Fast algorithms for sparse principal component analysis based on Rayleigh quotient iteration

Volodymyr Kuleshov KULESHOV@STANFORD.EDU
Department of Computer Science, Stanford University, Stanford, CA

Abstract

We introduce new algorithms for sparse principal component analysis (sPCA), a variation of PCA which aims to represent data in a sparse low-dimensional basis. Our algorithms possess a cubic rate of convergence and can compute principal components with k non-zero elements at a cost of $O(nk + k^3)$ flops per iteration. We observe in numerical experiments that these components are of equal or greater quality than ones obtained from current state-of-the-art techniques, but require between one and two orders of magnitude fewer flops to be computed. Conceptually, our approach generalizes the Rayleigh quotient iteration algorithm for computing eigenvectors, and can be viewed as a second-order optimization method. We demonstrate the applicability of our algorithms on several datasets, including the STL-10 machine vision dataset and gene expression data.

1. Introduction

A fundamental problem in statistics is to find simpler, low-dimensional representations for data. Such representations can help uncover previously unobserved patterns and often improve the performance of machine learning algorithms.

A basic technique for finding low-dimensional representations is principal component analysis (PCA). PCA produces a new K-dimensional coordinate system along which data exhibits the most variability. Although this technique can significantly reduce the dimensionality of the data, the new coordinate system it introduces is not easily interpretable. In other words, whereas initially, each coordinate x_i of a data point $x \in \mathbb{R}^n$ corresponds to a well-understood parameter (such as the expression level of a gene), in the PCA basis, the new coordinates x'_i are weighted combinations $\sum_i w_i x_i$ of every original parameter x_i, and it is no longer easy to assign a meaning to different values of x'_i.

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28. Copyright 2013 by the author(s).

An effective way of ensuring that the new coordinates are interpretable is to require that each of them be a weighted combination of only a small subset of the original dimensions. In other words, we may require that the basis vectors for our new coordinate system be *sparse*. This technique is referred to in the literature as sparse principal component analysis (sPCA), and has been successfully applied in areas as diverse as bioinformatics (Lee et al., 2010), natural language processing (Richtárik et al., 2012), and signal processing (D'Aspremont et al., 2008).

Formally, an instance of sPCA is defined in terms of the cardinality-constrained optimization problem (1), in which $\Sigma \in \mathbb{R}^{n \times n}$ is a symmetric matrix (normally a positive semidefinite covariance matrix), and $k > 0$ is a sparsity parameter.

$$\begin{aligned} \max \quad & \frac{1}{2} x^T \Sigma x \\ \text{s.t.} \quad & \|x\|_2 \leq 1 \\ & \|x\|_0 \leq k \end{aligned} \tag{1}$$

Problem (1) yields a single sparse component x^* and for $k = n$, reduces to finding the leading eigenvector of Σ (i.e. to the standard formulation of PCA).

Although there exists a vast literature on sPCA, most popular algorithms are essentially variations of the same technique called the generalized power method (GPM, see Journée et al. (2010)). This technique has been shown to match or outperform most other algorithms, including ones based on SDP formulations (D'Aspremont et al., 2007), bilinear programming (Witten et al., 2009), and greedy search (Moghaddam et al., 2006). The GPM is a straightforward extension of the power method for computing the largest eigenvector of a matrix (Parlett, 1998) and can also be interpreted as projected gradient ascent on a variation of problem (1).

Although the GPM is a very simple and intuitive algorithm, it can be slow to converge when the covariance matrix Σ is large. This should not come as unexpected, as the GPM generalizes two rather unsophisticated algorithms. In practice, eigenvectors are computed using a technique called Rayleigh quotient iteration (normally implemented as part of the QR algorithm). Similarly, in unconstrained optimization, Newton's method converges orders of magnitude faster than gradient descent.

Here, we introduce a new algorithm for problem (1) that generalizes Rayleigh quotient iteration, and that can also be interpreted as a second-order optimization technique similar to Newton's method. It converges to a local solution of (1) in about eight iterations or less on most problem instances, and generally requires between one and two orders of magnitude fewer floating point operations (flops) than the GPM. Moreover, the solutions it finds tend to be of as good or of better quality than those found by the GPM. Conceptually, our algorithm fills a gap in the family of eigenvalue algorithms and their generalizations that is left by the lack of second-order optimization methods in the sparse setting (Table 1).

2. Algorithm

We refer to our technique as *generalized Rayleigh quotient iteration* (GRQI). GRQI is an iterative algorithm that converges to local optima of (1) in a small number of iterations; see Algorithm 1 for a pseudocode definition.

Briefly, GRQI performs up to two updates at every iteration j: a Rayleigh quotient iteration update, followed by an optional power method update. The next iterate $x^{(j+1)}$ is set to the Euclidean projection $P_k(\cdot)$ on the set $\{||x||_0 \leq k\} \cap \{||x||_2 \leq 1\}$ of the result x_{new} of these steps.

The first update is an iteration of the standard Rayleigh quotient iteration algorithm on the set of *working* indices $\mathcal{W} = \{i | x_i^{(j)} \neq 0\}$ of the current iterate $x^{(j)}$. The indices which equal zero are left unchanged.

Rayleigh quotient iteration is a method that computes eigenvectors by iteratively performing the update $x_+ = (\Sigma - \mu I)^{-1} x$ for $\mu = x^T \Sigma x / x^T x$. To get an understanding of why this technique works, observe that when μ is close to an eigenvalue λ_i of Σ, the i-th eigenvalue of $(\Sigma - \mu I)^{-1}$ tends to infinity, and after the

Algorithm 1 GRQI(Σ, x_0, k, J, ϵ)

```
j ← 0
repeat
  // Compute Rayleigh quotient and working set:
  $\mu \leftarrow (x^{(j)})^T \Sigma x^{(j)} / (x^{(j)})^T x^{(j)}$
  $\mathcal{W} \leftarrow \{i | x_i^{(j)} \neq 0\}$

  // Rayleigh quotient iteration update:
  $x_{\mathcal{W}}^{(j)} \leftarrow (\Sigma_{\mathcal{W}} - \mu I)^{-1} x_{\mathcal{W}}^{(j)}$
  $x_{\text{new}} \leftarrow x^{(j)} / ||x^{(j)}||_2$

  // Power method update:
  if $j < J$ then
    $x_{\text{new}} \leftarrow \Sigma x_{\text{new}} / ||\Sigma x_{\text{new}}||_2$
  end if

  $x^{(j+1)} \leftarrow P_k(x_{\text{new}})$
  $j \leftarrow j + 1$
until $||x^{(j)} - x^{(j-1)}|| < \epsilon$
return $x^{(j)}$
```

Algorithm 2 SPCA-GRQI(Σ, κ, J, ϵ, K, δ)

```
for $k = 1$ to $K$ do
  $x_0 \leftarrow$ largest column of $\Sigma$
  $x^{(k)} \leftarrow \text{GRQI}(\Sigma, x_0, \kappa, J, \epsilon)$
  $\Sigma \leftarrow \Sigma - \delta((x^{(k)})^T \Sigma x^{(k)}) x^{(k)} (x^{(k)})^T$ // Deflation
end for
```

Algorithm 3 GPower0(D, x_0, γ, ϵ)

```
Let $S_0 : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ be defined as
  $(\mathcal{S}_0(a), \gamma)_i := a_i[\text{sgn}(a_i^2 - \gamma)]_+$.
$j \leftarrow 0$
repeat
  $y \leftarrow \mathcal{S}_0(D^T x^{(j)}, \gamma) / ||\mathcal{S}_0(D^T x^{(j)}, \gamma)||_2$
  $x^{(j)} \leftarrow Dy / ||Dy||_2$
  $j \leftarrow j + 1$
until $||x^{(j)} - x^{(j-1)}|| < \epsilon$

return $\frac{\mathcal{S}_0(D^T x^{(j)}, \gamma)}{||\mathcal{S}_0(D^T x^{(j)}, \gamma)||_2}$
```

Algorithm 4 GPower1(D, x_0, γ, ϵ)

```
Let $S_1 : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ be defined as
  $(\mathcal{S}_1(a, \gamma))_i := \text{sgn}(a_i)(|a_i| - \gamma)_+$.
$j \leftarrow 0$
repeat
  $y \leftarrow \mathcal{S}_1(D^T x^{(j)}, \gamma) / ||\mathcal{S}_1(D^T x^{(j)}, \gamma)||_2$
  $x^{(j)} \leftarrow Dy / ||Dy||_2$
  $j \leftarrow j + 1$
until $||x^{(j)} - x^{(j-1)}|| < \epsilon$

return $\frac{\mathcal{S}_1(D^T x^{(j)}, \gamma)}{||\mathcal{S}_1(D^T x^{(j)}, \gamma)||_2}$
```

Eigenvalue algorithm	Power method	Rayleigh quotient iteration
Equivalent optimization method	Gradient descent	Newton's method
Sparse generalization	Generalized power method	*Generalized Rayleigh quotient iteration*
Convergence rate	Linear	Cubic

Table 1. Conceptual map of eigenvalue algorithms and their generalizations. The light gray box indicates the previously unoccupied spot where our method can be placed.

multiplication $x_+ = (\Sigma - \mu I)^{-1}x$, x_+ becomes almost parallel to the i-th eigenvector. Since μ is typically a very good estimate of an eigenvalue of Σ, this happens after only a few multiplications in practice.

The second update is simply a power method step along all indices. This step starts from where the Rayleigh quotient update ended. In addition to improving the current iterate, it can introduce large changes to the working set at every iteration, which helps the algorithm to find a good sparsity pattern quickly.

Finally, the projection $P_k(x)$ on the intersection of the l_0 and l_2 balls ensures that $x^{(j+1)}$ is sparse. Note that this projection can be done by setting all but the k largest components of x (in absolute value) to zero and normalizing the resulting vector to a norm of one.

For the sake of simplicity, we defined Algorithm 1 to perform power method updates only at the first J iterations, with $J \in [0, \infty]$ being a parameter. We also implemented more sophisticated ways of combining Rayleigh quotient and power method updates, but they offered only modest performance improvements over Algorithm 1, and so we focus here on a very simple but effective method.

2.1. Convergence

In all our experiments, we simply set $J = \infty$, and we recommend this choice of parameter for most applications. However, to formally guarantee convergence, J must take on a finite value, in which case we can establish the following proposition.

Proposition. *Let $\Sigma \in \mathbb{R}^{n\times n}$ such that $\Sigma = \Sigma^T$, $x_0 \in \mathbb{R}^n$, $k > 0$, $J < \infty$ be input parameters to Algorithm 1. There exists an x^* such that $||x^*||_0 \leq k$ and such that the iterates $(x_j)_{j=1}^{\infty}$ generated by Algorithm 1 converge to x^* at a cubic rate:*

$$||x_{j+1} - x^*|| = O\left(||x_j - x^*||^3\right).$$

This proposition follows from the fact that when $j \geq J$, Algorithm 1 reduces to Rayleigh quotient iteration on a fixed set of k non-zero indices, and from the fact that standard Rayleigh quotient iteration converges at a cubic rate (Parlett, 1998).

2.2. Starting point

We recommend setting x_0 to the largest column of Σ, as previously suggested in Journée et al. (2010). Interestingly, we also observed that we could initialize Algorithm 1 randomly, as long as the first Rayleigh quotient $\mu^{(0)}$ was close the largest eigenvalue λ_1 of Σ. In practice, an accurate estimate of λ_1 can be obtained very quickly by performing only a few steps of Power iteration (O'Leary et al., 1979).

2.3. Multiple sparse principal components

Most often, one wants to compute more than one sparse principal component of Σ. Algorithm 1 easily lends itself to this setting; the key extra step that must be added is the *deflation* of Σ. In order to avoid describing variation in the data that has already been explained by an earlier principal component x, we replace Σ at the next iteration by $\Sigma_+ = \Sigma - (x^T\Sigma x)xx^T$. Thus, if a new principal component x_+ explains any variance in the direction of x (i.e. if $x^Tx_+ \neq 0$), that variance will not count towards the new objective function $x_+^T\Sigma_+x_+$.

In practice, there are settings where one may not want to perform a full deflation of Σ, but instead only give variance explained in the direction of x less weight. This can be done by changing the deflation step to $\Sigma_+ = \Sigma - \delta(x^T\Sigma x)xx^T$, where $0 \leq \delta \leq 1$ is a parameter. For example, this partial deflation turns out to be useful when analyzing image features, as we show in Section 5. In addition, there exist several other deflation schemes which could be used with our algorithm; we refer the reader to the survey by Mackey (2009) for details. The full version of our method for computing K principal components can be found in Algorithm 2.

Finally, we would like to point out that block algorithms that compute several components at once, like the ones in Journée et al. (2010), can also be derived from our work.

3. Discussion

Essentially, the GRQI algorithm modifies the power method by replacing the power iteration step by

Rayleigh quotient iteration, followed by an optional step of the power method. Rayleigh quotient iteration is a simple technique that converges to an eigenvector much faster than power iteration, and is responsible for the high speed of modern eigenvalue algorithms. Perhaps our most interesting observation is that this technique dramatically improves the performance of algorithms in the sparse setting as well.

Existing algorithms for solving (1) essentially do two tasks at once: (a) identifying a good sparsity pattern, and (b) computing a good eigenvector within that pattern. Intuitively, our algorithm works well because Rayleigh quotient iteration converges to high-variance eigenvectors within the current working set of indices $\mathcal{W}$ much faster than the power method. Therefore, it solves task (b) much more quickly than the GPM, at the cost of some extra computation.

3.1. Rayleigh quotient iteration as Newton's method

Interestingly, the effectiveness of Rayleigh quotient iteration can also be explained by the fact that it is equivalent to a slight variation of Newton's method (Tapia & Whitley, 1988). By the same logic, one can view GRQI as a second-order projected optimization method on the objective (1).

In fact, Algorithm 1 resembles in some ways the projected Newton methods of Gafni & Bertsekas (1984). Both approaches take at every iteration a gradient step and a Newton step; in both cases, Newton steps are restricted to a subspace to avoid getting stuck at bad local optima (see Bertsekas (1982) for an example). Unfortunately, projected Newton methods can only be used with very simple constraints, unlike those in (1).

3.2. Computational complexity

Perhaps the chief concern with second order methods is their per-step cost. Algorithm 1 performs at every iteration only $O(k^3 + nk)$ flops, where k is the number of non-zero components. For comparison, the GPM requires $O(n^2 + nk)$ flops (see next section and Algorithms 3 and 4). Thus, our algorithm has an advantage when $k \ll n$, which is precisely when the principal components are truly sparse.

3.3. Comparison to the generalized power method and to other sPCA algorithms

The generalized power method is a straightforward extension of the power method for computing eigenvalues; in its simplest formulation, it alternates between power iteration and projection steps until convergence. According to Journée et al. (2010), its most effective formulations are two algorithms called GPower0 and GPower1 (Algorithms 3 and 4). Both algorithms are equivalent to subgradient ascent on the objective function

$$\max_{x:||x||_2 \leq 1} \frac{1}{2} x^T D^T D x - \gamma ||x||, \tag{2}$$

where D is the data matrix, $\gamma > 0$ is a sparsity parameter and $|| \cdot ||$ can be the l_0 norm (in the case of GPower0), or the l_1 norm (in the case of GPower1).

Our method has several immediate advantages over Algorithms 3 and 4. For one, the user can directly specify the desired cardinality of the solution, instead of having to search for a penalty parameter γ. In fact, we found that varying γ by $< 1\%$ in equation (2) sometimes led to changes in the cardinality of the solution of more than 10%. Moreover, the desired cardinality is kept constant at every iteration of our algorithm; thus one can stop iterating when a desired level of variance has been attained.

Other algorithms for sPCA are often variants of the generalized power method, including some highly cited ones, such as Zou et al. (2006) or Witten et al. (2009). Methods that are not equivalent to the GPM include the SDP relaxation approach of D'Aspremont et al. (2007); since it cannot scale to more than a few hundred variables, we do not consider it in this study. Another class of algorithms includes greedy methods that search the exponential space of sparsity patterns directly (Moghaddam et al., 2006); such algorithms have been shown to be much slower than the GPM (Journée et al., 2010), and we don't consider them either.

4. Evaluation

We now proceed to compare the performance of generalized Rayleigh quotient iteration to that of Algorithms 3 and 4. We evaluate the algorithms on a series of standard tasks that are modeled after the ones of Journée et al. (2010). The experiments in this section are performed on random positive semidefinite matrices $\Sigma \in \mathbb{R}^{n \times n}$ of the form $\Sigma = A^T A$ for some $A \sim \mathcal{N}(0,1)^{n \times n}$. Although we show results only for $n = 1000$, our observations carry over to other matrix dimensions.

Throughout this section, we use as our convergence criterion the condition $||x - x_{\text{prev}}|| < 10^{-6}$. Note that we cannot use criteria based on the objective functions (1) and (2), as they are not comparable: one measures cardinality and one doesn't. Furthermore, we set $J = \infty$ in all experiments, in which case Algorithm 1 performs

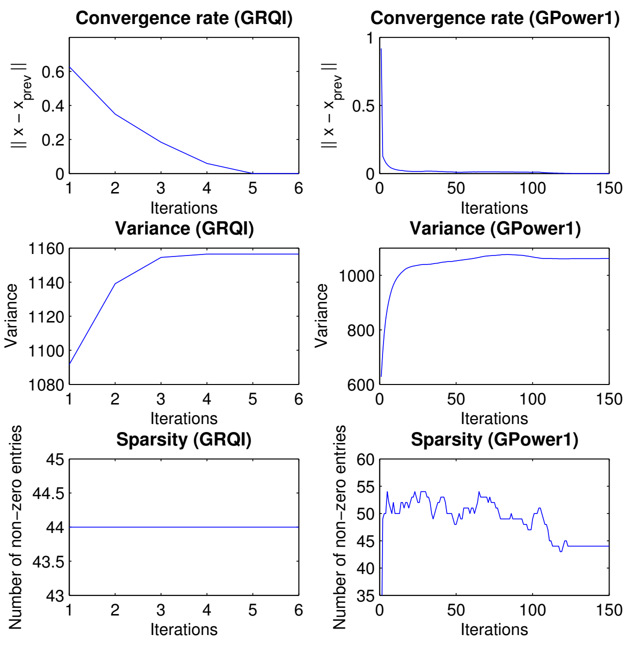


Figure 1. Comparison of generalized Rayleigh quotient iteration and the generalized power method on a random 1000×1000 matrix.

at every iteration a Rayleigh quotient step, followed by a full power method step.

4.1. Convergence rates

As a demonstration of the rapid convergence of generalized Rayleigh quotient iteration, we start by plotting in Figure 1 the variance, sparsity, and precision at every iteration of GRQI and GPower1. We set the sparsity parameter γ of GPower1 to 5, and set the parameter k to match the sparsity of the final solution returned by that algorithm, which was $k = 44$ (4.4% sparsity). We found that this setting was representative of what happens at other sparsities.

We observe that GRQI converges in six iterations, which is quite typical for the problem instances we considered. This observation is consistent with the speed of regular Rayleigh quotient iteration and appears to support empirically the cubic convergence guarantees of our algorithm. Interestingly, the rapid convergence in Figure 1 was achieved without setting $J < \infty$, as the convergence argument formally requires.

4.2. Time complexity analysis

To further evaluate the speed of GRQI, we counted the number of flops that were required to compute a sparse component on a number of random matrices. We chose to measure algorithm speed in flops because unlike time measurements, they do not depend on various implementation details. In fact, when implemented in a popular high-level language like MATLAB, GRQI would have an unfair advantage over the GPM, as it spends relatively more time in low-level subroutines.

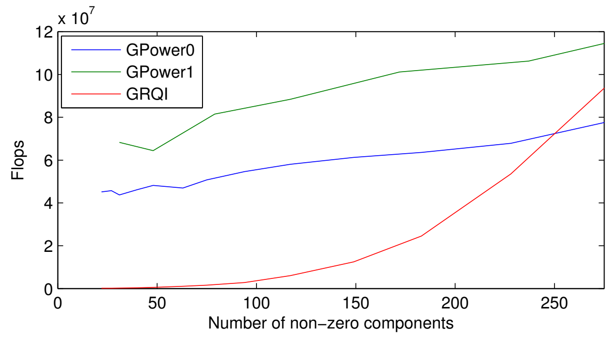


Figure 2. Number of total flops versus sparsity of the final solution. Each curve is an average of ten 1000×1000 random matrices.

To measure flops, we counted the number of matrix operations performed by each algorithm. For every $k \times n$ matrix-vector multiplication, we counted kn flops, whereas for $k \times k$ matrix inversions, we counted $\frac{1}{3}k^3 + 2k^2$. The latter is the complexity of inverting a Hermitian matrix through an LDL^T factorization. Operations taking $o(n^2)$ time were ignored, as their flop count was negligible.

Results of this experiment appear in Figure 2. GRQI uses between one and two orders of magnitude fewer flops at sparsities below 20%; at sparsities below 5%, it used a hundred times fewer flops. However, for large values of k, the complexity of GRQI outgrows that of the GPM. In this case, we found that we could bring down the number of GRQI flops to at least the level of the GPM by letting J be small; however at those levels of k the problem is not truly sparse, and so we do not give details.

In practice, the computational requirements of the GPM would appear somewhat smaller if we used a loose convergence criterion, such as $||x - x_{\text{prev}}|| < 10^{-2}$. However, even in that case, GRQI required at least an order of magnitude fewer flops. On the other hand, the GPM in our experiments needed about 3-4 restarts before we could find a penalty term that yielded the desired number of non-zero entires in the principal component. We did not count these restarts in our flop measurements.

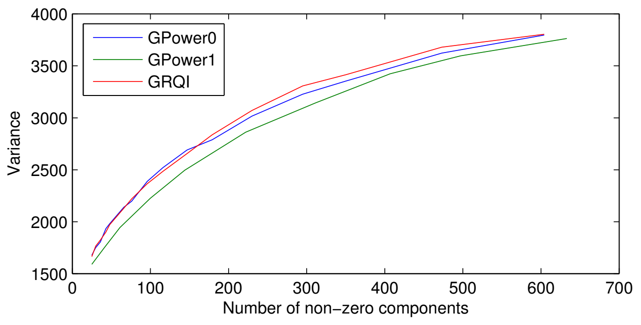


Figure 3. Variance explained versus sparsity of the leading principal component. Each curve represents an average of ten 1000 × 1000 random matrices.

4.3. Variance

Finally, we demonstrate that the quality of the solutions to which our algorithm quickly arrives matches that of the solutions obtained by the GPM. In Figure 3, we plot the tradeoff between the sparsity and the variance of a solution for both algorithms. The two curves are essentially identical, and this observation could be also made for other matrix dimensions.

5. Sparse SVD

Thus far, our methods have been presented only in the context of positive semidefinite covariance matrices. However, they can be easily extended to handle an arbitrary rectangular $m \times n$ matrix R. In that setting, the problem we solve becomes a generalization of the singular value decomposition (SVD); hence, we refer to it as the sparse singular value decomposition (sSVD).

Sparse SVD generalizes objective (1) as follows.

$$\begin{aligned} \max \;\; & u^T R v \\ \text{s.t.} \;\; & \|u\|_2 \le 1 \quad \|v\|_2 \le 1 \\ & \|u\|_1 \le k_u \quad \|v\|_1 \le k_v \end{aligned} \tag{3}$$

Problem (3) can be reduced to problem (1) by exploiting the well-known observation that

$$\begin{aligned} u^T R v &= \frac{1}{2}\begin{pmatrix} v \\ u \end{pmatrix}^T \begin{pmatrix} 0 & R^T \\ R & 0 \end{pmatrix} \begin{pmatrix} v \\ u \end{pmatrix} \\ &=: \frac{1}{2}\begin{pmatrix} v \\ u \end{pmatrix}^T \Sigma \begin{pmatrix} v \\ u \end{pmatrix}. \end{aligned}$$

Thus we can solve (3) by running Algorithm 1 on the $(m+n) \times (m+n)$ matrix Σ. Fortunately, this can be done without ever explicitly forming $(m+n) \times (m+n)$ matrices. Using blockwise inversion, we compute the inverse of $\Sigma - \mu I$ as follows.

$$\begin{pmatrix} -\mu I & R^T \\ R & -\mu I \end{pmatrix}^{-1} = \begin{pmatrix} \frac{1}{\mu^2} R^T S^{-1} R - \frac{1}{\mu} I & \frac{1}{\mu} R^T S^{-1} \\ \frac{1}{\mu} S^{-1} R & S^{-1} \end{pmatrix},$$

where $S = RR^T - \mu I$ is the Schur complement of μI. When R is a $k_1 \times k_2$ submatrix with $k_1 \le k_2$, $\Sigma - \mu I$ can be inverted using $O(k_1^2 k_2 + k_1^3)$ floating point operations. Note that when the matrix R is square, this complexity reduces to that of Algorithm 1. Details may be found in Algorithm 5.

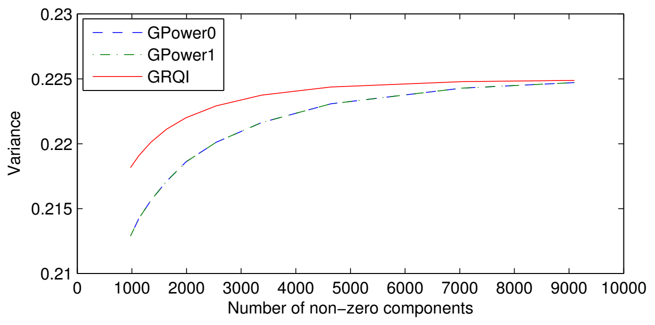
Algorithm 5 SSVD(R, u_0, v_0, k_u, k_v, J, ϵ)

```
j ← 0
repeat
  // Compute Rayleigh quotient and working sets
  $\mu \leftarrow (u^{(j)})^T R v^{(j)} / (\|u\|_2 \|v\|_2)$
  $\mathcal{W}_u \leftarrow \{i | u_i^{(j)} \neq 0\}$
  $\mathcal{W}_v \leftarrow \{i | v_i^{(j)} \neq 0\}$

  // Rayleigh quotient step
  $A \leftarrow R_{\mathcal{W}_u, \mathcal{W}_v}$
  $\begin{pmatrix} v^{(j)}_{\mathcal{W}_v} \\ u^{(j)}_{\mathcal{W}_u} \end{pmatrix} \leftarrow \begin{pmatrix} -\mu I & A^T \\ A & -\mu I \end{pmatrix}^{-1} \begin{pmatrix} v^{(j)}_{\mathcal{W}_v} \\ u^{(j)}_{\mathcal{W}_u} \end{pmatrix}$
  $\begin{pmatrix} v_{\text{new}} \\ u_{\text{new}} \end{pmatrix} \leftarrow \begin{pmatrix} v^{(j)} \\ u^{(j)} \end{pmatrix} / \left\| \begin{pmatrix} v^{(j)} \\ u^{(j)} \end{pmatrix} \right\|_2$

  // Power step
  if $j < J$ then
    $u_{\text{new}} \leftarrow R v_{\text{new}}$
    $v_{\text{new}} \leftarrow R^T u_{\text{new}}$
  end if

  $u^{(j+1)} \leftarrow P_{k_u}(u_{\text{new}})$
  $v^{(j+1)} \leftarrow P_{k_v}(v_{\text{new}})$
  $j \leftarrow j + 1$
until $\max(\|u^{(j)} - u^{(j-1)}\|, \|v^{(j)} - v^{(j-1)}\|) < \epsilon$
return $u^{(j)}, v^{(j)}$
```

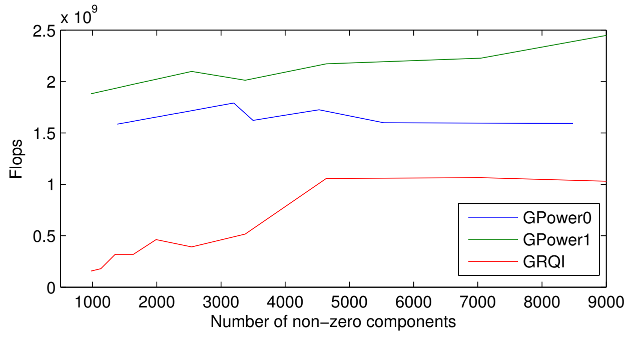
5.1. Gene expression data

We test the performance of Algorithm 5 on gene expression data collected in the prostate cancer study by Singh et al. (2002). The study measured expression profiles for 52 tumor and 50 normal samples over 12,600 genes, resulting in a $102 \times 12,600$ data matrix.

Figure 4 shows the variance-sparsity tradeoffs of GRQI and GPower for the first principal component of the data matrix, as well the number of flops they use. Both algorithms explain roughly the same variance (in fact, GRQI explains 2.35% more for small sparsities), whereas time complexity is again significantly smaller for Algorithm 1. Although the relative advantage is



(a) Variance/sparsity tradeoff



(b) Total flops required at different sparsity levels

Figure 4. Computing the leading sparse principal component on the Singh et al. (2002) gene expression dataset.

not as large as in Figure 2, GRQI still uses about an order of magnitude fewer flops at small sparsity levels.

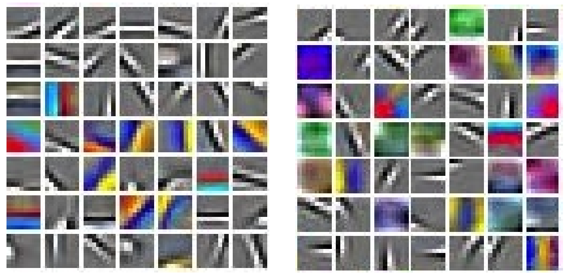
Yao et al. (2012) have shown that the principal components of the above data matrix map to known pathways associated with cancer. Our algorithms can identify these pathways at a fraction of the computational cost of existing methods.

5.2. Application in deep learning/machine vision

As a further example of how our methods can be applied in practice, we use Algorithm 5 to perform unsupervised feature learning on the STL-10 dataset of images.

Briefly, we sampled 100,000 random $3 \times 8 \times 8$ RGB image patches and computed sparse principal components on the resulting $196 \times 100,000$ matrix using a deflation parameter $\delta = 0.2$. This partial deflation allowed us to recover 400 principal components, which is more than twice the rank of the data matrix.

Curiously, we observe that when visualized as 8×8 color matrices, these principal components take on the shape of little "edges", as shown in Figure 5. We found



(a) sSVD (b) Autoencoder

Figure 5. Sparse components learned by sSVD on image patches (a) compared to features learned by an autoencoder neural net (b).

it challenging to reproduce the above results using the GPower algorithms because of the difficulties in tuning the sparsity parameter γ; therefore we do not report a speed comparison on this data.

In the vision literature, the edges shown in Figure 5 are known as *Gabor filters*, and are used as a basis for representing images in many machine vision algorithms. Their use considerably improves these algorithms' performance.

In the past several years, learning such features automatically from data has been a major research topic in artificial intelligence. The fact that sparse principal component analysis can be used as a technique for unsupervised feature extraction may lead to new feature learning algorithms that are simple, fast, and that use matrix multiplications that can be parallelized over many machines.

More generally, sparse principal component analysis addresses a key challenge in statistics and thus has applications in many other areas besides bioinformatics and machine vision. We believe that developing fast, simple algorithms for this problem will spread its use to an even wider range of interesting application domains.

Acknowledgements

We wish to thank Stephen Boyd for helpful discussions and advice, as well as Ben Poole for reviewing an early draft of the paper.

References

Bertsekas, D. P. Projected Newton Methods for Optimization Problems with Simple Constraints. *SIAM Journal on Control and Optimization*, 20(2):221–246+, 1982.

D'Aspremont, Alexandre, El Ghaoui, Laurent, Jordan, Michael I., and Lanckriet, Gert R. G. A Direct Formulation for Sparse PCA Using Semidefinite Programming. *SIAM Rev.*, 49(3):434–448, 2007.

D'Aspremont, Alexandre, Bach, Francis, and Ghaoui, Laurent El. Optimal Solutions for Sparse Principal Component Analysis. *J. Mach. Learn. Res.*, 9:1269–1294, 2008.

Gafni, Eli M. and Bertsekas, Dimitri P. Two-Metric Projection Methods for Constrained Optimization. *SIAM Journal on Control and Optimization*, 22(6): 936–964, 1984.

Journée, Michel, Nesterov, Yurii, Richt a rik, Peter, and Sepulchre, Rodolphe. Generalized Power Method for Sparse Principal Component Analysis. *J. Mach. Learn. Res.*, 11:517–553, 2010.

Lee, Donghwan, Lee, Woojoo, Lee, Youngjo, and Pawitan, Yudi. Super-sparse principal component analyses for high-throughput genomic data. *BMC Bioinformatics*, 11(1):296, 2010.

Mackey, Lester. Deflation Methods for Sparse PCA. In *Advances in Neural Information Processing Systems*, pp. 1017–2024. MIT Press, 2009.

Moghaddam, Baback, Weiss, Yair, and Avidan, Shai. Spectral Bounds for Sparse PCA: Exact and Greedy Algorithms. In *Advances in Neural Information Processing Systems*, pp. 915–922. MIT Press, 2006.

O'Leary, Dianne P., Stewart, G. W., and Vandergraft, James S. Estimating the Largest Eigenvalue of a Positive Definite Matrix. *Mathematics of Computation*, 33(148):pp. 1289–1292, 1979.

Parlett, Beresford N. *The symmetric eigenvalue problem.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

Richtárik, Peter, Takác, Martin, and Ahipasaoglu, Selin Damla. Alternating Maximization: Unifying Framework for 8 Sparse PCA Formulations and Efficient Parallel Codes. *CoRR*, abs/1212.4137, 2012.

Singh, Dinesh, Febbo, Phillip G, Ross, Kenneth, Jackson, Donald G, Manola, Judith, Ladd, Christine, Tamayo, Pablo, Renshaw, Andrew A, D'Amico, Anthony V, Richie, Jerome P, Lander, Eric S, Loda, Massimo, Kantoff, Philip W, Golub, Todd R, and Sellers, William R. Gene expression correlates of clinical prostate cancer behavior. *Cancer Cell*, 1(2): 203–209, 2002.

Tapia, R. A. and Whitley, David L. The Projected Newton Method has Order 1 + 2 for the Symmetric Eigenvalue Problem. *SIAM Journal on Numerical Analysis*, 25(6):pp. 1376–1382, 1988.

Witten, D M, Tibshirani, R, and Hastie, T. A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis. *Biostatistics*, 10(3):515–534, jun 2009.

Yao, Fangzhou, Coquery, Jeff, and Le Cao, Kim-Anh. Independent Principal Component Analysis for biologically meaningful dimension reduction of large biological data sets. *BMC Bioinformatics*, 13(1):24, 2012.

Zou, Hui, Hastie, Trevor, and Tibshirani, Robert. Sparse Principal Component Analysis. *Journal of Computational and Graphical Statistics*, 15(2):265–286, jun 2006.