An efficient algorithm for rank-1 sparse PCA

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

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

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

Abstract

Sparse principal component analysis (PCA) imposes extra constraints or penalty terms to the original PCA to achieve sparsity. In this paper, we introduce an efficient algorithm to find a single sparse principal component with a specified cardinality. The algorithm consists of two stages. In the first stage, it identifies an active index set with desired cardinality corresponding to the nonzero entries of the principal component. In the second one, it finds the best direction with respect to the active index set, using the power iteration method. Experiments on both 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 other methods.

1 Introduction

Principal Component Analysis (PCA) is a classical tool for performing data analysis such as dimensionality reduction, data modeling, feature extraction and further learning tasks. It can be widely used in all kinds of data analysis areas like image processing, gene microarray analysis and document mining. Basically, PCA consists of finding a few orthogonal directions in the data space which preserve the most information from the data cloud. This is done by maximizing the variance of the projections of the data points along these directions. However, standard PCA generally produces dense directions (i.e., whose components are most nonzero), and hence are too complex to explain the set of data points. 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 cheaply stored; and ii) it captures the simple and inherent structure associated with the (in principal, complex) 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 a Lasso-based algorithm. D'Aspremont et al.'s DSPCA algorithm [1] to perform rank-1 sparse PCA consists of solving a semi-definite relaxation of a certain formulation of rank-1 sparse PCA whose solution is

*The work of this author was partially supported by NSF Grants CCF-0808863 and CMMI-0900094 and ONR Grant ONR N00014-08-1-0033.

†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.

then post-processed to yield a sparse principal component. 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. Moghaddam et al. [8] perform sparse PCA by using a combinatorial greedy method called ESPCA obtaining good results, 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 principal component) and then using an algorithm such as power-iteration to obtain the final sparse principal component. Journée et al [5] recently formulate multiple 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. Finally, [7] proposes a different multiple sparse PCA approach based on a formulation enforcing near orthogonality of the PCs, which is then solved by an augmented Lagrangian approach.

In this paper, we propose a simple but efficient heuristic for finding a single sparse principal component. We then compare our approach with the one proposed in [5], namely GPower, which is widely viewed as one of the most efficient methods for performing sparse PCA. 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. Finally, a clear advantage of our method is that it can easily produce a single sparse principal component of a specified cardinality with just a single run while the GPower method in [5] may require several runs due to the fact it is based on a formulation which is not directly related to the given cardinality.

2 Rank-1 Sparse PCA

In this section, we define the rank-1 sparse PCA problem and present algorithms for approximately solving it.

2.1 Formulation

Given a data matrix $V \in \mathbf{R}^{n \times p}$ whose rows represent data points in $\mathbf{R}^p$ and a positive integer s, performing rank-1 sparse PCA on V consists of finding an s-sparse principal component of V, i.e., a direction $0 \neq x \in \mathbf{R}^p$ 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 $\max\{\|Vx\|^2/\|x\|^2 : \|x\|_0 \leq s\}$, where $\|x\|_0$ denotes the number of nonzero components of x. To eliminate redundancy, we can alternatively consider only the optimal directions of size $\sqrt{s}$, i.e., directions x which solve

$$\max\{\|Vx\|^2 : \|x\|_0 \leq s,\ \|x\|_2 \leq \sqrt{s}\}. \tag{1}$$

2.2 A Heuristic Algorithm

We will now give 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

$$\max\{\|Vx\| : \|x\|_2 \leq \sqrt{s}, x_j = 0, \forall j \notin J\}. \tag{2}$$

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

To describe the procedure to determine J, we make the following observations. First, note that under the condition that $\|x\|_0 \leq s$, the inequality $\|x\|_\infty \leq 1$ implies that $\|x\|_2 \leq \sqrt{s}$. Hence, the problem

$$\max\{\|Vx\|^2 : \|x\|_0 \leq s,\ \|x\|_\infty \leq 1\} \tag{3}$$

is a restricted version of (1). 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 components which are either 1 or -1. Ideally, we would like to choose J as the set of nonzero components of an optimal extreme point of (3). However, since solving (3) exactly is hard,

we instead propose a heuristic to find an approximate feasible solution of (3), which is then used to determine J in an obvious manner.

Our heuristic to approximately solve (3) 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 components which are all either 1 or -1. Also, let J_{k-1} denote the index set corresponding to the nonzero components 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

$$(j_k, \alpha_k) = \underset{j \notin J_{k-1},\, \alpha = \pm 1}{\arg\max} \|V(x^{(k-1)} + \alpha e_j)\|^2. \tag{4}$$

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

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

$$\|V(x^{(k-1)} + \alpha e_j)\|^2 = \|Vx^{(k-1)}\|^2 + \|v_j\|^2 + 2\alpha v_j^T V x^{(k-1)}, \tag{5}$$

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

$$j_k = \underset{j \notin J_{k-1}}{\arg\max} \|v_j\|^2 + 2|v_j^T V x^{(k-1)}|, \quad \alpha_k = \text{sign}(v_{j_k}^T V x^{(k-1)}). \tag{6}$$

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:

$$v_j^T V x^{(k-1)} = v_j^T V(x^{(k-2)} + \alpha_{k-1} e_{j_{k-1}}) = v_j^T V x^{(k-2)} + \alpha_{k-1} v_j^T v_{j_{k-1}}. \tag{7}$$

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. Otherwise, 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 Scol-SPA, is summarized below. 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.

Algorithm 1: Scol-PCA

Input: Centered data matrix V (or, sample covariance matrix $\Sigma = V^T V$) and desired sparsity s.
Initialization: Set $x^{(0)} = 0$, $J = \emptyset$.
Iteration: For $k = 1, \ldots, s$, do:
Update $j_k = \arg\max_{j \notin J_{k-1}} \|v_j\|^2 + 2|v_j^T V x^{(k-1)}|$ and $\alpha_k = \text{sign}(v_{j_k}^T V x^{(k-1)})$.
Set $x^{(k)} = x^{(k-1)} + \alpha_k e_{j_k}$ and add j_k to J
Postprocessing: Use the power-iteration method to solve (2) and output is optimal solution.

2.3 Complexity and Speed-up Strategy

We now briefly discuss the arithmetic complexity of the first phase of Algorithm 1 disregarding the complexity of its second phase where the power-iteration method is applied. The reason is that the latter method generally depends on measures other than the dimension of the underlying matrix. 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 arithmetic complexity of the first phase of Algorithm 1 is $\mathcal{O}(ps)$. Otherwise, when $V^T V$ is not explicity given, then this complexity becomes $\mathcal{O}(nps)$ in the dense case, and considerably smaller than $\mathcal{O}(s.nz + ps)$ in the sparse case, where 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, thereby reducing the overall arithmetic 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 (6), 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

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

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 arithmetic complexity of each loop is $\mathcal{O}(pn)$, thereby implying the aforementioned arithmetic complexity for the first phase of the new variant. We will refer to this variant as the Mcol-PCA method. It is considerably faster than the single column version described earlier at the expense of a small sacrifice in the quality of its solution (i.e., its variance). In our computational experiments, we set $c = s/10$ so that the Mcol-PCA method performs at most 10 loops.

In many applications, one has $V = (I - \frac{ee^T}{n})W$, where W is the uncentered data matrix while V is the centered data matrix. Moreover, the matrix W is generally sparse while the matrix V is dense. It is easy to see that our method can be implemented only in terms of W, without ever having to form V, thereby taking advantage of any available sparsity of the uncentered data.

3 Experiment result and comparison

3.1 Randomly Generated Data

In this subsection, we evaluate the quality and speed of both versions of our method by comparing them to GPower method [5] with L_0 penalty term, namely GPower0, using a set of randomly generated sparse matrices. All experiments are performed in MATLAB.

In our first experiment, we have randomly generated sparse square matrices W with dimension p varying from 200 to 4000, with their sparsity (i.e., proportion of nonzero entries) set to 20%. We also set the required cardinality s to $p/5$. In Figure 1, the first graph plots the running time against matrix size for all three methods and the second graph plots the matrix size against the solution variance for the two versions of our method. Observe that while the speed of Scol-PCA is comparable with GPower, Mcol-PCA can be much faster than the latter one at the expense of a little loss in solution quality.

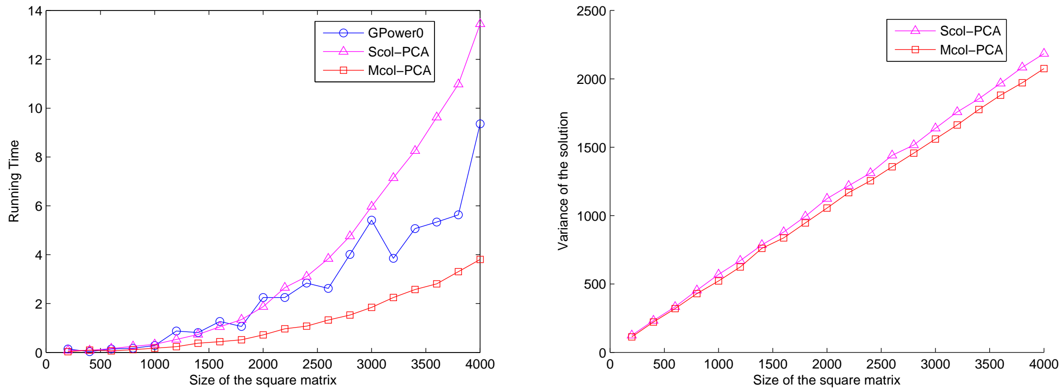

Figure 1: When the size of the matrix is increasing from 200 to 4000, the first graph displays the curves of the time for a single run of all three methods, and the second graph displays curves of the solution variance for Scol-PCA and Mcol-PCA. The cardinality of solution for Scol-PCA and Mcol-PCA is fixed as $p/5$, while the parameter in GPower method is chosen coarsely to get a close cardinality.

Our second experiment consists of two parts. In the first (resp., second) one, we have randomly generated sparse matrices with $n/p = .1$ (resp., $n/p = 10$), with the sparsity set to 20% and with their larger dimension increasing from 200 to 4000. The corresponding graphs of the running time

against the size of the larger dimension are given in Figure 2. Observe that while the speed of Mcol-PCA method is comparable to GPower when $n/p = .1$, it is faster than GPower when $n/p = 10$.

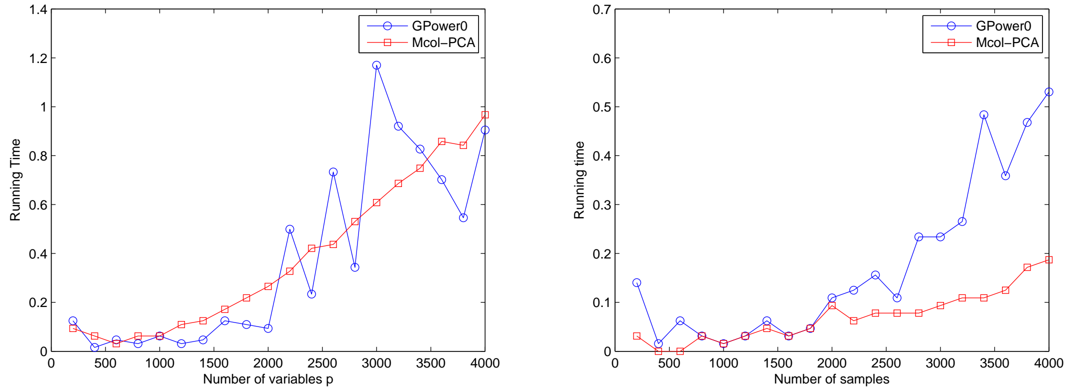

Figure 2: As the number of variables increases from 200 to 4000 and $n/p = 0.1$, the running time curve is shown in the left graph. As the number of samples increases from 200 to 4000 and $n/p = 10$, the running time curve is shown in the right graph.

In the third experiment, we have generated a set of square data matrices of size 5000, with their sparsity varying from 1% to 20%. The cardinality of the solution is still set as 20% of the matrix size. The plot of running time against sparsity of the matrix is displayed in Figure 3. It turns out that Mcol-PCA algorithm become considerably faster than GPower method as the data matrix becomes sparser.

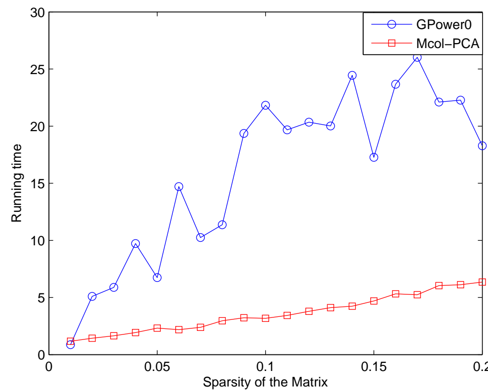

Figure 3: Plot of running time against sparsity of the matrix, ranging from 1% to 20%.

In the fourth experiment, we have input the cardinality of the solution by GPower to both versions of our method, so that we can compare their solution quality. The size of the square matrix is fixed as 5000, while the cardinality of the solution grows from 1 to 900. The trade-off curve of the variance against the cardinality of the solution is displayed in the first graph in Figure 4. The second graph plots running time against the cardinality. Observe that Scol-PCA method outperforms GPower method in terms of solution quality and speed. The running time of our speed-up algorithm Mcol-PCA barely increases as the cardinality increasing, at the expense of an acceptable sacrifice in solution quality.

3.2 Image data and document data

In this subsection, we compare our Mcol-PCA method with GPower method using two kinds of real-world data matrices. The first matrix comes from handwritten digits database MNIST [6]. The

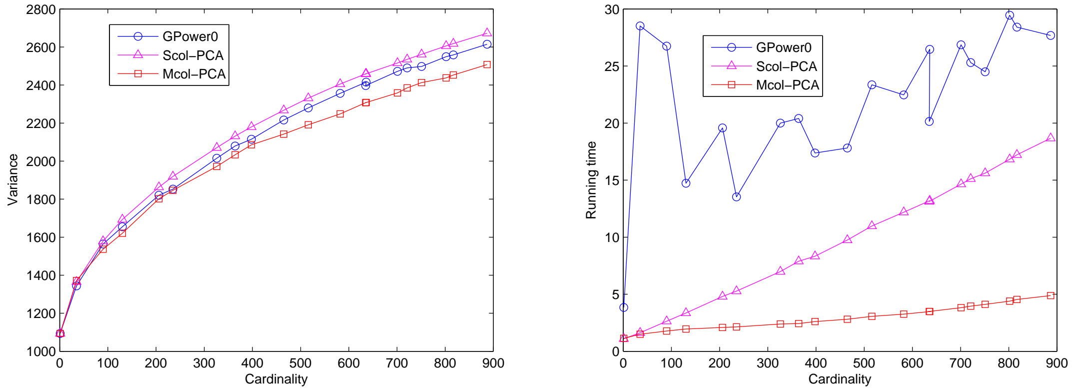

Figure 4: The trade-off curve of variance against cardinality is on the left. The curve of running time against cardinality is on the right. In this experiment, the cardinality of all three methods is set exactly the same.

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. In Figure 5, the first graph plots running time against the sparsity of the solution, while the second graph plots the variance of the solution against its sparsity. Observe that on this data set, Mcol-PCA method outperforms GPower method not only in terms of speed but also solution quality.

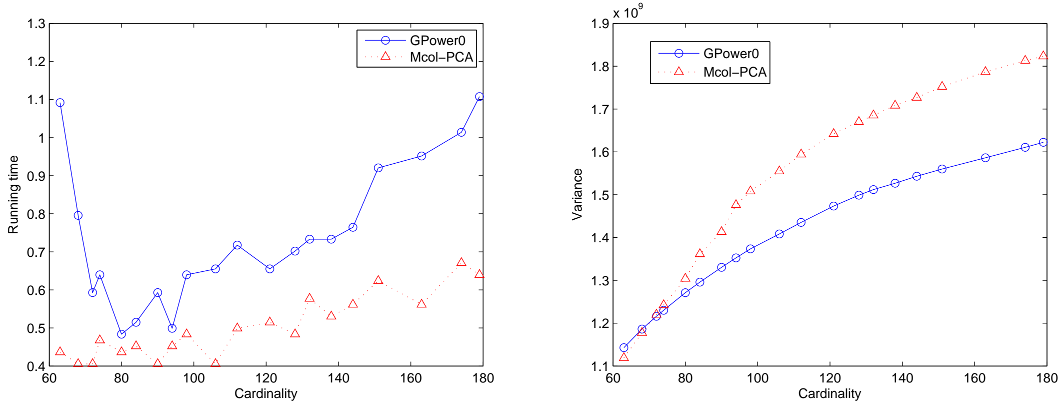

Figure 5: Experiments on 5000 handwritten digits images from MNIST database. The left one plots running time against cardinality curves while the right one plots variance against cardinality curves.

By experiments on huge document data from [3], we can show our algorithm is efficient in terms of both speed and storage. The first document data set we use is the NIPS full papers data set, with 1500 documents and 12419 words forming a sparse matrix of size 1500 by 12419. In Figure 6, the fist graph plotting running time against sparsity shows that Mcol-PCA is more efficient than GPower in terms of speed when the desired sparsity is less than 500. The second graph shows that the solution variances of Mcol-PCA is comparable to that of GPower although a little smaller.

The second document data set is the Enron Emails data set with 39861 documents and 28102 words. When we try to center the 39861 by 28102 sparse matrix W, MATLAB returns out of memory. However, using our algorithm there is no need to install the centered data explicitly, as we can compute the values in $W^TW - n(\mu_1, \dots \mu_p)^T(\mu_1, \dots \mu_p)$ instead of V^TV.

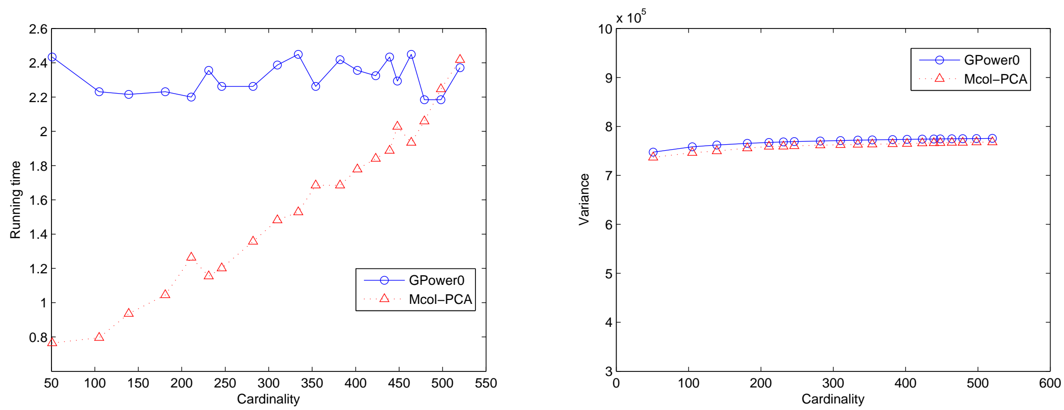

Figure 6: Experiments on NIPS document-word data.The left one plots running time against cardinality curves while the right one plots variance against cardinality curves.

4 Further discussion

The main contribution of this paper is that we propose a simple but very efficient algorithm for performing rank-1 sparse PCA. Our method allows users to set the cardinality of the solution explicitly, which is important in visualization systems. To find subsequent sparse principal components, we can use the classical deflation scheme described in [1].

The two versions of our method, namely Scol-PCA and Mcol-PCA, can be easily applied to perform sparse PCA on either the data matrix or the sample covariance matrix, as the key step in our algorithm is to update using values in the covariance matrix (see identity (7)). Due to this property, the method has potential applications to other sparse learning algorithms where covariance matrix, kernel matrix or general distance matrix is involved, which we will study in the future work.

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.

[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] 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.

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