A DIRECT FORMULATION FOR SPARSE PCA USING SEMIDEFINITE PROGRAMMING*

ALEXANDRE D'ASPREMONT†, LAURENT EL GHAOUI‡, MICHAEL I. JORDAN§, AND GERT R. G. LANCKRIET¶

Abstract. Given a covariance matrix, we consider the problem of maximizing the variance explained by a particular linear combination of the input variables while constraining the number of nonzero coefficients in this combination. This problem arises in the decomposition of a covariance matrix into sparse factors or sparse PCA, and has wide applications ranging from biology to finance. We use a modification of the classical variational representation of the largest eigenvalue of a symmetric matrix, where cardinality is constrained, and derive a semidefinite programming based relaxation for our problem. We also discuss Nesterov's smooth minimization technique applied to the semidefinite program arising in the semidefinite relaxation of the sparse PCA problem. The method has complexity $O(n^4\sqrt{\log(n)}/\epsilon)$, where n is the size of the underlying covariance matrix, and ϵ is the desired absolute accuracy on the optimal value of the problem.

Key words. Principal component analysis, Karhunen-Loève transform, factor analysis, semidefinite relaxation, Moreau-Yosida regularization, semidefinite programming.

AMS subject classifications. 90C27, 62H25, 90C22.

1. Introduction. Principal component analysis (PCA) is a popular tool for data analysis, data compression and data visualization. It has applications throughout science and engineering. In essence, PCA finds linear combinations of the variables (the so-called principal components) that correspond to directions of maximal variance in the data. It can be performed via a singular value decomposition (SVD) of the data matrix A, or via an eigenvalue decomposition if A is a covariance matrix.

The importance of PCA is due to several factors. First, by capturing directions of maximum variance in the data, the principal components offer a way to compress the data with minimum information loss. Second, the principal components are uncorrelated, which can aid with interpretation or subsequent statistical analysis. On the other hand, PCA has a number of well-documented disadvantages as well. A particular disadvantage that is our focus here is the fact that the principal components are usually linear combinations of *all* variables. That is, all weights in the linear combination (known as *loadings*) are typically non-zero. In many applications, however, the coordinate axes have a physical interpretation; in biology for example, each axis might correspond to a specific gene. In these cases, the interpretation of the principal components would be facilitated if these components involved very few non-zero loadings (coordinates). Moreover, in certain applications, e.g., financial asset trading strategies based on principal component techniques, the sparsity of the loadings has important consequences, since fewer non-zero loadings imply fewer transaction costs.

It would thus be of interest to discover *sparse principal components*, *i.e.*, sets of sparse vectors spanning a low-dimensional space that explain most of the variance present in the data. To achieve this, it is necessary to sacrifice some of the explained

*A preliminary version of this paper appeared in the proceedings of the Neural Information Processing Systems (NIPS) 2004 conference and the associated preprint is on arXiv as cs.CE/0406021.

†ORFE Dept., Princeton University, Princeton, NJ 08544. aspremon@princeton.edu

‡EECS Dept., U.C. Berkeley, Berkeley, CA 94720. elghaoui@eecs.berkeley.edu

§EECS and Statistics Depts., U.C. Berkeley, Berkeley, CA 94720. jordan@cs.berkeley.edu

¶ECE Dept., U.C. San Diego, La Jolla, CA 92093. gert@ece.ucsd.edu

variance and the orthogonality of the principal components, albeit hopefully not too much.

Rotation techniques are often used to improve interpretation of the standard principal components (see [10] for example). Vines or Kolda and O'Leary [27, 12] considered simple principal components by restricting the loadings to take values from a small set of allowable integers, such as 0, 1, and -1. Cadima and Jolliffe [4] proposed an ad hoc way to deal with the problem, where the loadings with small absolute value are thresholded to zero. We will call this approach "simple thresholding." Later, algorithms such as SCoTLASS [11] and SLRA [28, 29] were introduced to find modified principal components with possible zero loadings. Finally, Zou, Hastie and Tibshirani [30] propose a new approach called sparse PCA (SPCA) to find modified components with zero loading in very large problems, by writing PCA as a regression-type optimization problem. This allows the application of LASSO [24], a penalization technique based on the l_1 norm. All these methods are either significantly suboptimal (thresholding) or nonconvex (SCoTLASS, SLRA, SPCA).

In this paper, we propose a direct approach (called DSPCA in what follows) that improves the sparsity of the principal components by directly incorporating a sparsity criterion in the PCA problem formulation, then forming a *convex relaxation* of the problem. This convex relaxation turns out to be a semidefinite program.

Semidefinite programs (SDP) can be solved in polynomial time via general-purpose interior-point methods [23, 25]. This suffices for an initial empirical study of the properties of DSPCA and for comparison to the algorithms discussed above on small problems. For high-dimensional problems, however, the general-purpose methods are not viable and it is necessary to exploit problem structure. Our particular problem can be expressed as a saddle-point problem which is well suited to recent algorithms based on a *smoothing* argument combined with an optimal first-order smooth minimization algorithm [21, 17, 2]. These algorithms offer a significant reduction in computational time compared to generic interior-point SDP solvers. This also represents a change in the *granularity* of the solver, requiring a larger number of significantly cheaper iterations. In many practical problems this is a desirable tradeoff; interior-point solvers often run out of memory in the first iteration due to the necessity of forming and solving large linear systems. The lower per-iteration memory requirements of the first-order algorithm described here means that considerably larger problems can be solved, albeit more slowly.

This paper is organized as follows. In section 2, we show how to efficiently maximize the variance of a projection while constraining the cardinality (number of nonzero coefficients) of the vector defining the projection. We achieve this via a semidefinite relaxation. We briefly explain how to generalize this approach to non-square matrices and formulate a robustness interpretation of our technique. We also show how this interpretation can be used in the decomposition of a matrix into sparse factors. Section 5 describes how Nesterov's smoothing technique (see [21], [20]) can be used to solve large problem instances efficiently. Finally, section 6 presents applications and numerical experiments comparing our method with existing techniques.

Notation. In this paper, $\mathbf{S}^n$ is the set of symmetric matrices of size n, and Δ_n the spectahedron (set of positive semidefinite matrices with unit trace). We denote by $\mathbf{1}$ a vector of ones, while $\mathbf{Card}(x)$ denotes the cardinality (number of non-zero elements) of a vector x and $\mathbf{Card}(X)$ is the number of non-zero coefficients in the matrix X. For $X \in \mathbf{S}^n$, $X \succeq 0$ means that X is positive semidefinite, $\|X\|_F$ is the Frobenius norm of X, i.e., $\|X\|_F = \sqrt{\mathbf{Tr}(X^2)}$, $\lambda^{\max}(X)$ is the maximum eigenvalue

of X and $\|X\|_\infty = \max_{\{i,j=1,\ldots,n\}} |X_{ij}|$, while $|X|$ is the matrix whose elements are the absolute values of the elements of X. Finally, for matrices $X, Y \in \mathbf{S}^n$, $X \circ Y$ is the Hadamard (componentwise or Schur) product of X and Y.

2. Semidefinite Relaxation. In this section, we derive a semidefinite programming (SDP) relaxation for the problem of maximizing the variance explained by a vector while constraining its cardinality. We formulate this as a variational problem, then obtain a lower bound on its optimal value via an SDP relaxation (we refer the reader to [26] or [3] for an overview of semidefinite programming).

2.1. Sparse Variance Maximization. Let $A \in \mathbf{S}^n$ be a given symmetric matrix and k be an integer with $1 \leq k \leq n$. Given the matrix A and assuming without loss of generality that A is a covariance matrix (i.e. A is positive semidefinite), we consider the problem of maximizing the variance of a vector $x \in \mathbf{R}^n$ while constraining its cardinality:

$$\begin{array}{ll} \text{maximize} & x^T A x \\ \text{subject to} & \|x\|_2 = 1 \\ & \mathbf{Card}(x) \leq k. \end{array} \tag{2.1}$$

Because of the cardinality constraint, this problem is hard (in fact, NP-hard) and we look here for a convex, hence efficient, relaxation.

2.2. Semidefinite Relaxation. Following the *lifting procedure* for semidefinite relaxation described in [15], [1], [13] for example, we rewrite (2.1) as:

$$\begin{array}{ll} \text{maximize} & \mathbf{Tr}(AX) \\ \text{subject to} & \mathbf{Tr}(X) = 1 \\ & \mathbf{Card}(X) \leq k^2 \\ & X \succeq 0, \ \mathbf{Rank}(X) = 1, \end{array} \tag{2.2}$$

in the (matrix) variable $X \in \mathbf{S}^n$. Programs (2.1) and (2.2) are equivalent, indeed if X is a solution to the above problem, then $X \succeq 0$ and $\mathbf{Rank}(X) = 1$ mean that we have $X = xx^T$, while $\mathbf{Tr}(X) = 1$ implies that $\|x\|_2 = 1$. Finally, if $X = xx^T$ then $\mathbf{Card}(X) \leq k^2$ is equivalent to $\mathbf{Card}(x) \leq k$. We have made some progress by turning the convex maximization objective $x^T A x$ and the nonconvex constraint $\|x\|_2 = 1$ into a linear constraint and linear objective. Problem (2.2) is, however, still nonconvex and we need to relax both the rank and cardinality constraints.

Since for every $u \in \mathbf{R}^n$, $\mathbf{Card}(u) = q$ implies $\|u\|_1 \leq \sqrt{q}\|u\|_2$, we can replace the nonconvex constraint $\mathbf{Card}(X) \leq k^2$, by a weaker but convex constraint: $\mathbf{1}^T|X|\mathbf{1} \leq k$, where we exploit the property that $\|X\|_F = \sqrt{x^T x} = 1$ when $X = xx^T$ and $\mathbf{Tr}(X) = 1$. If we drop the rank constraint, we can form a relaxation of (2.2) and (2.1) as:

$$\begin{array}{ll} \text{maximize} & \mathbf{Tr}(AX) \\ \text{subject to} & \mathbf{Tr}(X) = 1 \\ & \mathbf{1}^T|X|\mathbf{1} \leq k \\ & X \succeq 0, \end{array} \tag{2.3}$$

which is a semidefinite program in the variable $X \in \mathbf{S}^n$, where k is an integer parameter controlling the sparsity of the solution. The optimal value of this program will be an upper bound on the optimal value of the variational problem in (2.1). Here, the relaxation of $Card(X)$ in $\mathbf{1}^T|X|\mathbf{1}$ corresponds to a classic technique which replaces the (non-convex) cardinality or l_0 norm of a vector x with its largest convex lower bound on the unit box: $|x|$, the l_1 norm of x (see [7] or [6] for other applications).

2.3. Extension to the Non-Square Case. Similar reasoning applies to the case of a non-square matrix $A \in \mathbf{R}^{m\times n}$, and the problem:

$$\begin{array}{ll} \text{maximize} & u^T A v \\ \text{subject to} & \|u\|_2 = \|v\|_2 = 1 \\ & \mathbf{Card}(u) \leq k_1, \ \ \mathbf{Card}(v) \leq k_2, \end{array}$$

in the variables $(u, v) \in \mathbf{R}^m \times \mathbf{R}^n$ where $k_1 \leq m$, $k_2 \leq n$ are fixed integers controlling the sparsity. This can be relaxed to:

$$\begin{array}{ll} \text{maximize} & \mathbf{Tr}(A^T X^{12}) \\ \text{subject to} & X \succeq 0, \ \ \mathbf{Tr}(X^{ii}) = 1 \\ & \mathbf{1}^T |X^{ii}| \mathbf{1} \leq k_i, \quad i = 1, 2 \\ & \mathbf{1}^T |X^{12}| \mathbf{1} \leq \sqrt{k_1 k_2}, \end{array}$$

in the variable $X \in \mathbf{S}^{m+n}$ with blocks X^{ij} for $i, j = 1, 2$. Of course, we can consider several variations on this, such as constraining $\mathbf{Card}(u, v) = \mathbf{Card}(u) + \mathbf{Card}(v)$.

3. A Robustness Interpretation. In this section, we show that problem (2.3) can be interpreted as a robust formulation of the maximum eigenvalue problem, with additive, componentwise uncertainty in the input matrix A. We again assume A to be symmetric and positive semidefinite.

In the previous section, we considered a cardinality-constrained variational formulation of the maximum eigenvalue problem:

$$\begin{array}{ll} \text{maximize} & x^T A x \\ \text{subject to} & \|x\|_2 = 1 \\ & \mathbf{Card}(x) \leq k. \end{array}$$

Here we look instead at a variation in which we penalize the cardinality and solve:

$$\begin{array}{ll} \text{maximize} & x^T A x - \rho \, \mathbf{Card}^2(x) \\ \text{subject to} & \|x\|_2 = 1, \end{array} \tag{3.1}$$

in the variable $x \in \mathbf{R}^n$, where the parameter $\rho > 0$ controls the magnitude of the penalty. This problem is again nonconvex and very difficult to solve. As in the last section, we can form the equivalent program:

$$\begin{array}{ll} \text{maximize} & \mathbf{Tr}(AX) - \rho \, \mathbf{Card}(X) \\ \text{subject to} & \mathbf{Tr}(X) = 1 \\ & X \succeq 0, \ \ \mathbf{Rank}(X) = 1, \end{array}$$

in the variable $X \in \mathbf{S}^n$. Again, we get a relaxation of this program by forming:

$$\begin{array}{ll} \text{maximize} & \mathbf{Tr}(AX) - \rho \mathbf{1}^T |X| \mathbf{1} \\ \text{subject to} & \mathbf{Tr}(X) = 1 \\ & X \succeq 0, \end{array} \tag{3.2}$$

which is a semidefinite program in the variable $X \in \mathbf{S}^n$, where $\rho > 0$ controls the magnitude of the penalty. We can rewrite this problem as:

$$\max_{X \succeq 0, \mathbf{Tr}(X)=1} \ \min_{|U_{ij}| \leq \rho} \mathbf{Tr}(X(A+U)) \tag{3.3}$$

in the variables $X \in \mathbf{S}^n$ and $U \in \mathbf{S}^n$. This yields the following dual to (3.2):

$$\begin{array}{ll} \text{minimize} & \lambda^{\max}(A+U) \\ \text{subject to} & |U_{ij}| \leq \rho, \quad i,j=1,\ldots,n, \end{array} \tag{3.4}$$

which is a maximum eigenvalue problem with variable $U \in \mathbf{S}^n$. This gives a natural robustness interpretation to the relaxation in (3.2): it corresponds to a worst-case maximum eigenvalue computation, with componentwise bounded noise of intensity ρ imposed on the matrix coefficients.

Let us first remark that ρ in (3.2) corresponds to the optimal Lagrange multiplier in (2.3). Also, the KKT conditions (see [3, §5.9.2]) for problem (3.2) and (3.4) are given by:

$$\left\{ \begin{array}{l} (A+U)X = \lambda^{\max}(A+U)X \\ U \circ X = \rho|X| \\ \mathbf{Tr}(X) = 1, \ X \succeq 0 \\ |U_{ij}| \leq \rho, \quad i,j=1,\ldots,n. \end{array} \right. \tag{3.5}$$

If the eigenvalue $\lambda^{\max}(A+U)$ is simple (when, for example, $\lambda^{\max}(A)$ is simple and ρ is sufficiently small), the first condition means that $\mathbf{Rank}(X) = 1$ and the semidefinite relaxation is *tight*, with in particular $Card(X) = Card(x)^2$ if x is the dominant eigenvector of X. When the optimal solution X is not of rank one because of degeneracy (i.e. when $\lambda^{\max}(A+U)$ has multiplicity strictly larger than one), we can truncate X as in [1, 13], retaining only the dominant eigenvector x as an approximate solution to the original problem. In that degenerate scenario however, the dominant eigenvector of X is not guaranteed to be as sparse as the matrix itself.

4. Sparse Decomposition. Using the results obtained in the previous two sections we obtain a sparse equivalent to the PCA decomposition. Given a matrix $A_1 \in \mathbf{S}^n$, our objective is to decompose it in factors with target sparsity k. We solve the relaxed problem in (2.3):

$$\begin{array}{ll} \text{maximize} & \mathbf{Tr}(A_1 X) \\ \text{subject to} & \mathbf{Tr}(X) = 1 \\ & \mathbf{1}^T|X|\mathbf{1} \leq k \\ & X \succeq 0. \end{array} \tag{4.1}$$

Letting X_1 denote the solution, we truncate X_1, retaining only the dominant (sparse) eigenvector x_1. Finally, we deflate A_1 to obtain

$$A_2 = A_1 - (x_1^T A_1 x_1) x_1 x_1^T,$$

and iterate to obtain further components. The question is now: When do we stop the decomposition?

In the PCA case, the decomposition stops naturally after $\mathbf{Rank}(A)$ factors have been found, since $A_{\mathbf{Rank}(A)+1}$ is then equal to zero. In the case of the sparse decomposition, we have no guarantee that this will happen. Of course, we can add an additional set of linear constraints $x_i^T X x_i = 0$ to problem (4.1) to explicitly enforce the orthogonality of $x_1, \ldots, x_n$ and the decomposition will then stop after a maximum of n iterations. Alternatively, the robustness interpretation gives us a natural stopping criterion: if all the coefficients in $|A_i|$ are smaller than the noise level $\rho^\star$ (computed in the last section) then we must stop since the matrix is essentially indistinguishable from zero. Thus, even though we have no guarantee that the algorithm will terminate with a zero matrix, in practice the decomposition will terminate as soon as the coefficients in A become indistinguishable from the noise.

5. Algorithm. For small problems, the semidefinite program (4.1) can be solved efficiently using interior-point solvers such as SEDUMI [23] or SDPT3 [25]. For larger-scale problems, we need to resort to other types of convex optimization algorithms because the $O(n^2)$ constraints implicitly contained in $\mathbf{1}^T|X|\mathbf{1} \leq k$ make the memory requirements of Newton's method prohibitive. Of special interest are the algorithms recently presented in [21, 17, 2]. These are first-order methods specialized to problems such as (3.3) having a specific saddle-point structure. These methods have a significantly smaller memory cost per iteration than interior-point methods and enable us to solve much larger problems. Of course, there is a price: for *fixed* problem size, the first-order methods mentioned above converge in $O(1/\epsilon)$ iterations, where ϵ is the required accuracy on the optimal value, while interior-point methods converge as $O(\log(1/\epsilon))$. Since the problem under consideration here does not require a high degree of precision, this slow convergence is not a major concern. In what follows, we adapt the algorithm in [21] to our particular constrained eigenvalue problem.

5.1. A Smoothing Technique. The numerical difficulties arising in large scale semidefinite programs stem from two distinct origins. First, there is an issue of *memory*: beyond a certain problem size n, it becomes essentially impossible to form and store any second order information (Hessian) on the problem, which is the key to the numerical efficiency of interior-point SDP solvers. Second, *smoothness* is an issue: the constraint $X \succeq 0$ is not smooth, hence the number of iterations required to solve problem (2.3) using first-order methods such as the bundle code of [8] (which do not form the Hessian) to an accuracy ϵ is given by $O(1/\epsilon^2)$. In general, this complexity bound is tight and cannot be improved without additional structural information on the problem. Fortunately, in our case we do have structural information available that can be used to bring the complexity down from $O(1/\epsilon^2)$ to $O(1/\epsilon)$. Furthermore, the cost of each iteration is equivalent to that of computing a matrix exponential (roughly $O(n^3)$).

Recently, [21] and [20] (see also [17]) proposed an efficient first-order scheme for convex minimization based on a smoothing argument. The main structural assumption on the function to minimize is that it has a *saddle-function* format:

$$f(x) = \hat{f}(x) + \max_u \{\langle Tx, u\rangle - \hat{\phi}(u) \ : \ u \in Q_2\} \tag{5.1}$$

where f is defined over a compact convex set $Q_1 \subset \mathbf{R}^n$, $\hat{f}(x)$ is convex and differentiable and has a Lipschitz continuous gradient with constant $M \geq 0$, T is an element of $\mathbf{R}^{n\times n}$ and $\hat{\phi}(u)$ is a continuous convex function over some closed compact set $Q_2 \subset \mathbf{R}^n$. This assumes that the function $\hat{\phi}(u)$ and the set Q_2 are simple enough so that the optimization subproblem in u can be solved very efficiently. When a function f can be written in this particular format, [21] uses a *smoothing technique* to show that the complexity (number of iterations required to obtain a solution with absolute precision ϵ) of solving:

$$\min_{x\in Q_1} f(x) \tag{5.2}$$

falls from $O(1/\epsilon^2)$ to $O(1/\epsilon)$. This is done in two steps.

Regularization. By adding a strongly convex penalty to the saddle function representation of f in (5.1), the algorithm first computes a *smooth* ϵ-approximation of f with Lipschitz continuous gradient. This can be seen as a generalized Moreau-Yosida regularization step (see [14] for example).

Optimal first-order minimization. The algorithm then applies the optimal first-order scheme for functions with Lipschitz continuous gradient detailed in [18] to the regularized function. Each iteration requires an efficient computation of the regularized function value and its gradient. As we will see, this can be done explicitly in our case, with a complexity of $O(n^3)$ and memory requirements of $O(n^2)$.

5.2. Application to Sparse PCA. Given a symmetric matrix $A \in \mathbf{S}^n$, we consider the problem given in (3.3) (where we can assume without loss of generality that $\rho = 1$):

$$\begin{array}{ll} \text{maximize} & \mathbf{Tr}(AX) - \mathbf{1}^T|X|\mathbf{1} \\ \text{subject to} & \mathbf{Tr}(X) = 1 \\ & X \succeq 0, \end{array} \tag{5.3}$$

in the variable $X \in \mathbf{S}^n$. Duality allows us to rewrite this in the saddle-function format:

$$\min_{U \in \mathcal{Q}_1} \; f(U), \tag{5.4}$$

where

$$\mathcal{Q}_1 = \{U \in \mathbf{S}^n : |U_{ij}| \leq 1,\ i,j = 1,\ldots,n\},\ \mathcal{Q}_2 = \{X \in \mathbf{S}^n : \mathbf{Tr}\, X = 1,\ X \succeq 0\}$$

$$f(U) := \max_{X \in \mathcal{Q}_2} \langle TU, X\rangle - \hat{\phi}(X), \quad \text{with } T = I_{n^2},\ \hat{\phi}(X) = -\,\mathbf{Tr}(AX).$$

As in [21], to $\mathcal{Q}_1$ and $\mathcal{Q}_2$ we associate norms and so-called prox-functions. To $\mathcal{Q}_1$, we associate the Frobenius norm in $\mathbf{S}^n$, and a prox-function defined for $U \in \mathcal{Q}_1$ by:

$$d_1(U) = \frac{1}{2} U^T U.$$

With this choice, the center U_0 of $\mathcal{Q}_1$, defined as:

$$U_0 := \arg\min_{U \in \mathcal{Q}_1} \; d_1(U),$$

is $U_0 = 0$, and satisfies $d_1(U_0) = 0$. Moreover, we have:

$$D_1 := \max_{U \in \mathcal{Q}_1} \; d_1(U) = n^2/2.$$

Furthermore, the function d_1 is strongly convex on its domain, with convexity parameter of $\sigma_1 = 1$ with respect to the Frobenius norm. Next, for $\mathcal{Q}_2$ we use the dual of the standard matrix norm (denoted $\|\cdot\|_2^*$), and a prox-function

$$d_2(X) = \mathbf{Tr}(X \log X) + \log(n),$$

where $\log X$ refers to the *matrix* (and not componentwise) logarithm, obtained by replacing the eigenvalues of X by their logarithm. The center of the set $\mathcal{Q}_2$ is $X_0 = n^{-1} I_n$, where $d_2(X_0) = 0$. We have

$$\max_{X \in \mathcal{Q}_2} \; d_2(X) \leq \log n := D_2.$$

The convexity parameter of d_2 with respect to $\|\cdot\|_2^*$, is bounded below by $\sigma_2 = 1$. (This non-trivial result is proved in [20].)

Next we compute the $(1,2)$ norm of the operator T introduced above, which is defined as:

$$\begin{aligned}\|T\|_{1,2} &:= \max_{X,U} \langle TX, U\rangle \ : \ \|U\|_F = 1,\ \|X\|_2^* = 1\\ &= \max_{X} \ \|X\|_2 : \|X\|_F \leq 1\\ &= 1.\end{aligned}$$

To summarize, the parameters defined above are set as follows:

$$D_1 = n^2/2,\ \sigma_1 = 1,\ D_2 = \log(n),\ \sigma_2 = 1,\ \|T\|_{1,2} = 1.$$

Let us now explicitly formulate how the regularization and smooth minimization techniques can be applied to the variance maximization problem in (5.3).

5.2.1. Regularization. The method in [21] first sets a regularization parameter

$$\mu := \frac{\epsilon}{2D_2}.$$

The method then produces an ϵ-suboptimal optimal value and corresponding suboptimal solution in a number of steps not exceeding

$$N = \frac{4\|T\|_{1,2}}{\epsilon}\sqrt{\frac{D_1 D_2}{\sigma_1\sigma_2}}.$$

The non-smooth objective $f(X)$ of the original problem is replaced with

$$\min_{U\in\mathcal{Q}_1} \ f_\mu(U),$$

where f_μ is the penalized function involving the prox-function d_2:

$$f_\mu(U) := \max_{X\in\mathcal{Q}_2} \langle TU, X\rangle - \hat{\phi}(X) - \mu d_2(X).$$

Note that in our case, the function f_μ and its gradient are readily computed; see below. The function f_μ is a smooth uniform approximation to f everywhere on $\mathcal{Q}_2$, with maximal error $\mu D_2 = \epsilon/2$. Furthermore, f_μ has a Lipschitz continuous gradient, with Lipschitz constant given by:

$$L := \frac{D_2\|T\|_{1,2}^2}{\epsilon 2\sigma_2}.$$

In our specific case, the function f_μ can be computed explicitly as:

$$f_\mu(U) = \mu\log\left(\mathbf{Tr}\exp((A+U)/\mu)\right) - \mu\log n,$$

which can be seen as a smooth approximation to the function $f(U) = \lambda_{\max}(A+U)$. This function f_μ has a Lipshitz-continuous gradient and is a uniform approximation of the function f.

5.2.2. First-order minimization. An optimal gradient algorithm for minimizing convex functions with Lipschitz continuous gradients is then applied to the smooth convex function f_μ defined above. The key difference between the minimization scheme developed in [18] and classical gradient minimization methods is that it is not a descent method but achieves a complexity of $O(L/k^2)$ instead of $O(1/k)$ for gradient descent, where k is the number of iterations and L the Lipschitz constant of the gradient. Furthermore, this convergence rate is provably optimal for this particular class of convex minimization problems (see [19, Th. 2.1.13]). Thus, by sacrificing the (local) properties of descent directions, we improve the (global) complexity estimate by an order of magnitude.

For our problem here, once the regularization parameter μ is set, the algorithm proceeds as follows.

Repeat:

1. Compute $f_\mu(U_k)$ and $\nabla f_\mu(U_k)$
2. Find $Y_k = \arg\min_{Y\in\mathcal{Q}_1} \langle \nabla f_\mu(U_k), Y\rangle + \frac{1}{2}L\|U_k - Y\|_F^2$
3. Find $W_k = \arg\min_{W\in\mathcal{Q}_1} \left\{ \frac{Ld_1(W)}{\sigma_1} + \sum_{i=0}^{k} \frac{i+1}{2}(f_\mu(U_i) + \langle \nabla f_\mu(U_i), W - U_i\rangle)\right\}$
4. Set $U_{k+1} = \frac{2}{k+3}W_k + \frac{k+1}{k+3}Y_k$

Until gap $\leq \epsilon$.

Step one above computes the (smooth) function value and gradient. The second step computes the *gradient mapping*, which matches the gradient step for unconstrained problems (see [19, p.86]). Step three and four update an *estimate sequence* see ([19, p.72]) of f_μ whose minimum can be computed explicitly and gives an increasingly tight upper bound on the minimum of f_μ. We now present these steps in detail for our problem (we write U for U_k and X for X_k).

Step 1. The most expensive step in the algorithm is the first, the computation of f_μ and its gradient. Setting $Z = A + U$, the problem boils down to computing

$$u^*(z) := \arg\max_{X\in\mathcal{Q}_2} \langle Z, X\rangle - \mu d_2(X) \tag{5.5}$$

and the associated optimal value $f_\mu(U)$. It turns out that this problem has a very simple solution, requiring only an eigenvalue decomposition for $Z = A + U$. The gradient of the objective function with respect to Z is set to the maximizer $u^*(Z)$ itself, so the gradient with respect to U is $\nabla f_\mu(U) = u^*(A + U)$.

To compute $u^*(Z)$, we form an eigenvalue decomposition $Z = VDV^T$, with $D = \mathrm{diag}(d)$ the matrix with diagonal d, then set

$$h_i := \frac{\exp(\frac{d_i - d_{\max}}{\mu})}{\sum_{j=1}^{n}\exp(\frac{d_j - d_{\max}}{\mu})}, \quad i = 1, \ldots, n,$$

where $d_{\max} := \max_{\{j=1,\ldots,n\}} d_j$ is used to mitigate problems with large numbers. We then let $u^*(z) = VHV^T$, with $H = \mathrm{diag}(h)$. The corresponding function value is given by:

$$f_\mu(U) = \mu \log\left(\mathbf{Tr}\exp\left(\frac{(A+U)}{\mu}\right)\right) = \mu\log\left(\sum_{i=1}^{n}\exp\left(\frac{d_i}{\mu}\right)\right) - \mu\log n,$$

which can be reliably computed as:

$$f_\mu(U) = d_{\max} + \mu\log\left(\sum_{i=1}^{n}\exp(\frac{d_i - d_{\max}}{\mu})\right) - \mu\log n.$$

Step 2. This step involves a problem of the form:

$$\arg\min_{Y\in\mathcal{Q}_1} \langle \nabla f_\mu(U), Y\rangle + \frac{1}{2}L\|U-Y\|_F^2,$$

where U is given. The above problem can be reduced to a Euclidean projection:

$$\arg\min_{\|Y\|_\infty\leq 1} \|Y-V\|_F, \tag{5.6}$$

where $V = U - L^{-1}\nabla f_\mu(U)$ is given. The solution is given by:

$$Y_{ij} = \mathbf{sgn}(V_{ij})\min(|V_{ij}|,1), \quad i,j=1,\ldots,n.$$

Step 3. The third step involves solving a Euclidean projection problem similar to (5.6), with V defined by:

$$V = -\frac{\sigma_1}{L}\sum_{i=0}^{k}\frac{i+1}{2}\nabla f_\mu(U_i).$$

Stopping criterion. We can stop the algorithm when the duality gap is smaller than ϵ:

$$\mathrm{gap}_k = \lambda_{\max}(A+U_k) - \mathbf{Tr}\, AX_k + \mathbf{1}^T|X_k|\mathbf{1} \leq \epsilon,$$

where $X_k = u^*((A+U_k)/\mu)$ is our current estimate of the dual variable (the function u^* is defined by (5.5)). The above gap is necessarily non-negative, since both X_k and U_k are feasible for the primal and dual problem, respectively. This is checked periodically, for example every 100 iterations.

Complexity. Since each iteration of the algorithm requires computing a matrix exponential (which requires an eigenvalue decomposition and $O(n^3)$ flops in our code), the predicted worst-case complexity to achieve an objective with absolute accuracy less than ϵ is [21]:

$$4\|T\|_{1,2}\frac{O(n^3)}{\epsilon}\sqrt{\frac{D_1D_2}{\sigma_1\sigma_2}} = O(n^4\sqrt{\log n}/\epsilon).$$

In some cases, this complexity estimate can be improved by using specialized algorithms for computing the matrix exponential (see [16] for a discussion). For example, computing only a few eigenvalues might be sufficient to obtain this exponential with the required precision (see [5]). In our preliminary experiments, the standard technique using Padé approximations, implemented in packages such as Expokit (see [22]), required too much scaling to be competitive with a full eigenvalue decomposition.

6. Numerical results & Applications. In this section, we illustrate the effectiveness of the proposed approach (called DSPCA in what follows) both on an artificial data set and a real-life data set. We compare with the other approaches mentioned in the introduction: PCA, PCA with simple thresholding, SCoTLASS and SPCA. The results show that our approach can achieve more sparsity in the principal components than SPCA (the current state-of-the-art method) does, while explaining as much variance. The other approaches can explain more variance, but result in principal components that are far from sparse. We begin by a simple example illustrating the link between k and the cardinality of the solution.

6.1. Artificial data. To compare the numerical performance with that of existing algorithms, we consider the simulation example proposed by [30]. In this example, three hidden factors are created:

$$V_1 \sim \mathcal{N}(0, 290), \quad V_2 \sim \mathcal{N}(0, 300), \quad V_3 = -0.3V_1 + 0.925V_2 + \epsilon, \quad \epsilon \sim \mathcal{N}(0, 300)$$

with V_1, V_2 and ϵ independent. Afterward, 10 observed variables are generated as follows:

$$X_i = V_j + \epsilon_i^j, \quad \epsilon_i^j \sim \mathcal{N}(0, 1),$$

with $j = 1$ for $i = 1, \ldots, 4$, $j = 2$ for $i = 5, \ldots, 8$ and $j = 3$ for $i = 9, 10$ and ϵ_i^j independent for $j = 1, 2, 3$, $i = 1, \ldots, 10$. We use the exact covariance matrix to compute principal components using the different approaches.

Since the three underlying factors have roughly the same variance, and the first two are associated with four variables while the last one is associated with only two variables, V_1 and V_2 are almost equally important, and they are both significantly more important than V_3. This, together with the fact that the first two principal components explain more than 99% of the total variance, suggests that considering two sparse linear combinations of the original variables should be sufficient to explain most of the variance in data sampled from this model [30]. The ideal solution would thus be to use only the variables (X_1, X_2, X_3, X_4) for the first sparse principal component, to recover the factor V_1, and only (X_5, X_6, X_7, X_8) for the second sparse principal component to recover V_2.

Using the true covariance matrix and the oracle knowledge that the ideal sparsity is four, [30] performed SPCA (with $\lambda = 0$). We carry out our algorithm with $k = 4$. The results are reported in Table 6.1, together with results for PCA, simple thresholding and SCoTLASS ($t = 2$). Notice that DSPCA, SPCA and SCoTLASS all find the correct sparse principal components, while simple thresholding yields inferior performance. The latter wrongly includes the variables X_9 and X_{10} (likely being misled by the high correlation between V_2 and V_3), moreover, it assigns higher loadings to X_9 and X_{10} than to each of the variables (X_5, X_6, X_7, X_8) that are clearly more important. Simple thresholding correctly identifies the second sparse principal component, probably because V_1 has a lower correlation with V_3. Simple thresholding also explains a bit less variance than the other methods.

TABLE 6.1

Loadings and explained variance for the first two principal components of the artificial example. Here, "ST" denotes the simple thresholding method, "other" is all the other methods: SPCA, DSPCA and SCoTLASS. PC1 and PC2 denote the first and second principal components.

	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	explained variance
PCA, PC1	.116	.116	.116	.116	-.395	-.395	-.395	-.395	-.401	-.401	60.0%
PCA, PC2	-.478	-.478	-.478	-.478	-.145	-.145	-.145	-.145	.010	.010	39.6%
ST, PC1	0	0	0	0	0	0	-.497	-.497	-.503	-.503	38.8%
ST, PC2	-.5	-.5	-.5	-.5	0	0	0	0	0	0	38.6%
other, PC1	0	0	0	0	.5	.5	.5	.5	0	0	40.9%
other, PC2	.5	.5	.5	.5	0	0	0	0	0	0	39.5%

6.2. Pit props data. The pit props data (consisting of 180 observations and 13 measured variables) was introduced by [9] and is another benchmark example used to test sparse PCA codes. Both SCoTLASS [11] and SPCA [30] have been tested on this data set. As reported in [30], SPCA performs better than SCoTLASS in the sense

that it identifies principal components with 7, 4, 4, 1, 1, and 1 non-zero loadings, respectively, as shown in Table 6.2. This is much sparser than the modified principal components by SCoTLASS, while explaining nearly the same variance (75.8% versus 78.2% for the 6 first principal components) [30]. Also, simple thresholding of PCA, with a number of non-zero loadings that matches the result of SPCA, does worse than SPCA in terms of explained variance.

Following this previous work, we also consider the first 6 principal components. We try to identify principal components that are sparser than those of SPCA, but explain the same variance. Therefore, we choose values for k of 5, 2, 2, 1, 1, 1 (two less than the values of SPCA reported above, but no less than 1). Figure 6.1 shows the cumulative number of non-zero loadings and the cumulative explained variance (measuring the variance in the subspace spanned by the first i eigenvectors). It can be seen that our approach is able to explain nearly the same variance as the SPCA method, while clearly reducing the number of non-zero loadings for the first six principal components. Adjusting the first value of k from 5 to 6 (relaxing the sparsity), we obtain results that are still better in terms of sparsity, but with a cumulative explained variance that is uniformly larger than SPCA. Moreover, as in the SPCA approach, the important variables associated with the six principal components do not overlap, which leads to a clearer interpretation. Table 6.2 shows the first three corresponding principal components for the different approaches (DSPCAw5 denotes runs with $k_1 = 5$ and DSPCAw6 uses $k_1 = 6$).

TABLE 6.2

Loadings for first three principal components, for the pit props data. DSPCAw5 (resp. DSPCAw6) shows the results for our technique with k_1 equal to 5 (resp. 6).

	topd	length	moist	testsg	ovensg	ringt	ringb	bowm	bowd	whorls	clear	knots	diaknot
SPCA 1	-.477	-.476	0	0	.177	0	-.250	-.344	-.416	-.400	0	0	0
SPCA 2	0	0	.785	.620	0	0	0	-.021	0	0	0	.013	0
SPCA 3	0	0	0	0	.640	.589	.492	0	0	0	0	0	-.015
DSPCAw5 1	-.560	-.583	0	0	0	0	-.263	-.099	-.371	-.362	0	0	0
DSPCAw5 2	0	0	.707	.707	0	0	0	0	0	0	0	0	0
DSPCAw5 3	0	0	0	0	0	-.793	-.610	0	0	0	0	0	.012
DSPCAw6 1	-.491	-.507	0	0	0	-.067	-.357	-.234	-.387	-.409	0	0	0
DSPCAw6 2	0	0	.707	.707	0	0	0	0	0	0	0	0	0
DSPCAw6 3	0	0	0	0	0	-.873	-.484	0	0	0	0	0	.057

6.3. Controlling sparsity with k. We present a simple example to illustrate how the sparsity of the solution to our relaxation evolves as k varies from 1 to n. We generate a 10×10 matrix U with uniformly distributed coefficients in $[0, 1]$. We let v be a sparse vector with:

$$v = (1, 0, 1, 0, 1, 0, 1, 0, 1, 0).$$

We then form a test matrix $A = U^T U + \sigma v v^T$, where σ is a signal-to-noise ratio that we set equal to 15. We sample 50 different matrices A using this technique. For each value of k between 1 and 10 and each A, we solve the following SDP:

$$\begin{array}{ll} \max & \mathbf{Tr}(AX) \\ \text{subject to} & \mathbf{Tr}(X) = 1 \\ & \mathbf{1}^T |X| \mathbf{1} \leq k \\ & X \succeq 0. \end{array}$$

We then extract the first eigenvector of the solution X and record its cardinality. In Figure 6.2, we show the mean cardinality (and standard deviation) as a function of

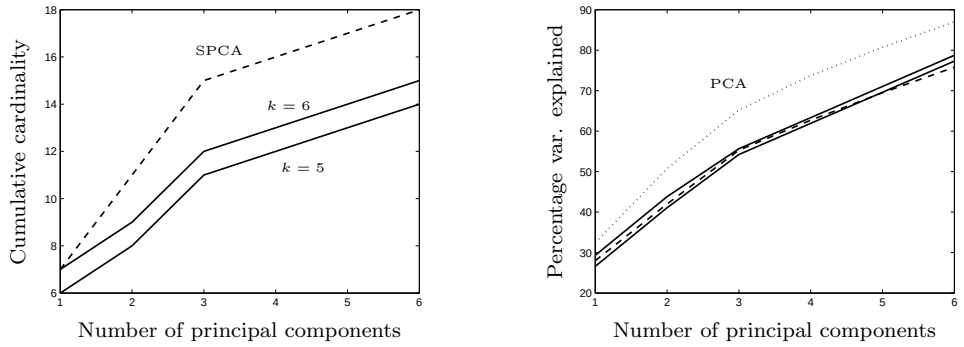

FIG. 6.1. *Cumulative cardinality and percentage of total variance explained versus number of principal components, for SPCA and DSPCA on the pit props data. The dashed lines are SPCA and the solid ones are DSPCA with $k_1 = 5$ and $k_1 = 6$. On the right, the dotted line also shows the percentage of variance explained by standard (non sparse) PCA. While explaining the same cumulative variance, our method (DSPCA) produces sparser factors.*

k. We observe that $k + 1$ is actually a good predictor of the cardinality, especially when $k + 1$ is close to the actual cardinality (5 in this case). In fact, in the random examples tested here, we always recover the original cardinality of 5 when $k + 1$ is set to 5.

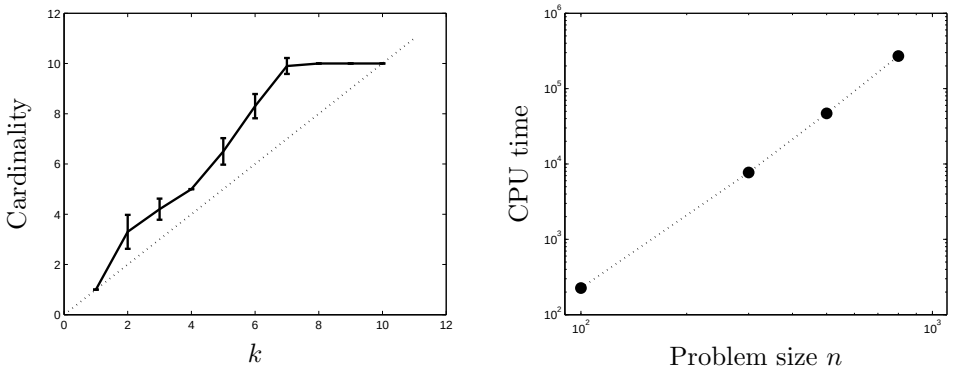

FIG. 6.2. *Left: Plot of average cardinality (and its standard deviation) versus k for 100 random examples with original cardinality 5. Right: Plot of CPU time (in seconds) versus problem size for randomly chosen problems.*

6.4. Computing Time versus Problem Size. In Figure 6.2 we plot the total CPU time used for randomly chosen problems of size n ranging from 100 to 800. The required precision was set to $\epsilon = 10^{-3}$, which was always reached in fewer than 60000 iterations. In these examples, the empirical complexity appears to grow as $O(n^3)$.

6.5. Sparse PCA for Gene Expression Data Analysis. We are given m data vectors $x_j \in \mathbf{R}^n$, with $n = 500$. Each coefficient x_{ij} corresponds to the expression of gene i in experiment j. For each vector x_j we are also given a class $c_j \in \{0, 1, 2, 3\}$. We form $A = xx^T$, the covariance matrix of the experiment. Our objective is to use PCA to first reduce the dimensionality of the problem and then look for *clustering* when the data are represented in the basis formed by the first three

principal components. Here, we do not apply any clustering algorithm to the data points, we just assign a color to each sample point in the three dimensional scatter plot, based on known experimental data.

The sparsity of the factors in sparse PCA implies that the clustering can be attributed to fewer genes, making interpretation easier. In Figure 6.3, we see clustering in the PCA representation of the data and in the DSPCA representation. Although there is a slight drop in the resolution of the clusters for DSPCA, the key feature here is that the total number of nonzero gene coefficients in the DSPCA factors is equal to 14 while standard PCA produces three dense factors, each with 500 nonzero coefficients.

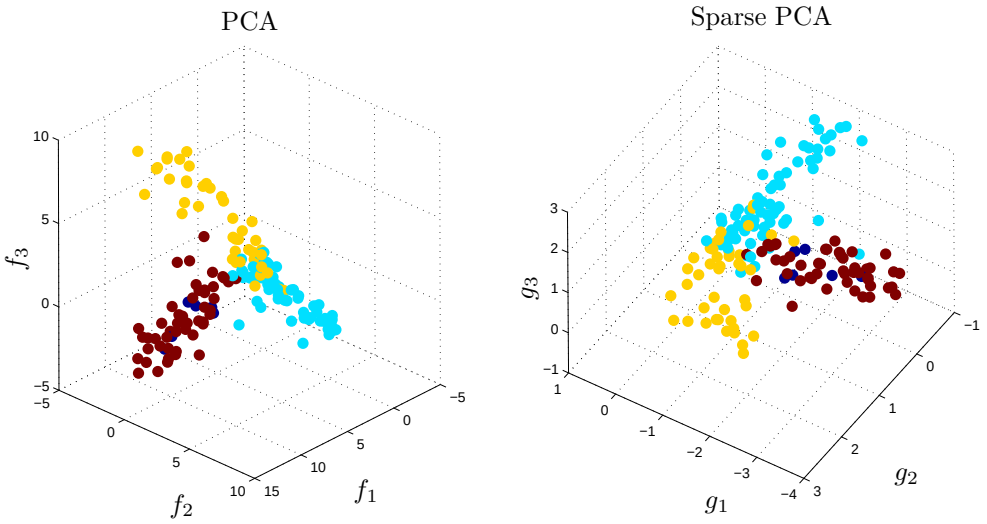

FIG. 6.3. *Clustering of the gene expression data in the PCA versus sparse PCA basis with 500 genes. The factors f on the left are dense and each use all 500 genes while the sparse factors g_1, g_2 and g_3 on the right involve 6, 4 and 4 genes respectively. (Data: Iconix Pharmaceuticals)*

Acknowledgments. Thanks to Andrew Mullhaupt and Francis Bach for useful suggestions. We would like to acknowledge support from NSF grant 0412995, ONR MURI N00014-00-1-0637, Eurocontrol-C20052E/BM/03 and C20083E/BM/05, NASA-NCC2-1428 and a gift from Google, Inc.

REFERENCES

[1] F. ALIZADEH, *Interior point methods in semidefinite programming with applications to combinatorial optimization*, SIAM Journal on Optimization, 5 (1995), pp. 13–51.

[2] A. BEN-TAL AND A. NEMIROVSKI, *Non-Euclidean restricted memory level method for large-scale convex optimization*, Mathematical Programming, 102 (2005), pp. 407–456.

[3] S. BOYD AND L. VANDENBERGHE, *Convex Optimization*, Cambridge University Press, 2004.

[4] J. CADIMA AND I. T. JOLLIFFE, *Loadings and correlations in the interpretation of principal components*, Journal of Applied Statistics, 22 (1995), pp. 203–214.

[5] A. D'ASPREMONT, *Smooth optimization for sparse semidefinite programs*, ArXiv: math.OC/0512344, (2005).

[6] D. L. DONOHO AND J. TANNER, *Sparse nonnegative solutions of underdetermined linear equations by linear programming*, Proceedings of the National Academy of Sciences, 102 (2005), pp. 9446–9451.

[7] M. Fazel, H. Hindi, and S. Boyd, *A rank minimization heuristic with application to minimum order system approximation*, Proceedings American Control Conference, 6 (2001), pp. 4734–4739.

[8] C. Helmberg and F. Rendl, *A spectral bundle method for semidefinite programming*, SIAM Journal on Optimization, 10 (2000), pp. 673–696.

[9] J. Jeffers, *Two case studies in the application of principal components*, Applied Statistics, 16 (1967), pp. 225–236.

[10] I. T. Jolliffe, *Rotation of principal components: choice of normalization constraints*, Journal of Applied Statistics, 22 (1995), pp. 29–35.

[11] I. T. Jolliffe, N.T. Trendafilov, and M. Uddin, *A modified principal component technique based on the LASSO*, Journal of Computational and Graphical Statistics, 12 (2003), pp. 531–547.

[12] T. G. Kolda and D. P. O'Leary, *Algorithm 805: computation and uses of the semidiscrete matrix decomposition*, ACM Transactions on Mathematical Software, 26 (2000), pp. 415–435.

[13] C. Lemaréchal and F. Oustry, *Semidefinite relaxations and Lagrangian duality with application to combinatorial optimization*, INRIA, Rapport de recherche, 3710 (1999).

[14] C. Lemaréchal and C. Sagastizábal, *Practical aspects of the Moreau-Yosida regularization: theoretical preliminaries*, SIAM Journal on Optimization, 7 (1997), pp. 367–385.

[15] L. Lovász and A. Schrijver, *Cones of matrices and set-functions and* 0-1 *optimization*, SIAM Journal on Optimization, 1 (1991), pp. 166–190.

[16] C. Moler and C. Van Loan, *Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*, SIAM Review, 45 (2003), pp. 3–49.

[17] A. Nemirovski, *Prox-method with rate of convergence O(1/T) for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle point problems*, SIAM Journal on Optimization, 15 (2004), pp. 229–251.

[18] Y. Nesterov, *A method of solving a convex programming problem with convergence rate* $O(1/k^2)$, Soviet Mathematics Doklady, 27 (1983), pp. 372–376.

[19] ———, *Introductory Lectures on Convex Optimization*, Springer, 2003.

[20] ———, *Smoothing technique and its application in semidefinite optimization*, CORE Discussion Paper No. 2004/73, (2004).

[21] ———, *Smooth minimization of nonsmooth functions*, Mathematical Programming, Series A, 103 (2005), pp. 127–152.

[22] R. B. Sidje, *Expokit: a software package for computing matrix exponentials*, ACM Transactions on Mathematical Software (TOMS), 24 (1998), pp. 120–156.

[23] J. Sturm, *Using SEDUMI 1.0x, a MATLAB toolbox for optimization over symmetric cones*, Optimization Methods and Software, 11 (1999), pp. 625–653.

[24] R. Tibshirani, *Regression shrinkage and selection via the LASSO*, Journal of the Royal statistical society, series B, 58 (1996), pp. 267–288.

[25] K. C. Toh, M. J. Todd, and R. H. Tutuncu, *SDPT3 – a MATLAB software package for semidefinite programming*, Optimization Methods and Software, 11 (1999), pp. 545–581.

[26] L. Vandenberghe and S. Boyd, *Semidefinite programming*, SIAM Review, 38 (1996), pp. 49–95.

[27] S. Vines, *Simple principal components*, Applied Statistics, 49 (2000), pp. 441–451.

[28] Z. Zhang, H. Zha, and H. Simon, *Low rank approximations with sparse factors I: basic algorithms and error analysis*, SIAM journal on matrix analysis and its applications, 23 (2002), pp. 706–727.

[29] ———, *Low rank approximations with sparse factors II: penalized methods with discrete Newton–like iterations*, SIAM journal on matrix analysis and its applications, 25 (2004), pp. 901–920.

[30] H. Zou, T. Hastie, and R. Tibshirani, *Sparse principal component analysis*, To appear in Journal of Computational and Graphical Statistics, (2004).