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

Volodymyr Kuleshov

Department of Computer Science, Stanford University

Highlights

New algorithms for sparse PCA that

- Perform $O(k^3 + nk)$ flops/step, for a sparsity of k;
- In practice, use 10-100x fewer flops than current state-of-the-art methods;
- Produce eigenvectors that are as good or better than ones from existing algorithms.
- Generalize Rayleigh quotient iteration;

Sparse PCA

And instance of sparse PCA is defined as

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

for $\Sigma \in \mathbb{R}^{n \times n}$, $\Sigma = \Sigma^T$ and $k > 0$.

Current state-of-the-art

Most popular methods are variations of the *generalized power method*: Zou et al. (2006), Witten et al., (2009), Journee et al. (2010).

Algorithm 1 GPower0/GPower1(D, x_0, γ, ϵ)

$j \leftarrow 0$
repeat
  $y \leftarrow \mathcal{S}(D^T x^{(j)}, \gamma) / \|\mathcal{S}(D^T x^{(j)}, \gamma)\|_2$
  $x^{(j)} \leftarrow Dy / \|Dy\|_2$
  $j \leftarrow j + 1$
until $\|x^{(j)} - x^{(j-1)}\| < \epsilon$

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

where $S : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ is

$$S(a, \gamma)_i := \begin{cases} a_i [\text{sgn}(a_i^2 - \gamma)]_+ & \text{for GPower0} \\ \text{sgn}(a_i)(|a_i| - \gamma)_+ & \text{for GPower1} \end{cases}$$

GPower outperforms other sparse PCA algorithms (Journee et al., 2010), including:

- SDP formulations (D'Aspremont et al., 2007);
- Greedy search (Moghaddam et al., 2006).

Generalized Rayleigh quotient iteration

The generalized power method is based on two rather unsophisticated algorithms: the power method for computing eigenvalues and gradient ascent. We introduce an algorithm that extends the state-of-the-art Rayleigh quotient iteration algorithm and that can be interpreted as a form of Newton's method.

	GPower	**GRQI**
Extends eigenvalue algorithm	Power method	Rayleigh quotient iteration
Interpretation in optimization	Subgradient ascent	A second-order method
Rate of convergence	Linear	Cubic
Time complexity	$O(nk + n^2)$ flops over ~100 iter.;	$O(nk + k^3)$ flops over ~10 iter.

Pseudocode

Algorithm 2 GRQI(Σ, x_0, k, J, ϵ)

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

  $x_{\mathcal{W}}^{(j)} \leftarrow (\Sigma_{\mathcal{W}} - \mu I)^{-1} x_{\mathcal{W}}^{(j)}$ // RQI Update on $\mathcal{W}$
  $x_{\text{new}} \leftarrow x^{(j)} / \|x^{(j)}\|_2$

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

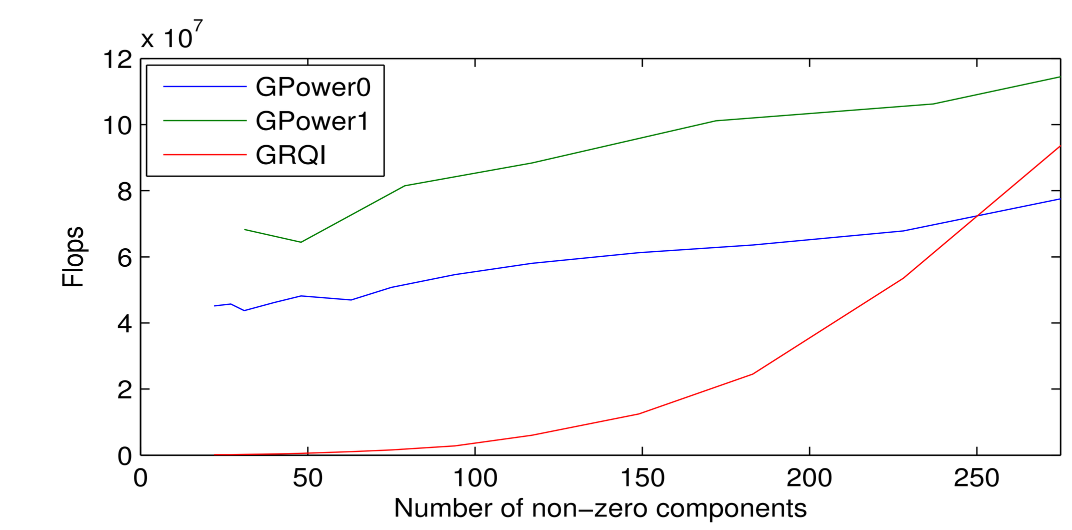
  $x^{(j+1)} \leftarrow \text{Project}_k(x_{\text{new}})$
  $j \leftarrow j + 1$
until $\|x^{(j)} - x^{(j-1)}\| < \epsilon$
return $x^{(j)}$

At every iteration, Algorithm 2 updates all non-zero indices using Rayleigh quotient iteration; for the first J iterations, it also performs a Power method step. It projects iterates on the l_0 ball.

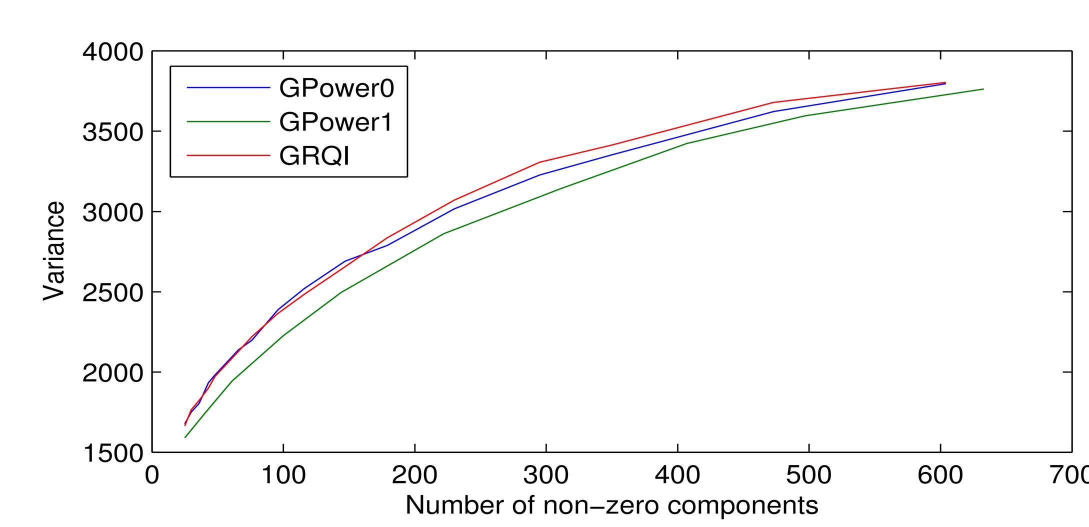
When $J < \infty$, the iterates $(x_j)_{j=1}^{\infty}$ of Algorithm 2 converge to a limit x^* at a cubic rate: $\|x_{j+1} - x^*\| = O(\|x_j - x^*\|^3)$.

Comparison to GPower

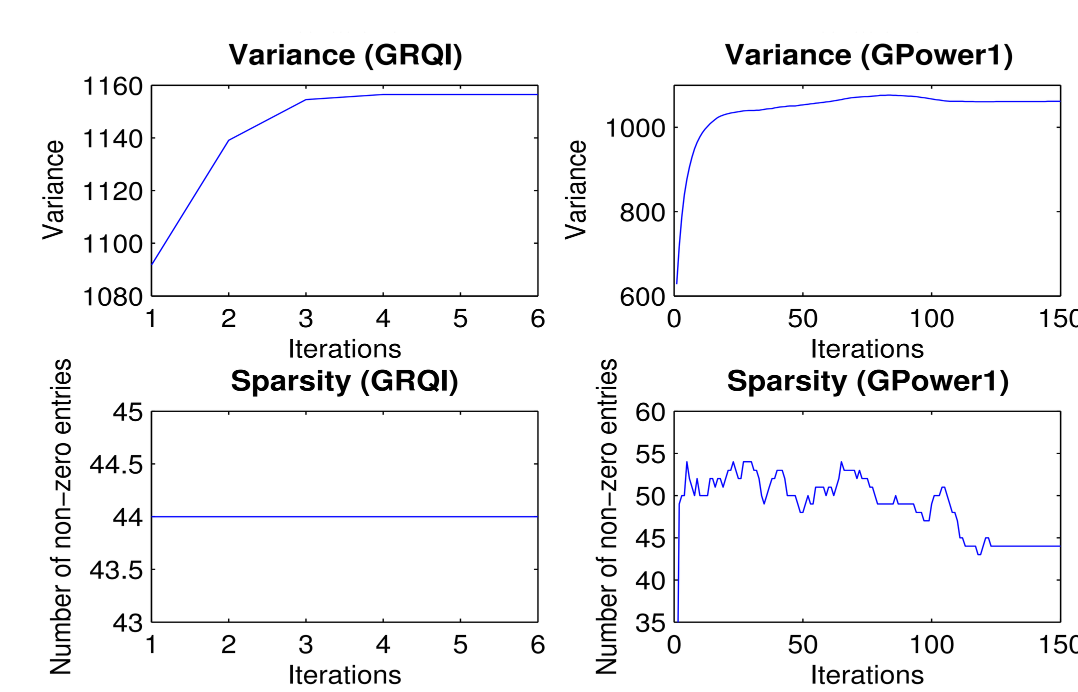
Basically, same results using ~10-100x fewer flops:



(a) Flops to compute eigenvector as a function of sparsity ($\mathbb{R}^{1000 \times 1000}$)



(b) Variance/sparsity tradeoff (random matrices in $\mathbb{R}^{1000 \times 1000}$)



Evolution of variance and sparsity across a run in $\mathbb{R}^{1000 \times 1000}$

Sparse SVD

Sparse SVD generalizes objective (1) as follows.

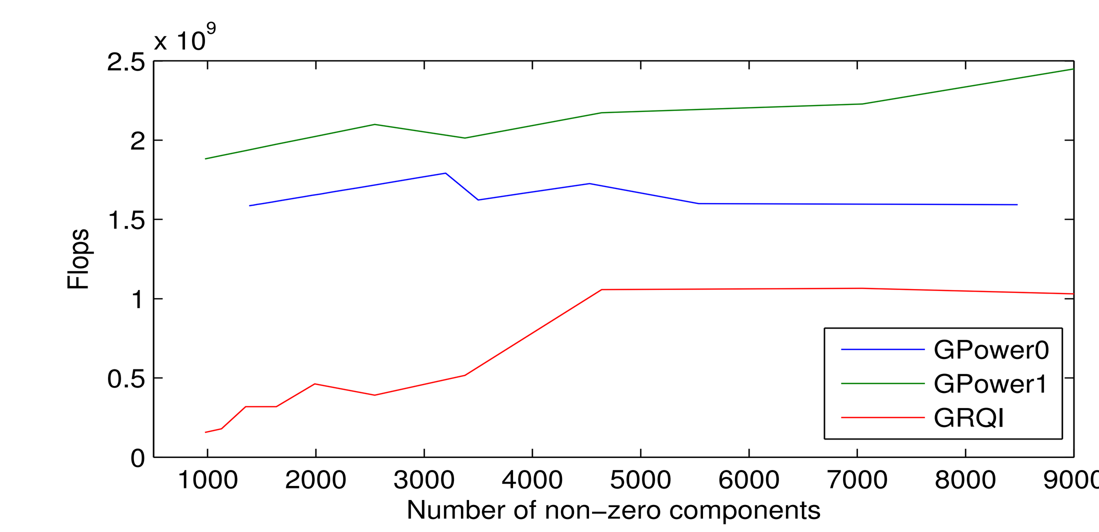
$$\begin{aligned} \max \quad & u^T R v \\ \text{s.t.} \quad & \|u\|_2 \leq 1 \quad \|v\|_2 \leq 1 \\ & \|u\|_1 \leq k_u \quad \|v\|_1 \leq k_v \end{aligned} \tag{2}$$

Problem (2) reduces to problem (1) using

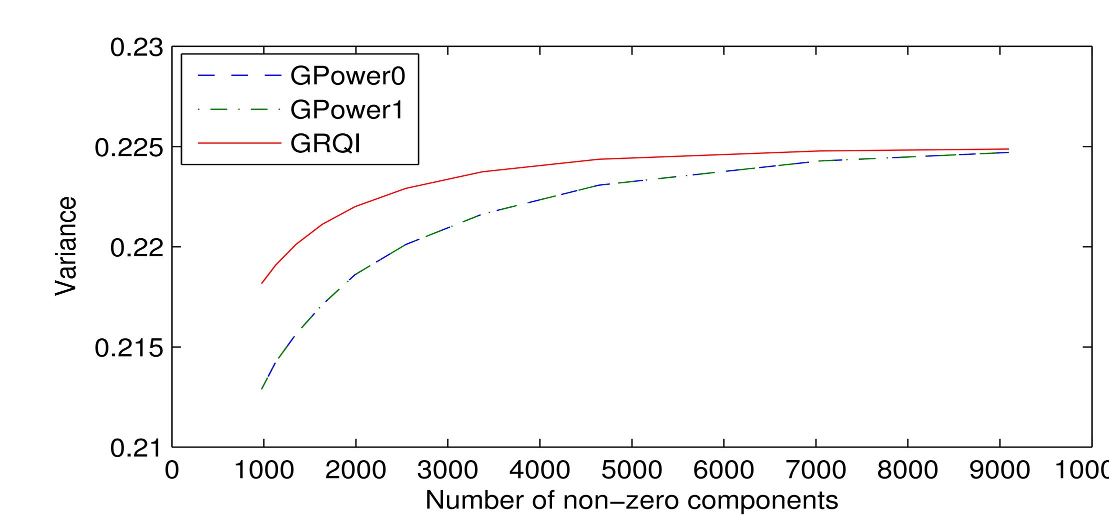
$$u^T R v = \frac{1}{2} \begin{pmatrix} v \\ u \end{pmatrix}^T \begin{pmatrix} 0 & R^T \\ R & 0 \end{pmatrix} \begin{pmatrix} v \\ u \end{pmatrix}.$$

Applications of the sparse SVD

Finding leading sparse principal component for a gene expression dataset using up to 10x fewer flops.

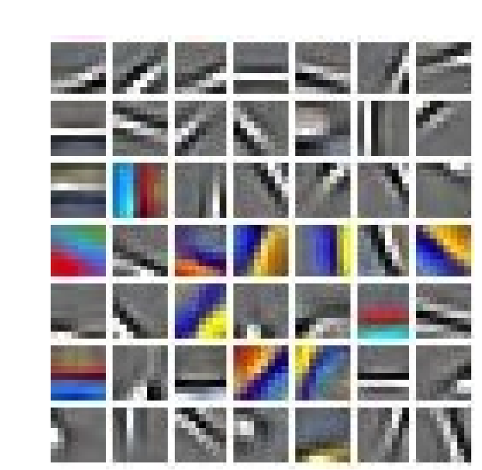


(c) Flops to compute eigenvector as a function of sparsity ($\mathbb{R}^{102 \times 12600}$)

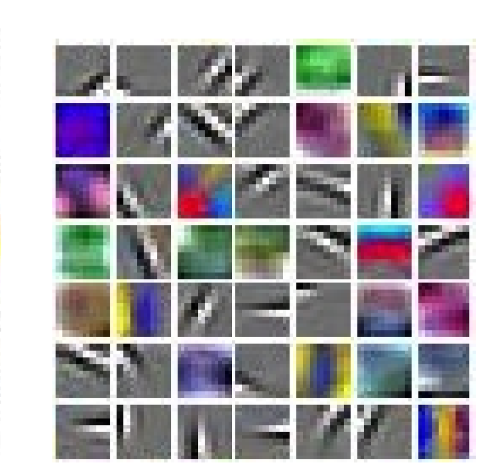


(d) Variance/sparsity tradeoff (gene expression matrix in $\mathbb{R}^{102 \times 12600}$)

Sparse SVD also uncovers Gabor filters from a matrix of image patches taken from the STL-10 dataset.



(e) sSVD



(f) Autoencoder