted algorithm is also unsuitable as the weighting needed to preserve the exact elements undesirably weights B. However Algorithm 4.1 produces the desired optimal matrix.

This SQP method is clearly capable of finding the optimal solution for suitable starting values, but as the timings show this method is too slow. The algorithm takes nearly three hours to converge for $n = 80$ and the finance application requires $n > 1000$.

However, the timings for the alternating projections algorithm are encouraging. Also in [10] its shown that the unweighted algorithm for a matrix of $n = 1399$ converges to the solution in 37 minutes, using the same MEX interface, on a 1Ghz Pentium III, so we conclude that this method is of practical use and is indeed being used by the finance company.

5 The Nearest Covariance Matrix Problem

Another problem posed by the finance company is concerned with covariance matrices. We wish to determine how the missing elements in P should be chosen to give the nearest covariance matrix to an approximate one.

5.1 The Problem

Again we have a $P \in \mathbb{R}^{m\times n}$ data matrix. And it is first required to compute $\mathrm{Ln}(P) \in \mathbb{R}^{(m-1)\times n}$, which is a standard procedure for financial data as the resulting matrix is considered easier and more appropriate to work with, where

$$\mathrm{Ln}(P) = l_{ij} = \ln(p_{i+1,j}/p_{i,j}), \qquad i = 1\colon m-1, \quad j = 1\colon n,$$

where both $p_{i+1,j}$ and $p_{i,j}$ are not missing. Clearly for each missing entry in P we have two undefined entries in $\mathrm{Ln}(P)$. If either $p_{i+1,j}$ or $p_{i,j}$ is missing then we set $l_{ij} = \mathrm{NaN}$. We note that $\mathrm{Ln}(P)$ can be any matrix in $\mathbb{R}^{(m-1)\times n}$ for suitable choice of P.

We then form the approximate sample covariance matrix $\overline{\mathrm{COV}}(\mathrm{Ln}(P)) \in \mathbb{R}^{n\times n}$ using the method described in Section 2.2.

Definition: An *extension* of a matrix $P \in \mathbb{R}^{m\times n}$ with missing data is defined by $P_E \in \mathbb{R}^{m\times n}$ having no missing data and if p_{ij} is not missing then $p_{E_{ij}} = p_{ij}$

The problem is to find the extension P_E of P that solves

$$\min_{P_E} \|\overline{\mathrm{COV}}(\mathrm{Ln}(P)) - \mathrm{COV}(\mathrm{Ln}(P_E))\|_F. \tag{5.1}$$

We make the observation that if we find the nearest covariance matrix to $\overline{\mathrm{COV}}(\mathrm{Ln}(P))$ of the form $\mathrm{COV}(L)$, then should we be able to find L we cannot recover P_E from $L = \mathrm{Ln}(P_E)$ as, in general,

$$p_{ij} = \frac{p_{i+1j}}{\exp(l_{ij})} \neq \exp(l_{i-1j})p_{i-1j}.$$

We also note that it is not clear that we can find the solution $\mathrm{COV}(\mathrm{Ln}(P_E))$ that is equal to the nearest covariance matrix. For example consider using the data in Table 3.1 to form a data matrix with one missing element:

$$P_1 = \begin{bmatrix} 59.875 & 42.734 & 47.938 & 60.359 & 54.016 & 69.625 & 61.500 & \mathrm{NaN} \\ 53.188 & 49.000 & 39.500 & 64.813 & 34.750 & 56.625 & 83.000 & 44.500 \\ 55.750 & 50.000 & 38.938 & 62.875 & 30.188 & 43.375 & 70.875 & 29.938 \\ 65.500 & 51.063 & 45.563 & 69.313 & 48.250 & 62.375 & 85.250 & 46.875 \\ 69.938 & 47.000 & 52.313 & 71.016 & 37.500 & 59.359 & 61.188 & 48.219 \\ 61.500 & 44.188 & 53.438 & 57.000 & 35.313 & 55.813 & 51.500 & 62.188 \\ 59.230 & 48.210 & 62.190 & 61.390 & 54.310 & 70.170 & 61.750 & 91.080 \\ 61.230 & 48.700 & 60.300 & 68.580 & 61.250 & 70.340 & 61.590 & 90.350 \\ 52.900 & 52.690 & 54.230 & 61.670 & 68.170 & 70.600 & 57.870 & 88.640 \\ 57.370 & 59.040 & 59.870 & 62.090 & 61.620 & 66.470 & 65.370 & 85.840 \end{bmatrix}.$$

Now, $S = \overline{\mathrm{COV}}(\mathrm{Ln}(P_1))$ is

$$S = \begin{bmatrix} 0.0117 & -0.0016 & 0.0090 & 0.0066 & 0.0096 & 0.0076 & -0.0000 & 0.0003 \\ -0.0016 & 0.0057 & -0.0036 & 0.0027 & 0.0003 & -0.0004 & 0.0128 & -0.0006 \\ 0.0090 & -0.0036 & 0.0152 & 0.0025 & 0.0202 & 0.0153 & -0.0039 & 0.0155 \\ 0.0066 & 0.0027 & 0.0025 & 0.0117 & 0.0073 & 0.0072 & 0.0118 & 0.0047 \\ 0.0096 & 0.0003 & 0.0202 & 0.0073 & 0.0901 & 0.0527 & 0.0161 & 0.0516 \\ 0.0076 & -0.0004 & 0.0153 & 0.0072 & 0.0527 & 0.0385 & 0.0136 & 0.0468 \\ -0.0000 & 0.0128 & -0.0039 & 0.0118 & 0.0161 & 0.0136 & 0.0425 & 0.0255 \\ 0.0003 & -0.0006 & 0.0155 & 0.0047 & 0.0516 & 0.0468 & 0.0255 & 0.0738 \end{bmatrix}$$

and the nearest covariance matrix is the nearest positive semidefinite matrix, given by $S_N = P_\Sigma(S)$:

$$S_N = \begin{bmatrix} 0.0118 & -0.0014 & 0.0089 & 0.0066 & 0.0097 & 0.0074 & -0.0002 & 0.0005 \\ -0.0014 & 0.0059 & -0.0037 & 0.0028 & 0.0004 & -0.0007 & 0.0126 & -0.0000 \\ 0.0089 & -0.0037 & 0.0152 & 0.0024 & 0.0202 & 0.0154 & -0.0039 & 0.0154 \\ 0.0066 & 0.0028 & 0.0024 & 0.0117 & 0.0073 & 0.0071 & 0.0118 & 0.0048 \\ 0.0097 & 0.0004 & 0.0202 & 0.0073 & 0.0901 & 0.0525 & 0.0160 & 0.0517 \\ 0.0074 & -0.0007 & 0.0154 & 0.0071 & 0.0525 & 0.0389 & 0.0138 & 0.0465 \\ -0.0002 & 0.0126 & -0.0039 & 0.0118 & 0.0160 & 0.0138 & 0.0426 & 0.0253 \\ 0.0005 & -0.0003 & 0.0154 & 0.0048 & 0.0517 & 0.0465 & 0.0253 & 0.0740 \end{bmatrix},$$

with

$$||S - S_N||_F = 0.0012.$$

Empirical tests show that there is no value to replace the NaN in P_1 that can give $S_N = \mathrm{COV}(\mathrm{Ln}(P_1))$. This is not surprising, we have far less variables than we had in the nearest correlation matrix problem.

We optimize with the missing elements as variables.

5.2 Multi-Directional Search Optimization

Multi-Directional Search (MDS) is a direct search method. Direct search methods use function values but not derivatives to determine the search direction, requiring only that the function be continuous. These methods are used when derivatives are not available or are ill-behaved in the domain of interest. At each iteration the function is evaluated on a given set of points including the current iterate. The MDS algorithm uses a simplex, and analysis of the function values generates the next set of points. See [18], [19] and [4] for details of this algorithm.

For r missing elements, e, in P we seek

$$\min_{e_1,\dots,e_r} ||\overline{\mathrm{COV}}(\mathrm{Ln}(P)) - \mathrm{COV}(\mathrm{Ln}(P(e_1,\dots,e_r)))||_F$$

where $e_1, \dots, e_r$ denote the missing elements of P, the $r = 9$ NaNs in (3.1) for example, which form a vector for our optimization.

We use `mdsmax`, which is part the Test Matrix Toolbox [9]. This routine aims to *maximise* a given function, thus we supply a function of the form

$$-\|\overline{\mathrm{COV}}(\mathrm{Ln}(P)) - \mathrm{COV}(\mathrm{Ln}(P(e_1, \dots, e_r)))\|_F,$$

and note that a function value of zero corresponds to a 'perfect' extension.

5.3 Experiments

From our test data we have L= Ln(P) (see `gen_lnp.m` in Appendix A.1):

$$L = \begin{bmatrix} -0.1184 & 0.1368 & -0.1936 & \mathrm{NaN} & -0.4411 & \mathrm{NaN} & 0.2998 & -0.3337 \\ 0.0470 & 0.0202 & -0.0143 & \mathrm{NaN} & -0.1407 & \mathrm{NaN} & -0.1579 & -0.3964 \\ 0.1612 & 0.0210 & 0.1571 & \mathrm{NaN} & 0.4690 & \mathrm{NaN} & 0.1847 & \mathrm{NaN} \\ 0.0656 & -0.0829 & 0.1381 & 0.0243 & \mathrm{NaN} & -0.0496 & -0.3316 & \mathrm{NaN} \\ -0.1286 & -0.0617 & 0.0213 & -0.2199 & \mathrm{NaN} & -0.0616 & -0.1724 & 0.2544 \\ -0.0376 & 0.0871 & 0.1517 & 0.0742 & 0.4305 & 0.2289 & 0.1815 & 0.3816 \\ 0.0332 & 0.0101 & -0.0309 & 0.1108 & 0.1203 & 0.0024 & \mathrm{NaN} & \mathrm{NaN} \\ -0.1462 & 0.0787 & -0.1061 & \mathrm{NaN} & 0.1070 & 0.0037 & \mathrm{NaN} & \mathrm{NaN} \\ 0.0811 & 0.1138 & 0.0989 & \mathrm{NaN} & -0.1010 & -0.0603 & 0.1219 & -0.0321 \end{bmatrix}.$$

This matrix gives the following approximate sample covariance matrix $S = \overline{\mathrm{COV}}(\mathrm{Ln}(P))$:

$$S = \begin{bmatrix} 0.0117 & -0.0016 & 0.0090 & 0.0102 & 0.0140 & -0.0016 & -0.0018 & -0.0082 \\ -0.0016 & 0.0057 & -0.0036 & 0.0063 & -0.0079 & 0.0040 & 0.0176 & -0.0081 \\ 0.0090 & -0.0036 & 0.0152 & 0.0024 & 0.0329 & 0.0034 & -0.0072 & 0.0330 \\ 0.0102 & 0.0063 & 0.0024 & 0.0222 & -0.0057 & 0.0100 & 0.0151 & 0.0187 \\ 0.0140 & -0.0079 & 0.0329 & -0.0057 & 0.1046 & 0.0270 & 0.0038 & 0.1155 \\ -0.0016 & 0.0040 & 0.0034 & 0.0100 & 0.0270 & 0.0123 & 0.0214 & 0.0260 \\ -0.0018 & 0.0176 & -0.0072 & 0.0151 & 0.0038 & 0.0214 & 0.0557 & -0.0023 \\ -0.0082 & -0.0081 & 0.0330 & 0.0187 & 0.1155 & 0.0260 & -0.0023 & 0.1192 \end{bmatrix}.$$

The eigenvalues of S are

$$\lambda_S = [\ -0.0244 \quad -0.0022 \quad -0.0011 \quad 0.0024 \quad 0.0241 \quad 0.0271 \quad 0.0760 \quad 0.2446\ ]^T.$$

We have a lower bound for (5.1), given by the nearest covariance matrix $P_\Sigma(S)$

$$\|S - P_\Sigma(S)\|_F = 0.0245.$$

We now call `mdsmax` with `[x,fmax,nf]=mdsmax(@mdsfun,x0,stop)` where the inputs are respectively our function to be maximised (see Appendix A.4 for `mdsfun.m`), a starting vector and a vector of stopping criteria and options. An iteration is terminated if the relative size of the simplex is less than or equal to `stop(1)` (we use 1e-04), the maximum number `stop(2)` of allowed function evaluations is exceeded (we use `inf`), or if the maximum allowed value `stop(3)` for the function evaluations is exceeded (we obviously use 0.) We also set `stop(4)=0` for a regular simplex and `stop(5)=1` to output progress of the iteration.

The outputs give the vector giving the maximum function value, the function value at that point and the number of function evaluations, respectively.

We try several starting vectors, ordered so the first element represents the (2,4) element in (3.1) and then continues column-wise from top to bottom.

First we try an initial vector of values such that their difference is equal to the two known entries above and below in P, that is,

$$x_0 = \begin{pmatrix} 63.3 & 66.3 & 65.3 & 41.5 & 67.2 & 65.0 & 59.8 & 39.0 & 89.8 \end{pmatrix}. \tag{5.2}$$

After 80 iteration with 1468 function calls the algorithm converged to the solution

$$x_1 = (75.8982 \quad 60.5820 \quad 59.2138 \quad 30.9588 \quad 72.69031$$
$$57.315 \quad 45.1410 \quad 58.3263 \quad 108.2271),$$

with

$$\|\overline{\mathrm{COV}}(\mathrm{Ln}(P)) - \mathrm{COV}(\mathrm{Ln}(P(x)))\|_F = 0.0605.$$

We now try initial values of the smallest integer value, rounded down, for each column containing the missing entry, that is

$$x_0 = \begin{pmatrix}57.0 & 57.0 & 57.0 & 30.0 & 55.0 & 55.0 & 57.0 & 29.0 & 29.0\end{pmatrix}. \tag{5.3}$$

After 70 iteration with 1324 function calls the algorithm converged to

$$x_2 = (72.5486 \quad 59.1316 \quad 71.4367 \quad 30.4268 \quad 72.5953$$
$$57.3579 \quad 76.4600 \quad 57.990 \quad 71.3162).$$

Here

$$\|\overline{\mathrm{COV}}(\mathrm{Ln}(P)) - \mathrm{COV}(\mathrm{Ln}(P(x)))\| = 0.0627.$$

We then try an initial vector of the highest integer value, rounded up, for each column containing the missing entry,

$$x_0 = \begin{pmatrix}71.0 & 71.0 & 71.0 & 69.0 & 71.0 & 71.0 & 86.0 & 92.0 & 92.0\end{pmatrix}.$$

This gave the same final vector, x_2, as (5.3) gave with the same number of iterations and function calls.

It appears we have two local minima, and we are uncertain if one is a global minimum. We also try very low and very high values (constant vectors of 2s and 200s) and they converge to the same minimum for that of (5.2).

Each convergence took around 90 seconds to compute.

If we replace our missing entries in (3.1) with these two minima, we have from x_1

$$P_{E1} = \begin{bmatrix}
59.875 & 42.734 & 47.938 & 60.359 & 54.016 & 69.625 & 61.500 & 62.125 \\
53.188 & 49.000 & 39.500 & \mathbf{75.898} & 34.750 & \mathbf{72.690} & 83.000 & 44.500 \\
55.750 & 50.000 & 38.938 & \mathbf{60.582} & 30.188 & \mathbf{57.315} & 70.875 & 29.938 \\
65.500 & 51.063 & 45.563 & 69.313 & 48.250 & 62.375 & 85.250 & \mathbf{58.326} \\
69.938 & 47.000 & 52.313 & 71.016 & \mathbf{30.959} & 59.359 & 61.188 & 48.219 \\
61.500 & 44.188 & 53.438 & 57.000 & 35.313 & 55.813 & 51.500 & 62.188 \\
59.230 & 48.210 & 62.190 & 61.390 & 54.310 & 70.170 & 61.750 & 91.080 \\
61.230 & 48.700 & 60.300 & 68.580 & 61.250 & 70.340 & \mathbf{45.141} & \mathbf{108.227} \\
52.900 & 52.690 & 54.230 & \mathbf{59.214} & 68.170 & 70.600 & 57.870 & 88.640 \\
57.370 & 59.040 & 59.870 & 62.090 & 61.620 & 66.470 & 65.370 & 85.840
\end{bmatrix},$$

and from x_2

$$P_{E2} = \begin{bmatrix} 59.875 & 42.734 & 47.938 & 60.359 & 54.016 & 69.625 & 61.500 & 62.125 \\ 53.188 & 49.000 & 39.500 & \mathbf{72.549} & 34.750 & \mathbf{72.595} & 83.000 & 44.500 \\ 55.750 & 50.000 & 38.938 & \mathbf{59.132} & 30.188 & \mathbf{57.358} & 70.875 & 29.938 \\ 65.500 & 51.063 & 45.563 & 69.313 & 48.250 & 62.375 & 85.250 & \mathbf{57.990} \\ 69.938 & 47.000 & 52.313 & 71.016 & \mathbf{30.427} & 59.359 & 61.188 & 48.219 \\ 61.500 & 44.188 & 53.438 & 57.000 & 35.313 & 55.813 & 51.500 & 62.188 \\ 59.230 & 48.210 & 62.190 & 61.390 & 54.310 & 70.170 & 61.750 & 91.080 \\ 61.230 & 48.700 & 60.300 & 68.580 & 61.250 & 70.340 & \mathbf{76.460} & \mathbf{71.316} \\ 52.900 & 52.690 & 54.230 & \mathbf{71.437} & 68.170 & 70.600 & 57.870 & 88.640 \\ 57.370 & 59.040 & 59.870 & 62.090 & 61.620 & 66.470 & 65.370 & 85.840 \end{bmatrix}.$$

Note we can reduce the amount of calculations in this method if we add an `if` statement to `cov_bar` to calculate only covariances for i or j greater than k. This makes a saving of $O(k^2m)$ floating point operations. And we make appropriate alterations in `mdsfun`, to obtain the correct value of the norm.

5.4 Conclusions

With careful choice of starting vectors this method can provide some insight into a possible solution. Now, no financial analysis is offered here, but it worth noting that the values 45.141 and 108.227 in P_{E_1} appear unrealistic (they represent the smallest and largest values in their column respectively) compared to their corresponding values in P_{E_2}, but $\mathrm{COV}(\mathrm{Ln}(P_{E_1}))$ is nearer to $\overline{\mathrm{COV}}(\mathrm{Ln}(P))$. Since the problem is to find the missing values of P it is not obvious that we can accept the nearest matrix without some financial interpretation.

6 Efficient Implementation

Our algorithms for the alternating projections method of solving the nearest correlation problem require us to find only the positive eigenvalues of a symmetric matrix, and their associated eigenvectors. Thus it is obvious that calling MATLAB's `eig` function is wasteful as it returns all the eigenvalues and vectors. MATLAB does, however, supply the `eigs` function that can return a specified range of eigenvalues. However, `eigs` uses an iterative method and is most suited to sparse matrices, and ours, of course, are dense. Obtaining these required eigenvalues and vectors is clearly the most expensive part of the algorithm, thus we attempt to speed this process up by writing a MATLAB MEX file that calls an appropriate LAPACK routine.

6.1 MEX files

MATLAB allows you to write Fortran and C subroutines and use them as if they were your own M-file routines. These *MEX* files are dynamically linked subroutines that the MATLAB interpreter can automatically load and execute.

The motivation for this feature is to allow users to use pre-existing Fortran and C code without the need to rewrite them as M-files and also to increase efficiency by overcoming bottlenecks in MATLAB such as its `for` loops. Here we implement a C MEX file to enable us to call an LAPACK routine directly.

6.2 LAPACK

LAPACK [1] is transportable collection of linear algebra subroutines designed to be efficient on a wide range of modern high-performance computers. MATLAB 6 itself is built on LAPACK. We use the library routine `desevr` (all LAPACK routines are supplied with MATLAB 6) to obtain the desired positive eigenvalues and their associated eigenvectors. This routine reduces the matrix to tridiagonal form and then uses a bisection method and inverse iteration.

6.3 Some Timings

We compare the performance of our MEX routine (see Appendix B for MEX source code file `eig_mex.c`) against MATLAB's `eigs`. We use an approximate correlation matrix, supplied by the finance company, of size 1399, and compute different numbers of its largest eigenvalues. See Table 6.1.

Number of eigenvalues	Time with `eigs` (secs)	Time with `eig_mex` (secs)
280	278.7	52.9
140	75.4	42.1
70	56.7	38.9
28	29.2	37.4

Table 6.1: Comparison of MATLAB's `eigs` vs. `eig_mex` MEX file for a dense correlation matrix

So the MEX subroutine is clearly more efficient than `eigs` for our dense matrix when we are computing more than a small number of eigenvalues.

7 Concluding Remarks

The Nearest Correlation Matrix Problem

For the problem of computing the nearest correlation matrix to a symmetric matrix with fixed elements, we have examined the suitability of three different approaches, namely alternating projections, sequential quadratic programming and matrix completion methods. We have found that the method of alternating projections is the only efficient method to guarantee a solution. This extends the theory and algorithm of [10]. Also, this method is fast enough for practical use.

The Nearest Covariance Matrix Problem

For the problem of computing the nearest covariance matrix we have investigated possible solutions using a multi-directional search optimization method. We have found that this method can produce a solution; however there is some uncertainty as to the usefulness of this solution. Further work is needed to establish the underlying theory of the problem and also some financial analysis of the solutions obtained is required.

Appendices

A MATLAB M-Files

A.1 Computation of S and R M-files

gen_cov.m

This routines produces the same output as `cov(P)` in MATLAB.

```
function S=gen_cov(P)
%GEN_COV  Calculates sample covariance matrix.
%
%   S=GEN_COV(P)
%
%   Produces an n-by-n covariance matrix based on
%   data of size m-by-n. n columns of different
%   random variables observed at m different times.
%
%   INPUT:  P  data matrix
%
%   OUTPUT: S  sample covariance matrix

[m,n]=size(P);
I=eye(m);
O=ones(m)/(m);

S=(1/(m-1))*P'*(I-O)*P;

% ensure symmetry
S=(S+S')/2;
```

gen_cor.m

This routines produces the same output as `corrcoef(P)` in MATLAB.

```
function R=gen_cor(P)
```

```
%GEN_COR  Calculates sample correlation matrix.
%
%   S=GEN_COR(P)
%
%   Produces an n-by-n correlation matrix based on
%   data of size m-by-n. n columns of different
%   random variables observed at m different times.
%
%   INPUT:  P  data matrix
%
%   OUTPUT: R  sample correlation matrix

[m,n]=size(P);
S=gen_cov(P);
D=diag(1./sqrt(diag(S)));

R=D*S*D;
```

cov_bar.m

```
function S = cov_bar(P)
%COV_BAR  Calculates approximate sample covariance matrix.
%
%   S=COV_BAR(P)
%
%   Produces an n-by-n approx covariance matrix based on
%   data of size m-by-n. n columns of different
%   random variables observed at m different times.
%   P has missing data represented by NaNs.
%
%   INPUT:  P  data matrix
%
%   OUTPUT: S  approx sample covariance matrix

[m,n] = size(P);
```

```
S = zeros(n);

for i = 1:n
    xi = P(:,i);

    for j=1:i
    xj = P(:,j);

        % create mask for data values that are 'common'
        p = ~isnan(xi) & ~isnan(xj);

        S(i,j) = (xi(p) - mean(xi(p)))'*( xj(p) - mean(xj(p)));

        % normalise over effective sample size i.e. sum(p)-1
        S(i,j) = 1/(sum(p)-1)*S(i,j);

        S(j,i) = S(i,j);

    end
end
```

cor_bar.m

```
function R = cor_bar(P)
%COR_BAR  Calculates approximate sample correlation matrix.
%
%   S=COR_BAR(P)
%
%   Produces an n-by-n approx correlation matrix based on
%   data of size m-by-n. n columns of different
%   random variables observed at m different times.
%   P has missing data represented by NaNs.
%
%   INPUT:  P  data matrix
%
%   OUTPUT: R  approx sample correlation matrix
```

```
[m,n]=size(P);
S=cov_bar(P);
D=diag(1./sqrt(diag(S)));

R=D*S*D;
```

gen_lnp.m

```
function L=gen_lnp(P)
%GEN_LNP  Compute L(i,j)=Ln(P(i+1,j)/P(i,j))
%
%   L = GEN_LNP(P)
%
%   If either P(i+1,j) or P(i,j) is NaN then
%   L(i,j)=NaN.
%
%   INPUT:  P   (m+1)-by-n matrix
%
%   OUTPUT: L   n-by-n matrix

[m,n]=size(P);

k=m-1;

for j=1:n
    for i=1:k

        L(i,j)=log(P(i+1,j)/P(i,j));

    end
end
```

A.2 Alternating Projection M-Files

The convergence criterion is taken from [10].

near_cor.m

```
function X=near_cor(A,tol,maxits)
%NEAR_COR  Computes the nearest correlation matrix.
%
%   X = NEAR_COR(A,tol,maxits)
%
%   Computes the nearest correlation matrix
%   to an approximate correlation matrix,
%   i.e. not positive semidefinite.
%
%   INPUT:  A        n-by-n approx correlation matrix
%           tol      vector of size three or omit for defaults
%           tol(1)   convergence tolerance for algorithm,
%                    default 1.0e-5
%           tol(2)   convergence tolerance for eig_mex mex
%                    routine, default 1.0e-5
%           tol(3)   defines relative positiveness of
%                    eigenvalues compared to largest,
%                    default 1.0e-4
%           maxits   maximum number of iterations allowed
%
%                    tol optional, maxits optional if tol incl.
%
%   OUTPUT: X        nearest correlation matrix to A

if  ~isequal(A,A')
    error('Error: Input matrix A must be square and symmetric')
end

if nargin < 2
    conv_tol = 1.0e-5;
    mex_tol = 1.0e-5;
    eig_tol = 1.0e-4;
else
```

```
    conv_tol = tol(1);
    mex_tol = tol(2);
    eig_tol = tol(3);
end

if nargin < 3, maxits = 100; end

[m,n]=size(A);

U=zeros(n);
Y=A;

iter=0;

[V,D]=eig(Y);
d=diag(D);

% define 'positiveness' relative to largest eigenvalue
num_pos= sum(d >= eig_tol*d(n));

while 1

    T=Y-U;

    % PROJECT ONTO PSD MATRICES
    [Q,d]=eig_mex(T,num_pos,mex_tol);

    D=diag(d);

    % create mask from relative positive eigenvalues
    p=(d>eig_tol*d(n));

    % use p mask to only compute 'positive' part
    X=Q(:,p)*D(p,p)*Q(:,p)';

    % UPDATE DYKSTRA'S CORRECTION
    U=X-T;

    % PROJECT ONTO UNIT DIAG MATRICES
    Y=X;
    for i=1:n
        Y(i,i)=1;
    end

    iter = iter + 1;

    if iter==maxits
        fprintf('Max its exceeded'), break, end

    % convergence test
    if norm(Y-X,'inf')/norm(Y,'inf') <= conv_tol, break,end

end

fprintf('||A-X||_F: %2.4f\n',norm(A-X,'fro'))
fprintf('Number of iterations taken: %4.0f\n',iter)
```

cor_exact.m

```
function X=cor_exact(A,k,tol,maxits)
%COR_EXACT  Computes the nearest correlation matrix w/exact part.
%
%   X = COR_EXACT(A,k,tol,maxits)
%
%   Computes the nearest correlation matrix to an approximate
%   correlation matrix (not positive semidefinite) w/exact part.
%
%   INPUT:  A        n-by-n approx correlation matrix,
%                    with exact part of the form | E B^T |
%                                                | B C   |
%                    E is k-by-k and is exact
%                    C is (n-k)-by-(n-k), B and C are approx.
%                    E must be psd
%           k        size of E
%           tol      vector of size three or omit for defaults
%           tol(1)   convergence tolerance for algorithm,
%                    default 1.0e-5
%           tol(2)   convergence tolerance for eig_mex mex
%                    routine, default 1.0e-5
%           tol(3)   defines relative positiveness of
%                    eigenvalues compared to largest,
%                    default 1.0e-4
%           maxits   maximum number of iterations allowed
%
%                    tol optional, maxits optional if tol incl.
%
%   OUTPUT: X        nearest correlation matrix to A

[m,n]=size(A);

if (nargin > 1) & (k <= n)
    E=A(1:k,1:k);
    d=eig(E);
end

if nargin < 3
    conv_tol = 1.0e-5;
    mex_tol = 1.0e-5;
    eig_tol = 1.0e-4;
else
    conv_tol = tol(1);
    mex_tol = tol(2);
    eig_tol = tol(3);
end

if  ~isequal(A,A')
    error('Error: Input matrix A must be square and symmetric')
elseif nargin < 2
    error('Error: k must be specified')
elseif k > n
    error('Error: k too large')
elseif sum(d >= eig_tol*d(k)) ~= k
    error('Error: E must be positive semidefinite')
end
```

```
if nargin < 4, maxits = 100; end

U=zeros(n);
Y=A;

iter=0;

[V,D]=eig(Y);
d=diag(D);

% define 'positiveness' relative to largest eigenvalue
num_pos= sum(d >= eig_tol*d(n));

while 1

    T=Y-U;

    % PROJECT ONTO SIGMA
    [Q,d]=eig_mex(T,num_pos,mex_tol);

    D=diag(d);

    % create mask from relative positive eigenvalues
    p=(d>eig_tol*d(n));

    % use p mask to only compute 'positive' part
    X=Q(:,p)*D(p,p)*Q(:,p)';

    % UPDATE DYKSTRA'S CORRECTION
    U=X-T;

    % PROJECT ONTO SIGMA_E
    Y=X;
    Y(1:k,1:k)=E;

    for i=k+1:n
        Y(i,i)=1;
    end
    iter = iter + 1;

    if iter==maxits
        fprintf('Max its exceeded'), break, end

    % convergence test
    if norm(Y-X,'inf')/norm(Y,'inf') <= conv_tol, break,end

end

fprintf('||A-X||_F: %2.4f\n',norm(A-X,'fro'))
fprintf('Number of iterations taken: %4.0f\n',iter)
```

cor_weight.m

```
function X=cor_weight(A,W,tol,maxits)
%COR_WEIGHT  Computes nearest correction matrix, weighted.
%
%   X = COR_WEIGHT(A,W,tol,maxits)
%
%   Computes the nearest correlation matrix to an approximate
%   correlation matrix (not positive semidefinite) subject
%   to weighting, i.e min|| W^(1/2) ( A-X ) W^(1/2)||_F
%   where W is diagonal. Note: input W is W^(1/2).
%
%   INPUT:  A        n-by-n approx correlation matrix
%           W        diaganol matrix of weights
%           tol      vector of size three or omit for defaults
%           tol(1)   convergence tolerance for algorithm,
%                    default 1.0e-5
%           tol(2)   convergence tolerance for eig_mex mex
%                    routine, default 1.0e-5
%           tol(3)   defines relative positiveness of
%                    eigenvalues compared to largest,
%                    default 1.0e-4
%           maxits   maximum number of iterations allowed
%
%                    tol optional, maxits optional if tol incl.
%
%   OUTPUT: X        nearest correlation matrix to A

[m,n]=size(A);
[mw,nw]=size(W);

if  ~isequal(A,A’)
    error(’Error: Input matrix A must be square and symmetric’)
elseif nargin < 2
    error(’Error: W must be specified’)
elseif ~isequal(A,A’)
    error(’Error: W must be square and symmetric’)
elseif n ~= nw
    error(’Error: A and W must be conformable’)
end

if nargin < 3
    conv_tol = 1.0e-5;
    mex_tol = 1.0e-5;
    eig_tol = 1.0e-4;
else
    conv_tol = tol(1);
    mex_tol = tol(2);
    eig_tol = tol(3);
end

if nargin < 4, maxits = 100; end

U=zeros(n);
Y=A;

iter=0;

Winv=W^(-1);

% weighting preserves inertia we can use Y
```

```
[V,D]=eig(Y);
d=diag(D);

% define 'positiveness' relative to largest eigenvalue
num_pos= sum(d >= eig_tol*d(n));

while 1

    T=Y-U;

    % PROJECT ONTO PSD MATRICES
    [Q,d]=eig_mex(W*T*W,num_pos,mex_tol);

    D=diag(d);

    % create mask from relative positive eigenvalues
    p=(d>eig_tol*d(n));

    % use p mask to only compute 'positive' part
    X=Winv*(Q(:,p)*D(p,p)*Q(:,p)')*Winv;

    % UPDATE DYKSTRA'S CORRECTION
    U=X-T;

    % PROJECT ONTO UNIT DIAG MATRICES
    Y=X;
    for i=1:n
        Y(i,i)=1;
    end

    iter = iter + 1;

    if iter==maxits
        fprintf('Max its exceeded \n'), break, end

    % convergence test
    if norm(Y-X,'inf')/norm(Y,'inf') <= conv_tol, break,end

end
```

```
fprintf('||A-X||_F: %2.4f\n',norm(A-X,'fro'))
fprintf('Number of iterations taken: %4.0f\n',iter)
```

A.3 M-Files for fmincon

sqp_run.m

```
% Script file for finding the nearest correlation matrix to
% A below using FMINCON

% Set random starting vector
for i=1:48
    x0(i)=rand;
end

% A is our approx correlation matrix
A= [1.0000 -0.3250  0.1881  0.5760  0.0064 -0.6111 -0.0724 -0.1589;
   -0.3250  1.0000  0.2048  0.2436  0.4058  0.2730  0.2869  0.4241;
    0.1881  0.2048  1.0000 -0.1325  0.7658  0.2765 -0.6172  0.9006;
    0.5760  0.2436 -0.1325  1.0000  0.3041  0.0126  0.6452 -0.3210;
    0.0064  0.4058  0.7658  0.3041  1.0000  0.6652 -0.3293  0.9939;
   -0.6111  0.2730  0.2765  0.0126  0.6652  1.0000  0.0492  0.5964;
   -0.0724  0.2869 -0.6172  0.6452 -0.3293  0.0492  1.0000 -0.3983;
   -0.1589  0.4241  0.9006 -0.3210  0.9939  0.5964 -0.3983  1.0000];

% We know constant values
n=8;
t=6;   %t is actually n-t
k=3;

tic
% Set options for fmincon, mediumscale algorithm
% and need many function evaluations. Default tolerance
opt=optimset('Largescale','off','MaxFunEvals',10000);
[x,f,fl,out]=fmincon(@fun,x0,[],[],[],[],[],[],@con,opt,A,n,t,k)

toc

% check constraints
[c,ceq]=con(x,A,n,t,k)

% reconstruct X
X=zeros(n);
```

```
for i=1:t

    y=x((i-1)*n+1:i*n);
    X=X+y'*y;

end

X
```

fun.m

```
function f=fun(x,A,n,t,k)
%FUN  Function to be minimises in FMINCON.
%
%   f = FUN(x,A,n,t,k)
%
%   Constructs matrix X from the latest vector x
%   and calculates Frobenius norm of matrix minus
%   the approximate correlation matrix.
%
%   INPUT:  x   current vector
%           A   approx correlation matrix
%           n   size of A
%           t   number of positive eigenvalues
%           k   size of exact part of A (Not used)
%
%   OUTPUT: f   function value


% construct correlation matrix from vector
X=zeros(n);

for i=1:t

    y=x((i-1)*n+1:i*n);
    X=X+y'*y;
```

```
end

f=norm(A-X,’fro’);
```

con.m

```
function [c,ceq]=con(x,A,n,t,k)
%CON  Nonlinear equality constraint for FMINCON.
%
%   [c,ceq] = CON(x,A,n,t,k)
%
%   Supplies the constraints necessary to obtain unit
%   diagonal of correlation matrix and preserve
%   exact part of approximate correlation matrix ’E’.
%
%   INPUT:  x     current vector
%           A     approx correlation matrix
%           n     size of A
%           t     number of positive eigenvalues
%           k     size of exact part of A
%
%   OUTPUT: c     inequality constraint (zero)
%           ceq   equality constraint vector

% empty inequality constraint
c=0;

% equality constraint for unit diaganol of X
for i=1:n
    ceq(i)=sum(x(i:n:(t-1)*n + i).^2) - 1;
end

l=n+1;

% constraint for preserving ’E’
if k >0
```

```
    for i=1:k-1
       for j=i+1:k
          ceq(l)=sum(x(i:n:(t-1)*n+i).*x(j:n:(t-1)*n+j))-A(i,j);
          l=l+1;
       end
    end

end
```

A.4 M-File Function for `mdsmax`

mdsfun.m

```
function f=mdsfun(x)
%MDSFUN  Function for use with MDSMAX to find nearest cov matrix
%
%   f = MDSFUN(x)
%
%   INPUT:  x   current vector
%
%   OUTPUT: f   function value
%

% data matrix P with missing elements
P=[59.875  42.734  47.938  60.359  54.016  69.625  61.500  62.125;
   53.188  49.000  39.500   x(1)   34.750   x(5)   83.000  44.500
   55.750  50.000  38.938   x(2)   30.188   x(6)   70.875  29.938;
   65.500  51.063  45.563  69.313  48.250  62.375  85.250   x(8) ;
   69.938  47.000  52.313  71.016   x(4)   59.359  61.188  48.219;
   61.500  44.188  53.438  57.000  35.313  55.813  51.500  62.188;
   59.230  48.210  62.190  61.390  54.310  70.170  61.750  91.080;
   61.230  48.700  60.300  68.580  61.250  70.340   x(7)    x(9);
   52.900  52.690  54.230   x(3)   68.170  70.600  57.870  88.640;
   57.370  59.040  59.870  62.090  61.620  66.470  65.370  85.840];

% S = cov_bar(gen_lnp(P)) the approx covariance matrix
% computed from P above (with NaNs instead of the x(i)
S=[0.0117 -0.0016  0.0090  0.0102  0.0140 -0.0016 -0.0018 -0.0082;
  -0.0016  0.0057 -0.0036  0.0063 -0.0079  0.0040  0.0176 -0.0081;
   0.0090 -0.0036  0.0152  0.0024  0.0329  0.0034 -0.0072  0.0330;
   0.0102  0.0063  0.0024  0.0222 -0.0057  0.0100  0.0151  0.0187;
   0.0140 -0.0079  0.0329 -0.0057  0.1046  0.0270  0.0038  0.1155;
  -0.0016  0.0040  0.0034  0.0100  0.0270  0.0123  0.0214  0.0260;
  -0.0018  0.0176 -0.0072  0.0151  0.0038  0.0214  0.0557 -0.0023;
  -0.0082 -0.0081  0.0330  0.0187  0.1155  0.0260 -0.0023  0.1192];


% Generate new Ln(P) with x(i) values, and corresponding
% covariance matrix
L=gen_lnp(P);
V=cov_bar(L);
```

```
% we seek to min||S-V||
% hence minus sign since using mdsmax

f=-norm(S-V,'fro');
```

B MEX file for Partial Eigendecomposition

eig_mex.c

```
/*
 *  C mex file for MATLAB that implements LAPACK dsyevr_ for
 *  for finding largest 'num' eigenvalues and their corresponding
 *  vectors of a symmetric real matrix
 *
 *  [Q,d]=eig_mex(A,num,tol)
 *
 *  INPUT:  A            need only have upper triangular part
 *          num          number of largest eigs required
 *          tol          as required by dsyevr_
 *
 *  OUTPUT: d(1:num)     required eigenvalues
 *          Q(1:num,:)   orthonormal eigenvectors
 */

#include "mex.h"
#include "matrix.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const
mxArray *prhs[]) {

  /* jobz=V to get eigvectors, range=I for ILth to IUth eigs */
  /* vu, vl  not referenced by LAPACK routine */

  char *jobz = "V", *range = "I", *uplo = "U", msg[80];
  int n, num, lda, il, iu, *m, ldz, lwork, *iwork;
  int liwork, *isuppz, info;
  double  *a,  *vu, *vl, abstol, *w, *z, *work;
  mxArray *org;

  /* expect 3 inputs and 2 outputs */
  if ((nrhs != 3) || (nlhs != 2)){
     mexErrMsgTxt("Expected 3 inputs and 2 outputs");
  }
```

```
/* copy input matrix so it's not destroyed */
org = mxDuplicateArray(prhs[0]);
a=mxGetPr(org);

/* get dimension of A via number of cols */
n = mxGetN(prhs[0]);

/* assume input array is square */
lda = n;
/* set dimension of output */
ldz = n;

/* get biggest 'num' eigs */
num = mxGetScalar(prhs[1]);
iu = n;
il = n-num+1;

/* set work space dimensions (not optimised) */
lwork = 26*n;
liwork = 10*n;

/* set tolerances for eigs */
abstol = mxGetScalar(prhs[2]);

/* allocate all workspace */
work = (double *)mxCalloc(lwork,sizeof(double));
iwork = (int *)mxCalloc(liwork,sizeof(int));
isuppz = (int *)mxCalloc(2*num,sizeof(int));

/* must allocate m, it's referenced */
m=(int *)mxCalloc(1,sizeof(int));

/* must also allocate variables NOT referenced */
vl = (double *)mxCalloc(1,sizeof(double));
vu = (double *)mxCalloc(1,sizeof(double));

/* set output, then set pointers to them */
plhs[0]=mxCreateDoubleMatrix(n,n,mxREAL);
z=mxGetPr(plhs[0]);
```

```
  plhs[1]=mxCreateDoubleMatrix(n,1,mxREAL);
  w=mxGetPr(plhs[1]);

  info=0;

  dsyevr_(jobz,range,uplo,&n,a,&lda,vl,vu,&il,&iu,&abstol,m,w,z,
          &ldz,isuppz,work,&lwork,iwork,&liwork,&info);

  if(info < 0){
    sprintf(msg, "input %d to DSYEVR had illegal input",-info);
    mexErrMsgTxt(msg);
  }

  /* Free up memory */
  mxFree(work);
  mxFree(iwork);
  mxFree(isuppz);
  mxFree(m);

  mxFree(vl);
  mxFree(vu);

}
```

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorenson. *LAPACK Users' Guide, 3rd Edition.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[2] J. P. Boyle and R. L. Dykstra. A method for finding projections onto the intersection of convex sets in Hilbert spaces. In *Advances in Order Restricted Inference*, volume 37 of *Lecture Notes in Statistics*, Springer-Verlag, Berlin, 1985, pages 28–47.

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

[4] J. E. Dennis, Jr. and V. Torczon. Direct search methods on parallel machines. *SIAM Journal of Optimization*, 1(4):448–474, November 1991.

[5] Richard L. Dykstra. An algorithm for restricted least squares regression. *J. Amer. Stat. Assoc.*, 78(384):837–842, 1983.

[6] FEA. URL: `http://www.fea.com/home.htm`.

[7] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization.* path—Academic Press, Boston, MA, USA, 1981. xvi + 401 pp.

[8] Shih-Ping Han. A successive projection method. *Math prog*, 40:1–14, November 1988.

[9] Nicholas J. Higham. The Test Matrix Toolbox for Matlab (version 3.0). Numerical Analysis Report No. 276, Manchester Centre for Computational Mathematics, Manchester, England, September 1995.

[10] Nicholas J. Higham. Computing the nearest correlation matrix—A problem from finance. Numerical Analysis Report No. 369, Manchester Centre for Computational Mathematics, Manchester, England, October 2000. 14 pp. Revised July 2001.

[11] Roger A. Horn and Charles R. Johnson. *Matrix Analysis.* Cambridge University Press, 1985. xiii+561 pp. ISBN 0-521-30586-1.

[12] C. R. Johnson. Matrix completion problems: a survey. In *Matrix theory and applications*, Amer. Math. Soc., Providence, RI, 1990, pages 171–198.

[13] LISTREL. Frequently asked questions number 3. URL: `http://www.utexas.edu/cc/faqs/stat/lisrel/lisrel3.html`.

[14] David G. Luenberger. *Optimization by Vector Space Methods.* Wiley, New York, 1969. xvii+326 pp. ISBN 0-471-55359-X.

[15] MathWorks. Optimization toolbox user's guide version 2. URL: `http://www.mathworks.com/access/helpdesk/help/pdf_doc/optim/optim_tb.pdf`.

[16] MathWorks. Solution number: 23117. URL: `http://www.mathworks.com/support/solutions/data/23117.shtml`.

[17] R. Tyrrell Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, NJ, USA, 1970. xviii+451 pp. ISBN 0-691-08069-0.

[18] Virginia J. Torczon. *Multi-Directional Search: A Direct Search Algorithm for Parallel Machines.* PhD thesis, Rice University, Houston, TX, USA, May 1989. vii+85 pp.

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

[20] J. von Neumann. *Functional Operators (Vol. II).* Princeton University Press, Princeton, NJ, 1950.