An efficient algorithm for sparse PCA

Yunlong He
Georgia Institute of Technology
School of Mathematics
heyunlong@gatech.edu

Renato D.C. Monteiro
Georgia Institute of Technology
School of Industrial & System Engineering
renato.monteiro@isye.gatech.edu

Haesun Park
Georgia Institute of Technology
School of Computational Science and Engineering
hpark@cc.gatech.edu

Abstract

Sparse principal component analysis (PCA) imposes extra constraints or penalty terms to the standard PCA to achieve sparsity. In this paper, we introduce an efficient algorithm for finding a single sparse principal component (PC) with specified cardinality. 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. Experiments on 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.

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. 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. [9] formulate sparse PCA as a regression-type optimization problem which is then solved by Lasso-type algorithms. Shen and Huang [8]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. [7] 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 [4] 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. A different multiple sparse PCA approach is proposed in [6] based on a formulation enforcing near orthogonality of the PCs, which is then solved by an augmented Lagrangian approach. Throughout this paper, we mostly compare our approach with the GPower method proposed in [4], 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 principal component. 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. In this paper, we concentrate on the problem of computing a single sparse PC. Though our algorithm, combined with Schur complement deflation, can sequentially find multiple sparse PCs while greedily maximizes *the adjusted variance* [9] explained by the sparse PCs, this extension is not presented in this paper due to space limit.

2 Algorithms for single sparse PCA

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

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

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 (1) satisfying $x_j = 0$ for all $j \notin J$, i.e., it solves the problem

$$\max\{\|Vx\|/\|x\| : x_j = 0, \forall j \notin J\}. \tag{2}$$

We note that once J is determined, x can be efficiently computed by using the power-iteration method (see for example [3]). 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

$$\max\{\|Vx\|^2 : \|x\|_0 \leq s,\ \|x\| \leq \sqrt{s}\}. \tag{3}$$

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

$$\max\{\|Vx\|^2 : \|x\|_0 \leq s,\ \|x\|_\infty \leq 1\} \tag{4}$$

is a restricted version of (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 (4). However, since computing (4) is hard, we instead propose an algorithm to find an approximate solution of (4), which is then used to determine J.

Our method to find an approximate solution for (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

Algorithm 1 S_1-SPCA

Given a centered data matrix $V \in \mathbf{R}^{n\times p}$ (or, sample covariance matrix $\Sigma = V^TV \in \mathbf{R}^{p\times p}$) 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 $k = 1, \dots, s$ 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 $\alpha_k = \text{sign}(v_{j_k}^T V x^{(k-1)})$.
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.
```

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

$$(j_k, \alpha_k) = \underset{j \notin J_{k-1},\, \alpha = \pm 1}{\arg\max} \|V(x^{(k-1)} + \alpha e_j)\|^2. \tag{5}$$

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 $\alpha = \pm 1$,

$$\|V(x^{(k-1)} + \alpha e_j)\|^2 = \|Vx^{(k-1)}\|^2 + \|v_j\|^2 + 2\alpha v_j^T V x^{(k-1)}, \tag{6}$$

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

$$j_k = \underset{j \notin J_{k-1}}{\arg\max} \|v_j\|^2 + 2|v_j^T V x^{(k-1)}|, \alpha_k = \text{sign}(v_{j_k}^T V x^{(k-1)}). \tag{7}$$

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:

$$v_j^T V x^{(k-1)} = v_j^T V(x^{(k-2)} + \alpha_{k-1} e_{j_{k-1}}) = v_j^T V x^{(k-2)} + \alpha_{k-1} v_j^T v_{j_{k-1}}. \tag{8}$$

There are two cases to discuss at this point. If V^TV 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^TV 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 our second algorithm 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 [3]. Moreover, our computational experiments show that the first phase is generally by far the more expensive one. When V^TV 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^TV 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 (7), say $j_{k,1}, \dots, j_{k,c}$, 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).

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 (8) in our algorithms, which asks for the computation of j_{k-1}-th column of the sample covariance matrix $V^T V$. 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 $V^T V$ based on the observation that

$$V^T V = W^T W - n\mu\mu^T, \tag{9}$$

where $\mu = W^T e/n$ 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 Experiment results and comparison

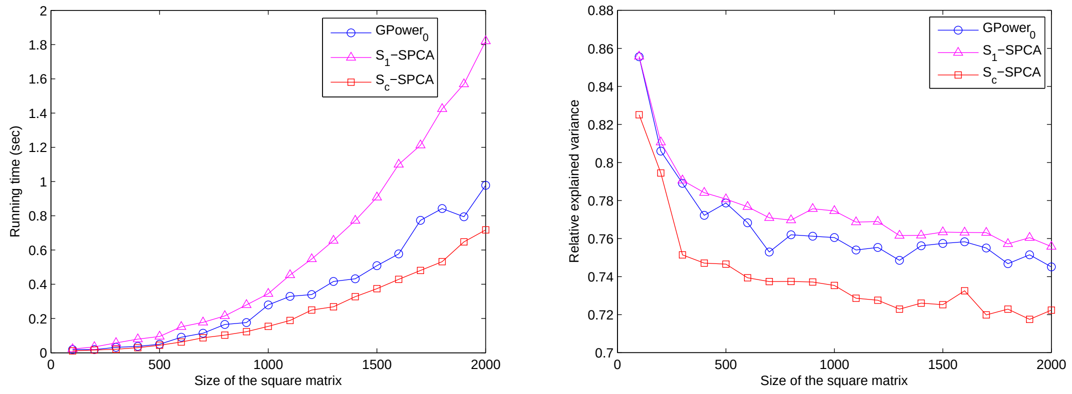
3.1 Randomly Generated Data

In this section, we evaluate the numerical quality and speed of both versions of our method S_1-SPCA and S_c-SPCA($c > 1$) using a set of randomly generated sparse matrices. Since our algorithms use L_0 constraint, we choose $GPower_0$ as the counterpart, which is the the L_0 penalized version of the state-of-art sparse PCA algorithm GPower method [4]. Another reason is that the experiments in [4] show that the L_0 version of GPower is generally more efficient than the L_1 version. 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.

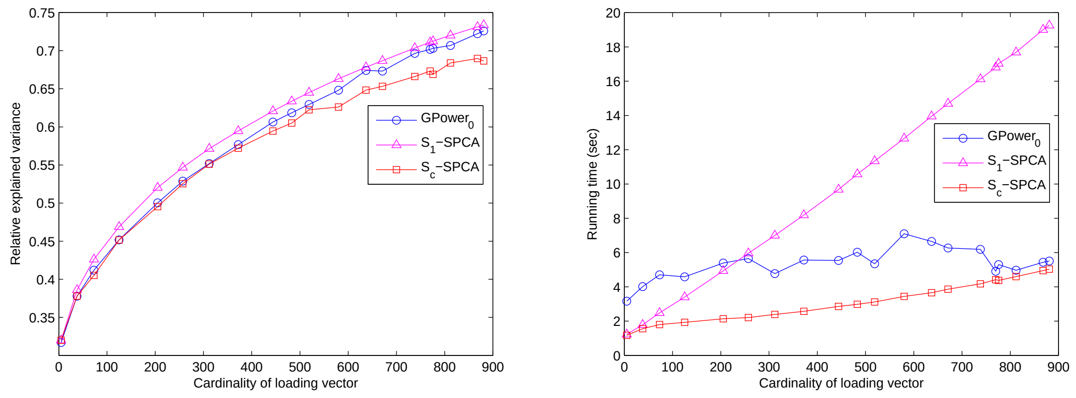
In the first experiment, we randomly generate sparse square matrices W with dimension p varying from 100 to 2000, with their sparsity (i.e., proportion of nonzero entries) set to 20%. For S_1-SPCA and S_c-SPCA, we set the required cardinality s to be $p/5$. For $GPower_0$, we set the parameter $\gamma = 0.002 \max_i \|v_i\|^2/p$, 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 1(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 matrix. Notice here we directly feed $GPower_0$ the parameter γ rather than using line search, and we compare the time for just a single run of different methods. But in practice, it usually takes several times longer for GPower method to obtain a certain level of sparsity.

Using the same set of matrices, we compare the numerical quality of three algorithms using the proportion of 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 right graph of Figure 1(a), we plot the curve of explained 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 $GPower_0$ at the expense of a little loss in solution quality.

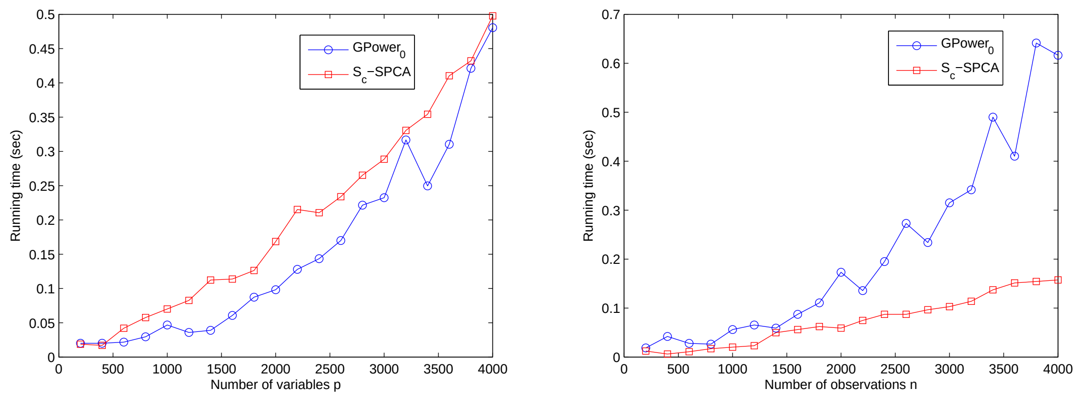
In the second experiment, the size of the square matrix p is fixed as 5000. We input the cardinality of the solution z computed by $GPower_0$ to both versions of our method, so that we can compare their solution quality based on explained variance $\|Vz\|^2/\sigma_1^2$. To obtain z with different cardinality, we choose 20 parameters $\gamma = 0.01 \max_i \|v_i\|^2/p/\sqrt{j}, j = 1, \ldots, 20$ for $GPower_0$. The trade-off curve of the explained variance against the cardinality of the solution is displayed in the first graph in Figure 1(b). The second graph plots running time against the cardinality, where again we only compare the time for a single run. 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



(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 line search to obtain z with 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 1: Experiments on randomly generated matrix.

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 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 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 plots of the running time against the size of the larger dimension are given in Figure 1 (c). While the speed of S_c-SPCA method is comparable to $GPower_0$ when $n/p = .1$, it is faster than $GPower_0$ when $n/p = 10$.

3.2 Image data

In this subsection, we compare our method with GPower method using real-world data matrix from handwritten digits database MNIST [5]. 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 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 $GPower_0$.

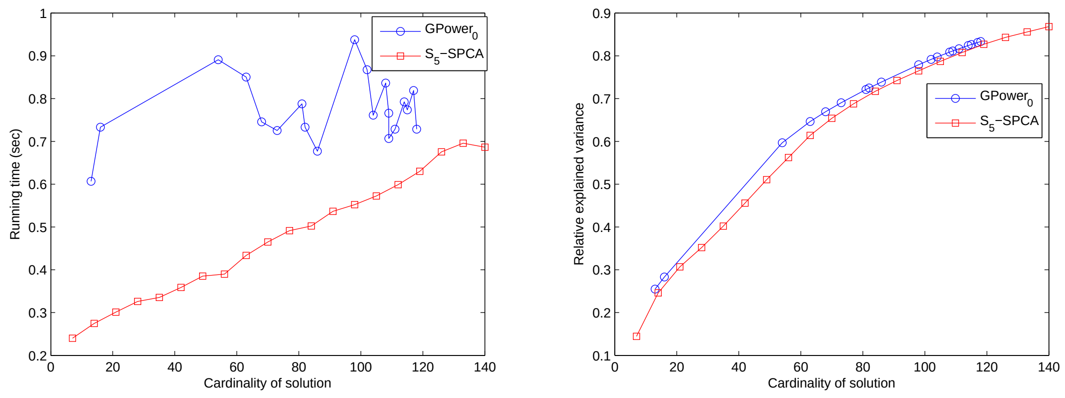


Figure 2: Experiments on 5000 handwritten digits images. We show the plot of running time against cardinality on the left, and the plot of variance against cardinality on the right.

3.3 Biology data sets

In this subsection we compare the speed of S_1-SPCA with another greedy method proposed in [2] –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 problem size. Both of these two methods can be either directly applied on the covariance matrix or applied on the Cholesky factorization of the covariance matrix. For $GPower_0$ and $GPower_1$, we use the Cholesky factorization of the covariance matrix as input. We fix the parameter $\gamma = 0.1$ for $GPower_0$ and $GPower_1$, with no concern of the sparsity level. The accumulated running time (in seconds) for 10 runs is shown in the table below.

Data set	S_1-SPCA-cov	S_1-SPCA-data	PathSPCA-cov	PathSPCA-data	$GPower_0$	$GPower_1$
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

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

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

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

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

[7] 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.

[8] 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.

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