# Shrinking the Sparse PCA problem using projection and perturbation

Daniel Ringwalt

January 24, 2023

## 1 Background

PCA is understood to produce the eigenvector having the largest eigenvalue in a positive definite matrix (the covariance matrix $\Sigma$). The eigenvector is written as:

$$\max_v v^T \Sigma v \text{ such that } ||v||_2 = 1$$

$k$-sparse PCA makes a cut to the system (applying a rank-$k$ projection matrix using $k$ standard basis vectors), such that $||v||_0 = k$. Selecting $k$ standard basis vectors is equivalent to selecting a subset of $k$ variables, out of $n$ variables.

Consider increasing $k$ by one (including another row and column in $\Sigma$ after projection). In general, all eigenvalues, all eigenvectors, and the new data to be added all influence the new spectrum, but the previous state can be used to compute the new state (rank-one update; the original matrix will be diagonalized, and the added data will undergo a change-of-basis). We believe that eagerly computing the diagonalization, each time that we increment $k$, will lead to novel tactics for Sparse PCA. However, if new tactics are not used based on the properties of the subsystem, then it is better not to solve the data SVD/covariance eigenproblem incrementally, waiting until all $k$ variables are chosen.

Here, we will propose one application of the updated diagonalization, for making a cut to the system by projecting onto the current principal component. Simpler tactics could be possible. For our graph search tactics, our objective is to shrink the number of nodes visited before terminating on realistic data sets. We will also need to note and evaluate the wall time of our evaluation at each step, which is becoming more complex.

## 2 Tree search Sparse PCA

Berk et. al described a tree search for the Sparse PCA problem. Each node of the tree contains a lower bound and upper bound on each entry of the principal component. At the root, the lower bound at the node is $\mathcal{L} = \mathbf{0}$ and the upper bound is $\mathcal{U} = \mathbf{1}$. Child nodes are found by disallowing exactly one variable where $\mathcal{L}_i \neq \mathcal{U}_i$ (updating $\mathcal{U}_i$ from 1 to 0), or committing to that variable's inclusion (updating $\mathcal{L}_i$ from 0 to 1). The root has an empty support for $\mathcal{L}$. Denote the support of $\mathcal{L}$ as $i$. This represents a subsystem which we can update using a rank-one update for a child node, if $\mathcal{L}$ has changed.

Time complexity is expected to be average-case $O\left(\binom{n}{k}\right)$. Berk et. al's branch-and-bound of the tree ought to perform better on real data sets suited to PCA, where there should already be some $k$-variable subset of the system where all variables have good correlation with each other.

## 2.1 Lower bounds

The global lower bound on any tree node is stored, for evaluating each node which will be explored. This lower bound could be continuously updated using any stochastic Sparse PCA or sparse recovery algorithm. In particular, there is not a data dependency between this lower bound exploration and the upper bound exploration.

## 2.2 Upper bounds

We will move on to upper bounds, which require analysis to prove upper bound. Performance depends on skipping nodes where the node upper bound is lower than the global lower bound. Positive definite eigenvalues are bounded by the matrix trace, but without an improved upper bound, the runtime of the system would degrade to $O\left(\binom{n}{k}\right)$. After taking a projection of $\Sigma$, identities such as the Gershgorin circle theorem can be applied to bound all eigenvalues.

# 3 Recursive Sparse PCA problem

Our objective function, on $\Sigma$ projected in order to keep only $k$ rows/columns, is related to the operator norm of the matrix after this projection. PCA explained variance consists of the dominant eigenvalue, while the operator norm of $\Sigma$ is simply the maximum absolute value singular value. $\Sigma$ is a positive definite matrix, and its diagonalization can be simultaneously used as a singular value decomposition, giving $||\Sigma||$.

We should make a cut to the system, based on the $i$ variables selected for our sparse system so far ($\mathcal{L}_i = 1$). As one example, we could remove the selected variables as well as the rejected variables (having $\mathcal{U}_i = 0$), then invoke our program recursively to select one of $\binom{n-i}{k-i}$ possible sparse systems. This reduced system would have an additive effect on our upper bound for the current node, using Weyl's inequalities.

## 4 Shrinking the problem using projection

We can shrink the difficulty of the recursive Sparse PCA program, by projecting onto a good principal component guess. A judicious guess for the principal component would be the left-singular vector of the $i$-sparse system solved at the current node. We will evaluate whether projecting onto a worse-performing choice of initial variable(s) would shrink the problem more, resulting in a tight upper bound, or would be challenging to produce a tight upper bound for.

We would project each variable's observations onto the current best-guess principal component, taking all projections (rank-one system) held separately from the residuals (recursive problem). After adding at least $k - i$ variables on top of the solved system (probably analyzing most of the $n$ variables for the upper bound), the residual observations will no longer be orthogonal to the rank-one projection (the original left-singular vector can no longer diagonalize the new system). In fact, the full spectrum of eigenvalues when selecting the $k-i$ new variables influences our overall objective. A tight bound would require a highly complex system, which would need more considerations than maximizing the dominant eigenvalue.

Let's have our problem, after shrinkage, run Sparse PCA exactly. After projection, the residual variables would have very little correlation to the rank-one data matrix of projected variables, if the best principal component already explained almost all of the variance. Now, we need an improved bound over Weyl's inequalities, but cannot take the smaller eigenvalues of the residual data into account. We might produce a particular sparse solution, but if we use that solution in a more complex analytical upper bound, then we do not know whether it is the global optimum solution to use.

So far, states would be given a very large upper bound if they contain both highly correlated and uncorrelated variables (maximize the rank-one projection norm, while maximizing the almost-orthogonal system using a different set of variables). We can modify the Sparse PCA program for recursive use. When a variable is selected in the rank-one system, then the increase to the principal component of the system is additive and can be computed ahead of time (dot product with each data column when producing the rank-one data). The Sparse PCA program simply needs an addition to the objective, adding a certain positive value for each variable whenever it is selected. This addition to the objective function fits well into the greedy upper bounds which are in use (Gershgorin's circle theorem).

## 5 Selecting a left-singular vector (principal component)

Performance depends upon choosing a left-singular vector which the data can be projected onto. We haven't provided instructions for an upper bound in a state where $i = 0$. We could use the entire data matrix's SVD and the exact PCA as a seed, but an effective sparsity constraint can actually rotate the large principal components significantly. $k$-sparse PCA can produce a principal component with strikingly low cosine similarity to non-sparse PCA. However, the two-dimensional plots are strikingly similar to robust PCA, which can shrink the magnitude of outlier observations. We suggest investigating any dense robust PCA, which penalizes outlier observations, as a seed which may allow us to compute the improved upper bound, from beginning to end of the graph search.

## 6 Ipsen

Make a best guess for principal component loadings $u$ for the SPCA result. This comes from an SVD of the data matrix subsetted where $\mathcal{L}_i = 1$ (which we solved efficiently). By solving the small covariance matrix/right singular vectors, we can multiply through by $A$ and find the corresponding left singular vector, which is a linear combination of the variables selected so far. $u$ should be unit-normed, because it will be used for projecting other variables.

After this step, we can subset $A$ as desired; consider for the rest of this procedure that $A$ can have any of the columns which are desired. The projected covariance matrix, which is rank-one, is $(A^T u)(u^T A)$, and it has an operator norm (for this PSD matrix, the dominant eigenvalue): $u^T A A^T u$. No further norming of values is needed. This is a (not tight) lower bound on the dominant eigenvalue of the covariance matrix.

The additive update is given by the covariance of residual data: $A^T(I - vv^T)A$. For some dominant right eigenvector $w$ of this new matrix, we have the operator norm bounding the update as: $||A^T(I - uu^T)A|| = ||A^T(I - uu^T)Aw||$. However, this is not a tight bound.

Instead of the dominant eigenvector of the update matrix, we may use the eigenvector of interest of the original matrix, which is given by $A^T u$ (but must be unit-normed). The eigenvalue after update is upper-bounded by:

$$u^T A A^T u + \frac{||(A^T(I - uu^T)A)(A^T u)||}{||A^T u||}$$

We are dealing with a quotient of column vectors. For each column vector, use a dot product and square root.

$$u^T A A^T u + \sqrt{\frac{u^T A(A^T(I - uu^T)A)(A^T(I - uu^T)A)A^T u}{u^T A A^T u}}$$

Note - We want to introduce a diagonal matrix to take certain columns from $A$, and zeroes otherwise. We need the global optimum of this upper bound, or else we might have skipped a possible variable selection for SPCA. That is starting to look like a linear programming problem. We can't continue here trying to calculate the gradient (convex optimization) and approximate solution using rounding, because we are looking for an upper bound, not a lower bound.

## 7 Future Directions

We should consider the ability of this Sparse PCA graph search framework to select $d$ principal components with a support of $k$.

Upper and lower bounding the $d+1$ largest eigenvalues in the recursive subproblem could improve bounds when recombining the system. Instead of passing a single additive value to prioritize selecting each variable, an analytic objective function callable would need to be passed for each recursive function call. Multiple features of the subproblem (upper and lower bounds on $d$ eigenvalues, and on their eigenvectors' dot products with the projection vector) would be used to determine the score of the overall problem.

## 8 Supplementary Notes

### 8.1 Linearized Ipsen formula

First, project the data onto a unit vector $v$ of length $m$ (the guess for the left singular vector corresponding to our PCA's right singular vector). The variance explained after the projection can be added up, and is linear. It is still linear after a semidefinite relaxation, and it only uses the diagonal entries of the semidefinite $S$ matrix.

The perturbation (Ipsen) of the rank-one (projected) system is bounded by:

$$\frac{n}{d} = \frac{||(S \circ \Sigma)v||}{\sqrt{\text{Tr } Svv^T}}$$

Consider the gradient with respect to the entries of $S$. The denominator can be treated as $d = \sqrt{kS_{ij} + \ell}$, where $k = v_i v_j$ and $\ell$ contains the remaining terms held constant. The numerator can be treated as $n = \sqrt{(pS_{ij} + q)^2 + r}$, where the squared term $pS_{ij} + q$ represents the $i$th component of the matrix-vector product. $p = \Sigma_{ij} \cdot v_j$, and $q = (S \circ \Sigma)_{i*} v - \Sigma_{ij} \cdot S_{ij} \cdot v_j$. The remaining rows of $S \circ \Sigma$ can be ignored, except when the entire quantity $n = ||(S \circ \Sigma)v||$ is needed.

$$\frac{\delta}{\delta S_{ij}} \frac{\sqrt{(pS_{ij} + q)^2 + r}}{\sqrt{kS_{ij} + \ell}} = \frac{p(pS_{ij} + q)}{nd} - \frac{kn}{2d^3} = \frac{\Sigma_{ij} \cdot v_j \cdot (S \circ \Sigma)_{i*} v}{nd} - \frac{v_i v_j n}{2d^3}$$

We should create a linear program of the entries of $S$ (convex programming; we need a solver which can constantly keep the $S$ matrix within the semidefinite cone). The linearization (gradient of our perturbation bound) will be added to the coefficients of the objective function. The gradient can be recomputed at the current $S$ efficiently, so the convex solver will be stopped after every several iterations, and restarted at the latest $S$ and the current linear program.