# A tutorial on sparse principal components analysis

Keshav Motwani

December 10, 2020

## 1 Background and motivation

First we review and derive important properties about regular principal components analysis (PCA) that will be useful in sparse extensions. Consider an observed data matrix $X \in \mathbb{R}^{n \times p}$. PCA seeks to find a lower dimensional representation of this data $Z \in \mathbb{R}^{n \times r}$ by maximizing the variance of each of the $r$ resulting components. Specifically, for $i \in \{1, \ldots, r\}$, PCA seeks to find

$$\underset{w_i:\, ||w_i||_2^2=1}{\arg\max}\ \mathrm{Var}(Xw_i), \quad w_j' w_k = 0 \text{ for all } j \neq k.$$

Each $w_i$ is called the $i$th loading vector and $Z_{\cdot,i} = Xw_i$ the $i$th principal component. Without loss of generality, we assume that the observed data are centered, i.e. $1'X = 0'$. Then, up to a constant, $\hat{\mathrm{Var}}(X) = X'X$, so $\mathrm{Var}(Xw_i) = w_i' \mathrm{Var}(X) w_i = w_i' X' X w_i$. Therefore, we can rewrite the optimization problem as

$$\underset{w_i:\, ||w_i||_2^2=1}{\arg\max}\ w_i' X' X w_i, \quad w_j' w_k = 0 \text{ for all } j \neq k.$$

For the first loading vector $w_i$, we only have one constraint that $||w_1||_2^2 = w_1' w_1 = 1$. Thus the Lagrangian is $w_1' X' X w_1 - \lambda_1 (w_1' w_1 - 1)$, and taking the derivative with respect to $w_1$ and setting it to 0 gives $X'Xw_1 = \lambda_1 w_1$, so $w_1$ is some eigenvector of $X'X$. Taking the derivative with respect to $\lambda_1$ and setting it to 0 gives us our constraint that $w_1' w_1 = 1$. Combining these two equations, we get that our objective function value $w_1' X' X w_1 = w_1' \lambda_1 w_1 = \lambda_1 w_1' w_1 = \lambda_1$, so the eigenvector that maximizes this is the eigenvector corresponding to the largest eigenvalue. For $w_2$, we have two constraints: $w_2' w_2 = 1$ and $w_1' w_2 = 0$. Thus the Lagrangian for this is $w_2' X' X w_2 - \lambda_2 (w_2' w_2 - 1) - \gamma_1 w_1' w_2$. Taking the derivative with respect to $w_2$ and setting it to 0 gives $2X'Xw_2 - 2\lambda_2 w_2 - \gamma_1 w_1 = 0$. Left multiplying this by $w_1'$ gives $2w_1' X' X w_2 - 2\lambda_2 w_1' w_2 - \gamma_1 w_1 = 0$. By definition of $w_1$, we have $w_1' X' X = (X'Xw_1)' = \lambda_1 w_1'$, so with the constraint that $w_1' w_2 = 0$, we have $w_1' X' X w_2 = \lambda_1 w_1' w_2 = 0$. Therefore, $0 - 0 - \gamma_1 w_1 = 0$ and thus $\gamma_1$ must be 0. Then the same argument as for $w_1$ holds, and we choose the $w_2$ to be the eigenvector with the second largest eigenvalue. Using a similar argument repeatedly for the remaining loading vectors, we can see that we can obtain all of the loading vectors at once by taking the first $r$ eigenvectors of $X'X$ sorted by eigenvalue.

With this observation that the loading vectors from PCA are simply the first $r$ eigenvectors of $X'X$, we have another convenient way of computing the loading vectors using singular value decomposition. Letting $UDV' = X$ be the singular value decomposition of $X$, we see that $X'X =$

$VDU'UDV'$. Since $U'U = I$, we get that $X'X = VD^2V'$, and thus the eigenvectors of $X'X$ are $V$. This means we can get the loading vectors directly from the right singular vectors.

Notably, it turns out that maximizing the variance as done above is equivalent to minimizing reconstruction error (by projecting the principal components back on to the loading vectors) in the Frobenius norm sense. If we let $W = [w_1, \dots, w_r]$, then the principal components are $Z = XW$, and to project these principal components back onto the loading vectors, we compute $ZW' = XWW'$. Therefore, we want to show that the optimization problem given by

$$\underset{W:W'W=I}{\arg\min} ||X - XWW'||_F^2$$

is equivalent to that of maximizing variance of each of the principal components. We can see this, as

$$\begin{aligned}
||X - XWW'||_F^2 &= \text{tr}((X - XWW')'(X - XWW')) \\
&= \text{tr}(X'X - WW'X'X - X'XWW' + WW'X'XWW') \\
&= \text{tr}(X'X) - \text{tr}(WW'X'X) - \text{tr}(X'XWW') + \text{tr}(WW'X'XWW') \\
&= \text{tr}(X'X) - \text{tr}(W'X'XW) - \text{tr}(W'X'XW) + \text{tr}(W'X'XWW'W) \\
&= \text{tr}(X'X) - \text{tr}(W'X'XW) - \text{tr}(W'X'XW) + \text{tr}(W'X'XW) \\
&= \text{tr}(X'X) - \text{tr}(W'X'XW)
\end{aligned}$$

and thus minimizing $||X - XWW'||_F^2$ is equivalent to maximizing $\text{tr}(W'X'XW)$, which is done component-wise (for each $w_i$) in the maximum variance formulation.

From these various formulations, we can see PCA gives us loading vectors that likely include all features in $X$ with nonzero weight. However, for increased interpretability of the principal components, it would be useful to only use a subset of features in each principal component, meaning having some 0 entries in the loading vectors. This is what sparse PCA tries to accomplish by adding an $L_1$ penalty on each $w_i$. In this tutorial, we describe two such approaches to this problem.

The rest of the tutorial is organized as follows. In sections 2 and 3, we walk through derivations of the methods of Zou, Hastie, and Tibshirani in 2006 and Witten, Hastie, and Tibshirani in 2010, respectively. In section 4, we provide an implementation in R.

## 2 Sparse PCA as penalized reduced-rank regression

In this section, we cover the method of sparse PCA as proposed by Zou, Hastie, and Tibshirani in 2006, and the arguments presented here follow closely from the original text but with additional details. First, we introduce how regular PCA can be framed as a regression problem in the case when $n > p$ and then when $p > n$. Then we cover the extension to sparse PCA by adding a penalty term that induces sparsity in the loading vectors.

### 2.1 Regular PCA ($n > p$) as reduced-rank regression

Consider the following multivariate regression problem:

$$X = XBA' + E$$

where $B, A \in \mathbb{R}^{p\times r}$ and $E \in \mathbb{R}^{n\times p}$. Here, $BA'$ is a rank-$r$ matrix of regression coefficients and $E$ is an error matrix. To minimize the errors, we want the following:

$$\underset{A,B:\,A'A=I_r}{\arg\min}\ ||X - XBA'||_F^2$$

This is very similar to the formulation of PCA that involves minimizing the reconstruction error. However, the difference here is that we have two separate matrices, $B$ and $A$, where the columns of $B$ are related to the loadings matrix and the columns of $A$ are the new basis vectors. Having two separate matrices gives us greater flexibility in later sections, but for now, we show that in the case where $n > p$, it turns out that $B = A$, and as a result the solution to this optimization gives us regular PCA.

Suppose $n > p$. First, we find the optimal $B$ by treating $A$ as fixed. Since $A \in \mathbb{R}^{p\times r}$ and $A'A = I_r$, we can construct a matrix $A_\perp \in \mathbb{R}^{p\times(p-r)}$ such that $[A, A_\perp]'[A, A_\perp] = [A, A_\perp][A, A_\perp]' = I_p$ (i.e. it is orthogonal). Then since for any matrix $U$ and orthogonal matrix $V$, we have

$$||UV||_F^2 = \text{tr}(V'U'UV) = \text{tr}(U'UVV') = \text{tr}(U'U) = ||U||_F^2,$$

we get that

$$\begin{aligned}||X - XBA'||_F^2 &= ||(X - XBA')\,[A, A_\perp]\,||_F^2\\ &= ||X\,[A, A_\perp] - XBA'\,[A, A_\perp]\,||_F^2\\ &= ||\,[XA, XA_\perp] - [XB, 0]\,||_F^2\\ &= ||XA - XB||_F^2 + ||XA_\perp - 0||_F^2\\ &= ||XA - XB||_F^2 + ||XA_\perp||_F^2.\end{aligned}$$

Taking the gradient with respect to $B$ and setting it to 0, we get

$$\begin{aligned}&-X'(XA - XB) = 0\\ &\implies X'XA = X'XB\\ &\implies B = (X'X)^{-1}X'XA\\ &\implies \hat{B} = A\end{aligned}$$

since $X'X$ is of full rank. Therefore with $B = \hat{B} = A$ fixed, the minimization problem is the same as minimizing reconstruction error in regular PCA.

### 2.2 Regular PCA ($p > n$) as reduced-rank regression with ridge penalty

At the end of the last section, we relied on the fact that $X'X$ is of full rank. However, when $p > n$, this is no longer the case. A common solution to this is adding a penalty, and since we needed this when solving for $B$ with $A$ fixed, we add a ridge penalty to the columns of $B$. In this section, we show that we can still recover the regular PCA solution with this method.

Now we want to solve the optimization problem

$$\underset{A,B:\,A'A=I_r}{\arg\min}\ ||X - XBA'||_F^2 + \lambda||B||_F^2$$

where $\lambda$ is any value greater than 0.

As in the last section, we first find the optimal $B$ with $A$ fixed. With $A$ fixed, we have

$$C_\lambda(A,B) = ||X - XBA'||_F^2 + \lambda||B||_F^2 = ||XA - XB||_F^2 + ||XA_\perp||_F^2 + \lambda||B||_F^2.$$

based on the same argument as the last section. Therefore, taking the gradient with respect to $B$ and setting it to 0, we get

$$\begin{aligned}
-X'(XA - XB) + \lambda B &= 0 \\
\implies X'XA &= (X'X + \lambda I_p)B \\
\implies \hat{B} &= (X'X + \lambda I_p)^{-1}X'XA.
\end{aligned}$$

Note that from this, we also get the following

$$\begin{aligned}
-X'(XA - XB) + \lambda B &= 0 \\
\implies \lambda B &= X'(XA - XB) \\
\implies \lambda B'B &= B'X'(XA - XB) \\
\implies \mathrm{tr}(\lambda B'B) &= \mathrm{tr}(B'X'(XA - XB)) \\
\implies \lambda||\hat{B}||_F^2 &= \mathrm{tr}(\hat{B}'X'(XA - X\hat{B})).
\end{aligned}$$

This is useful, since with $B = \hat{B}$ fixed, we can write our partially optimized objective function as follows:

$$\begin{aligned}
C_\lambda(A,\hat{B}) &= ||X - X\hat{B}A'||_F^2 + \lambda||\hat{B}||_F^2 \\
&= \mathrm{tr}((X - X\hat{B}A')'(X - X\hat{B}A')) + \mathrm{tr}(\hat{B}'X'(XA - X\hat{B})) \\
&= \mathrm{tr}(X'X) - \mathrm{tr}(X'X\hat{B}A') - \mathrm{tr}(A\hat{B}'X'X) + \mathrm{tr}(A\hat{B}'X'X\hat{B}A') + \mathrm{tr}(\hat{B}'X'(XA - X\hat{B})) \\
&= \mathrm{tr}(X'X) - \mathrm{tr}(A'X'X\hat{B}) - \mathrm{tr}(\hat{B}'X'XA) + \mathrm{tr}(\hat{B}'X'X\hat{B}A'A) + \mathrm{tr}(\hat{B}'X'XA) - \mathrm{tr}(\hat{B}'X'X\hat{B}) \\
&= \mathrm{tr}(X'X) - \mathrm{tr}(A'X'X\hat{B}) \\
&= \mathrm{tr}(X'X) - \mathrm{tr}(A'X'X(X'X + \lambda I_p)^{-1}X'XA)
\end{aligned}$$

Therefore, minimizing $C_\lambda(A,\hat{B})$ is equivalent to maximizing $\mathrm{tr}(A'X'X(X'X + \lambda I_p)^{-1}X'XA)$, still with the constraint that $A'A = I_r$. With a similar argument to that of maximizing the variance in regular PCA, we can see that $\hat{A} = \mathrm{eig}_r(X'X(X'X + \lambda I_p)^{-1}X'X)$ maximizes this, where $\mathrm{eig}_r(\cdot)$ denotes a matrix with the columns containing the first $r$ eigenvectors (sorted by eigenvalue) of the input matrix.

Now we show that these results can give us the regular PCA result. Letting $UDV' = X$ be the singular value decomposition of $X$, we have that

$$\begin{aligned}
X'X(X'X + \lambda I_p)^{-1}X'X &= VD^2V'(VD^2V' + \lambda I_p)^{-1}VD^2V' \\
&= VD^2V'(VD^2V' + V\lambda I_pV')^{-1}VD^2V' \\
&= VD^2V'(V(D^2 + \lambda I_p)V')^{-1}VD^2V' \\
&= VD^2V'V(D^2 + \lambda I_p)^{-1}V'VD^2V' \\
&= VD^2(D^2 + \lambda I_p)^{-1}D^2V'.
\end{aligned}$$

Therefore, $\hat{A} = \text{eig}_r(X'X(X'X + \lambda I_p)^{-1}X'X) = V_{\cdot,1:r}$. Additionally, we have

$$\begin{aligned}\hat{B} &= (X'X + \lambda I_p)^{-1}X'XA \\ &= (VD^2V' + \lambda I_p)^{-1}VD^2V'A \\ &= V(D^2 + \lambda I_p)^{-1}V'VD^2V'A \\ &= V(D^2 + \lambda I_p)^{-1}D^2V'A\end{aligned}$$

and with $A = \hat{A} = V_{\cdot,1:r}$, we have that $\hat{B} = V_{\cdot,1:r}\left[(D^2 + \lambda I_p)^{-1}D^2\right]_{1:r,1:r}$. This can be seen as $V'A = V'V_{\cdot,1:r}$ is a tall matrix with 1 on the diagonal and 0s elsewhere, effectively subsetting columns, and $(D^2 + \lambda I_p)^{-1}D^2$ is a diagonal matrix which scales the columns of V. Most importantly, this means that $\hat{B}$ is simply a scaled version of the regular loading vectors. Since the regular loading vectors are unit vectors, we can recover them with $w_i = \frac{B_{\cdot,i}}{||B_{\cdot,i}||_2}$.

In conclusion, we have shown that even when $p > n$, regular PCA can be formulated as a regression problem. Note that the results shown here are true for all $\lambda \geq 0$, as long as it results in an invertible matrix. Therefore, the results in the previous section are just a special case of these results, when $\lambda = 0$.

## 2.3 Sparse PCA as reduced-rank regression with elastic net penalty

In the last section, we showed how adding a ridge penalty to $B$ allows us to recover regular PCA with a regression problem even when $p > n$. With this penalized regression formulation, we can add an additional lasso penalty to $B$ that induces sparsity. Specifically, the goal is to solve the following optimization problem:

$$\underset{A,\,B\colon A'A = I_r}{\arg\min}\ ||X - XBA'||_F^2 + \lambda||B||_F^2 + ||B\,\text{diag}(\lambda_{1,1}, \ldots, \lambda_{1,r})||_1$$

where $\lambda_{1,1}, \ldots, \lambda_{1,r}$ are tuning parameters to control the degree of sparsity in each loading vector separately.

Once again, we start with treating $A$ fixed and compute the optimal $B$. We have that

$$\begin{aligned}C_{\lambda,\lambda_1}(A, B) &= ||X - XBA'||_F^2 + \lambda||B||_F^2 + ||B\,\text{diag}(\lambda_{1,1}, \ldots, \lambda_{1,r})||_1 \\ &= ||XA - XB||_F^2 + ||XA_\perp||_F^2 + \lambda||B||_F^2 + ||B\,\text{diag}(\lambda_{1,1}, \ldots, \lambda_{1,r})||_1.\end{aligned}$$

The terms with $B$ can be rewritten as

$$\sum_{i=1}^{r} ||XA_{\cdot,i} - XB_{\cdot,i}||_2^2 + \lambda||B_{\cdot,i}||_2^2 + \lambda_{1,r}||B_{\cdot,i}||_1$$

and thus solving this is equivalent to solving $r$ independent elastic net regression problems, for which we can use existing fast software such as `glmnet`.

With $B$ fixed, we want to minimize $||X - XBA'||_F^2$. Since

$$\begin{aligned}||X - XBA'||_F^2 &= \text{tr}((X - XBA')'(X - XBA')) \\ &= \text{tr}(X'X) - 2\,\text{tr}(X'XBA') + \text{tr}(AB'X'XBA') \\ &= \text{tr}(X'X) - 2\,\text{tr}(X'XBA') + \text{tr}(B'X'XB),\end{aligned}$$

this is equivalent to maximizing $\text{tr}(X'XBA')$, still subject to $A'A = I_r$. If we let $UDV' = X'XB$ be the singular value decomposition of $X'XB$, we get

$$\text{tr}(X'XBA') = \text{tr}(UDV'A') = \text{tr}(V'A'UD).$$

Since $D$ is diagonal with nonnegative elements, maximizing the trace of the product is equivalent to maximizing $\text{tr}(V'A'U)$. By Cauchy-Schwartz inequality, we have that $\text{tr}(V'A'U) \leq \sqrt{\text{tr}(V'A'AV)}\sqrt{\text{tr}(U'U)}$, with equality when $AV = U$. Therefore, it is maximized when $AV = U$, or equivalently $A = UV'$.

Therefore, with these two update steps, we can update $B$ with $A$ fixed and then update $A$ with $B$ fixed, until convergence. We can then get the unit loading vectors $w_i = \frac{B_{\cdot,i}}{||B_{\cdot,i}||_2}$, which are now sparse due to the $L_1$ penalty. However, the solution depends on the initial choice of $A$, so the authors recommend initializing this at the loading vectors from regular PCA.

## 3 Sparse PCA as penalized matrix decomposition

In this section, we cover the method of sparse PCA as proposed by Witten, Hastie, and Tibshirani in 2010, and once again, the arguments presented here follow closely from the original text but with additional details.

This method is primarily motivated by the relationship between singular value decomposition and regular PCA. It is a well known result by Eckart and Young about the best low-rank approximation to a matrix that

$$\underset{\hat{X}\in M(r)}{\arg\min}\, ||X - \hat{X}||_F^2 = U_{\cdot,1:r} D_{1:r,1:r} V'_{\cdot,1:r}$$

where $M(r)$ denotes the set of matrices of rank $r$ and $UDV' = X$ is the singular value decomposition of $X$. In this paper, the authors proposed a new low-rank matrix decomposition that approximates the original matrix, while penalizing the elements in $U$ and $V$ with various penalties. They showed many applications of this decomposition, one of which was sparse PCA. Since the columns of $V_{\cdot,1:r}$ are also the loading vectors in regular PCA, they add an $L_1$ penalty to each column of $V$ to get sparse loading vectors.

### 3.1 Setup with one principal component

Starting with a rank-1 approximation to $X$ (to obtain the first loading vector), we have the following optimization problem:

$$\underset{d,u,v}{\arg\min}\, ||X - duv'||_F^2 \quad \text{subject to} \quad u'u = 1, \quad v'v = 1, \quad ||v||_1 \leq c, \quad d \geq 0.$$

For any orthogonal matrices $U, V \in \mathbb{R}^{n\times r}$ and diagonal matrix $D \in \mathbb{R}^{r\times r}$, we have that

$$\begin{aligned}
||X - UDV'||_F^2 &= \operatorname{tr}((X - UDV')'(X - UDV')) \\
&= \operatorname{tr}(X'X) - \operatorname{tr}(X'UDV') - \operatorname{tr}(VDU'X) + \operatorname{tr}(VDU'UDV') \\
&= \operatorname{tr}(X'X) - 2\operatorname{tr}(DU'XV) + \operatorname{tr}(D^2V'V) \\
&= \operatorname{tr}(X'X) - 2\operatorname{tr}(DU'XV) + \operatorname{tr}(D^2) \\
&= \operatorname{tr}(X'X) - 2\sum_{i=1}^{r} D_{i,i}U'_{\cdot,i}XV_{\cdot,i} + \sum_{i=1}^{r} D_{i,i}^2.
\end{aligned}$$

Therefore, for the rank-1 approximation ($r = 1$), the above optimization problem is the same as

$$\arg\min_{d,u,v} -2du'Xv + d^2 \quad \text{subject to} \quad u'u = 1, \quad v'v = 1, \quad ||v||_1 \le c, \quad d \ge 0.$$

The $u$ and $v$ that minimize this must the solution to the following:

$$\arg\max_{u,v} u'Xv \quad \text{subject to} \quad u'u = 1, \quad v'v = 1, \quad ||v||_1 \le c$$

and taking the derivative with respect to $d$ and setting it to 0, we see the optimal $d = u'Xv$. Therefore, to obtain the first sparse loading vector, we solve this optimization problem. With $u$ fixed, we have

$$\arg\max_{v} u'Xv \quad \text{subject to} \quad v'v = 1, \quad ||v||_1 \le c,$$

which is linear in $v$, but not convex due to the quadratic equality constraint. However, to get around this, the authors change this to an inequality constraint and note that as long as $c$ is chosen such that

$$\arg\max_{v} u'Xv \quad \text{subject to} \quad ||v||_1 \le c$$

has $L_2$ norm greater than 1, the solutions are equivalent. Similarly, with $u$ fixed and the analogous change, the optimization problem is convex. Therefore, we solve

$$\arg\max_{u,v} u'Xv \quad \text{subject to} \quad u'u \le 1, \quad v'v \le 1, \quad ||v||_1 \le c,$$

which is biconvex in $u$ and $v$, and this suggests an iterative algorithm for solving this problem.

With $v$ fixed, we want to solve

$$\arg\min_{u} -u'Xv \quad \text{subject to} \quad u'u \le 1.$$

The Lagrangian for this is $-u'Xv + \lambda u'u$, so the KKT conditions consist of

$$-Xv + \lambda u = 0, \quad \lambda(u'u - 1) = 0.$$

Therefore, $u = \frac{Xv}{\lambda}$ and since $\lambda = 0$ does not work, $\lambda$ must be chosen to satisfy $u'u = 1$ and thus $u = \frac{Xv}{||Xv||_2}$.

With $u$ fixed, we want to solve

$$\arg\min_{v} -u'Xv \quad \text{subject to} \quad v'v \le 1, \quad ||v||_1 \le c.$$

The Lagrangian for this is $-u'Xv + \lambda v'v + \Delta||v||_1$, so the KKT conditions consist of

$$-X'u + 2\lambda v + \Delta\Gamma = 0, \quad \lambda(v'v-1) = 0, \quad \Delta(||v||_1 - 1) = 0$$

where $\Gamma_i = \text{sgn}(v_i)$ if $|v_i| > 0$ or $\Gamma_i \in [-1,1]$ if $v_i = 0$. Therefore $X'u = 2\lambda v + \Delta\Gamma$. To find $v$ that satisfies these first-order conditions, we consider cases on each element of $X'u$. First suppose $\frac{[X'u]_i - \Delta}{2\lambda} > 0$. Then set $v_i = \frac{[X'u]_i - \Delta}{2\lambda}$. Then since $v_i > 0$ by hypothesis, we have $\Gamma_i = 1$. Therefore we see that $v_i$ satisfies the first condition, since

$$-\left[X'u\right]_i + 2\lambda\frac{[X'u]_i - \Delta}{2\lambda} + \Delta * 1 = 0.$$

Next suppose $\frac{[X'u]_i + \Delta}{2\lambda} < 0$. Then set $v_i = \frac{[X'u]_i + \Delta}{2\lambda}$. Then since $v_i < 0$ by hypothesis, we have $\Gamma_i = -1$. Therefore we see that $v_i$ satisfies the first condition, since

$$-\left[X'u\right]_i + 2\lambda\frac{[X'u]_i + \Delta}{2\lambda} + \Delta * (-1) = 0.$$

Finally, suppose $|[X'u]_i| < \Delta$. Then set $v_i = 0$. Therefore, since $\frac{[X'u]_i}{\Delta} \in [-1,1]$, we have that the first condition is satisfied:

$$-\left[X'u\right]_i + 2\lambda * 0 + \Delta\frac{[X'u]_i}{\Delta} = 0.$$

In summary, we have

$$v_i = \begin{cases} \frac{[X'u]_i - \Delta}{2\lambda} & \text{if } \frac{[X'u]_i - \Delta}{2\lambda} > 0, \\ \frac{[X'u]_i + \Delta}{2\lambda} & \text{if } \frac{[X'u]_i + \Delta}{2\lambda} < 0, \\ 0 & \text{if } |[X'u]_i| < \Delta. \end{cases}$$

This can also be written in terms of the soft-thresholding operator $S$, where $S(a,\Delta) = \text{sgn}(a)(|a| - \Delta)_+$. Then $v = \frac{S(X'u,\Delta)}{2\lambda}$. Now $\lambda$ and $\Delta$ have to be chosen to satisfy the inequality constraints. Since $\lambda = 0$ does not work, we set $\lambda = \frac{||S(X'u,\Delta)||_2}{2}$ to get $v = \frac{S(X'u,\Delta)}{||S(X'u,\Delta)||_2}$. If $\Delta = 0$ results in $||v||_1 \leq c$, then $\Delta = 0$, otherwise $\Delta$ can be found such that $||v||_1 = c$ with binary search.

Therefore, with these two update steps, we can update $u$ with $v$ fixed, update $v$ with $u$ fixed, and then set $d = u'Xv$. However, the solution to this depends on the initial $v$ chosen, so the authors recommend initializing $v$ to be the normal first singular vector of $X$. At the end of this iteration, our final $v$ is now the first sparse loading vector.

## 3.2 Extension to multiple sparse principal components

The first extension to computing multiple loading vectors proposed in the paper is quite simple. After obtaining $d_1$, $u_1$, and $v_1$ using the approach in the last section, the same approach can be repeated on the matrix $X - d_1u_1v_1'$ to obtain $d_2$, $u_2$, and $v_2$, and so on until a total of $r$ loading vectors are obtained.

### 3.3 Extension to multiple sparse principal components with orthogonal $U$

The alternative extension to multiple loading vectors is based on constraining $u_k$ to be orthogonal to $u_1, \ldots, u_{k-1}$. While this does not give orthogonal loading vectors, as in regular PCA, since each $u_i$ is associated with a corresponding $v_i$, it should at least make the $v_i$ somewhat unrelated. Specifically, for $k > 1$, if we let $U_{k-1} = [u_1, \ldots, u_{k-1}]$, we want to solve the following optimization problem in this setup:

$$\arg\max_{u_k, v_k} u_k' X v_k \quad \text{subject to} \quad u_k' u_k = 1, \quad v_k' v_k = 1, \quad ||v_k||_1 \leq c, \quad U_{k-1}' u_k = 0$$

If we let $U_{k-1}^{\perp}$ be a matrix consisting of basis vectors orthogonal to $U_{k-1}$, we want that $u_k$ is in the column space of $U_{k-1}^{\perp}$. In other words, we want that $u_k = U_{k-1}^{\perp}\theta$ for some $\theta$. Since $U_{k-1}^{\perp}$ is an orthogonal matrix, we have that $u_i' u_i = (U_{k-1}^{\perp}\theta)'(U_{k-1}^{\perp}\theta) = \theta' U_{k-1}^{\perp'} U_{k-1}^{\perp}\theta = \theta'\theta$. Therefore, the constraint that $\theta'\theta \leq 1$ is equivalent to $u_i' u_i \leq 1$. Therefore, with $v_k$ fixed, we can solve the following:

$$\arg\max_{\theta} \theta' U_{k-1}^{\perp'} X v_k \quad \text{subject to} \quad \theta'\theta \leq 1$$

By an identical argument to that of the update for $u$ with $v$ fixed, we can see that the optimal $\theta = \frac{U_{k-1}^{\perp'} X v_k}{||U_{k-1}^{\perp'} X v_k||_2}$, and thus $u_k = \frac{U_{k-1}^{\perp} U_{k-1}^{\perp'} X v_k}{||U_{k-1}^{\perp'} X v_k||_2} = \frac{U_{k-1}^{\perp} U_{k-1}^{\perp'} X v_k}{||U_{k-1}^{\perp} U_{k-1}^{\perp'} X v_k||_2}$. It is not immediately apparent how to compute this, as we need a specific orthogonal basis. However, $U_{k-1}^{\perp} U_{k-1}^{\perp'}$ is a projection matrix onto the subspace orthogonal to the column space of $U$. One could prove this directly, or note the connection to the least squares projection matrix. First, note that $U_{k-1}^{\perp'} U_{k-1}^{\perp} = I_r$. Therefore, we have

$$U_{k-1}^{\perp} U_{k-1}^{\perp'} X v_k = U_{k-1}^{\perp} (U_{k-1}^{\perp'} U_{k-1}^{\perp})^{-1} U_{k-1}^{\perp'} X v_k.$$

Since it is an orthogonal projection, we have the following:

$$U_{k-1}^{\perp} (U_{k-1}^{\perp'} U_{k-1}^{\perp})^{-1} U_{k-1}^{\perp'} = I_r - U_{k-1} (U_{k-1}' U_{k-1})^{-1} U_{k-1}'$$

And thus

$$\begin{aligned} u_k^* &= U_{k-1}^{\perp} U_{k-1}^{\perp'} X v_k \\ &= (I_r - U_{k-1} U_{k-1}') X v_k \end{aligned}$$

This is exactly the result of the residuals of least squares regression of $X v_k$ with $U_{k-1}$ as predictors. Finally, we can get $u_k$ with $u_k = \frac{u_k^*}{||u_k^*||_2}$. Therefore, with this method, we can iteratively compute multiple sparse loading vectors, that have some notion of "independence" between each loading vector estimated.

## 4 Implementations

In this section, we discuss implementation details and provide implementations in R for the two methods of sparse PCA described above.

## 4.1 Sparse PCA as penalized reduced-rank regression

First, we implement a function to update $B$ with $A$ fixed. This involves solving $r$ separate elastic net regression problems, and we use `glmnet` to solve these. However, the objective function that we want to minimize is slightly different than the objective function that `glmnet` solves. We also have a different parameterization of tuning parameters. Specifically, for each $i \in \{1, \ldots, r\}$, we want to solve

$$\arg\min_{B_{\cdot,i}} ||XA_{\cdot,i} - XB_{\cdot,i}||_2^2 + \lambda||B_{\cdot,i}||_2^2 + \lambda_{1,i}||B_{\cdot,i}||_1$$

with some fixed $\lambda$ and tuning parameter $\lambda_{1,i}$ . However, `glmnet` would solve the following objective function (with `intercept = FALSE` and `standardize = FALSE`):

$$\arg\min_{B_{\cdot,i}} \frac{1}{2N}||XA_{\cdot,i} - XB_{\cdot,i}||_2^2 + \lambda_g\left(\frac{1-\alpha}{2}||B_{\cdot,i}||_2^2 + \alpha||B_{\cdot,i}||_1\right)$$

where $N$ is the length of $XA_{\cdot,i}$ and $\lambda_g$ and $\alpha$ are different tuning parameters. We can solve for $\lambda_g$ and $\alpha$ in terms of $\lambda$ and $\lambda_{1,i}$ and then use these in `glmnet` to solve our desired minimization problem. We can see that we have the following correspondence:

$$\lambda = 2N\lambda_g\frac{1-\alpha}{2} = N\lambda_g(1-\alpha), \quad \lambda_{1,i} = 2N\lambda_g\alpha.$$

We can solve in terms of $\lambda_g$ and $\alpha$ to get

$$\lambda_g = \frac{\lambda_{1,i} + 2\lambda}{2N}, \quad \alpha = \frac{\lambda_{1,i}}{\lambda_{1,i} + 2\lambda}.$$

With this, we write a function that takes in these values of $\lambda_g$ and $\alpha$. We do the conversion outside of this function, so that conversion only has to be done once. It also takes in our data matrix $X$ and the current iterate of $A$. It fits the elastic net problems and returns the new iterate of $B$.

```r
update_B = function(X, A, alpha, lambda_g) {

  B = A

  for (i in 1:ncol(B)) {

    B[, i] = coef(glmnet::glmnet(x = X, y = X %*% A[, i],
                                 alpha = alpha[i], lambda = lambda_g[i],
                                 intercept = FALSE, standardize = FALSE))[-1]

  }

  return(B)

}
```

Next, we implement a function to update $A$ with a fixed $B$. This function takes in $X'X$ and the current iterate of $B$ and returns the new iterate of $A = UV'$ where $UDV' = X'XB$ is the singular value decomposition of $X'XB$.

```r
update_A = function(XtX, B) {

  svd_A = svd(XtX %*% B)
  A = tcrossprod(svd_A$u, svd_A$v)

  return(A)

}
```

We then implement a function that normalizes the columns of $B$ to give us the final loading vectors, where the $i$th column of the matrix it returns is $\frac{B_{\cdot,i}}{||B_{\cdot,i}||_2}$.

```r
normalize_B = function(B) {

  norm = sqrt(crossprod(rep(1, nrow(B)), B^2))

  B %*% diag(c(1/ifelse(norm == 0, 1, norm)))

}
```

To identify when to stop the algorithm, we implement a function that takes in two estimates of $B$ and determines if they are sufficiently close. Since the final loading vectors are what we are concerned with, we first obtain the normalized loading vectors, and get the max absolute difference.

```r
check_convergence = function(B_1, B_2) {

  max(abs(normalize_B(B_1) - normalize_B(B_2)))

}
```

Finally, we implement the main function that uses all the pieces we've implemented so far. This function takes in our data matrix, a scalar value $\lambda$, a vector of length $r$ of each $\lambda_{1,i}$, as well as a tolerance value and max number of iterations. This function first converts our values of $\lambda$ and $\lambda_{1,i}$ into $\lambda_g$ and $\alpha$ for use in `glmnet`. Then we initialize $A$ to the regular PCA loading vectors, and update $B$ with the initial $A$. Until convergence, we keep updating $A$ and $B$, and the final result is the sparse loading vectors.

```r
#' @export
sPCA_Zou = function(X, lambda, lambda_1, gram, tolerance = 0.001, max_iter = 200) {

  r = length(lambda_1)

  alpha = lambda_1 / (lambda_1 + 2 * lambda)
  lambda_g = (lambda_1 + 2 * lambda)/(2 * nrow(X))

  A_old = svd(X)$v[, 1:r, drop = FALSE]
  B_old = update_B(X, A_old, alpha, lambda_g)

  XtX = crossprod(X)

  iter = 0
  diff = tolerance * 10

  while((iter < max_iter) & (diff > tolerance)) {

    A_new = update_A(XtX, B_old)
    B_new = update_B(X, A_new, alpha, lambda_g)

    diff = check_convergence(B_new, B_old)
    A_old = A_new
    B_old = B_new
    iter = iter + 1

  }

  return(normalize_B(B_new))

}
```

## 4.2 Sparse PCA as penalized matrix decomposition

First, we implement a function for the soft-thresholding operator, $S(a, \Delta) = \text{sgn}(a)(|a| - \Delta)_+$.

```
soft_threshold = function(a, delta) {

  pmax(0, abs(a) - delta) * sign(a)

}
```

Next, we implement a function that takes in a vector $a$ and the amount to soft-threshold, $\Delta$. It then performs the soft-thresholding, normalizes it to a unit vector, and then returns the $L_1$ norm.

```
compute_l1_norm = function(a, delta) {

  thresholded = soft_threshold(a, delta)

  normed = thresholded / sqrt(sum(thresholded ^ 2))

  return(sum(abs(normed)))

}
```

Now we implement binary search to choose $\Delta$ such that the soft-thresholded and normalized vector has max $L_1$ norm of $c$. We do this by searching for values between 0 and the maximum absolute value in the input vector.

```
binary_search = function(a, c) {

  max_iter = 100
  tolerance = 1e-6

  left = 0
  right = abs(max(a))

  if (compute_l1_norm(a, 0) < c) {

    delta = 0

  } else {

    iter = 1

    while((iter < max_iter) & (right - left) > tolerance) {

      mid = (left + right) / 2
      l1_norm = compute_l1_norm(a, mid)

      if (l1_norm > c) {
        left = mid
      } else {
        right = mid
      }

    }

    delta = (left + right) / 2

  }

  return(delta)

}
```

Next, we implement the update step for $u$, with $v$ fixed, where $u = \frac{Xv}{||Xv||_2}$.

```
update_u = function(X, v) {

  Xv = X %*% v
  norm = sqrt(sum(Xv ^ 2))

  u = Xv / norm

  return(u)

}
```

We also implement the update step for $u$, when we constraint the $u_i$ to be orthogonal to each other. Here, $u = \frac{u^*}{||u^*||_2}$, where $u^* = (I_r - U_{k-1}U'_{k-1})Xv$.

```
update_orthogonal_u = function(X, v, U_prev) {

  Xv = X %*% v

  u_star = Xv - tcrossprod(U_prev) %*% Xv

  u = u_star / sqrt(sum(u_star ^ 2))

  return(u)

}
```

Next we implement the update for $v$ with $u$ fixed, where $v = \frac{S(X'u,\Delta)}{||S(X'u,\Delta)||_2}$. This is where we use the binary search function to find a $\Delta$ that satisfies the constraints of $L_1$ norm being less than $c$.

```
update_v = function(X, u, c) {

  Xtu = crossprod(X, u)

  delta = binary_search(Xtu, c)

  v = soft_threshold(Xtu, delta)

  v = v / sqrt(sum(v ^ 2))

  return(v)

}
```

Combining the previous functions, we now write a function to compute a single loading vector. This consists of iterating until convergence, where we define convergence by the max absolute difference in the estimated $v$ vectors between iterations.

```
compute_single_loading = function(X, c, k, V_init, U_result, orth, tolerance, max_iter
    ) {

  v_old = V_init[, k, drop = TRUE]

  diff = tolerance * 10
  iter = 1

  while((iter < max_iter) & (diff > tolerance)) {

    if (orth & (k > 1)) {
      u_new = update_orthogonal_u(X, v_old, U_result[, 1:(k - 1), drop = FALSE])
    } else {
      u_new = update_u(X, v_old)
    }
    v_new = update_v(X, u_new, c)
    d = crossprod(u_new, X) %*% v_new

    diff = max(abs(v_new - v_old))
    v_old = v_new
    iter = iter + 1

  }

  return(list(u = u_new, d = d, v = v_new))

}
```

Finally, we can wrap this all into one function that computes all of the loading vectors.

```
#' @export
sPCA_Witten = function(X, r, c, orth, tolerance = 1e-7, max_iter = 20) {

  V_init = svd(X)$v[, 1:r, drop = FALSE]

  U_result = matrix(nrow = nrow(X), ncol = r)
  V_result = matrix(nrow = ncol(X), ncol = r)
  d_result = numeric(r)

  for (k in 1:r) {

    if (!orth & k > 1) {
      X = X - result$u %*% result$d %*% t(result$v)
    }

    result = compute_single_loading(X, c, k, V_init, U_result, orth, tolerance,
        max_iter)

    U_result[, k] = result$u
    V_result[, k] = result$v
    d_result[k] = result$d

  }

  return(V_result)

}
```