An algorithm for sparse PCA based on a new sparsity control criterion

Yunlong He [*] Renato D.C. Monteiro [†] Haesun Park [*]

Abstract

Sparse principal component analysis (PCA) imposes extra constraints or penalty terms to the standard PCA to achieve sparsity. In this paper, we first introduce an efficient algorithm for finding a single sparse principal component (PC) with a specified cardinality. Experiments on synthetic data, randomly generated data and real-world data sets show that our algorithm is very fast, especially on large and sparse data sets, while the numerical quality of the solution is comparable to the state-of-the-art algorithm. Moreover, combining our algorithm for computing a single sparse PC with the Schur complement deflation scheme, we develop an algorithm which sequentially computes multiple PCs by greedily maximizing the *adjusted variance* explained by them. On the other hand, to address the difficulty of choosing the proper sparsity and parameter in various sparse PCA algorithms, we propose a new PCA formulation whose aim is to minimize the sparsity of the PCs while requiring that their *relative adjusted variance* is larger than a given fraction. We also show that a slight modification of the aforementioned multiple component PCA algorithm can also find sharp solutions of the latter formulation.

1 Introduction

Principal Component Analysis (PCA) is a classical tool for performing data analysis such as dimensionality reduction, data modeling, feature extraction and other learning tasks. It can be widely used in all kinds of data analysis areas like image processing, gene microarray analysis and document analysis. Basically, PCA consists of finding a few orthogonal directions in the data space which preserve the most information in the data. This is done by finding directions that would maximize the variance of the projections of the data points along these directions. However, standard PCA generally produces dense directions (i.e., whose entries are mostly nonzero), and hence are too complex to explain the data set. Instead, a standard approach in the learning community is to pursue sparse directions which in some sense approximate the directions produced by standard PCA. Sparse PCA has a few advantages, namely: i) it can be effectively stored and ii) it allows the simpler interpretation of the inherent structure and important information associated with the data set. For these reasons, sparse PCA is a subject which has received a lot of attention from the learning community in the last decade.

Several formulations and algorithms have been proposed to perform sparse PCA. Zou et al.[12] formulate sparse PCA as a LASSO-type optimization problem. Shen and Huang [10] combine simple linear regression and thresholding to solve a regularized SVD problem, which achieves sparse PCA. D'Aspremont et al.'s DSPCA algorithm [1] consists of solving a semidefinite programming relaxation of a certain formulation of sparse PCA whose solution is then post-processed to yield a sparse principal component (PC). Paper [2] by d'Aspremont et al. proposes a greedy algorithm PathSPCA to solve a new semidefinite programming relaxation and provides a sufficient condition for optimality. ESPCA algorithm in Moghaddam et al. [9] obtains good numerical quality by using a combinatorial greedy method, although their method can be slow on large data set. Their method, like ours, consists of identifying an active index set (i.e., the indices corresponding to the nonzero entries of the PC) and then using an algorithm such as power-iteration to obtain the final sparse PC. Journée et al.'s GPower method [5] formulates sparse PCA as a nonconcave maximization problem with a penalty term to achieve sparsity, which is then reduced to an equivalent problem of maximizing a convex function over a compact set. The latter problem is then solved by an algorithm which is essentially a generalization of the power-iteration method. Different deflation methods have been studied in [8], which are used to find multiple sparse PCs sequentially. A different multiple sparse PCA approach is proposed in

[*]School of Mathematics, Georgia Institute of Technology (heyunlong@gatech.edu); School of Computational Science and Engineering, Georgia Institute of Technology (hpark@cc.gatech.edu). The work of these authors was supported by the National Science Foundation grant CCF-0808863 and CCF-0732318. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

[†]School of Industrial & System Engineering, Georgia Institute of Technology (renato.monteiro@isye.gatech.edu). The work of this author was partially supported by NSF Grants CCF-0808863 and CMMI-0900094 and ONR Grants N00014-08-1-0033 and N00014-11-1-0062.

[7] based on a formulation enforcing near orthogonality of the PCs, which is then solved by an augmented Lagrangian approach. Throughout this paper, we mainly compare our approach with the GPower method proposed in [5], which is widely viewed as one of the most efficient methods for performing sparse PCA.

Our contributions in this paper are as follows. First, we propose a simple but effective algorithm for finding a single sparse PC. The algorithm consists of two stages. In the first stage, it identifies an active index set with a desired cardinality corresponding to the nonzero entries of the PC. In the second one, it uses the power iteration method to find the best direction with respect to the active index set. The complexity of this algorithm is proportional to the pre-specified cardinality of the solution, but we show that it can be accelerated by adding multiple indices to the active set in every iteration and optimizing it for sparse matrix. An important advantage of our method is that it can easily produce a single sparse PC of a specified cardinality with just a single run while the GPower method may require several runs due to the fact it is based on a formulation which is not directly related to the given cardinality. Experiments show that our algorithm can perform better than GPower in some data instances, and hence provides an alternative tool to efficiently perform sparse PCA.

Second, using this efficient algorithm for computing a single PC together with the Schur complement deflation scheme, we also develop an algorithm which sequentially computes multiple PCs by greedily maximizing their *adjusted variance*, i.e., a measure of total variance explained by the sparse PCs proposed in [12]. We also show in a rigorous manner why the *Schur complement deflation* scheme proposed in [8] is most suitable for the goal of maximizing the *adjusted variance*. One of the critical issues which have not been properly addressed in the existing sparse PCA algorithms is that of deciding the sparsity of each principal component. In order to address this issue, we also propose a new sparsity-controlled PCA approach whose goal is to reach a certain specified level of *relative adjusted variance* while minimizing the overall sparsity of the principal component.

Our paper is organized as follows. We present the details of our new algorithm to compute a single sparse PC in Section 2. In Section 3, we present our method for computing multiple sparse PCs based on the single sparse PCA algorithm and the *Schur complement deflation* scheme. Our sparsity-controlled PCA approach is described in Section 4. Finally, we illustrate the effectiveness of our methods by comparing them with other state-of-the-art methods on synthetic, randomly generated, and real-world, data sets.

2 Sparse PCA for finding a single PC

In this section, we introduce the formulation for the PCA problem of computing a single sparse PC with a given cardinality and present two algorithms for solving it.

2.1 Formulation Throughout this paper, we consider sparse PCA of a data matrix $V \in \mathbf{R}^{n\times p}$ whose n rows represent data points in $\mathbf{R}^p$. We assume that V is a centered matrix, i.e., a matrix whose average of its rows is the zero vector (see Section 2.3). Given a positive integer $s \le p$, single-unit sparse PCA on V consists of finding an s-sparse PC of V, i.e., a direction $0 \ne x \in \mathbf{R}^p$ with at most s nonzero entries that maximizes the variance of the projections of these data points along x. Mathematically, this corresponds to finding a vector x that solves the optimization problem

$$\max\{\|Vx\|^2/\|x\|^2 : \|x\|_0 \le s\}, \tag{2.1}$$

where $\|\cdot\|$ denotes the Euclidean norm and $\|x\|_0$ denotes the number of nonzero entries of x.

2.2 Algorithm We now present the basic ideas behind our method. The method consists of two stages. In the first stage, an active index set J of cardinality s is determined. The second stage then computes the best feasible direction x with respect to (2.1) satisfying $x_j = 0$ for all $j \notin J$, i.e., it solves the problem

$$\max\{\|Vx\|/\|x\| : x_j = 0, \forall j \notin J\}. \tag{2.2}$$

We note that once J is determined, x can be efficiently computed by using the power-iteration method (see for example [4, 11]). Hence, from now on, we will focus our attention on the determination of the index set J.

Based on the following observations, we design the procedure to determine J. First, we can alternatively consider only the optimal vectors of size $\sqrt{s}$, i.e., x which solve

$$\max\{\|Vx\|^2 : \|x\|_0 \le s,\ \|x\| \le \sqrt{s}\}. \tag{2.3}$$

Note that under the condition that $\|x\|_0 \le s$, the inequality $\|x\|_\infty \le 1$ implies that $\|x\| \le \sqrt{s}$. Hence, the problem

$$\max\{\|Vx\|^2 : \|x\|_0 \le s,\ \|x\|_\infty \le 1\} \tag{2.4}$$

is a restricted version of (2.3) and their index sets are expected to be similar. Since the objective function of problem (2.4) is convex, one of its extreme points must be an optimal solution. Note also that its set of extreme points consists of those vectors x with exactly s nonzero entries which are either 1 or -1. Ideally, we would like to choose J as the set of nonzero entries of an optimal extreme point of (2.4). However, since computing the solution (2.4) is difficult, we instead propose an algorithm to find an approximate solution of (2.4), which is then used to determine the index set J for the original problem (2.1).

Our method to find an approximate solution for (2.4) proceeds in a greedy manner as follows. Starting from $x^{(0)} = 0$, assume that at the k-th step, we have a vector $x^{(k-1)}$ with exactly $k-1$ nonzero entries which are all either 1 or -1. Also, let J_{k-1} denote the index set corresponding to the nonzero entries of $x^{(k-1)}$. We then set $x^{(k)} := x^{(k-1)} + \alpha_k e_{j_k}$, where e_i denotes the i-th unit vector and we solve

$$(j_k, \alpha_k) = \underset{j \notin J_{k-1},\, \alpha = \pm 1}{\arg\max} \|V(x^{(k-1)} + \alpha e_j)\|^2. \tag{2.5}$$

Clearly, $x^{(k)}$ is a vector with exactly k nonzero entries which are all either 1 or -1. It differs from $x^{(k-1)}$ only in the j_k-th entries which changes from 0 in $x^{(k-1)}$ to α_k in $x^{(k)}$.

Since, for fixed $j \notin J_{k-1}$ and $\alpha = \pm 1$,

$$\begin{aligned} & \|V(x^{(k-1)} + \alpha e_j)\|^2 \\ =& \|Vx^{(k-1)}\|^2 + \|v_j\|^2 + 2\alpha v_j^T V x^{(k-1)}, \end{aligned} \tag{2.6}$$

where v_j is the j-th column of V, α that maximizes the above expression is the sign of $v_j^T V x^{(k-1)}$. Hence, it follows that

$$\begin{aligned} j_k &= \underset{j \notin J_{k-1}}{\arg\max} \|v_j\|^2 + 2|v_j^T V x^{(k-1)}|, \\ \alpha_k &= \operatorname{sign}(v_{j_k}^T V x^{(k-1)}). \end{aligned} \tag{2.7}$$

Hence, we need to compute $v_j^T V x^{(k-1)}$ for every $j \notin J_{k-1}$ to find j_k. A key point to observe is that there is no need to compute $v_j^T V x^{(k-1)}$ from scratch. Instead, this quantity can be updated based on the following identity:

$$\begin{aligned} v_j^T V x^{(k-1)} &= v_j^T V(x^{(k-2)} + \alpha_{k-1} e_{j_{k-1}}) \\ &= v_j^T V x^{(k-2)} + \alpha_{k-1} v_j^T v_{j_{k-1}}. \end{aligned} \tag{2.8}$$

There are two cases to discuss at this point. If $V^T V$ is explicitly given, then the quantity $v_j^T v_{j_{k-1}}$ is just its (j, j_{k-1})-th entry, and hence there is no need to compute it. If $V^T V$ is not explicitly given, it is necessary to essentially compute its j_{k-1}-column and then extract the entries of this column corresponding to the indices $j \notin J_{k-1}$.

Our first algorithm, referred to as S_1-SPCA, is summarized in Algorithm 1. Its main difference from

Algorithm 1 S_1-SPCA

Given a centered data matrix $V \in \mathbf{R}^{n \times p}$ (or, sample covariance matrix $\Sigma = V^T V \in \mathbf{R}^{p \times p}$) and desired cardinality s, this algorithm computes an s-sparse loading vector x.

```
1: Initialization: set $x^{(0)} = 0$, $J_0 = \emptyset$.
2: Phase I: find the active index set $J$ for nonzero
   entries of $x$.
3: for $k = 1, \ldots, s$ do
4:     Find $j_k = \arg\max_{j \notin J_{k-1}} \|v_j\|^2 + 2|v_j^T V x^{(k-1)}|$
       and set $\alpha_k = \operatorname{sign}(v_{j_k}^T V x^{(k-1)})$.
5:     Set $x^{(k)} = x^{(k-1)} + \alpha_k e_{j_k}$ and $J_k = J_{k-1} \cup j_k$.
6: end for
7: Phase II: compute the solution of (2) with index
   set $J = J_s$ using the power-iteration method.
```

our second algorithm (see next section) is that it adds to J exactly one index (instead of several indices) per loop.

2.3 Complexity and Speed-up Strategy We now briefly discuss the computational complexity of the first phase of Algorithm 1. The complexity of the second phase where the power-iteration method is applied generally depends on measures other than the dimension of the underlying matrix [4]. Moreover, our computational experiments show that the first phase is generally by far the more expensive one. When $V^T V$ is explicitly given, it is easy to see that the computational complexity of the first phase of Algorithm 1 is $\mathcal{O}(ps)$. When $V^T V$ is not explicitly given, then this complexity becomes $\mathcal{O}(nps)$ in the dense case, and considerably smaller than $\mathcal{O}(sn_{nz} + ps)$ in the sparse case, where n_{nz} denotes the number of nonzero entries of V.

It is possible to develop a variant of the above algorithm which includes a constant number, say c, of indices into J in the same loop instead of just one index as in S_1-SPCA, thereby reducing the overall computational complexity of the first phase to $\mathcal{O}(nps/c)$. This simple idea consists of adding the c best indices $j \notin J_{k-1}$ according to the criteria in (2.7), say $j_{k,1}, \ldots, j_{k,c}$, to the set J_{k-1} to obtain the next index set J_k, and then set

$$x^{(k)} = x^{(k-1)} + \alpha_{j_{k,1}} e_{j_{k,1}} + \cdots + \alpha_{j_{k,c}} e_{j_{k,c}},$$

where $\alpha_{j_{k,i}}$ is the sign of $v_{j_{k,i}}^T V x^{(k-1)}$ for $i = 1, \ldots, c$.

It is easy to see that such variant performs at most $\lceil s/c \rceil$ loops and that the computational complexity of each loop is $\mathcal{O}(pn)$, thereby implying the computational complexity $\mathcal{O}(nps/c)$ for the first phase. We will refer to this variant as the S_c-SPCA method, where the c indicates the number of indices added to J in each iteration. It is considerably faster than the single index version S_1-SPCA at the expense of a small sacrifice in the quality of its solution (i.e., its variance). In our computational experiments, we usually set $c = \lceil s/10 \rceil$ so that the S_c-SPCA method performs at most 10 iterations.

One of the advantages of our algorithm is that it is very efficient especially when the data matrix is sparse. In many applications such as text mining, the data matrix W is extremely sparse. However, the centered data matrix $V = (I - \frac{ee^T}{n})W$, where e is a vector whose elements are all one's, is usually dense. Note that V is only used in the key update (2.8) in our algorithms, which asks for the computation of j_{k-1}-th column of the sample covariance matrix V^TV. The proposed algorithms can actually be implemented without explicitly forming the centered matrix V, but keeping the raw data W and computing the columns of V^TV based on the observation that

$$V^TV = W^TW - n\mu\mu^T, \tag{2.9}$$

where $\mu = W^Te/n$ consists of the average of the rows of W. As a result, we can take advantage of any available sparsity on the uncentered data W.

3 Sparse PCA with Multiple PCs

For further tasks like dimension reduction and feature extraction, more than one PC need to be computed. In this section, we discuss how our algorithm for finding a single sparse PC, together with the Schur complement deflation approach [8], can be used to develop an efficient method for finding multiple sparse PCs.

Throughout this section, we assume that $k \le \min\{n, p\}$ is the number of sparse PCs that need to be computed. Given the centered data matrix V and desired cardinalities $s_1, \ldots, s_k$ for the k loading vectors $z_1, \ldots, z_k$, our method to compute these vectors is based on the following general scheme for computing multiple sparse PCs.

```
1: Set $V_{(1)} = V$.
2: for  $i = 1 : k$ do
3:     Step 1: find a $s_i$-sparse loading vector $z_i$ for $V_{(i)}$
4:     Step 2: deflate $z_i$ from $V_{(i)}$ to obtain $V_{(i+1)}$
5: end for
```

Our multiple sparse PCA method, which we refer to as M_c-SPCA, uses the algorithm S_c-SPCA from the last section to obtain the vector z_i in above step 1. The next subsection discusses the deflation scheme used by our method to implement the above step 2.

Before describing the deflation method, we first discuss how to measure the quality of a set of k PCs. If $z_1, \ldots, z_k \in \mathbf{R}^p$ are the loading vectors of the exact PCs for a given centered data matrix $V \in \mathbf{R}^{n\times p}$, then the total variance explained by the corresponding multiple PCs $Vz_1, \ldots, Vz_k$ is given by

$$\|VZ\|_F^2 = tr(Z^TV^TVZ) = \sigma_1^2 + \cdots + \sigma_k^2, \tag{3.10}$$

where $Z = [z_1, \ldots, z_k]$ and the scalars $\sigma_1, \ldots, \sigma_k$ are the largest singular values of V. However, in the case where the z_i's are loading vectors for approximate sparse PCs of V, the quantity (3.10) is no longer suitable to measure the aggregate variance explained by $Vz_1, \ldots, Vz_k$, since it does not take into account the correlation (i.e., lack of orthogonality) between these components. A proper way of measuring the variance explained by these components is the *adjusted variance* introduced by Zou et al [12], namely:

$$AdjVar(VZ) = tr(R^2), \tag{3.11}$$

where $VZ = QR$ is the reduced QR factorization of VZ, i.e., R is a $k \times k$ upper-triangular matrix and Q is an $n \times k$ matrix satisfying $Q^TQ = I$.

3.1 Schur complement deflation versus adjusted variance In this subsection, we discuss a deflation technique proposed in [8], which is used as a key ingredient by our multiple sparse PCA method.

Given $V_{(i)} \in \mathbf{R}^{n\times p}$ and $z_i \in \mathbf{R}^p$, the Schur complement deflation scheme computes $V_{(i+1)}$ according to

$$V_{(i+1)} \leftarrow \left(I - \frac{V_{(i)}z_iz_i^TV_{(i)}^T}{\|V_{(i)}z_i\|^2} \right) V_{(i)}. \tag{3.12}$$

It is closely related to the goal of maximizing adjusted variance in a greedy manner, as the following two results show.

Proposition 3.1. *Assume that $V \in \mathbf{R}^{n\times p}$ and a set of loading vectors $Z_i = [z_1, \ldots, z_i] \in \mathbf{R}^{p\times i}$ whose corresponding PCs $Vz_1, \ldots, Vz_i$ are linearly independent are given. If $V_{(1)} = V$ and $V_{(2)}, \ldots, V_{(i+1)}$ are generated according to* (3.12)*, then*

$$V_{(i+1)} = (I - Q_iQ_i^T)V, \tag{3.13}$$

where Q_iR_i is the reduced QR factorization of VZ_i.

Proof. We will prove by induction on j that $V_{(j+1)} = (I - Q_jQ_j^T)V$ for every $j = 1, \ldots, i$, where Q_jR_j is the reduced QR factorization of VZ_j.

When $j = 1$, we have $VZ_1 = Vz_1 = Q_1R_1$, where $R_1 = \|Vz_1\|$ and $Q_1 = Vz_1/\|Vz_1\|$. The latter formula for Q_1, (3.12) with $i = 1$ and the assumption $V = V_{(1)}$ then imply that $V_{(2)} = (I - Q_1 Q_1^T)V$. Now assume that

$$V_{(j)} = (I - Q_{j-1} Q_{j-1}^T)V \tag{3.14}$$

is true for some $2 \le j \le i$. Since by assumption Vz_j does not belong to the subspace spanned by $Vz_1, \dots, Vz_{j-1}$, and hence to the column space of Q_{j-1}, we conclude that

$$V_{(j)} z_j = (I - Q_{j-1} Q_{j-1}^T) V z_j \neq 0 \tag{3.15}$$

and the columns of $Q_j := [Q_{j-1}, q_j]$ are orthonormal, where

$$q_j := \frac{V_{(j)} z_j}{\|V_{(j)} z_j\|}. \tag{3.16}$$

Using the above definition of q_j and (3.15), we easily see that the reduced QR factorization of VZ_j is $Q_j R_j$, where

$$R_j = \begin{bmatrix} R_{j-1} & Q_{j-1}^T V z_j \\ 0 & \|V_{(j)} z_j\| \end{bmatrix}.$$

Moreover, by (3.12), (3.14) and (3.16), we have

$$\begin{aligned} V_{(j+1)} = & \left(I - \frac{V_{(j)} z_j z_j^T V_{(j)}^T}{\|V_{(j)} z_j\|^2} \right) V_{(j)} \\ = & (I - q_j q_j^T)(I - Q_{j-1} Q_{j-1}^T)V \\ = & (I - Q_j Q_j^T)V, \end{aligned}$$

which concludes our proof. $\square$

PROPOSITION 3.2. *Let $Z_i = [z_1, \dots, z_i] \in \boldsymbol{R}^{p \times i}$ be given and assume that $VZ_i = Q_i R_i$ is the reduced QR factorization of VZ_i. Then for any $z \in \boldsymbol{R}^p$,*

$$AdjVar(V[Z_i, z]) - AdjVar(VZ_i) = \|(I - Q_i Q_i^T)Vz\|^2. \tag{3.17}$$

Proof. If Vz lies in the subspace spanned by the columns of the matrix VZ_i, then one can easily see that both sides of (3.17) are zero.

Assume then that Vz is not in the above subspace and note that the vector $w := (I - Q_i Q_i^T)Vz \neq 0$. Defining $q = w/\|w\|$, we can easily see that the columns of $[Q_i, q]$ are orthonormal and

$$V[Z_i, z] = [Q_i, q] \begin{bmatrix} R_i & Q_i^T V z \\ 0 & \|w\| \end{bmatrix}$$

is the reduced QR factorization of $V[Z_i, z]$. Hence, we conclude that the $AdjVar(V[Z_i, z]) = tr(R_i^2) + \|w\|^2$, which is equivalent to (3.17). $\square$

A greedy method for finding k sparse loading vectors with given cardinalities $s_1, \dots, s_k$ would proceed as follows. Assuming that $i < k$ sparse loadings vectors $z_1, \dots, z_i$ have already been computed, the $(i+1)$-th loading vector z_{i+1} is chosen so as to maximize $AdjVar(V[Z_i, z])$ subject to the condition that $\|z\| = 1$ and $\|z\|_0 = s_{i+1}$, which, according to the Proposition 3.2, is equivalent to the max-variance problem

$$\max\{\|(I - Q_i Q_i^T)Vz\|^2 : \|z\| = 1, \|z\|_0 = s_{i+1}\}.$$

Moreover, according to the Proposition 3.1, this is equivalent to

$$\max\{\|V_{(i+1)} z\|^2 : \|z\| = 1, \|z\|_0 = s_{i+1}\}.$$

The latter problem can be solved using the algorithms discussed in Section 2.

4 Sparsity-controlled PCA

According to our best knowledge, there is no general guideline for choosing the cardinality or penalty parameter when performing sparse PCA. For example, there is no clear answer for how to assign given cardinalities among the multiple PCs. However, in practice, one can expect the PCs to explain a certain proportion of the variance explained by standard PCs (i.e. with no constraints such as sparsity), using as few variables as possible. For the sake of convenience, we define the *relative adjusted variance* as

$$rAdjVar(VZ) = \frac{AdjVar(VZ)}{\sigma_1^2 + \cdots + \sigma_k^2}. \tag{4.18}$$

Based on this measure, we propose a problem which minimizes the cardinality of the loading vectors, with the constraint that the *relative adjusted variance* (4.18) is larger than or equal to a certain specified threshold $\rho \in (0, 1)$.

For the single PC case, the problem is formulated as

$$\min\{\|z\|_0 : \|Vz\|^2 \ge \rho\sigma_1^2, \|z\| \le 1\}, \tag{4.19}$$

while for the multiple PC case, the problem is formulated as

$$\begin{aligned} \min \ & \sum_{i=1}^{k} \|z_i\|_0 \\ \text{s.t. } & rAdjVar(V[z_1, \dots, z_k]) \ge \rho \\ & \|z_i\| \le 1, \ i = 1, \dots, k. \end{aligned} \tag{4.20}$$

To solve problem (4.19) and (4.20), we use a variant of our algorithm S_c-SPCA combined with the

Algorithm 2 Sparsity-Controlled PCA (SC_c-PCA)

Given the objective proportion ρ, a centered data matrix V and the desired number k of sparse PCs, this algorithm computes k sparse loading vectors $z_1, \dots, z_k$.

```
1: Compute true variance:
   compute the first $k$ singular values $\sigma_1, \dots, \sigma_k$ of $V$.
2: for $i$ from 1 to $k$ do
3:     Set $J = \emptyset$ and $z_i = 0$.
4:     while $rAdjVar(V[z_1, \dots, z_i]) < \rho$ do
5:         Find the top $c$ best indices $j_1, \dots, j_c \notin J$ maximizing the increase of the variance, and add $j_1, \dots, j_c$ to $J$ (see Subsection 2.3).
6:         Compute $z_i = \arg\max\{\|Vx\|^2 : \|x\| \leq 1, x_j = 0, \forall j \notin J\}$ by using the power-iteration method starting from current $z_i$.
7:     end while
8:     Schur complement deflation:
       $V \leftarrow (I - \frac{Vz_iz_iV^T}{\|Vz_i\|^2})V$.
9: end for
```

Schur complement deflation scheme (3.12) to sequentially compute the loading vectors z_i's . Basically it differs from the algorithm outlined at the end of Section 3 in that it stops updating z_i whenever the criteria $rAdjVar(V[z_1, \dots, z_i]) \geq \rho$ is satisfied. Our sparsity-controlled PCA heuristic, which we refer to as SC_c-PCA, is summarized in Algorithm 2.

Finally, note that the relative adjusted variance defined by (4.18) can be sequentially updated according to (3.17).

5 Experiment results and comparison

5.1 Synthetic Data In this subsection, we use synthetic data to show that our algorithm is able to recover the "true" sparse PC. The procedure, proposed by Shen and Huang [10], consists of generating random data with a covariance matrix having dominant sparse eigenvectors. The first step is to artificially specify the two dominant sparse eigenvectors of the covariance matrix. Then the matrix of eigenvectors U are constructed by randomly generating the remaining vectors and applying the Gram-Schmidt procedure to the resulting matrix to obtain an orthogonal one. Next, we set the covariance matrix as $\Sigma = UDU^T$, where D is a positive diagonal matrix whose two largest diagonal elements are associated with the specified dominant eigenvectors. Then, observations are sampled from a zero mean normal distribution with covariance matrix Σ.

In our experiments, observations are drawn from $\mathbf{R}^{500}$ with 500×500 covariance matrix $\Sigma = UDU^T$. In the diagonal matrix D, the first ten eigenvalues are set as $400, 300, 100, 100, 50, 50, 50, 50, 30, 30$ and all the rest eigenvalues are 1's. The first two eigenvectors u_1 and u_2, associated with the two dominant eigenvalues of the covariance matrix, are chosen to be 10% sparse as follows:

$$\begin{cases} u_{1i} = \frac{1}{\sqrt{50}} & 1 \leq i \leq 50 \\ u_{1i} = 0 & i > 50 \end{cases} \qquad \begin{cases} u_{2i} = -\frac{1}{\sqrt{50}} & 31 \leq i \leq 40 \\ u_{2i} = \frac{1}{\sqrt{50}} & 41 \leq i \leq 80 \\ u_{2i} = 0 & \texttt{otherwise}. \end{cases}$$

To test whether our algorithms S_1-SPCA and S_c-SPCA can efficiently recover the dominant eigenvectors, we use them to compute two unit-norm sparse loading vectors $z_1, z_2 \in \mathbf{R}^{500}$, which are expected to be close to u_1 and u_2. We perform our tests on two sets of matrices. The first set consists of 200 data matrices $V \in \mathbf{R}^{50\times500}$, whose rows are sampled observations as explained above. The other set consists of 200 data matrices $V \in \mathbf{R}^{200\times500}$ generated in the same way. We then measure the scalar products $|z_1^T z_2|$, $|u_1^T z_1|$ and $|u_2^T z_2|$ and average them over the corresponding 200 data matrices. When both quantities $|u_1^T z_1|$ and $|u_2^T z_2|$ are greater than 0.95, we name it a *successful identification* of u_1 and u_2 and the total number of *successful identifications* are provided in the last column of Table 1 and 2.

In our experiments, we feed the cardinality 50 to S_1-SPCA and S_c-SPCA. We choose $c = 5$ for S_c-SPCA so that it only performs 10 iterations. We compare these two variants with the GPower variants $GPower_0$ and $GPower_{0,k}$[5], both based on the L_0 penalization, since the experiments in [5] show that the L_0 version of GPower is generally more efficient than its L_1 version. The difference between $GPower_0$ and $GPower_{0,k}$ is that the former one finds one sparse PC at a time while the latter one finds k sparse PCs simultaneously. Both versions of the GPower methods consist of solving a nonconcave maximization penalized problem depending on a parameter γ, which affects the sparsity level of the solution. To search for the parameter γ that yields the desired solution sparsity, we use a naive bisection scheme, initiating γ according to the empirical experiment in [5]. We stop the bisection scheme when the resulting vector has a cardinality between 45 and 55 or the number of trials reaches 40. For S_1-SPCA, S_5-SPCA and $GPower_0$, we employ Schur complement deflation (3.12) before searching for the second PC. For comparison, we measure the average time for *a single run* of each algorithm, even though GPower methods actually cost several times more due to the search of the proper value of γ. The results of the two problem sets are presented respectively in Table 1 and 2.

Table 1: Sparse PCA on 50 × 500 synthetic data matrix. Four algorithms are performed to compute the first two sparse loading vectors, which are then compared to the loading vectors of true PCs. Measurements are averaged over 200 repeated experiments, except that the last column counts the number of successful identifications out of 200.

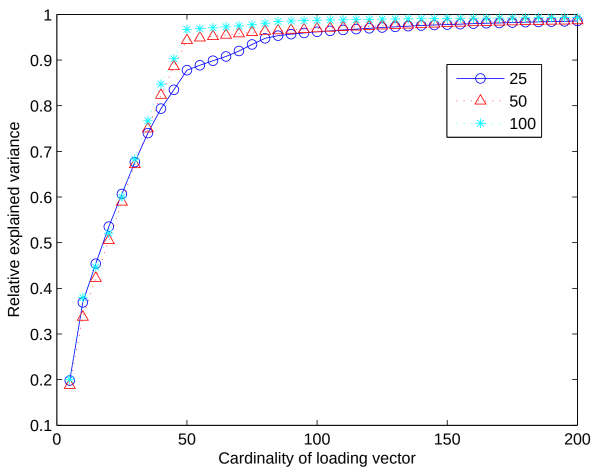
	$\lvert z_1^T z_2\rvert$	$\lvert u_1^T z_1\rvert$	$\lvert u_2^T z_2\rvert$	# trials	single run time(10^{-2} seconds)	# success
S_1-SPCA	0.0350	0.8067	0.8029	1	4.8	155
S_5-SPCA	0.0383	0.8659	0.8626	1	3.6	164
$GPower_0$	0.0324	0.8142	0.8122	3.9	2.3	147
$GPower_{0,2}$	0.0375	0.8119	0.7931	7.1	1.9	149

Table 2: Sparse PCA on 200 × 500 synthetic data matrix. Four algorithms are performed to compute the first two sparse loading vectors, which are then compared to the loading vectors of true PCs. Measurements are averaged over 200 repeated experiments, except that the last column counts the number of successful identifications out of 200.

	$\lvert z_1^T z_2\rvert$	$\lvert u_1^T z_1\rvert$	$\lvert u_2^T z_2\rvert$	# trials	single run time(10^{-2} seconds)	# success
S_1-SPCA	0.0172	0.9882	0.9892	1	10.4	198
S_5-SPCA	0.0180	0.9883	0.9893	1	7.6	198
$GPower_0$	0.0191	0.9812	0.9830	1.9	6.7	194
$GPower_{0,2}$	0.0157	0.9774	0.8748	9.7	4.4	171

On the other hand, even though we feed the cardinality 50 to our algorithm S_1-SPCA and S_5-SPCA in the above experiments, we can actually detect the inherent cardinality of the PCs by plotting the variance-cardinality (i.e., $\|Vz\|^2/\sigma_1^2$ vs. $\|z\|_0$) trade-off curve, where σ_1 is the largest singular value of V. On synthetic data matrices $V \in \mathbf{R}^{25\times 500}, \mathbf{R}^{50\times 500}$ or $\mathbf{R}^{100\times 500}$ with different number of observations, a single run of S_5-SPCA algorithm is able to find the critical point which indicates the inherent sparsity associated with the first PC. It is shown in Figure 1 that as the algorithm progresses and the cardinality of the loading vector z increases, the increase rate of the relative adjusted variance $\|Vz\|^2/\sigma_1^2$ changes drastically when the cardinality of z bypasses 50.

Figure 1: The variance-cardinality trade-off curves on synthetic data matrices, where number of observations n is 25, 50 and 100. Each curve is obtained from a single run of S_5-SPCA algorithm with 5 indices updated in each iteration.



5.2 Randomly Generated Data In this subsection, we evaluate the numerical quality and speed of both versions of our method S_1-SPCA and S_c-SPCA($c > 1$) using a set of randomly generated sparse matrices. Since both of our algorithms are based on an L_0 constraint, we choose $GPower_0$ as the counterpart, which is the L_0 penalized version of the state-of-the-art sparse PCA algorithm GPower method [5]. Another reason is that the experiments in [5] show that the L_0 version of GPower is generally more efficient than the L_1 version. For S_c-SPCA, we set $c = \lceil s/10 \rceil$ as the number of indices added to the active index set at each iteration, where s is the desired cardinality. Experiments are performed in MATLAB with the codes for all three methods optimized for sparse matrix computation. All results are averaged over ten repeated measurements.

In the first experiment, we randomly generate sparse square matrices W with dimension p varying from 100 to 2000, with their sparsity (i.e., proportion of nonzero entries) set to 20%. For S_1-SPCA and S_c-SPCA, we set the required cardinality s to $p/5$. For $GPower_0$, we set the penalty parameter $\gamma = 0.002 \max_i \|v_i\|^2/p$, where v_i's are the columns of the centered data matrix V. We then measure the average cpu time for a single run of each algorithm. In Figure 2(a), the left graph plots the curve of the running time (in seconds) against matrix size, which indicates that S_1-SPCA and S_c-SPCA are fast on large sparse matrices. Note that here we directly input the parameter γ to $GPower_0$ rather than using bisection to compute it, and hence the running times for a single run of all methods are compared. However, we observe that it usually takes several runs of $GPower_0$ to obtain a sparse PC with the desired level of sparsity.

Using the same set of matrices, we compare in the right graph of 2(a) the relative explained variance of a single 20%-sparse PC computed by S_1-SPCA, S_c-SPCA and $GPower_0$. We input the desired cardinality $s = p/5$ for S_1-SPCA and S_c-SPCA while for $GPower_0$ we use a bisection scheme on the penalty parameter γ to obtain a 20%-sparse PC. The graph plots the curve of explained variance vs. matrix size. Observe that S_1-SPCA achieves better numerical quality than $GPower_0$ and that S_c-SPCA with $c > 1$ is faster than $GPower_0$ at the expense of a little loss in solution quality.

In the second experiment, the size of the square matrix p is fixed to 5000. $GPower_0$ is then run for 20 different penalty parameters, namely $\gamma = 0.01 \max_i \|v_i\|^2/p/\sqrt{j}, j = 1, \ldots, 20$ and their cardinalities are input to both versions of our method to obtain corresponding solutions of identical cardinalities. The trade-off curve of the explained variance against the cardinality of the solution is displayed in the first graph of Figure 2(b). The second graph plots the running time against the cardinality for each method, where only the time for a single run of $GPower_0$ is reported. Observe that S_1-SPCA outperforms $GPower_0$ in terms of solution quality but, according to this experiment, its running time increase proportionally with the desired solution cardinality. On the other hand, the running time of S_c-SPCA barely increases as the desired solution cardinality does, at the expense of a slight sacrifice in solution quality.

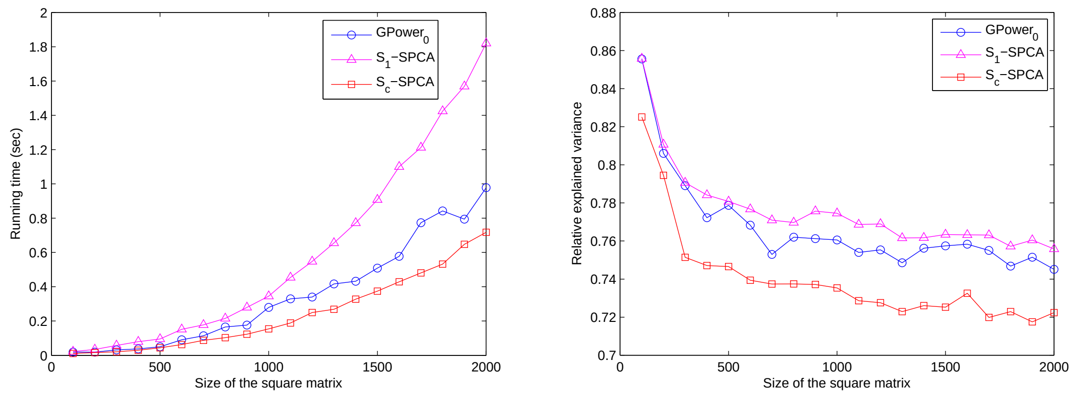
Our third experiment consists of two parts. In the first (resp., second) one, we randomly generate $n \times p$ matrices with $n/p = 0.1$ (resp., $n/p = 10$), with the sparsity set to 20% and with their larger dimension increasing from 200 to 4000. For S_c-SPCA, we set the required cardinality s to be $p/10$. For $GPower_0$, we set the penalty parameter $\gamma = 0.01 \max_i \|v_i\|^2/n$ (resp., $\gamma = 0.00005 \max_i \|v_i\|^2/n$). The corresponding plots of the running time against the size of the larger dimension are given in Figure 2(c). Observe that, while the speed of S_c-SPCA method is comparable to $GPower_0$ when $n/p = 0.1$, it is faster than $GPower_0$ when $n/p = 10$.

5.3 Biology data sets In this subsection we compare the speed of S_1-SPCA with another greedy method proposed in [2], namely PathSPCA, as well as $GPower_0$ and $GPower_1$, on several biology data sets, which are available as covariance matrices of size 500, 500 and 118. For $S_1 - SPCA$ and $PathSPCA$, we fix the desired cardinality as 10% of the matrix dimension. Both of these two methods can be implemented using either the covariance matrix or the Cholesky factor [4] of the covariance matrix. For $GPower_0$ and $GPower_1$, we use the Cholesky factor of the covariance matrix as input. We fix the parameter $\gamma = 0.1$ for $GPower_0$ and $GPower_1$, without being concerned with the obtained solution sparsity. The accumulated running time (in seconds) for 10 runs of each method is shown in Table 3. The two times reported for each one of $S_1 - SPCA$ and $PathSPCA$ correspond to implementations based on the covariance matrix and the Cholesky factor.

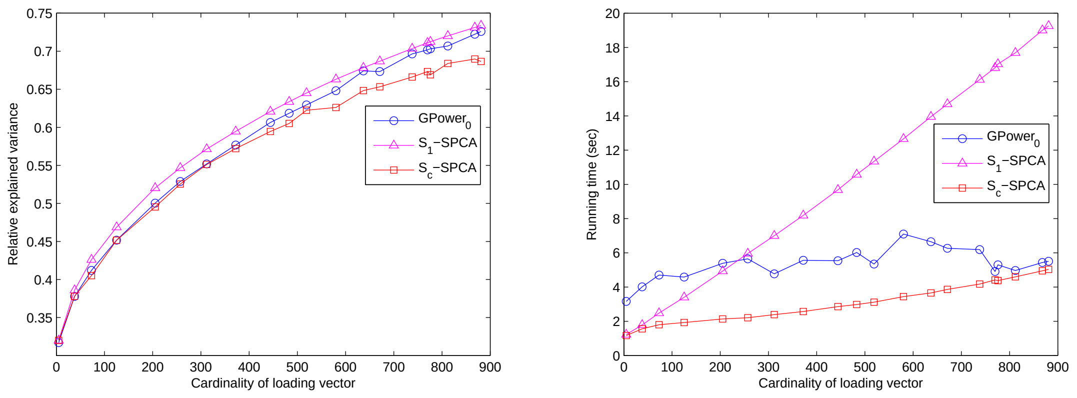
5.4 Image data In this subsection, we compare the algorithms S_5-SPCA and SC_5-PCA with corresponding variants of GPower method using a real-world data matrix of size 5000×784 from the handwritten digits database MNIST [6]. Each row of the matrix corresponds to a image with 28 by 28 pixels, and hence a vector of size 784.

Figure 3 compares S_5-SPCA with $GPower_0$ in terms of speed and explained variance performance. For $GPower_0$ we choose 20 different parameter values, i.e., $\gamma = 0.00002 \max_i \|v_i\|^2/n/j$ for $j = 1, \ldots, 20$, to obtain 20 PCs of different sparsity levels in order to draw the corresponding curves in the two graphs of Figure 3. On the other hand for our method S_5-SPCA, we draw the corresponding curves by generating PCs whose sparsity levels are uniformly increased. In Figure 5, the first graph plots running time against the cardinality of solution, while the second graph plots the relative explained variance of the solution against its cardinality. Observe that on this data set, S_5-SPCA method outperforms $GPower_0$ in terms of speed and generates sparse PCs with quality close to $GPower_0$.

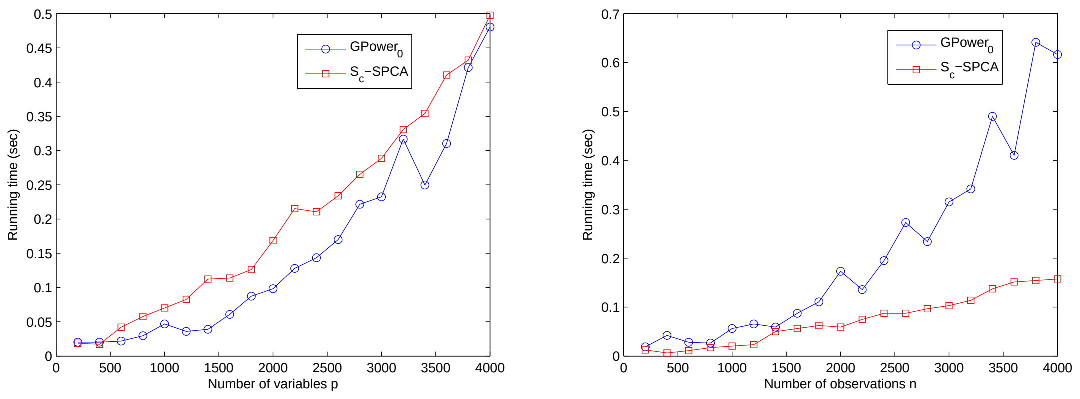
In our second experiment with the above data matrix, we are interested in finding six sparse PCs $z_1, \ldots, z_6$ to explain at least 70% of the variance obtained by standard PCA (i.e., $rAdjVar(VZ) \geq 0.7$).



(a) In these two plots, the size of the matrix is increasing from 100 to 2000. The left plot displays the curves of the time for a single run of all three methods versus the size of the matrix. The cardinality of solution for S_1-SPCA and S_c-SPCA is fixed as $p/5$, while the parameter in $GPower_0$ is given by $\gamma = 0.002 \max_i \|v_i\|^2/p$ so that no line search is applied. The right plot displays curves of the explained variance $\|Vz\|^2/\sigma_1^2$. $GPower_0$ uses a bisection searching scheme to obtain z with desired cardinality $p/5$, which can be directly achieved by S_1-SPCA and S_c-SPCA.



(b) In the second experiment, we fix the data matrix as a square sparse matrix of size 5000 and run $GPower_0$ with different parameters. Then we feed the cardinality of the solution z computed by $GPower_0$ to both versions of our method so that solutions of all three methods have exactly the same cardinality. The left plot displays the trade-off curve of variance against cardinality, and right plot displays the curve of running time against cardinality.



(c) In the third experiment, as the number of variables p increases from 200 to 4000 and $n/p = 0.1$, the running time curve is shown on the left, and as the number of observations n increases from 200 to 4000 and $n/p = 10$, the running time curve is shown on the right.

Figure 2: Experiments on randomly generated matrix.

Table 3: Comparison of running time of S_1-SPCA, $PathSPCA$, $GPower_0$ and $GPower_1$ on three biology data sets, which are available as covariance matrices of size 500, 500 and 118. The columns ending with "-cov" (resp., "-factor") correspond to the implementations of S_1-SPCA and PathSPCA based on the covariance matrix (resp., Cholesky factor).

Data set	S_1-SPCA-cov	S_1-SPCA-factor	PathSPCA-cov	PathSPCA-factor	$GPower_0$	$GPower_1$
Lymphoma	0.0296	0.2714	3.3509	2.3977	0.1435	0.1934
Colon	0.0312	0.2761	3.3166	2.3884	0.1232	0.1576
Eisen	0.0062	0.0125	0.3307	0.1716	0.0078	0.0156

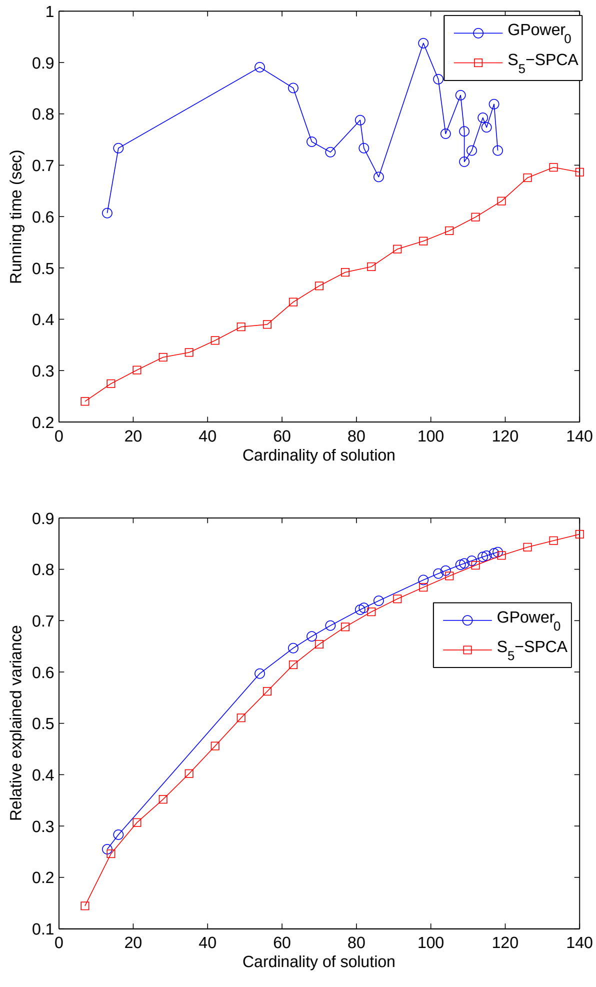


Figure 3: Experiments on 5000 handwritten digits images from MNIST database. The top one plots running time against cardinality curves while the bottom one plots variance against cardinality curves.

Towards this goal, we have also implemented a variant of $GPower_0$ similar to SC_c-PCA, which consecutively computes the z_i's using $GPower_0$ with a modified bisection scheme for finding γ based on the stopping criteria $rAdjVar(V[z_1, \ldots, z_i]) \geq 0.7$. Table 4 compares SC_5-PCA with the aforementioned variant of $GPower_0$ and a variant based on $GPower_{0,6}$ (see [5]) in terms of speed and sparsity performance. As opposed to the first variant, the latter one computes the six sparse PCs simultaneously by solving a nonconcave maximization penalized problem with matrix variables, whose penalty parameter is searched via a bisection scheme to yield the desired 70% relative adjusted variance. It turns out that $GPower_0$ with γ randomly initialized took on average 24 runs per PC. We observe that $GPower_{0,6}$ fails to find a set of sparse PCs $z_1, \ldots, z_6$ with relative adjusted variance in the interval $[0.7, 0.75]$. Hence we include in Table 4 the solution with relative adjusted variance closest to, but smaller than, 0.7. Note that such solution is of poor quality in terms of sparsity, even though it explains less relative adjusted variance than the solution obtained by other methods. In Table 4, the first 6 columns specifies the cardinality of each PC, the 7th column describes the sum of the 6 cardinalities, the 8th column specifies the average number of runs per PC for each method, and the 9th and 10th column give the total running time and the actual relative adjusted variance obtained.

5.5 Sparsity-controlled PCA on Pitprops data

The pitprops data is a benchmark data for performing sparse PCA, which consists of 180 observations and 13 measured variables. While the first 6 PCs can explain 86.9% of the total variance explained by all 13 principal components, they are all dense and hard to explain. Previous studies (see [5, 9]) have found 6 sparse PCs based on the 13×13 sample covariance matrix, but the ways they choose the cardinality of each PC are very ad hoc. In contrast, the sparsity-controlled PCA approach of Section 4 provides a specific recipe for computing sparse PCs with a given relative adjusted variance. To illustrate the effectiveness of our approach,

Table 4: Sparsity-controlled PCA on digits image data. Four algorithms are used to compute the first six sparse loading vectors with relative adjusted variance at least 0.7. Here, $GPower_0$ denotes the variant of $GPower_0$ which consecutively computes the 6 sparse PCs. Since $GPower_{0,6}$, which computes the 6 sparse PCs simultaneously, fails to obtain a solution with relative adjusted variance in the interval $[0.7, 0.75]$, we give the best available result for it.

	$\|z_1\|_0$	$\|z_2\|_0$	$\|z_3\|_0$	$\|z_4\|_0$	$\|z_5\|_0$	$\|z_6\|_0$	total cardinality	# trials	total time	$rAdjVar(VZ)$
SC_1-PCA	76	73	57	62	41	43	352	1	25.38	0.7004
SC_5-PCA	80	75	60	55	45	40	355	1	11.82	0.7036
$GPower_0$	88	76	60	39	71	61	395	24	25.69	0.7308
$GPower_{0,6}$	535	0	0	340	0	317	1192	NA	NA	0.6755

Table 5: Sparsity-controlled PCA on pitprops data with at least 0.9 relative adjusted variance .

Variables	z_1	z_2	z_3	z_4	z_5	z_6
topdiam	0.4229	0	0	0	0	0
length	0.4295	0	-0.2610	0	0	0
moist	0	0.6676	0	0	0	0
testsg	0	0.6435	0	0	0	0
ovensg	0	0	0.5377	0	0	0.7157
ringtop	0.2695	0	0.4897	0	0.2898	0
ringbut	0.4043	0	0.3682	0	0	0
bowmax	0.3131	0	0	0	-0.3549	0
bowdist	0.3782	0	0	0	0	0
whorls	0.3994	0	0	0	-0.3332	0
clear	0	0.2030	0	0.8723	0.4030	0
knots	0	0.3147	0	-0.4890	0.7188	0
diaknot	0	0	-0.5172	0	0	0.6984
Cardinality	7	4	5	2	5	2

Table 6: Sparsity-controlled PCA on Leukemia data matrix. Four algorithms are performed to compute the first two sparse loading vectors explaining 70% relative adjusted variance. Here, $GPower_0$ denotes the variant of $GPower_0$ which consecutively computes the 2 sparse PCs. Since $GPower_{0,2}$, which computes the 2 sparse PCs simultaneously, fails to obtain a solution with relative adjusted variance in the interval $[0.7, 0.75]$, we give the best available result for it.

	$\|z_1\|_0$	$\|z_2\|_0$	$\|z_1\|_0 + \|z_2\|_0$	$\|z_1^T z_2\|$	# trials	total time(seconds)	$rAdjVar(Vz_1, Vz_2)$
SC_{10}-PCA	1930	1650	3580	0.0290	1	54.4	0.7003
$SC_1$43-PCA	2145	1430	3575	0.0043	1	9.8	0.7076
$GPower_0$	1962	1659	3621	0.0383	31	7.1	0.7059
$GPower_{0,2}$	6677	0	6677	NA	NA	NA	0.6840

we consider the problem of finding 6 sparse PCs with relative adjusted variance at least 0.9. It turns out that a single run of SC_1-PCA finds a set of 6 sparse PCs with cardinality pattern $7-4-5-2-5-2$ explaining 90.69% of the variance explained by the first 6 standard PCs. The resulting loading vectors are shown in the Table 5. To achieve the same goal, we also use $GPower_{0,6}$ method with balancing parameters $N = (1, 1/2, 1/3, 1/4, 1/5, 1/6)$ (see [5]), while the penalty parameter γ is again obtained via bisection scheme. It usually takes $GPower_{0,6}$ more than 20 runs to obtain the set of 6 PCs with relative adjusted variance close to 0.9. The best result given by $GPower_{0,6}$ is a set of 6 sparse PCs with cardinality pattern $12-11-11-2-1-1$ that explains 90.12% of the variance explained by the first 6 standard PCs. *Observe that the total cardinality of the solution obtained by SC_1-PCA is 25, while the one obtained by $GPower_{0,6}$ is* 38.

5.6 Sparsity-controlled PCA on Leukemia data The Leukemia data set is a DNA microarray data set consisting of 72 dense observations in $\mathbf{R}^{7129}$. This a typical problem with many more variables than observations. Using SC_c-PCA algorithm, we minimize the total cardinality of two PCs z_1 and z_2 to explain at least 70% of the variance explained by the first two dense PCs(i.e., $rAdjVar(Vz_1, Vz_2) = 0.7$). In our experiment, we use $c = 10$ and $c = \lceil p/50 \rceil = 143$. The motivation for the latter choice of c comes from the fact that if each PC were 20%-sparse, Algorithm 2 would perform 10 inner loops per PC. We compare our method with the variant of $GPower_0$ outlined in the third paragraph of Subsection 5.4. It turns out that with a randomly initialized parameter, it takes dozens of runs for $GPower_0$ to obtain z_1 and z_2 with desired relative adjusted variance. We also include the result obtained by the block version of GPower method, namely $GPower_{0,2}$, with penalty parameter γ found via a bisection scheme. Again, $GPower_{0,2}$ fails to find z_1 and z_2, with relative adjusted variance in the interval $[0.7, 0.75]$. In Table 6, the first three columns specifies the cardinality of each PC and their sum, the 4th column gives the cosine of the angle between them, the 5th column specifies the average number of runs per PC for each method, and the 6th and 7th column give the total running time and the actual relative adjusted variance obtained.

5.7 Sparsity-controlled PCA on document data The document data set we use is the NIPS full papers data set [3] with 1500 documents and 12419 words forming a large sparse matrix of size 1500 by 12419. Using (2.9) and (3.13), we carefully designed our code and also modified $GPower_{0,k}$'s code to avoid loss of sparsity due to centering and deflation. It takes 220 seconds for $GPower_{0,6}$ to find 6 PCs with relative adjusted variance close to 0.9. With balancing parameter $N = (1, 1/2, 1/3, 1/4, 1/5, 1/6)$ and a bisection scheme for computing the penalty parameter γ, the loading vectors found by $GPower_{0,6}$ have cardinality pattern $(3679, 50, 120, 43, 89, 42)$ with $rAdjVar = 0.9175$. On the other hand, SC_{10}-PCA with $\rho = 0.9$ finds 6 sparse PCs with cardinality pattern $(20, 140, 70, 110, 170, 50)$ and relative adjusted variance 0.9006 in only 11.8 seconds. *Observe that both the running time of the latter method and the cardinality of its solution are substantially smaller than the corresponding ones for the first method.*

References

[1] A. d Aspremont, L. El Ghaoui, M.I. Jordan, and G.R.G. Lanckriet. A direct formulation for sparse PCA using semidefinite programming. *SIAM review*, 49(3):434, 2007.

[2] A. d'Aspremont, F. Bach, and L.E. Ghaoui. Optimal solutions for sparse principal component analysis. *The Journal of Machine Learning Research*, 9:1269–1294, 2008.

[3] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[4] G.H. Golub and C.F. Van Loan. *Matrix computations.* Johns Hopkins Univ Pr, 1996.

[5] M. Journée, Y. Nesterov, P. Richtárik, and R. Sepulchre. Generalized power method for sparse principal component analysis. *The Journal of Machine Learning Research*, 11:517–553, 2010.

[6] Y. LeCun and C. Cortes. The MNIST database of handwritten digits, 2009.

[7] Z. Lu and Y. Zhang. An Augmented Lagrangian Approach for Sparse Principal Component Analysis. *Arxiv preprint arXiv:0907.2079*, 2009.

[8] L. Mackey. Deflation methods for sparse pca. *Advances in Neural Information Processing Systems*, 21:1017–1024, 2009.

[9] B. Moghaddam, Y. Weiss, and S. Avidan. Spectral bounds for sparse PCA: Exact and greedy algorithms. *Advances in Neural Information Processing Systems*, 18:915, 2006.

[10] H. Shen and J.Z. Huang. Sparse principal component analysis via regularized low rank matrix approximation. *Journal of multivariate analysis*, 99(6):1015–1034, 2008.

[11] G.W. Stewart and GW Stewart. *Introduction to matrix computations.* Academic press New York, 1973.

[12] H. Zou, T. Hastie, and R. Tibshirani. Sparse principal component analysis. *Journal of computational and graphical statistics*, 15(2):265–286, 2006.