pcaL1: An Implementation in R of Three Methods for L1-Norm Principal Component Analysis

J. Paul Brooks
Virginia Commonwealth University

Sapan Jot
Virginia Commonwealth University

Abstract

pcaL1 is a package for the R environment for finding principal components using methods based on the L1 norm. The principal components derived using traditional principal component analysis (PCA) can be interpreted as an optimal solution to several optimization problems involving the L2 norm. Using the L1 norm in these problems provides an alternative that is more robust to outlier observations in moderate-sized datasets. Replacing the L2 norm with the L1 norm in the different optimization problems yields different principal components. The package **pcaL1** implements three algorithms: PCA-L1, L1-PCA, and L1-PCA*. Results are presented for test datasets that indicate the conditions under which each method performs best and the computation time required by each implementation.

Keywords: R, L1 norm, principal component analysis.

1. Introduction

Principal component analysis (PCA) has a variety of uses including dimensionality reduction and has been applied to problems in computer vision, face reconstruction, and chemometrics. As shown by Jolliffe (2002), calculating principal components can be viewed as finding an optimal solution to several different optimization problems as well as a decomposition of the covariance matrix into its eigenvectors and eigenvalues. PCA can be viewed as finding successive orthogonal directions of maximum variation in data, finding successive orthogonal directions of minimum variation in data, and finding a subspace that minimizes the sum-of-squared distances of points to their projections on the subspace.

Variants of traditional L2-norm PCA (L2-PCA) can be derived by altering the optimization problems or the covariance matrix. The L1 norm is often used in order to reduce the influence of outlier observations on the results. Each of the interpretations of L2-PCA can be adapted to incorporate the L1 norm; however, the results when using the L1 norm do not all coincide as they do for L2-PCA. Therefore, there are many L1-norm PCA methods possible. This paper describes the implementation in R (**?**) of three L1-norm PCA methods that all have their basis in a geometric interpretation of PCA.

Previous related work includes an R implementation of a projection pursuit PCA method by Filzmozer, Fritz, and Kalcher (2011) in a package called **pcaPP**. The concept of projection pursuit has been applied to finding successive directions of maximum dispersion in data, where dispersion can be measured using the L1 norm or an approximation (Choulakian 2006;

Maronna 2005; Croux and Ruiz-Gazen 2005). The implementation in **pcaPP** is described in detail by Croux, Filzmoser, and Oliveira (2007).

In this paper, we describe the implementation of three methods for L1-norm PCA as an R package called **pcaL1**. The methods that we implement are PCA-L1 (Kwak 2008), L1-PCA (Ke and Kanade 2003, 2005), and L1-PCA* (Brooks, Dulá, and Boone 2012). PCA-L1 is a method for finding successive directions of maximum dispersion in data based on approximating successive, orthogonal L1-norm best-fit lines. L1-PCA is a method for estimating the L1-norm best-fit subspace. L1-PCA* is a method for finding successive, orthogonal L1-norm best-fit hyperplanes that contain the origin. In each interpretation, the L1 norm is more resistant against the magnifying effects of outliers than the traditional L2 norm.

The remainder of the paper is organized as follows. In the next section, notation is introduced. Section 3 describes three L1-norm PCA methods using the common notation, presents an analysis of their computational complexity, and describes their implementation in the R package **pcaL1**. Section 4 contains computational results on simulated and real-world datasets that give insight into the relative strengths of each method, the empirical computational complexity, and the benefits of an R implementation.

2. Notation and definitions

Boldface uppercase letters represent matrices, and boldface lowercase letters represent vectors. A dataset is represented by an $n \times m$ matrix $\boldsymbol{X}$ with one row for each observation, or by a set of observations $\boldsymbol{x}_i \in \mathbb{R}^m$, $i = 1, \ldots, n$. We assume that the rank of $\boldsymbol{X}$ is $\min\{n, m\}$ and that the data are centered. Let q be the dimension of the subspace into which we wish to project data. The $m \times q$ matrix $\boldsymbol{V}$ corresponds to an *internal representation* (Rockafellar 1970) of the subspace $S = \{\boldsymbol{x} \in \mathbb{R}^m | \boldsymbol{x} = \boldsymbol{V}\boldsymbol{u} \text{ for some } \boldsymbol{u}\}$. The columns of $\boldsymbol{V}$ are denoted $\boldsymbol{v}_j$, $j = 1, \ldots, q$. They span the subspace S and are the *principal components*. The $n \times q$ matrix $\boldsymbol{U}$ is the matrix of *scores*, and the score of observation $\boldsymbol{x}_i$ is $\boldsymbol{u}_i$. The $n \times m$ matrix $\boldsymbol{E}$ is the matrix of *reconstructions*, and is the matrix of projected points in terms of the original coordinates.

3. L1-norm PCA methods

In this section, we describe three L1-norm PCA methods and their implementation in R package **pcaL1** as functions `pcal1`, `l1pca`, and `l1pcastar`. The source code is available at `http://cran.r-project.org/web/packages/pcaL1/`. Each function takes a data matrix or data table and returns principal components. Options for the functions include the specifying the projection dimension, returning scores, and returning the proportion of L_1 dispersion explained by the principal components.

Each R function uses the `.C` interface to call a C implementation. The C programming language is used for its speed and dynamic memory allocation. The implementations of algorithms L1-PCA and L1-PCA* require the solution of linear programs (LPs) as subroutines. All linear programs are solved using **Clp**, a free linear programming solver available in the COIN-OR repository (`http://www.coin-or.org`).

3.1. pcal1

pcal1 is our implementation of PCA-L1 (Kwak 2008), a procedure for finding a local optimum to the problem of finding a direction along which L1 dispersion is maximized. This direction is the first principal component. The optimization problem may be written as

$$\max_{\boldsymbol{v}} \sum_{i=1}^{n} |\boldsymbol{v}^\top \boldsymbol{x}_i|, \tag{1}$$

subject to

$$\boldsymbol{v}^\top \boldsymbol{v} = 1.$$

For a solution $\boldsymbol{v}$, the inner product $(\boldsymbol{v}^\top \boldsymbol{x}_i)\boldsymbol{v}$ gives the projection of $\boldsymbol{x}_i$ onto the direction $\boldsymbol{v}$, and $|\boldsymbol{v}^\top \boldsymbol{x}_i|$ gives the length of the projection.

Principal components $j = 2, \ldots, m$ may be found by finding a local optimum to (1) where the original data have been replaced by the projection into the orthogonal complement of the subspace spanned by principal components $1, \ldots, j-1$. Pseudocode is given in Algorithm 1. Step 1 initializes the data so that on the first pass through the `for` loop in Step 2, the original data is used. Step 3 projects data into the orthogonal complement spanned by principal components $1, \ldots, j-1$.

Step 4 requires an initial guess for the direction of maximum L1 dispersion. Possiblities include setting $\boldsymbol{v}(0) = \arg\max \|\boldsymbol{x}_i^j\|_2$, setting $\boldsymbol{v}(0)$ to be the first principal component from L2-PCA applied to $\boldsymbol{X}^j$, or using a random vector. Steps 6 through 12 conduct a polarity check to establish whether the data vectors are oriented in the same general direction as the current estimate $\boldsymbol{v}(t)$. In Steps 14 and 15, the estimate is calculated based on the current polarity of the points. Steps 18 through 20 prevent the possibility that the algorithm will get stuck at a local minimum.

The complexity of Algorithm 1 is $O(nmq \times T)$, where T is an upper bound on the number of iterations required for convergence in Steps 5 through 22. Kwak (2008) claims that T depends only on n so that computational time does not depend as heavily on the number of variables as on the number of observations.

A flowchart detailing the implementation of pcal1 in **pcaL1** is in Figure 3. The function pcal1 is defined in the script pcal1.R that passes the data to pcal1_R.c where further memory is allocated as needed. The algorithm is implemented in pcal1.c.

A call to pcal1 produces the loadings matrix $\boldsymbol{V}$, comprised of the principal components as columns. One may specify the number of principal components q to calculate, whether to center the data by subtracting the coordinatewise median, and the method for initialization of $\boldsymbol{v}(0)$. Initialization options include using the first L2-PCA principal component, using $\arg\max_{\boldsymbol{x}_i, i=1,\ldots,n} \|x\|_2$, or using a random vector. Options are available to also calculate the scores, the L1 dispersion explained by each component, and the reconstructed points. As the calculation of principal components proceeds, the component number is printed to the screen.

```
R> library("pcaL1")
R> data("USArrests")
R> mypcal1 <- pcal1(USArrests[,c("Murder", "Assault", "Rape")],
              + projDim=2, center=TRUE, scores=TRUE,
              + projPoints=TRUE, dispExp=TRUE, initialize="l2pca")
```

Algorithm 1 PCA-L1

Given an $n \times m$ data matrix $\boldsymbol{X}$.

Set $\boldsymbol{v}_0 = \boldsymbol{0}, \boldsymbol{X}^0 = \boldsymbol{X}$.
for $(j = 1; j \leq q; ++j)$ **do**
  Set $\boldsymbol{X}^j = \boldsymbol{X}^{j-1} - \boldsymbol{v}_{j-1}(\boldsymbol{v}_{j-1}^\top \boldsymbol{X}^{j-1})$.
  Initialize $\boldsymbol{v}(0)$.
  Set $t = 0$
  for $(i = 1; i \leq n; ++i)$ **do**
    if $\boldsymbol{v}(t)^\top \boldsymbol{x}_i^j < 0$ **then**
      $\boldsymbol{p}_i(t) = -1$
    else
      $\boldsymbol{p}_i(t) = 1$.
    end if
  end for
  Set $t = t + 1$.
  Set $\boldsymbol{v}(t) = \sum_{i=1}^n \boldsymbol{p_i}(t-1)\boldsymbol{x}_i^j$.
  Set $\boldsymbol{v}(t) \leftarrow \boldsymbol{v}(t) / \parallel \boldsymbol{v}(t) \parallel_2$.
  if $\boldsymbol{v}(t) \neq \boldsymbol{v}(t-1)$ **then**
    Go to Step 6.
  else if There exists i such that $\boldsymbol{v}^\top(t)\boldsymbol{x}_i^j = 0$ **then**
    Perturb $\boldsymbol{v}(t)$ with a small nonzero random vector ϵ:
    Set $\boldsymbol{v}(t) = (\boldsymbol{v}(t) + \boldsymbol{\epsilon})/\|\boldsymbol{v}(t) + \boldsymbol{\epsilon}\|_2$ and go to Step 6.
  else
    Set $\boldsymbol{v}_j = \boldsymbol{v}(t)$.
  end if
end for

```
1 2
R> mypcal1

$projPoints
                      [,1]          [,2]        [,3]
Alabama         2.851277376   77.10710869   1.4875374
Alaska          6.581845450  103.86755060  23.9207756
Arizona         5.943769727  134.82393165  10.2629548
.
.
.
West Virginia  -4.005451580  -77.91512625 -10.4929124
Wisconsin      -4.824096429 -105.99398228  -9.2782269
Wyoming        -0.492932597    2.00148398  -4.4946307

$dispExp
[1] 0.86994879 0.06249969

$scores
                      [,1]         [,2]
Alabama         -77.0298940  -4.71637503
Alaska         -105.6357564  15.65500684
Arizona        -135.3434281  -0.55286092
.
.
.
West Virginia    78.6050386  -4.26141462
Wisconsin       106.5057412  -0.78023788
Wyoming          -1.6122357  -4.67453976

$loadings
            [,1]        [,2]
[1,] -0.04440952  0.12076723
[2,] -0.99581560 -0.08471306
[3,] -0.07986917  0.98905964

attr(,"class")
[1] "pcal1"
```

The scores, the projected points in terms of the components given in $\boldsymbol{V}$, are calculated as the $n \times q$ matrix $\boldsymbol{U} = \boldsymbol{X}\boldsymbol{V}$. The L1 dispersion explained by component j is calculated as

$$\frac{\sum_{i=1}^{n} |u_{ij}|}{\sum_{i=1}^{n} \sum_{k=1}^{m} |x_{ik}|}.$$

Note that the sum of L1 dispersion explained over the components does not sum to 1 as for L2-PCA because of the lack of rotational invariance in the L1 components. The projected points in terms of the original data are given by the $n \times m$ matrix $\boldsymbol{E} = (\boldsymbol{V}\boldsymbol{V}^\top \boldsymbol{X})^\top$.

3.2. `l1pca`

We describe an implementation L1-PCA (Ke and Kanade 2003, 2005) as function `l1pca`, a method for estimating the best-fitting L1 subspace. The optimization problem is written as

$$\min_{\boldsymbol{V},\boldsymbol{U}} \|\boldsymbol{X}^\top - \boldsymbol{V}\boldsymbol{U}^\top\|_1 = \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{V}\boldsymbol{u}_i\|_1. \tag{2}$$

The measure of best fit is the sum of L1 distances from the original points in $\boldsymbol{X}$ to their projections in the subspace $\boldsymbol{E} = \boldsymbol{U}\boldsymbol{V}^\top$. The columns of $\boldsymbol{V}$ define the principal components and therefore the subspace. Note that there is no orthonormality constraint so that the principal components may be oblique.

The algorithm finds a local optimum to (2) by alternately fixing $\boldsymbol{U}$ and $\boldsymbol{V}$ and solving the resultant linear program. Pseudocode is given in Algorithm 2. In Step 1, an initial guess for $\boldsymbol{V}$ is specified. The guess may be the rotation matrix given by L2-PCA or a randomly-generated matrix.

Step 3 finds an estimate for the matrix of scores $\boldsymbol{U}$ by fixing $\boldsymbol{V}$ at its current estimate. As suggested by Ke and Kanade (2003, 2005), this problem may be decomposed into n small linear programs. There is one linear program for each observation $\boldsymbol{x}_i$:

$$\min_{\boldsymbol{\delta},\boldsymbol{u}_i} \sum_{i=1}^{m} \delta_i, \tag{3}$$

subject to

$$-\boldsymbol{\delta} \leq \boldsymbol{x}_i - \boldsymbol{V}(t-1)\boldsymbol{u}_i \leq \boldsymbol{\delta}.$$

Each linear program produces the new estimate of the score $\boldsymbol{u}_i$ for observation $\boldsymbol{x}_i$.

Step 4 derives a new estimate for $\boldsymbol{V}$ based on the current estimate for $\boldsymbol{U}$. Let $u_i(t)$ be the vector of score estimates for observation $\boldsymbol{x}_i$ at iteration t. The optimization problem may be written and solved as a linear program:

$$\min_{\boldsymbol{V},\boldsymbol{\lambda}^+,\boldsymbol{\lambda}^-} \sum_{i=1}^{n}\sum_{j=1}^{m}(\lambda_{ij}^+ + \lambda_{ij}^-), \tag{4}$$

subject to

$$\begin{aligned} \boldsymbol{x}_i - \boldsymbol{V}\boldsymbol{u}_i(t) + \boldsymbol{\lambda}_i^+ - \boldsymbol{\lambda}_i^- &= 0 \quad i = 1,\ldots,n, \\ \boldsymbol{\lambda}_i^+, \boldsymbol{\lambda}_i^- &\geq 0 \quad i = 1,\ldots,n. \end{aligned}$$

Step 5 normalizes the columns of $\boldsymbol{V}$ by dividing by the L2 norm so that the principal component have unit length. The estimation process is repeated until convergence is reached as checked in Step 6.

The complexity of Algorithm 2 is $O((nLP_1 + LP_2)T)$ where LP_1 is the complexity of solving (3), LP_2 is the complexity of solving (4), and T is the number of iterations required for convergence. According to Chvátal (1983), the complexity of solving a linear program with v variables and c constraints using the simplex method is typically $O(c \log v)$. The linear program (3) has $q + m$ variables and $2m$ constraints. The linear program (4) has $2nm + mq$ variables and nm constraints. The complexity may then be approximated as $O(nm \log(2nm + mq)T)$.

Algorithm 2 L1-PCA

Given an $n \times m$ data matrix $\boldsymbol{X}$.

1: Initialize $\boldsymbol{V}(0)$.
2: Set $t = t + 1$.
3: $\boldsymbol{U}(t) = \arg\min_{\boldsymbol{U}} \|\boldsymbol{X}^\top - \boldsymbol{V}(t-1)\boldsymbol{U}^\top\|_1$.
4: $\boldsymbol{V}(t) = \arg\min_{\boldsymbol{V}} \|\boldsymbol{X}^\top - \boldsymbol{V}\boldsymbol{U}^\top(t)\|_1$.
5: Normalize the columns of $\boldsymbol{V}(t)$.
6: **if** $\boldsymbol{V}(t) \neq \boldsymbol{V}(t-1)$ **then**
7: Go to Step 2.
8: **else**
9: Set $\boldsymbol{V} = \boldsymbol{V}(t)$ and $\boldsymbol{U} = \boldsymbol{U}(t)$.
10: **end if**

A flowchart detailing the implementation of `l1pca` in **pcaL1** is in Figure 4. The function `l1pca` is defined in the script `l1pca.R` that passes the data to `l1pca_R.c` where further memory is allocated as needed. The algorithm is implemented in `pcal1.c`, which calls functions in the **Clp** callable library for solving linear programming instances.

A call to `l1pca` produces the loadings matrix $\boldsymbol{V}$, the scores matrix $\boldsymbol{U}$, and the L1 dispersion explained by each component. One may specify the dimension of the projected subspace q, whether to center the data by subtracting the coordinatewise median, and the method for initialization of $\boldsymbol{V}(0)$. Options for initialization include using the first q principal components from L2-PCA or using a randomly-generated matrix. One may also specify whether to calculate the reconstructed points, the maximum number of iterations, and a tolerance parameter that measures convergence. The tolerance parameter allows one to specify an upper bound on $\max\{|v_{ij}(t) - v_{ij}(t-1)| : i = 1, \ldots, n; j = 1, \ldots, m\}$ that must be satisfied for convergence. As the calculation of principal components proceeds, the iteration number is printed to the screen.

```
R> library("pcaL1")
R> data("USArrests")
R> myl1pca <- l1pca(USArrests[,c("Murder", "Assault", "Rape")],
               + projDim=2, center=TRUE, projPoints=TRUE,
               + initialize="l2pca", tolerance=0.0001, iterations=10)
1 2 3
R> myl1pca

$projPoints
                        [,1]          [,2]        [,3]
Alabama         3.1146903688  7.710284e+01   1.3086579
Alaska          5.3065250952  1.039066e+02  24.3714659
Arizona         5.8616196121  1.347930e+02  10.7769448
.
.
.
West Virginia  -3.6110836144 -7.791753e+01 -10.7705013
```

```
Wisconsin      -4.6544881752 -1.059938e+02  -9.3964668
Wyoming        -0.1429585415  1.983520e+00  -4.5139013

$dispExp
[1] 0.8698929 0.0622168

$scores
                       [,1]         [,2]
Alabama          -76.9316809  -4.68982596
Alaska          -105.6543784  16.53935706
Arizona         -135.5640716   0.73310352
.
.
.
West Virginia     78.6690923  -4.91781243
Wisconsin        106.4994205  -1.31578915
Wyoming           -1.6530327  -4.64235766

$loadings
            [,1]        [,2]
[1,] -0.04309571  0.04585239
[2,] -0.99625047 -0.07607457
[3,] -0.07501836  0.99604730

attr(,"class")
[1] "l1pca"
```

The matrix of scores is the matrix $\boldsymbol{U}$ in Algorithm 2 at the time of convergence. The L1 dispersion explained by each component is calculated as for `pcal1`. The reconstructed points are given by the $n \times m$ matrix $\boldsymbol{E} = (\boldsymbol{V}\boldsymbol{V}^\top \boldsymbol{X})^\top$.

3.3. `l1pcastar`

L1-PCA* is an L1-norm method for finding successive best-fit hyperplanes containing the origin. The principal components are taken to be the directions orthogonal to the hyperplanes, in reverse order. The problem of finding the best-fit L1-norm hyperplane containing the origin may be written as

$$\min_{\boldsymbol{V},\boldsymbol{U}} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{V}\boldsymbol{u}_i\|_1, \tag{5}$$

where $\boldsymbol{V}$ is an $m \times (m-1)$ matrix. It can be shown that the L1 projection onto a hyperplane always occurs along a unit direction and the direction of projection is independent of location (Brooks and Dulá 2012). Therefore, a best-fit hyperplane is found by computing m L1 linear regressions, where each variable serves as the dependent variable, and selecting the regression hyperplane with the smallest error.

At each iteration, the algorithm finds a global optimum to (5), uses the optimal L1 regression coefficients to find the coefficients of a principal component, then projects the data onto the

best-fitting hyperplane. Pseudocode is given in Algorithm 3. Step 1 checks if the number of variables m is larger than the number of observations n. Note that if $m > n$, then the data lie in an n-dimensional subspace. A representation of the points in an n-dimensional subspace may be obtained by singular value decomposition. In Step 4, the initial data and projection matrix $\boldsymbol{Q}^{m+1}$ are defined. The loop in Step 5 indicates that the principal components are calculated in reverse order as we project data down from m dimensions to one dimension.

Steps 6-12 find the best of j L1 linear regressions by allowing each variable to take a turn as the response. An L1 linear regression hyperplane may be found by solving a linear program in Step 8 (Charnes, Cooper, and Ferguson 1955; Wagner 1959). Fixing the k^{th} coefficient of $\boldsymbol{\beta}$ to be -1 specifies that the k^{th} variable is the response. The vector $\boldsymbol{\beta}^*$ is the direction orthogonal to the L1-norm best-fit hyperplane in terms of the j-dimensional projection of the data. This direction defines the j^{th} principal component, calculated by projecting the vector out to the original m dimensions in Step 16.

Step 14 projects the points in j dimensions into a $(j-1)$-dimensional subspace by changing one variable value; i.e., the projection is along one unit direction. Steps 15-18 calculate the score of each point as the rows of $\boldsymbol{X}^{j-1}$. In Step 15, a set of spanning vectors for the projected points is calculated. The method implemented in **pcaL1** is to use the singular value decomposition of $\boldsymbol{Z}^j$. The projection matrices $\boldsymbol{Q}^\ell$ are used in Step 16 to project the principal components back to the original m-dimensional subspace. In Step 19, the first principal component is defined as the direction orthogonal to the previous $m-1$ best-fit hyperplanes.

The solution of the linear programs associated with finding L1 regression hyperplanes is the most computationally-intensive step in each iteration j of Algorithm 3. The complexity of Algorithm 3 is $O(\sum_{j=2}^{m'} jLP(2n'+j, n'))$, where $LP(2n'+j, n')$ is the complexity of solving the linear program associated with finding the L1 linear regression hyperplane and $m' = \min\{n, m\}$ and $n' = \max\{n, m\}$. Each LP has $2n'+j$ variables and n' constraints, so with the approximation due to Chvátal (1983), the complexity of Algorithm 3 may be approximated as $O(\sum_{j=2}^{m'} jn' \log(2n'+j))$.

A flowchart detailing the implementation of `l1pcastar` in **pcaL1** is in Figure 5. The function `l1pcastar` is defined in the script `l1pcastar.R` that passes the data to `l1pcastar_R.c` where further memory is allocated as needed. The algorithm is implmented in `l1pcastar.c`. The L1 regression hyperplanes are calculated by solving linear programs using **Clp**, and singular value decompositions are calculated using the function `dgesvd` in LAPACK (Anderson, Bai, Bischof, Blackford, Demmel, Dongarra, Croz, Greenbaum, Hammarling, McKenney, and Sorensen 1999).

A call to `l1pcastar` produces the loadings matrix $\boldsymbol{V}$. One may specify the dimension of the projected subspace q, whether to center the data by subtracting the coordinatewise median, whether to calculate the scores in q dimensions, whether to calculate the reconstructions of the scores in q dimensions, and whether to calculate the L1 dispersion contained in the scores in q dimensions. Regardless of the specified value of q, the data are projected into a one-dimensional subspace, and all of the principal components are returned. The projected dimension q is used to determine the dimension of the space containing the scores and the dimension of the subspace spanned by the reconstructions. The calculations of scores, L1 dispersion explained, and reconstructions are given as options because of the additional memory required.

As the calculation proceeds, the dimension of the projected subspace j is printed to the screen.

Algorithm 3 L1-PCA*

Given an $n \times m$ data matrix $\boldsymbol{X}$.

if m > n **then**
  Represent points in an n-dimensional subspace using the singular value decomposition of $\boldsymbol{X}$.
end if
Set $\boldsymbol{X}^m = \boldsymbol{X}$; let $\boldsymbol{Q}^{m+1}$ be an $m \times m$ identity matrix.
for ($j = m$; $j > 1$; --j) **do**
  Set $k^* = 0$, $R_0(\boldsymbol{X}^j) = \infty$.
  for ($k = 1$; $k \leq j$; ++k) **do**
    Solve $R_k(\boldsymbol{X}) = \min\limits_{\boldsymbol{\beta},\boldsymbol{\lambda}^+,\boldsymbol{\lambda}^-} \sum_{i=1}^n (\lambda_i^+ + \lambda_i^-)$
    subject to

$$\begin{aligned} \boldsymbol{\beta}^\top \boldsymbol{x}_i^j + \lambda_i^+ - \lambda_i^- &= 0 \quad i = 1, \ldots, n \\ \beta_k &= -1 \\ \lambda_i^+, \lambda_i^- &\geq 0 \quad i = 1, \ldots, n \end{aligned}$$

    if $R_k(\boldsymbol{X}^j) < R_{k^*}(\boldsymbol{X}^j)$ **then**
      Set $k^* = k$, $\boldsymbol{\beta}^* = \boldsymbol{\beta}$.
    end if
  end for
  Calculate $\boldsymbol{\alpha} \in \mathbb{R}^m$ by setting $\alpha_{k^*} = 0$ and $\alpha_\ell = \beta_\ell^* / \|\boldsymbol{\beta}^*\|_2$ for $\ell \neq k^*$.
  Set $\boldsymbol{Z}^j = \boldsymbol{X}^j$. Replace column k^* of $\boldsymbol{Z}^j$ with $\boldsymbol{X}^j \boldsymbol{\alpha}$.
  Set $\boldsymbol{Q}^j$ to be a $j \times (j-1)$ matrix whose columns are a set of vectors spanning the subspace containing the rows of $\boldsymbol{Z}^j$.
  Set $\boldsymbol{v}_j = \left(\Pi_{\ell=m+1}^{j+1} \boldsymbol{Q}^\ell \right) \boldsymbol{\beta}^* / \|\boldsymbol{\beta}^*\|_2$.
  Set $\boldsymbol{X}^{j-1} = \boldsymbol{Z}^j \boldsymbol{Q}^j$.
end for
Set $\boldsymbol{v}_1 = \Pi_{\ell=m+1}^{2} \boldsymbol{Q}^\ell$.

```
R> library("pcaL1")
R> data("USArrests")
R> myl1pcastar <- l1pcastar(USArrests[,c("Murder", "Assault", "Rape")],
                  + projDim=2, center=TRUE, scores=TRUE,
                  + projPoints=TRUE, dispExp=TRUE)
2 1
R> myl1pcastar

$projPoints
                      [,1] [,2]  [,3]
Alabama         3.10038554   77   1.1
Alaska          5.31161925  104  24.4
Arizona         5.87584419  135  10.9
.
.
.
West Virginia  -3.61579367  -78 -10.8
Wisconsin      -4.65000000 -106  -9.3
Wyoming        -0.14162457    2  -4.5

$dispExp
[1] 0.86989847 0.06220124

$scores
                       [,1]         [,2]
Alabama         -76.9254159  -4.72258251
Alaska         -105.6764388  16.49435028
Arizona        -135.5650376   0.67537402
.
.
.
Washington       13.4978267   7.14727377
West Virginia    78.6756475  -4.88430746
Wisconsin       106.5011666  -1.27043607
Wyoming          -1.6468379  -4.64305746

$loadings
           [,1]        [,2]        [,3]
[1,] 0.04331743  0.04560367 -0.99801999
[2,] 0.99586535 -0.08181134  0.03948561
[3,] 0.07984866  0.99560394  0.04895897

attr(,"class")
[1] "l1pcastar"
```

The scores, the points projected into the fitted q-dimensional space, are given by $\boldsymbol{U} = \boldsymbol{X}^q$. The projections in the j-dimensional subspace in terms of the original m dimensions is given

by $\boldsymbol{E} = \left(\Pi_{\ell=m+1}^{q+1} \boldsymbol{Q}^{\ell}(\boldsymbol{X}^{q})^{\top}\right)^{\top}$. The L1 dispersion explained for the components is calculated in the same manner as for `pcal1` and `l1pca`.

4. Computational results

The implementations `pcal1`, `l1pca`, and `l1pcastar` in the R environment facilitates a comparison of the methods using simulated data. Comparisons to other methods implemented in R are accessible as well. We apply the methods of **pcaL1** and **pcaPP** (Filzmozer *et al.* 2011) to outlier-contaminated datasets to evaluate the robustness to outliers. We then apply the methods to datasets with varying dimensions to observe the effect on computational complexity.

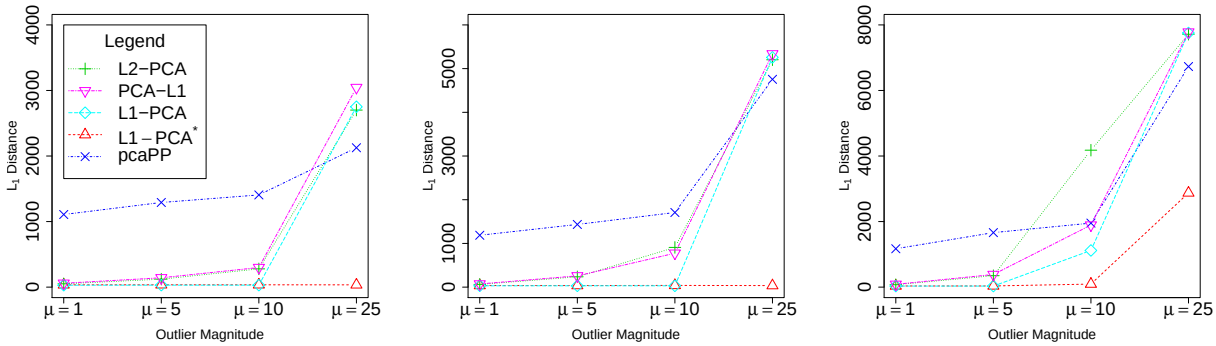
The methods of **pcaL1** are tested using datasets with clustered leverage outliers. Datasets are constructed so that most observations are near a "true" subspace while some of the observations are outliers. The outlier observations have one or more variables with values that are far from the true subspace, but are near to each other. The unbalanced outliers have the effect of pulling the fitted subspace away from the true subspace. To evaluate each method, we calculate the sum of the distances of projected points (i.e., the reconstructions) to the true subspace. The effects of varying the number of contaminated variables, the magnitude of contamination, and dataset size are investigated.

In these experiments, the method for **pcaPP** depends on the size of n and m. If $n \geq m$, the function `PCAgrid` is used; if $n < m$, the function `PCAproj` is used.

Preliminary tests indicated that for `pcal1` and `l1pca`, the best results are obtained when using L2-PCA for initialization. Therefore, L2-PCA is used for initialization in all experiments.

Experiment 1. In the first experiment, we investigate the effects of outlier magnitude and outlier contamination on the ability to fit a subspace. Each dataset consists of $n = 1000$ observations and $m = 10$ dimensions, and the true subspace has dimension $q = 5$ and is spanned by the first five unit directions. The first five columns for each observation are sampled from a $U(-10, 10)$ distribution. For clean data, the remaining five dimensions are sampled from a Laplace$(0, 0.1)$ distribution. Outliers comprise ten percent of each dataset. For outliers, p columns are sampled from a Laplace$(\mu, 0.01)$ distribution, and the last $5 - p$ columns are sampled from a Laplace$(0, 0.1)$ distribution. The outlier contamination p is varied over 1, 2, 3, and the outlier magnitude μ is varied over 1, 5, 10, 25. For each configuration, 100 datasets are generated. The error for a point is measured using the L1 distance of its projection into the fitted five-dimensional space to the true subspace.

The results are in Figure 1 and Table 1. For $p = 1$ and $\mu = 1$, 5, and 10, the L1-norm PCA methods in **pcaL1** resist the effects of outliers and have a lower error when compared to L2-PCA. The method in **pcaPP** has the highest average error when $\mu \leq 10$. When $\mu = 25$, the PCA-L1 and L1-PCA begin to fit the outliers instead of recovering the true subspace, and have higher error rates than L2-PCA. The method pcaPP has lower error rates than L2-PCA, but is still affected by the outliers. L1-PCA* appears unaffected by the magnitudes of outlier contamination for $p = 1$. A similar phenomenon is observed for $p = 2$ and $p = 3$. For $p = 3$ and $\mu = 10$, the approach of a *breakdown point* is observed as the average and standard deviation of the errors begins to increase rapidly. The standard deviation of errors for pcaPP is high for every configuration, indicating a sensitivity to small changes in input data. In the presence of low outlier contamination ($\mu \leq 5$), L1-PCA has the lowest average



(a) $p = 1$ (b) $p = 2$ (c) $p = 3$

Figure 1: The error versus outlier magnitude for (a) $p = 1$, (b) $p = 2$, and (c) $p = 3$ outlier contaminated-dimensions. For each replication, error is measured as the sum of L1 distances of projected points in a five-dimensional space to the "true" five-dimensional subspace of the data. The average over 100 replications is plotted.

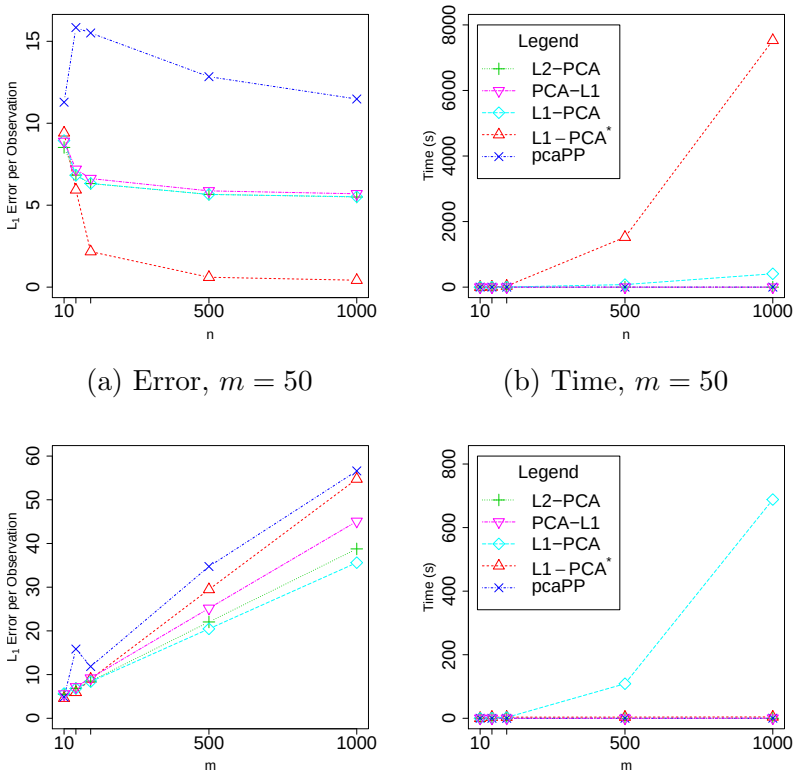
error rates followed by L1-PCA*. As the contamination magnitude is increased, L1-PCA* has the lowest average error and has a breakdown point only when $p = 3$ and $\mu > 10$.

Experiment 2. In the second experiment, we investigate the effects of dataset size on the ability to fit a subspace and on computation time. The computational complexities of the methods implemented in **pcaL1** are difficult to compare because tight bounds are unknown for the convergence rates of the methods implemented in `pcal1` and `l1pca`. This experiment provides insight into the computational time required and performance for each method for varying numbers of observations n and variables m.

Each dataset in this experiment is constructed as described for Experiment 1, with the following changes. Fora all datasets, $q = 5$, $p = 2$, and $\mu = 25$. For $m = 10$ and 50, datasets are constructed with $n = 10$, 50, 100, 500, and 1000 to observe the dependence on n. For $n = 10$ and 50, datasets are constructed with m =10, 50, 100, 500, and 1000 to observe the dependence on m. For each configuration, 100 datasets are generated.

The results are in Figure 2 and Tables 2-5. For $n \geq m$, L1-PCA* has the lowest average error among all methods, with pronounced improvement as n is increased. When $n = 1000$ and $m = 50$, the average error for other methods is at least ten times that of L1-PCA*. When $n \simeq m$, the standard deviation for L1-PCA* is higher than L2-PCA and the other methods in **pcaL1**, indicating a sensitivity to small changes in input data for those configurations. As in Experiment 1, for $n \geq m$, the other L1 methods perform worse than L2-PCA and appear to be fitting the outliers rather than the points near the true subspace. As n is increased, the computation times for all methods except L1-PCA and L1-PCA* remain negligible. For $n = 1000$, L1-PCA requires more than six minutes, and L1-PCA* requires more than two hours. The computation time for L1-PCA* increases slower than $n \log n$ as expected by the worst-case analysis. The differences in error rates appear to justify the extra computation time for L1-PCA*.

When $m > n$, L2-PCA and L1-PCA have the lowest error rates. As m is increased, an advantage is seen for L1-PCA. The error rates for L1-PCA are 10-30% lower than for the



(a) Error, $m = 50$ (b) Time, $m = 50$

(c) Error, $n = 50$ (d) Time, $n = 50$

Figure 2: The average L1 error per observation and computational time for different sizes of data matrices. (a) The L1 error per observation for $m = 50$ as a function of sample size n, (b) the total computational time for $m = 50$ as a function of sample size n, (c) the L1 error per observation for $n = 50$ as a function of m, (d) the total computational time for $n = 50$ as a function of m. Each plotted point represents 100 replications.

other methods. A tradeoff in performance and computation time is again observed. The computation times for all methods remains under five seconds except for L1-PCA. For $n = 50$ and $m = 1000$, L1-PCA requires over 10 minutes on average.

5. Conclusions

This paper describe the implementation of three L1-norm PCA methods. Each method is presented with common notation so that the approaches may be easily compared. Their implementation in an R package facilitates performance comparisons. Worst-case and emprirical computational complexities are presented. The performance of the methods on simulated data shows that under different conditions, different methods are indicated based on error rates and computation time. Unlike L2-norm PCA, there are many L1-norm PCA methods possible. The software package presented here provides a basic framework for comparing new methods.

Acknowledgements

The first author is supported in part by NIH-NIAID awards UH2AI083263-01 and UH3AI08326-01 and NASA award NNX09AR44A. The authors would like to acknowledge the Center for High Performance Computing at VCU for providing computational infrastructure and support.

References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J, Croz JD, Greenbaum A, Hammarling S, McKenney A, Sorensen D (1999). *LAPACK Users' Guide*. Third edition. Society for Industrial and Applied Mathematics, Philadelphia, PA. ISBN 0-89871-447-8 (paperback).

Brooks J, Dulá J (2012). "The L1-norm best-fit hyperplane problem." *Applied Mathematics Letters*, **in press**.

Brooks J, Dulá J, Boone E (2012). "A pure L1-norm principal component analysis." Submitted; available at http://www.optimization-online.org/DB_HTML/2010/01/2513.html.

Charnes A, Cooper W, Ferguson R (1955). "Optimal estimation of executive compensation by linear programming." *Management Science*, **1**, 138–150.

Choulakian V (2006). "L_1-norm projection pursuit principal component analysis." *Computational Statistics and Data Analysis*, **50**, 1441–1451.

Chvátal V (1983). *Linear Programming*. Freeman.

Croux C, Filzmoser P, Oliveira M (2007). "Algorithms for Projection-Pursuit robust principal components analysis." *Chemometrics and Intelligent Laboratory Systems*, **87**, 218–225.

Croux C, Ruiz-Gazen A (2005). "High breakdown estimators for prinicpal components: the projection-pursuit approach revisited." *Journal of Multivariate Analysis*, **95**, 206–226.

Filzmozer P, Fritz H, Kalcher K (2011). *pcaPP: Robust PCA by projection pursuit*. R package version 1.9-3, URL http://CRAN.R-project.org/package=pcaPP.

Jolliffe I (2002). *Principal Component Analysis*. 2nd edition. Springer.

Ke Q, Kanade T (2003). "Robust subspace computation using L1 norm." *Technical Report CMU-CS-03-172*, Carnegie Mellon University, Pittsburgh, PA.

Ke Q, Kanade T (2005). "Robust L1 norm factorization in the presence of outliers and missing data by alternative convex programming." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Kwak N (2008). "Principal component analysis based on L1-norm maximization." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **30**, 1672–1680.

Maronna R (2005). "Principal components and orghogonal regression based on robust scales." *Technometrics*, **47**, 264–273.

Rockafellar R (1970). *Convex analysis*. Princeton University Press.

Wagner H (1959). "Linear programming techniques for regression analysis." *Journal of the American Statistical Association*, **54**, 206–212.

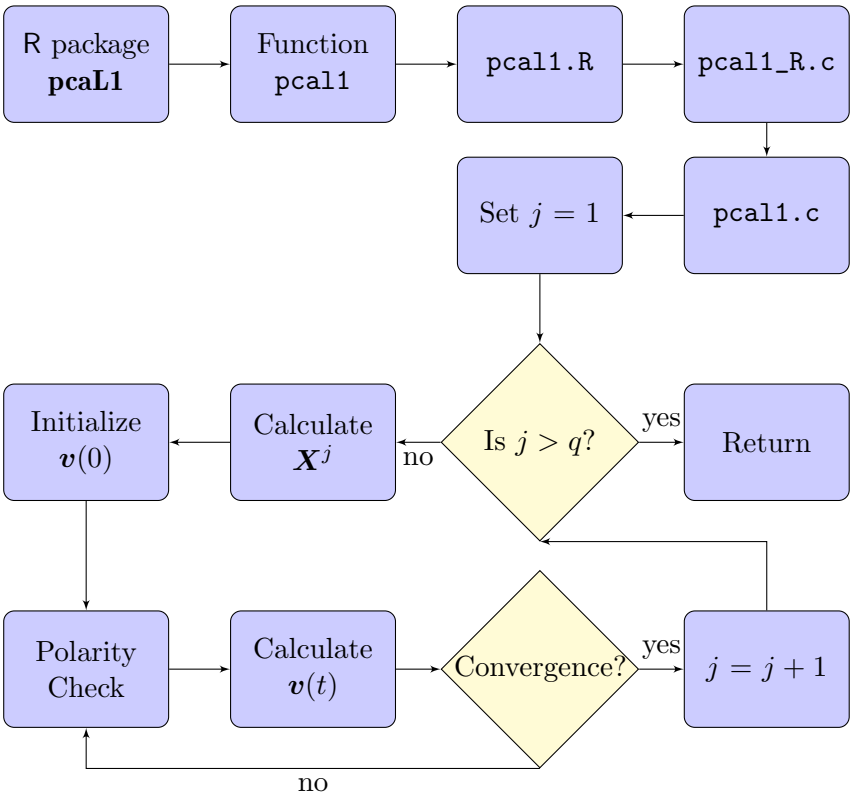


Figure 3: Flowchart of steps in implementation of function `pcal1` in R package **pcaL1**.

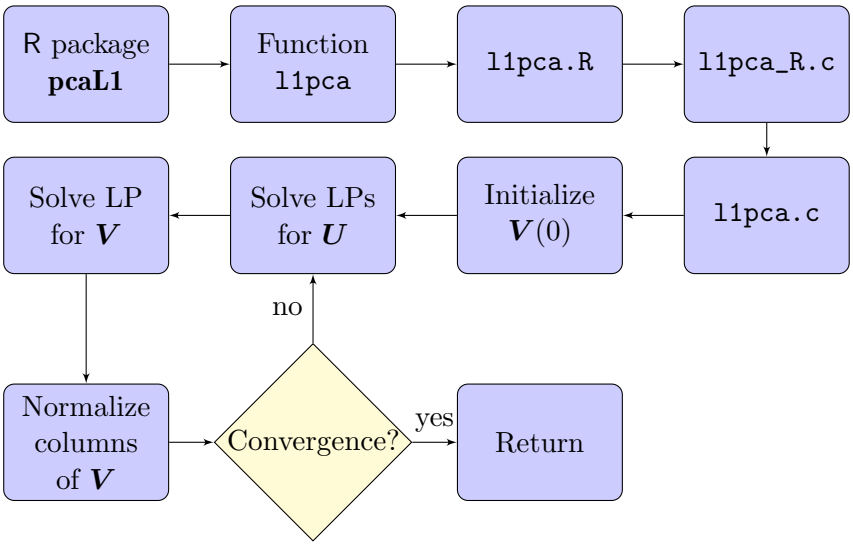


Figure 4: Flowchart of steps in implementation of function `l1pca` in R packagae **pcaL1**.

Affiliation:

J. Paul Brooks
Department of Statistical Sciences & Operations Research
Virginia Commonwealth University
P.O. Box 843083 Richmond, VA, USA 23284
E-mail: jpbrooks@vcu.edu

Table 1: Average (standard deviation) of L1 distance to true subspace for $m = 10$. The number of outlier-contaminated dimensions is p. The outlier magnitude is μ. The results reflect 100 replications for each configuration.

p	μ	L2-PCA	PCA-L1	L1-PCA	L1-PCA*	pcaPP
0	0	36.6 (6.5)	41.9 (7.7)	27.8 (5.4)	39.4 (6.8)	823.5 (195.1)
1	1	44.9 (9.2)	51.4 (10.4)	28.5 (5.6)	37.5 (7.4)	875.6 (206.3)
2	1	52.9 (11.2)	61.7 (14.6)	29.0 (6.1)	39.4 (7.2)	894.7 (211.2)
3	1	57.3 (16.9)	64.4 (18.7)	29.2 (5.8)	38.5 (7.5)	933.8 (218.2)
1	5	87.5 (29.2)	101.1 (33.3)	28.2 (5.3)	38.0 (7.0)	993.9 (246.3)
2	5	158.2 (64.8)	174.2 (66.2)	27.8 (5.1)	39.0 (7.2)	1151.3 (294.9)
3	5	228.2 (100.0)	249.3 (109.6)	29.4 (5.0)	38.4 (7.8)	1205.2 (382.2)
1	10	187.1 (78.6)	192.2 (82.0)	27.2 (5.1)	38.0 (7.0)	1126.0 (323.6)
2	10	539.9 (252.0)	448.5 (196.0)	28.7 (5.8)	40.4 (7.4)	1267.9 (365.6)
3	10	2820.0 (1177.7)	1071.9 (448.5)	401.0 (1271.2)	62.9 (22.9)	1499.1 (386.4)
1	25	2718.4 (100.4)	3164.8 (224.0)	2719.6 (104.6)	37.0 (6.5)	1590.2 (582.7)
2	25	5201.5 (52.8)	5417.6 (348.8)	5258.1 (95.1)	38.8 (7.5)	3500.3 (997.3)
3	25	7718.2 (38.0)	7786.8 (123.7)	7745.3 (63.3)	38.5 (7.8)	4549.3 (784.2)
0	0	39.7 (6.3)	46.6 (7.7)	29.1 (4.5)	37.2 (5.7)	1061.5 (240.3)
1	1	50.4 (7.5)	58.8 (10.3)	29.1 (4.6)	34.8 (5.0)	1107.8 (287.9)
2	1	62.2 (11.6)	72.3 (12.8)	30.4 (4.6)	36.5 (5.5)	1186.6 (574.1)
3	1	73.5 (20.2)	85.6 (21.2)	31.8 (4.7)	35.0 (5.8)	1171.1 (278.6)
1	5	121.3 (26.1)	142.3 (35.4)	30.2 (4.0)	35.5 (5.0)	1291.3 (263.5)
2	5	236.0 (70.0)	257.5 (68.5)	30.3 (4.9)	36.2 (5.6)	1434.2 (304.4)
3	5	349.2 (106.7)	385.0 (124.2)	31.8 (5.0)	35.7 (6.1)	1663.2 (372.8)
1	10	278.6 (73.8)	298.9 (79.1)	30.1 (4.9)	35.3 (5.4)	1406.5 (322.5)
2	10	908.5 (295.9)	775.8 (281.3)	30.3 (4.8)	39.2 (6.3)	1707.9 (382.2)
3	10	4177.9 (783.2)	1893.4 (691.8)	1118.9 (1981.8)	93.1 (31.7)	1953.5 (424.8)
1	25	2700.8 (79.3)	3044.7 (368.0)	2751.1 (97.4)	34.8 (5.3)	2126.1 (640.4)
2	25	5202.6 (49.7)	5336.8 (164.4)	5257.1 (84.7)	35.0 (5.6)	4754.9 (811.1)
3	25	7715.5 (33.7)	7781.4 (111.3)	7753.5 (73.4)	2873.9 (3726.6)	6729.9 (770.2)

Table 2: Average (standard deviation) of L1 distance to true subspace for $m = 50$. The dimension of the "true" underlying subspace is five, the number of outlier-contaminated dimensions is two, and the outlier magnitude is 25. The results reflect 100 replications for each configuration.

n	L2-PCA	PCA-L1	L1-PCA	L1-PCA*	pcaPP
10	85.2 (3.3)	89.0 (3.4)	89.4 (5.0)	94.1 (5.1)	112.8 (23.6)
50	341.8 (12.2)	359.5 (15.7)	341.2 (14.8)	296.7 (117.0)	792.0 (116.6)
100	631.7 (13.8)	662.4 (23.3)	632.9 (21.4)	216.6 (187.0)	1550.8 (187.5)
500	2830.3 (35.6)	2937.0 (97.6)	2827.1 (55.1)	301.3 (14.8)	6420.7 (845.1)
1000	5497.0 (50.2)	5695.8 (153.8)	5515.4 (87.3)	421.3 (20.5)	11478.5 (1326.5)

Table 3: Average (standard deviation) of computation time for $m = 50$. The dimension of the "true" underlying subspace is five, the number of outlier-contaminated dimensions is two, and the outlier magnitude is 25. The results reflect 100 replications for each configuration.

n	L2-PCA	PCA-L1	L1-PCA	L1-PCA*	pcaPP
10	0.0032 (0.00064)	0.0023 (0.00048)	0.073 (0.014)	0.010 (0.0012)	0.0015 (5e-04)
50	0.0062 (0.00068)	0.013 (0.00044)	0.77 (0.15)	3.6 (0.085)	0.0018 (0.00042)
100	0.0082 (9e-04)	0.019 (0.00072)	2.7 (0.56)	33.1 (0.54)	0.0023 (0.00046)
500	0.018 (0.0012)	0.054 (0.0027)	77.6 (13.3)	1522.8 (14.1)	0.0053 (0.00045)
1000	0.031 (0.0021)	0.11 (0.0077)	406.8 (62.0)	7529.2 (154.9)	0.0096 (0.00062)

Table 4: Average (standard deviation) of L1 distance to true subspace for $n = 50$. The dimension of the "true" underlying subspace is five, the number of outlier-contaminated dimensions is two, and the outlier magnitude is 25. The results reflect 100 replications for each configuration.

m	L2-PCA	PCA-L1	L1-PCA	L1-PCA*	pcaPP
10	274.0 (12.1)	277.9 (13.9)	283.1 (14.6)	230.0 (101.6)	243.8 (58.0)
50	341.8 (12.2)	359.5 (15.7)	341.2 (14.8)	296.7 (117.0)	792.0 (116.6)
100	425.3 (12.2)	459.7 (15.5)	419.4 (15.7)	449.2 (96.4)	592.6 (143.3)
500	1102.2 (18.3)	1258.2 (27.4)	1023.3 (22.6)	1473.5 (109.4)	1736.7 (264.5)
1000	1938.0 (23.4)	2253.1 (45.4)	1780.3 (41.6)	2737.1 (115.8)	2830.3 (273.0)

Table 5: Average (standard deviation) of computation time for $n = 50$. The dimension of the "true" underlying subspace is five, the number of outlier-contaminated dimensions is two, and the outlier magnitude is 25. The results reflect 100 replications for each configuration.

m	L2-PCA	PCA-L1	L1-PCA	L1-PCA*	pcaPP
10	0.0024 (0.00064)	0.0018 (0.00045)	0.064 (0.0089)	0.09 (0.0035)	0.00065 (0.00048)
50	0.0062 (0.00068)	0.013 (0.00044)	0.77 (0.15)	3.6 (0.085)	0.0018 (0.00042)
100	0.0093 (0.00087)	0.024 (0.00063)	2.7 (0.44)	3.6 (0.065)	0.0037 (0.00049)
500	0.030 (0.001)	0.12 (0.0048)	108.7 (39.7)	3.7 (0.059)	0.066 (0.012)
1000	0.056 (0.0017)	0.33 (0.023)	688.2 (201.0)	4.2 (0.11)	0.23 (0.032)

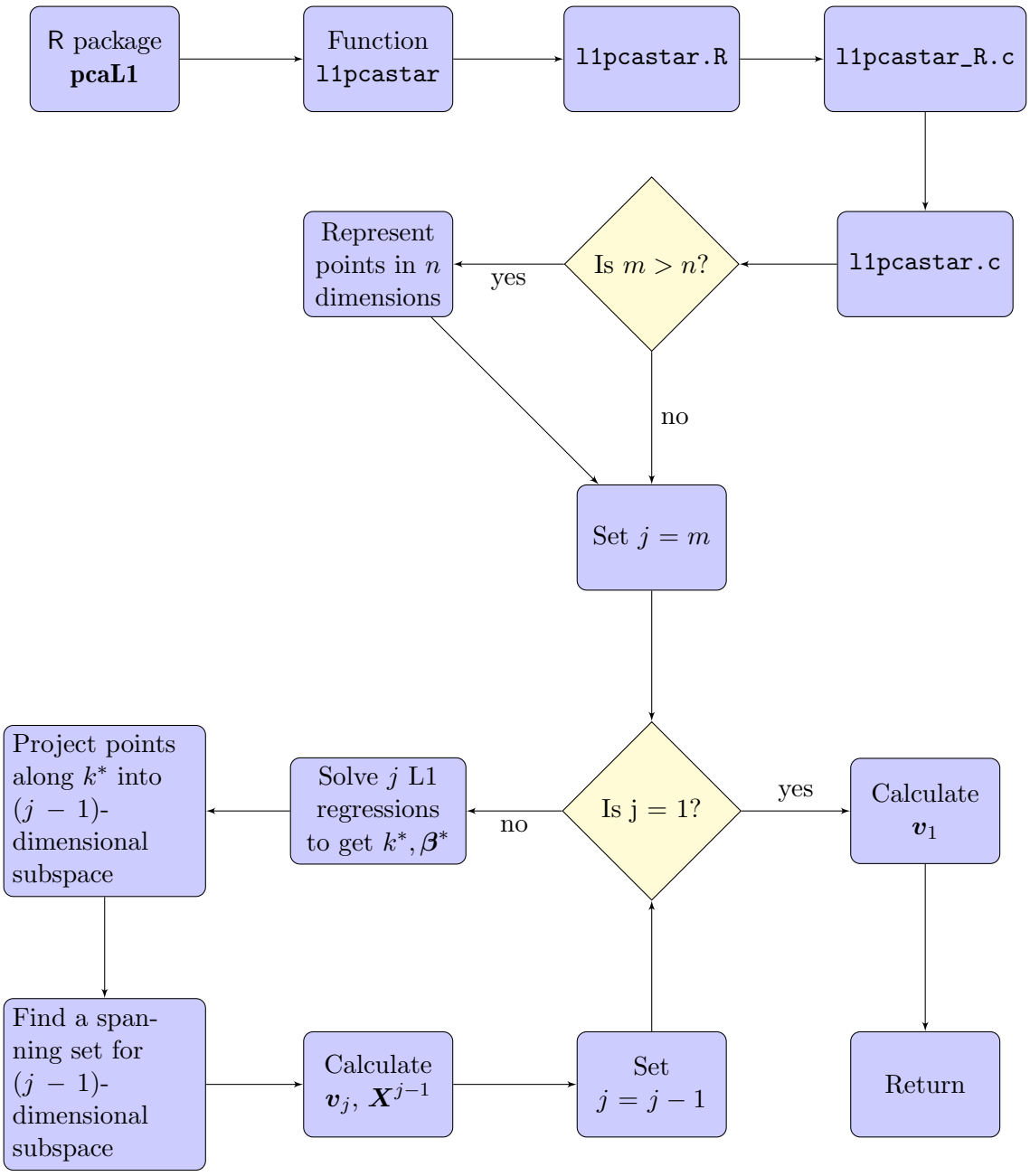


Figure 5: Flowchart of steps in implementation of function `l1pcastar` in R package **pcaL1**.