A Fast Algorithm for Sparse PCA and a New Sparsity Control Criteria

Yunlong He * Renato D.C. Monteiro † Haesun Park ‡

October 15, 2010

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 datasets 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-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 prespecified 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 nonzeros), 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 regression-type optimization problem which is then solved by Lasso-type algorithms. 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] for sparse PCA consists of solving a semi-definite 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 to solve a new semi-definite 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 [5] recently formulate 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.

*School of Mathematics, Georgia Institute of Technology (heyunlong@gatech.edu).

†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 Grant ONR N00014-08-1-0033.

‡School of Computational Science and Engineering, Georgia Institute of Technology (hpark@cc.gatech.edu).The work of this author was also 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.

For methods that find multiple sparse PCs sequentially, projection matrix deflation is used to find subsequent directions. Other deflation methods have been studied in [8]. A different multiple sparse PCA approach is proposed in [7] based on a formulation enforcing near orthogonality of the PCs, which is then solved by an augmented Lagrangian approach. Throughout this paper, we 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.

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 at 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 considerably better than GPower in some data instances, and hence provides an alternative tool to efficiently perform sparse PCA. 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 rigorous manner why the *Schur complement deflation* scheme proposed in [8] is the most suitable one when the goal is to maximize the *adjusted variance*.

Though couples of sparse PCA algorithms emerges in the past a few years, it is still not clear how much sparsity we need to impose on the PCs. As another contribution of this paper, we try to address this problem by formulating a new sparsity-controlled PCA problem based on *relative adjusted variance*. We provide an algorithm to control sparsity of PCs combining a variant of our first algorithm and Schur complement deflation. Last but not the least, we will show the application of this sparsity-controlled PCA framework on real-world data sets.

We organize the rest of our paper as follows. We present the details of our new algorithm to compute single sparse PC in Section 2, which includes formulation of problem, description of algorithm, complexity analysis and speed-up strategy. In Section 3, we focus on computing multiple sparse PCs. The relation between *adjusted variance* and *Schur complement deflation* is also discussed in this section. A new sparsity-controlled PCA problem is proposed and approximately solved in Section 4. We includes all the experiment and comparison results in the final section.

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 prespecifed cardinality and present two algorithms for solving it.

2.1 Formulation Throughout this paper, we consider sparse PCA on 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 $s \leq p$, single-unit sparse PCA on V consists of finding an s-sparse PC of V, i.e., a direction $0 \neq x \in \mathbf{R}^p$ 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

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

where $\|x\|_0$ 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

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

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

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

is a restricted version of (2.3). Since its objective function 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 (2.4) is hard, we instead propose an algorithm to find an approximate solution of (2.4), which is then used to determine J.

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 (j_k, α_k) solves

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

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 $j \notin J_{k-1}$.

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 (2.2) with index set $J = J_s$ using the power-iteration method.

Our first algorithm, referred to as S_1-SPCA, is summarized in Algorithm 2. Its main difference from 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 all one vector, is usually completely 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 $k \leq \min\{n, p\}$ 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 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 $\sigma_1, \ldots, \sigma_k$ are the largest singular values of V. However, in the case where the z_i's are loading vectors for approximate 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 maximizing adjusted variance in a greedy manner, as the following two results show.

PROPOSITION 3.1. *Given $V_{(1)} = V \in \boldsymbol{R}^{n\times p}$ and a set of loading vectors $Z_i = [z_1, \ldots, z_i] \in \boldsymbol{R}^{p\times i}$, assuming that the PCs $Vz_1, \ldots, Vz_i$ are linearly independent, the $V_{(i+1)}$ generated by Schur complement deflation (3.12) satisfies*

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

where Q_iR_i is the reduced QR factorization of VZ_i.

Proof. When $i = 1$, we have $VZ_1 = Vz_1 = Q_1R_1$, where $Q_1 = Vz_1/\|Vz_1\|$ and $R_1 = \|Vz_1\|$. Therefore, by Schur complement deflation (3.12) with $i = 1$, we

have $V_{(2)} = (I - Q_1 Q_1^T)V$. Now assume (3.13) is true for $j-1$, i.e., $V_{(j)} = (I - Q_{j-1}Q_{j-1}^T)V$, where $VZ_{j-1} = Q_{j-1}R_{j-1}$, we prove (3.13) is also true for j.

Since Vz_j is not in the subspace spanned by $Vz_1, \ldots, Vz_{j-1}$ so that the vector $w_j := (I - Q_{j-1}Q_{j-1}^T)Vz_j \neq 0$. Defining $q_j = w_j/\|w_j\|$ and $\alpha = \|w_j\|$, we can easily see that the columns of $[Q_{j-1}, q_j]$ are orthonormal and $q_j = \frac{V_{(j)}z_j}{\|V_{(j)}z_j\|}$.

The reduced QR factorization VZ_j is therefore

$$Q_j R_j = [Q_{j-1}, q_j] \begin{bmatrix} R_{j-1} & Q_{j-1}^T V z_j \\ 0 & \alpha \end{bmatrix}.$$

According to Schur complement deflation (3.12),

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

which concludes our proof. □

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

(3.14)
$$AdjVar(V[Z_i, z]) - AdjVar(VZ_i) = \|(I - Q_i Q_i^T)Vz\|^2.$$

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 (3.14) 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 $q = w/\|w\|$ and $\alpha = \|w\|$, we can easily see that the columns of $[Q_i, q]$ are orthonormal and

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

is the reduced QR factorization of $V[Z_i, z]$. Hence, we conclude that the $AdjVar(V[Z_i, z]) = tr(R_i^2) + \alpha^2$, which is equivalent to (3.14). □

A greedy method for finding k sparse loading vectors with prespecified cardinalities $s_1, \ldots, s_k$ would proceed as follows. Assuming that $i < k$ sparse loadings vectors $z_1, \ldots, z_i$ 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

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

where $V_{(i+1)}$ is obtained by recursively applying the Schur complement deflation scheme. The latter problem can be solved using algorithms from Section 2. Notice that Schur complement deflation includes a Gram-Schmidt process on the matrix of PCs VZ so that a byproduct is the QR factorization of VZ and hence the adjusted variance can be updated sequentially.

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 hoe to assign prespecified 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

$$rAdjVar(VZ) = \frac{AdjVar(VZ)}{\sigma_1^2 + \cdots + \sigma_k^2}. \tag{4.15}$$

Based on this measure, we propose a problem which minimizes the cardinality of the loading vectors, with the constraint that the *relative adjusted variance* (4.15) is larger than or equal to a certain specified threshold $\rho \in (0, 1)$.

In the single PC case, the problem is formulated as

$$\min\{\|z\|_0 : \|Vz\|^2 \geq \rho\sigma_1^2, \|z\| \leq 1\}. \tag{4.16}$$

In the multiple PC case, the problem is formulated as

$$\begin{aligned} \min & \sum_{i=1}^{k} \|z_i\|_0 \\ s.t. \;& rAdjVar(V[z_1, \ldots, z_k]) \geq \rho \\ & \|z_i\| \leq 1, i = 1, \ldots, k. \end{aligned} \tag{4.17}$$

To solve problem (4.16) and (4.17), we use a variant of our algorithm S_c-SPCA, combined with the Schur complement deflation scheme (3.12). Basically, for each z_i, we start z_i from zero vector and add c indices to the active index set J in every iteration. Instead of stopping the algorithm based on the cardinality of the current loading vector z_i, we stop it based on the relative adjusted variance, i.e., when $rAdjVar(V[z_1, \ldots, z_i]) \geq \rho$. Notice that even though the relative adjusted variance (4.15), it is sequentially updated according to (3.14). The heuristic algorithm is summarized in Algorithm 2, which we refer to as SC_c-PCA.

Algorithm 2 Sparsity-Controlled PCA (SC_c-PCA)

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

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

5 Experiment results and comparison

5.1 Synthetic Data We first 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 orthogonalization via a Gram-Schmidt process. Next, we set the covariance matrix as $\Sigma = UDU^T$, where D is a diagonal matrix with several dominant components. Then, observations are sampled from a zero mean normal distribution with covariance matrix Σ.

In our experiments, points 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 of the covariance matrix are 10% sparse and associated with two dominant eigenvalues. The nonzero pattern of the first two eigenvectors u_1 and u_2 are pre-specified as follows:

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

To test our algorithms S_1-SPCA and S_5-SPCA, 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 problem sets. The first set of problem corresponds 200 data matrices $V \in \mathbf{R}^{50\times500}$ generated according to the method described above. The other set of problem consists of 200 data matrices $V \in \mathbf{R}^{200\times500}$ generated in the same way. We 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.

We use the state-of-art algorithm GPower methods $GPower_0$ and $GPower_{0,k}$[5] with L_0 penalization as counterparts since our algorithms use L_0 constraints as well. 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. The difference between $GPower_0$ and $GPower_{0,k}$ is that the former one finds single sparse PC at a time while the latter one finds k PCs simultaneously. In our experiments, we feed the cardinality 50 to S_1-SPCA and S_5-SPCA. For S_5-SPCA, number of identified index c in one iteration is 5, so that only 10 iterations are performed. For GPower methods, we use line search to tune the parameter with an initial guess proportional to the maximal column norm of the matrix V(see [5]). We stop the trials when the resulting vector has a cardinality between 45 and 55 or the maximal trial number 40 is reached. For S_1-SPCA, S_5-SPCA and $GPower_0$, we employ Schur complement deflation (3.12) before searching 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 trials and errors process. The results of the two problem sets are presented respectively in Table 1 and 2.

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 $V \in \mathbf{R}^{25\times500}, \mathbf{R}^{50\times500}$ or $\mathbf{R}^{100\times500}$ 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 the first PC. It is shown in Figure 5.1 that as we increase the cardinality of the loading vector z, the increasing rate of relative adjusted variance $\|Vz\|^2/\sigma_1^2$ changes drastically when the cardinality gets to 50.

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 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 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

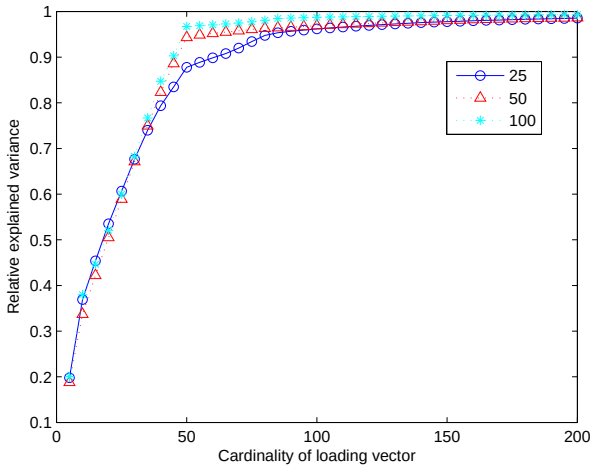

Figure 1: The variance-cardinality trade-off curves on synthetic data matrices, where number of observations n is 25, 50 and 100. Each curve is obtained from a single run of S_5-SPCA algorithm with 5 indices updated in each iteration.

5.2 Randomly Generated Data In this section, we evaluate the numerical quality and speed of both versions of our method S_1-SPCA and S_c-SPCA($c > 1$), by comparing them to GPower method [5] with L_0 penalty term, namely $GPower_0$, using a set of randomly generated sparse matrices. For the speed-up version S_c-SPCA, we set $c = \lceil s/10 \rceil$ as the number of added indices in each iteration, where s is the desired cardinality. Experiments are performed in MATLAB with codes of all three methods optimized for sparse matrix computation. All results are averaged over 10 repeated measurements.

In the first experiment, we have randomly generated 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 be $p/5$. For $GPower_0$, we set the parameter $\gamma = 0.002 \max_i \|v_i\|^2/n$, where v_i's are columns of the centered data matrix V. Then we measure the average cpu time for a single run of each algorithm. In Figure 5.2, the plot of the running time (in seconds) against matrix size indicates that S_1-SPCA and S_c-SPCA are fast on large sparse matrix.

Using the same set of matrices, we compare the numerical quality of three algorithms using the proportion of relative explained variance $\|Vz\|^2/\sigma_1^2$, where σ_1 is the largest singular value of V. Since this value is closely related to the cardinality of z, we set the required cardinality s to be $p/5$ for S_1-SPCA and S_c-SPCA, while we use line search for $GPower_0$ to obtain solutions with the same sparsity, i.e., 20% nonzero components. In the Figure 5.2, we plot the curve of relative explained

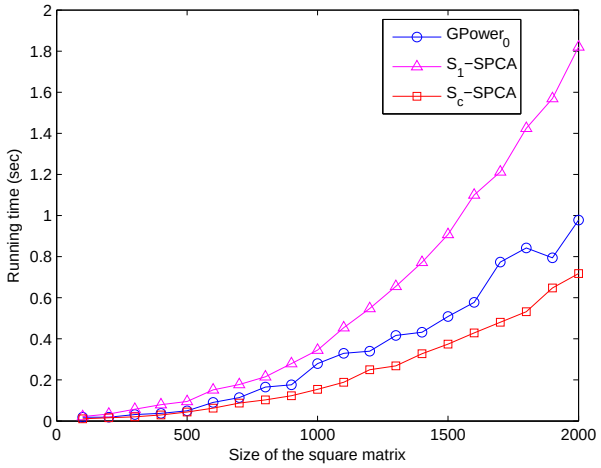

Figure 2: When the size of the matrix is increasing from 100 to 2000, this figure displays the curves of the time for a single run of all three methods. The cardinality of solution for S_1-SPCA and S_c-SPCA is fixed as $p/5$, while the parameter in $GPower_0$ is $\gamma = 0.002 \max_i \|v_i\|^2/n$.

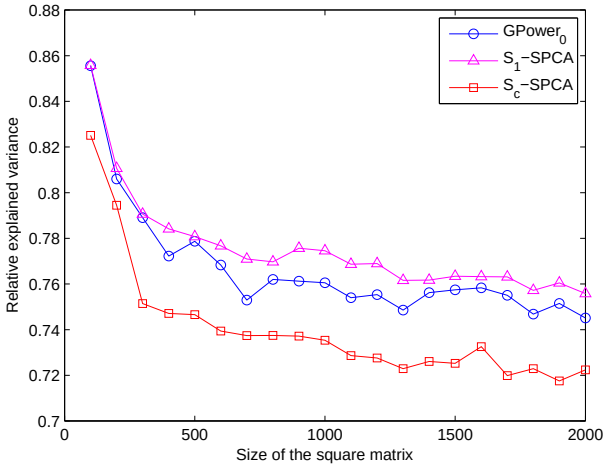

Figure 3: When the size of the matrix is increasing from 100 to 2000, this figure displays curves of the relative explained variance $\|Vz\|^2/\sigma_1^2$. $GPower_0$ uses line search to obtain z with cardinality $p/5$, which can be directly achieved by S_1-SPCA and S_c-SPCA.

variance against matrix size. Observe that while S_1-SPCA achieves better numerical quality compared to $GPower_0$, S_c-SPCA with $c > 1$ can be faster than the latter one at the expense of a little loss in solution quality. Another important observation is that the PC with 20% sparsity can explain a large portion of variance explained by the dense PC.

In the second experiment, the size of the square matrix is fixed as 5000. We input the cardinality of the solution z computed by $GPower_0$ to both versions of our method, so that we can compare their solution quality based on relative explained variance $\|Vz\|^2/\sigma_1^2$. To obtain z with different cardinality, we choose 20 parameters $\gamma = 0.01 \max_i \|v_i\|^2/n/\sqrt{j}, j = 1, \ldots, 20$ for $GPower_0$. The trade-off curve of the relative explained variance against the cardinality of the solution is displayed in the first graph in Figure 5.2. The second graph plots running time against the cardinality. Observe that S_1-SPCA method outperforms $GPower_0$ in terms of solution quality but the running time is proportional to the cardinality of solution. The running time of our speed-up algorithm S_c-SPCA barely increases as the cardinality increasing, at the expense of an acceptable sacrifice in solution quality.

Our third experiment consists of two parts. In the first (resp., second) one, we have randomly generated sparse 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 parameter $\gamma = 0.01 \max_i \|v_i\|^2/n$ and $\gamma = 0.00005 \max_i \|v_i\|^2/n$ respectively to obtain solutions with similar sparsity. The corresponding graphs of the running time against the size of the larger dimension are given in Figure 5.2. Observe that while the speed of S_c-SPCA method is comparable to $GPower_0$ when $n/p = .1$, it is faster than $GPower_0$ when $n/p = 10$.

5.3 Image data In this subsection, we compare our method with GPower method using real-world data matrix from handwritten digits database MNIST [6]. The matrix we use has size 5000 by 784. Each row of the matrix corresponds to a image with 28 by 28 pixels, and hence of size 784. To obtain PCs with different sparsity, we choose 20 parameters $\gamma = 0.00002 \max_i \|v_i\|^2/n/j, j = 1, \ldots, 20$ for $GPower_0$ and then directly control the cardinality constraint in S_5-SPCA. 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 $GPower_0$.

Using SC_c-PCA algorithm, we minimize the total cardinality of six PCs $z_1, \ldots, z_6$ to explain at least 70% of the variance explained by dense PCs(i.e., $rAdjVar(VZ) \leq 0.7$). For comparison, we implement

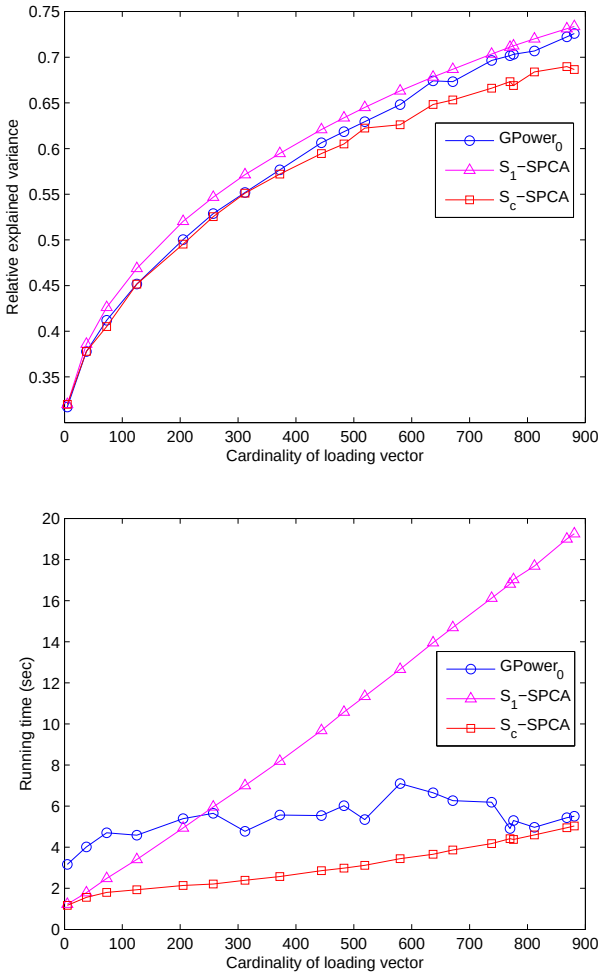

Figure 4: We fix the data matrix as a square sparse matrix of size 5000. The trade-off curve of variance against cardinality is on the top. The curve of running time against cardinality is on the bottom. In this experiment, we input the cardinality of the solution z computed by $GPower_0$ to both versions of our method so that solutions of all three methods have exactly the same cardinality.

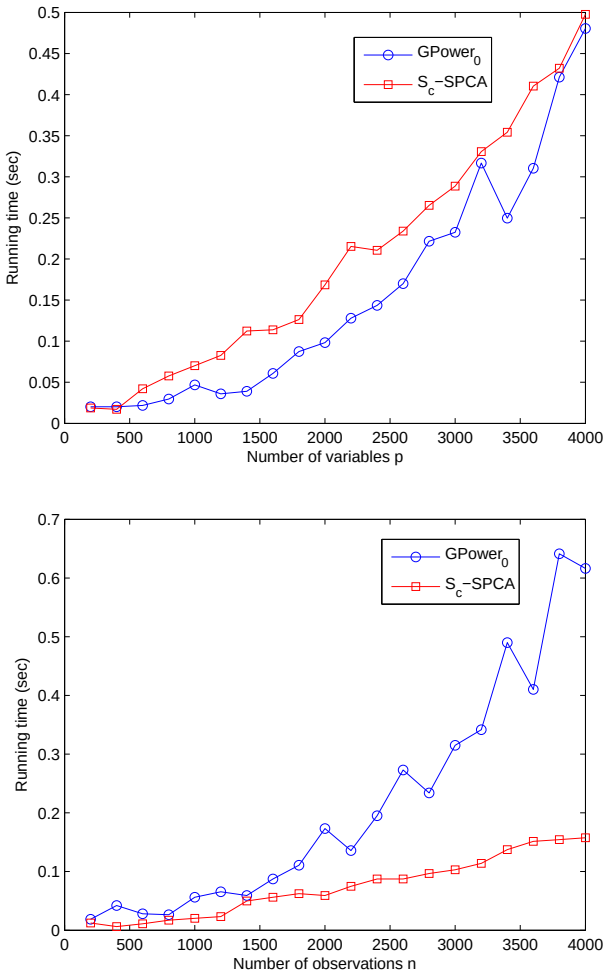

Figure 5: As the number of variables p increases from 200 to 4000 and $n/p = 0.1$, the running time curve is shown in the top graph. As the number of observations n increases from 200 to 4000 and $n/p = 10$, the running time curve is shown in the bottom graph.

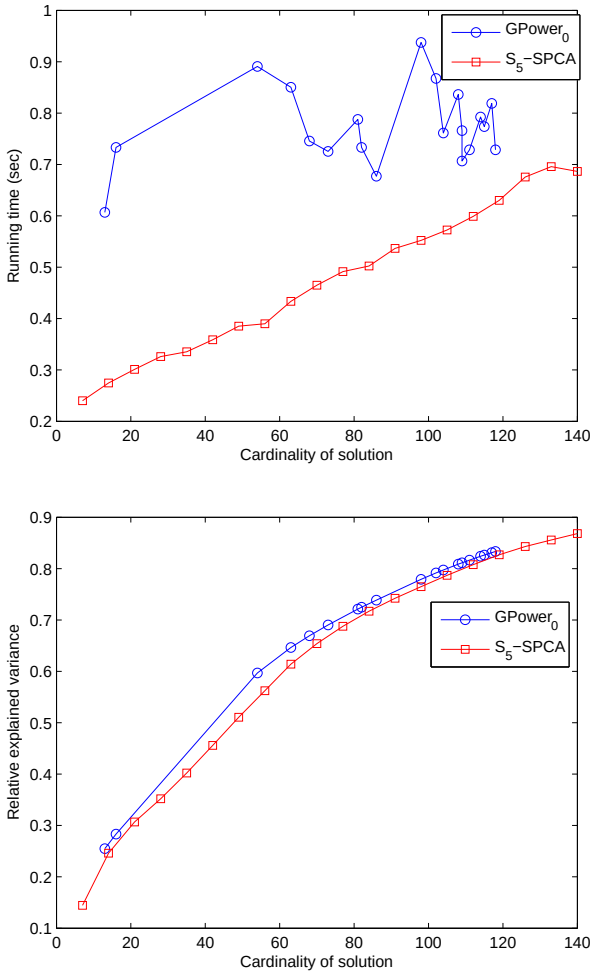

Figure 6: Experiments on 5000 handwritten digits images from MNIST database. The top one plots running time against cardinality curves while the bottom one plots variance against cardinality curves.

the $GPower_0$ with line search for parameter under the framework of sparsity-controlled PCA . It turns out that with a random initial parameter, it takes dozens of runs for $GPower_0$ to obtain sparse PCs with desired relative adjusted variance. We also include the result by block version algorithm $GPower_{0,6}$ with line search technique choosing parameter. It is shown that $GPower_{0,2}$ fails to find such $z_1, \ldots, z_6$ with relative adjusted variance closely above 0.7 in finite time. In Table 5.3, we show the cardinality of each sparse PC, the total running time and the actual value of relative adjusted variance.

5.4 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. The first 6 PCs can explain 86.9% of the total variance, but are dense and hard to explain. Previous studies tried to find 6 sparse PCs based on the 13×13 sample covariance matrix. However, the way they found the sparse PCs was to try different parameters or cardinalities for each sparse PC since it is not clear how to distribute the total cardinality among PCs. Our method is the first one to address this problem. To avoid the trials and errors process, we formulate the problem as sparsity-controlled PCA (4.17) to find 6 sparse as possible PCs with relative adjusted variance at least 0.9. It turns out that in just a single run we can find a set of 6 sparse PCs with cardinality pattern $7-4-5-2-5-2$ explaining 90.69% of the variance explained by dense PCs. The resulting loading vectors are shown in the Table 5.4. To achieve the same goal, we use $GPower_{0,6}$ method with balancing parameter $N = (1, 1/2, 1/3, 1/4, 1/5, 1/6)$ and random initial parameter γ and line search technique. 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 $12-11-11-2-1-1$ which explaining 90.12% of the variance explained by dense PCs.

5.5 Sparsity-controlled PCA on Leukemia data

The Leukemia data set is a DNA microarray dataset 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 dense PCs(i.e., $rAdjVar(Vz_1, Vz_2) = 0.7$). In our experiment, we try $c = 10$ and $c = \lceil 20\%p/10 \rceil$ since it is reasonable to expect 20%-sparse PC can explain 70% of the total variance. For comparison, we implement the $GPower_0$ with line search for parameter under the framework

Table 3: Sparsity-controlled PCA on digits image data. Four algorithms are performed to compute the first six sparse loading vectors with relative adjusted variance at least 0.7. $GPower_{0,6}$ fails to find such PCs so we present the best available result.

	$\|z_1\|_0$	$\|z_2\|_0$	$\|z_3\|_0$	$\|z_4\|_0$	$\|z_5\|_0$	$\|z_6\|_0$	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 4: Sparsity-controlled PCA on pitprops data with relative adjusted variance at least 0.9.

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

of sparsity-controlled PCA . It turns out that with a random initial 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 by block version algorithm $GPower_{0,2}$ with parameter search technique. It is shown that $GPower_{0,2}$ fails to find z_1 and z_2, with relative adjusted variance closely above 0.7. In Table 5.5, we show the cardinalities of the sparse PCs, the cosine of the angle between them, the total running time and the actual value of relative adjusted variance using SC_{10}-PCA, SC_c-PCA and $GPower_0$. For $GPower_{0,2}$, we show the best available result.

5.6 Sparsity-controlled PCA on document data

By implementing sparsity-controlled PCA on huge document data from [3], we give a flavor of sparse PCs as combinations of words. The document data set we use is the NIPS full papers data set, 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. By SC_{10}-PCA with objective proportion $\rho = 0.9$, we find 6 sparse PCs with cardinality $(20, 140, 70, 110, 170, 50)$ in only 11.8 seconds. The actual relative adjusted variance is 0.9006. The top ten weighted words in each sparse PCs are given in the Table 5.6. In contrast, it takes 220 seconds on average 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 line search technique for another parameter γ, the loading vectors found by $GPower_{0,k}$ have cardinality $(3679, 50, 120, 43, 89, 42)$, with $rAdjVar = 0.9175$.

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. JournŽe, Y. Nesterov, P. Richtarik, and R. Sepulchre. Generalized power method for sparse principal component analysis. *CORE Discussion Papers*, 2008.

Table 5: Sparsity-controlled PCA on Leukemia data matrix. Four algorithms are performed to compute the first two sparse loading vectors explaining 70% relative adjusted variance. For S_c-SPCA, $c = 20\%p/10$.

	$\|z_1\|_0$	$\|z_2\|_0$	$\|z_1\|_0 + \|z_2\|_0$	$\|z_1^T z_2\|$	# trials	total time(seconds)	$rAdjVar(Vz_1, Vz_2)$
SC_{10}-PCA	1930	1650	3580	0.0290	1	54.4	0.7003
SC_c-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

Table 6: Sparsity-controlled PCA on NIPS full paper data with relative adjusted variance at least 0.9. The cardinality and ten most weighted words in each sparse PC are given.

	z_1	z_2	z_3	z_4	z_5	z_6
Cardinality	20	140	70	110	170	50
Top ten weighted words	'compactly'	'sodium'	'wafer'	'sodium'	'ology'	'tdnns'
	'ology'	'original'	'iiii'	'wafer'	'tdnns'	'wechsler'
	'probabil'	'embody'	'compacity'	'ology'	'compactly'	'ology'
	'wafer'	'estimators'	'sodium'	'iiii'	'nique'	'probabil'
	'nique'	'tdnns'	'tdnns'	'bianchi'	'embody'	'sodium'
	'nonvanishing'	'nonvanishing'	'iiii'	'normalizsing'	'probabil'	'nonvanishing'
	'normalising'	'bianchi'	'reformulation'	'electric'	'normalising'	'embody'
	'synap'	'exclusive'	'original'	'result'	'ultimate'	'wafer'
	'ultimate'	'normalising'	'bianchi'	'probabil'	'bianchi'	'compactly'
	'exclusive'	'probabil'	'nique'	'embody'	'abstractions'	'electric'

[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.