

# Package ‘epca’

May 8, 2026

**Type** Package

**Title** Exploratory Principal Component Analysis

**Version** 1.1.0

**Date** 2023-07-10

**Description** Exploratory principal component analysis for large-scale dataset, including sparse principal component analysis and sparse matrix approximation.

**URL** <https://github.com/fchen365/epca>

**BugReports** <https://github.com/fchen365/epca/issues>

**License** GPL-3

**Depends** R ( $\geq 3.5$ ),

**Imports** clue, irlba, Matrix, GPArotation,

**Suggests** elasticnet, ggcorrplot, tidyverse, rmarkdown, reshape2, markdown, RSpectra, matlabr, knitr, PMA, testthat ( $\geq 3.0.0$ )

**VignetteBuilder** knitr, rmarkdown

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**NeedsCompilation** no

**Author** Fan Chen [aut, cre] (ORCID: <https://orcid.org/0000-0003-4508-6023>)

**Maintainer** Fan Chen <fan.chen@wisc.edu>

**Repository** CRAN

**Date/Publication** 2023-07-10 19:50:11 UTC

## Contents

epca-package . . . . .	2
absmin . . . . .	3
absmin.criteria . . . . .	3
cpve . . . . .	4
dist.matrix . . . . .	5

distance . . . . .	6
exp.frac . . . . .	6
hard . . . . .	7
inner . . . . .	7
labelCluster . . . . .	8
misClustRate . . . . .	8
norm.Lp . . . . .	9
permColumn . . . . .	10
pitprops . . . . .	10
polar . . . . .	11
print.sca . . . . .	11
print.sma . . . . .	12
prs . . . . .	12
pve . . . . .	14
rootmatrix . . . . .	15
rotation . . . . .	15
sca . . . . .	16
shrinkage . . . . .	19
sma . . . . .	20
soft . . . . .	23
varimax . . . . .	23
varimax.criteria . . . . .	24
vgQ.absmin . . . . .	25
<b>Index</b>	<b>26</b>

---

 epca-package

*Exploratory Principal Component Analysis*


---

## Description

epca is for comprehending any data matrix that contains *low-rank* and *sparse* underlying signals of interest. The package currently features two key tools: (1) sca for **s**parse **p**incipal **c**omponent **a**nalysis and (2) sma for **s**parse **m**atrix **a**pproximation, a two-way data analysis for simultaneously row and column dimensionality reductions.

## References

Chen, F. and Rohe K. (2020) "A New Basis for Sparse PCA".

---

absmin	<i>Absmin Rotation</i>
--------	------------------------

---

**Description**

Given a  $p \times k$  matrix  $x$ , finds the orthogonal matrix (rotation) that minimizes the [absmin.criteria](#).

**Usage**

```
absmin(x, r0 = diag(ncol(x)), normalize = FALSE, eps = 1e-05, maxit = 1000L)
```

**Arguments**

<code>x</code>	a matrix or Matrix, initial factor loadings matrix for which the rotation criterion is to be optimized.
<code>r0</code>	matrix, initial rotation matrix.
<code>normalize</code>	logical. Should Kaiser normalization be performed? If so the rows of $x$ are re-scaled to unit length before rotation, and scaled back afterwards.
<code>eps</code>	The tolerance for stopping: the relative change in the sum of singular values.
<code>maxit</code>	integer, maximum number of iteration (default to 1,000).

**Value**

A list with three elements:

<code>rotated</code>	the rotated matrix.
<code>rotmat</code>	the (orthogonal) rotation matrix.
<code>n.iter</code>	the number of iteration taken.

**See Also**

`GPARotation::GPForth`

---

absmin.criteria	<i>Absmin Criteria</i>
-----------------	------------------------

---

**Description**

Calculate the absmin criteria. This is a helper function for [absmin](#).

**Usage**

```
absmin.criteria(x)
```

**Arguments**

`x` a `matrix` or `Matrix`, initial factor loadings matrix for which the rotation criterion is to be optimized.

---

`cpve`*Cumulative Proportion of Variance Explained (CPVE)*

---

**Description**

Calculate the CPVE.

**Usage**

```
cpve(x, v, is.cov = FALSE)
```

**Arguments**

`x` `matrix` or `Matrix`, the original data matrix or the Gram matrix.  
`v` `matrix` or `Matrix`, coefficients of linear transformation, e.g., loadings (in PCA).  
`is.cov` `logical`, whether the input matrix is a covariance matrix (or a Gram matrix).

**Value**

a numeric vector of length `ncol(v)`, the *i*-th value is the CPVE of the first *i* columns in `v`.

**See Also**

[pve](#)

**Examples**

```
## use the "swiss" data
## find two sparse PCs
s.sca <- sca(swiss, 2, gamma = sqrt(ncol(swiss)))
ld <- loadings(s.sca)
cpve(as.matrix(swiss), ld)
```

---

dist.matrix	<i>Matrix Column Distance</i>
-------------	-------------------------------

---

### Description

Compute the distance between two matrices. The distance between two matrices is defined as the sum of distances between column pairs. This function matches the columns of two matrices, such that the matrix distance (i.e., the sum of paired column distances) is minimized. This is accomplished by solving an optimization over column permutation. Given two matrices, `x` and `y`, find permutation `p()` that minimizes  $\sum_i \text{similarity}(x[,p(i)], y[,i])$ , where the `similarity()` can be "euclidean" distance,  $1 - \text{"cosine"}$ , or "maximum" difference (manhattan distance). The solution is computed by `clue::solve_LSAP()`.

### Usage

```
dist.matrix(x, y, method = "euclidean")
```

### Arguments

<code>x, y</code>	matrix or Matrix, of the same number of rows. The columns of <code>x</code> and <code>y</code> will be scaled to unit length.
<code>method</code>	distance measure, "maximum", "cosine", or "euclidean" are implemented.

### Value

a list of four components:

<code>dist</code>	dist, the distance matrix.
<code>match</code>	<code>solve_LSAP</code> , the column matches.
<code>value</code>	numeric vector, the distance between pairs of columns.
<code>method</code>	character, the distance measure used.
<code>nrow</code>	integer, the dimension of the input matrices, i.e., <code>nrow(x)</code> .

### See Also

[clue::solve\\_LSAP](#)

### Examples

```
x <- diag(4)
y <- x + rnorm(16, sd = 0.05) # add some noise
y = t(t(y) / sqrt(colSums(y ^ 2))) ## normalize the columns
## euclidian distance between column pairs, with minimal matches
dist.matrix(x, y, "euclidean")
```

---

distance	<i>Matrix Distance</i>
----------	------------------------

---

**Description**

Matrix Distance

**Usage**

```
distance(x, y, method = "euclidean")
```

**Arguments**

x, y	matrix or Matrix, of the same number of rows. The columns of x and y will be scaled to unit length.
method	distance measure, "maximum", "cosine", or "euclidean" are implemented.

**Value**

numeric, the distance between two matrices.

---

exp.frac	<i>Calculate fractional exponent/power</i>
----------	--

---

**Description**

Calculate fractional exponent/power,  $a^{(\text{num}/\text{den})}$ , where a could be negative.

**Usage**

```
## S3 method for class 'frac'
exp(a, num, den)
```

**Arguments**

a	numeric(1), base (could be negative).
num	a positive integer, numerator of the exponent.
den	a positive integer, denominator of the exponent.

**Value**

numeric, the evaluated  $a^{(\text{num}/\text{den})}$

---

hard	<i>Hard-thresholding</i>
------	--------------------------

---

**Description**

Perform hard-thresholding given the cut-off value.

**Usage**

```
hard(x, t)
```

**Arguments**

x	any numerical matrix or vector.
t	numeric, the amount to hard-threshold, i.e., $\text{sgn}(x_{ij})( x_{ij} - t )_+$ .

---

inner	<i>Matrix Inner Product</i>
-------	-----------------------------

---

**Description**

Calculate the custom matrix inner product  $z$  of two matrices,  $x$  and  $y$ , where  $z[i, j] = \text{FUN}(x[, i], y[, j])$ .

**Usage**

```
inner(x, y, FUN = "crossprod", ...)
```

**Arguments**

x, y	matrix or Matrix.
FUN	function or a character(1) name of base function. The function should take in two vectors as input and output a numeric(1) result.
...	additional parameters for FUN.

**Value**

matrix, inner product of  $x$  and  $y$ .

**Examples**

```
x <- matrix(1:6, 2, 3)
y <- matrix(7:12, 2, 3)
## The default is equivalent to `crossprod(x, y)`
inner(x, y)
## We can compute the pair-wise Euclidean distance of columns.
EuclideanDistance = function(x, y) crossprod(x, y)^2
inner(x, y, EuclideanDistance)
```

---

**labelCluster***Label Cluster*

---

**Description**

Assign cluster labels to each row from the membership matrix.

**Usage**

```
labelCluster(x, ties.method = "random")
```

**Arguments**

**x** matrix with non-negative entries, where  $x[i, j]$  is the estimated likelihood (or any equivalent measure) of node  $i$  belongs to block  $j$ . The higher the more likely.

**ties.method** character, how should ties be handled, "random", "first", "last" are allowed. See [base::rank\(\)](#) for details.

**Value**

integer vector of the same length as  $x$ . Each entry is one of 1, 2, ...,  $\text{ncol}(x)$ .

---

**misClustRate***Mis-Classification Rate (MCR)*

---

**Description**

Compute the empirical MCR, assuming that  $\#cluster = \#block$ , This calculation allows a permutation on clusters.

**Usage**

```
misClustRate(cluster, truth)
```

**Arguments**

`cluster` vector of integer or factor, estimated cluster membership.  
`truth` a vector of the same length as `clusters`, the true cluster labels.

**Value**

numeric, the MCR.

**Examples**

```
truth = rep(1:3, each = 30)
cluster = rep(3:1, times = c(25, 32, 33))
misClustRate(cluster, truth)
```

---

`norm.Lp`*Element-wise Matrix Norm*

---

**Description**

Compute element-wise matrix Lp-norm. This is a helper function to [shrinkage\(\)](#).

**Usage**

```
norm.Lp(x, p = 1)
```

**Arguments**

`x` a matrix or Matrix.  
`p` numeric(1), the p for defining the Lp norm.

**Value**

numeric(1), the absolute sum of all elements.

permColumn

*Permute columns of a block membership matrix*

---

**Description**

Perform column permutation of block membership matrix for aesthetic visualization. That is, the k-th column gives k-th cluster. This is done by ranking the column sums of squares (by default).

**Usage**

```
permColumn(x, s = 2)
```

**Arguments**

x                    a non-negative matrix, nNode x nBlock,  
s                    integer, order of non-linear

---

pitprops

*Pitprops correlation data*

---

**Description**

The pitprops data is a correlation matrix that was calculated from 180 observations. There are 13 explanatory variables. Jeffers (1967) tried to interpret the first six PCs. This is a classical example showing the difficulty of interpreting principal components.

**References**

Jeffers, J. (1967) "Two case studies in the application of principal component", *Applied Statistics*, 16, 225-236.

**Examples**

```
## NOT TEST  
data(pitprops)  
ggcorrplot::ggcorrplot(pitprops)
```

---

polar

*Polar Decomposition*

---

### Description

Perform the polar decomposition of an  $n \times p$  ( $n > p$ ) matrix  $x$  into two parts:  $u$  and  $h$ , where  $u$  is an  $n \times p$  unitary matrix with orthogonal columns (i.e. `crossprod(u)` is the identity matrix), and  $h$  is a  $p \times p$  positive-semidefinite Hermitian matrix. The function returns the  $u$  matrix. This is a helper function of `prs()`.

### Usage

```
polar(x)
```

### Arguments

$x$  a matrix or Matrix, which is presumed full-rank.

### Value

a matrix of the unitary part of the polar decomposition.

### References

Chen, F. and Rohe, K. (2020) "A New Basis for Sparse Principal Component Analysis."

### Examples

```
x <- matrix(1:6, nrow = 3)
polar_x <- polar(x)
```

---

print.sca

*Print SCA*

---

### Description

Print SCA

### Usage

```
## S3 method for class 'sca'
print(x, verbose = FALSE, ...)
```

**Arguments**

x                    an sca object.  
 verbose            logical(1), whether to print out loadings.  
 ...                additional input to generic [print](#).

**Value**

Print an sca object interactively.

---

print.sma	<i>Print SMA</i>
-----------	------------------

---

**Description**

Print SMA

**Usage**

```
## S3 method for class 'sma'
print(x, verbose = FALSE, ...)
```

**Arguments**

x                    an sma object.  
 verbose            logical(1), whether to print out loadings.  
 ...                additional input to generic [print](#).

**Value**

Print an sma object interactively.

---

prs	<i>Polar-Rotate-Shrink</i>
-----	----------------------------

---

**Description**

This function is a helper function of [sma\(\)](#). It performs polar decomposition, orthogonal rotation, and soft-thresholding shrinkage in order. The three steps together enable sparse estimates of the SMA and SCA.

**Usage**

```
prs(x, z.hat, gamma, rotate, shrink, normalize, order, flip, epsilon)
```

## Arguments

<code>x, z.hat</code>	the matrix product <code>crossprod(x, z.hat)</code> is the actual Polar-Rotate-Shrink object. <code>x</code> and <code>z.hat</code> are input separately because the former is additionally used to compute the proportion of variance explained, in the case when <code>order = TRUE</code> .
<code>gamma</code>	numeric, the sparsity parameter.
<code>rotate</code>	character(1), rotation method. Two options are currently available: "varimax" (default) or "absmin" (see details).
<code>shrink</code>	character(1), shrinkage method, either "soft"- (default) or "hard"-thresholding (see details).
<code>normalize</code>	logical, whether to rows normalization should be done before and undone afterward the rotation (see details).
<code>order</code>	logical, whether to re-order the columns of the estimates (see Details below).
<code>flip</code>	logical, whether to flip the signs of the columns of estimates such that all columns are positive-skewed (see details).
<code>epsilon</code>	numeric, tolerance of convergence precision (default to 0.00001).

## Details

`rotate`: The `rotate` option specifies the rotation technique to use. Currently, there are two build-in options—"varimax" and "absmin". The "varimax" rotation maximizes the element-wise L4 norm of the rotated matrix. It is faster and computationally more stable. The "absmin" rotation minimizes the absolute sum of the rotated matrix. It is sharper (as it directly minimizes the L1 norm) but slower and computationally less stable.

`shrink`: The `shrink` option specifies the shrinkage operator to use. Currently, there are two build-in options—"soft"- and "hard"-thresholding. The "soft"-thresholding universally reduce all elements and sets the small elements to zeros. The "hard"-thresholding only sets the small elements to zeros.

`normalize`: The argument `normalize` gives an indication of if and how any normalization should be done before rotation, and then undone after rotation. If `normalize` is FALSE (the default) no normalization is done. If `normalize` is TRUE then Kaiser normalization is done. (So squared row entries of normalized `x` sum to 1.0. This is sometimes called Horst normalization.) For `rotate="absmin"`, if `normalize` is a vector of length equal to the number of indicators (i.e., the number of rows of `x`), then the columns are divided by `normalize` before rotation and multiplied by `normalize` after rotation. Also, If `normalize` is a function then it should take `x` as an argument and return a vector which is used like the vector above.

`order`: In PCA (and SVD), the principal components (and the singular vectors) are ordered. For this, we order the sparse components (i.e., the columns of `z` or `y`) by their explained variance in the data, which is defined as  $\sum(x \% \% y)^2$ , where `y` is a column of the sparse component. Note: not to be confused with the cumulative proportion of variance explained by `y` (and `z`), particularly when `y` (and `z`) is may not be strictly orthogonal.

`flip`: The argument `flip` gives an indication of if and the columns of estimated sparse component should be flipped. Note that the estimated (sparse) loadings, i.e., the weights on original variables, are column-wise invariant to a sign flipping. This is because flipping of a principal direction does not influence the amount of the explained variance by the component. If `flip=TRUE`, then the columns of loadings will be flip accordingly, such that each column is positive-skewed. This means that for each column, the sum of cubic elements (i.e.,  $\sum(x^3)$ ) are non-negative.

**Value**

a matrix of the sparse estimate, of the same dimension as `crossprod(x, z.hat)`.

**References**

Chen, F. and Rohe, K. (2020) "A New Basis for Sparse Principal Component Analysis."

**See Also**

[sma](#), [sca](#), [polar](#), [rotation](#), [shrinkage](#)

---

pve

*Proportion of Variance Explained (PVE)*

---

**Description**

Calculate the Proportion of variance explained by a set of linear transformation, (e.g. eigenvectors).

**Usage**

```
pve(x, v, is.cov = FALSE)
```

**Arguments**

<code>x</code>	matrix or Matrix, the original data matrix or the Gram matrix.
<code>v</code>	matrix or Matrix, coefficients of linear transformation, e.g., loadings (in PCA).
<code>is.cov</code>	logical, whether the input matrix is a covariance matrix (or a Gram matrix).

**Value**

a numeric value between 0 and 1, the proportion of total variance in `x` explained by the PCs whose loadings are in `v`.

**References**

Shen, H., & Huang, J. Z. (2008). "Sparse principal component analysis via regularized low rank matrix approximation." *Journal of multivariate analysis*, 99(6), 1015-1034.

**Examples**

```
## use the "swiss" data
## find two sparse PCs
s.sca <- sca(swiss, 2, gamma = sqrt(ncol(swiss)))
ld <- loadings(s.sca)
pve(as.matrix(swiss), ld)
```

---

rootmatrix	<i>Find root matrix</i>
------------	-------------------------

---

**Description**

Find the root matrix ( $x$ ) from the Gram matrix (i.e., `crossprod(x)`). This is also useful when the input is a covariance matrix, up to a scaling factor of  $n-1$ , where  $n$  is the sample size.

**Usage**

```
rootmatrix(x)
```

**Arguments**

$x$  a symmetric matrix (will trigger error if not symmetric).

---

rotation	<i>Varimax Rotation</i>
----------	-------------------------

---

**Description**

Perform varimax rotation. Flip the signs of columns so that the resulting matrix is positive-skewed.

**Usage**

```
rotation(
  x,
  rotate = c("varimax", "absmin"),
  normalize = FALSE,
  flip = TRUE,
  eps = 1e-06
)
```

**Arguments**

$x$  a matrix or Matrix.

rotate character(1), rotation method. Two options are currently available: "varimax" (default) or "absmin" (see details).

normalize logical, whether to rows normalization should be done before and undone afterward the rotation (see details).

flip logical, whether to flip the signs of the columns of estimates such that all columns are positive-skewed (see details).

eps numeric precision tolerance.

## Details

`rotate`: The `rotate` option specifies the rotation technique to use. Currently, there are two build-in options—“`varimax`” and “`absmin`”. The “`varimax`” rotation maximizes the element-wise L4 norm of the rotated matrix. It is faster and computationally more stable. The “`absmin`” rotation minimizes the absolute sum of the rotated matrix. It is sharper (as it directly minimizes the L1 norm) but slower and computationally less stable.

`normalize`: The argument `normalize` gives an indication of if and how any normalization should be done before rotation, and then undone after rotation. If `normalize` is `FALSE` (the default) no normalization is done. If `normalize` is `TRUE` then Kaiser normalization is done. (So squared row entries of normalized  $x$  sum to 1.0. This is sometimes called Horst normalization.) For `rotate="absmin"`, if `normalize` is a vector of length equal to the number of indicators (i.e., the number of rows of  $x$ ), then the columns are divided by `normalize` before rotation and multiplied by `normalize` after rotation. Also, If `normalize` is a function then it should take  $x$  as an argument and return a vector which is used like the vector above.

`flip`: The argument `flip` gives an indication of if and the columns of estimated sparse component should be flipped. Note that the estimated (sparse) loadings, i.e., the weights on original variables, are column-wise invariant to a sign flipping. This is because flipping of a principal direction does not influence the amount of the explained variance by the component. If `flip=TRUE`, then the columns of loadings will be flip accordingly, such that each column is positive-skewed. This means that for each column, the sum of cubic elements (i.e.,  $\sum(x^3)$ ) are non-negative.

## Value

the rotated matrix of the same dimension as  $x$ .

## References

Chen, F. and Rohe, K. (2020) "A New Basis for Sparse Principal Component Analysis."

## See Also

[prs](#), [varimax](#)

## Examples

```
## use the "swiss" data
fa <- factanal( ~., 2, data = swiss, rotation = "none")
rotation(loadings(fa))
```

## Description

`sca` performs sparse principal components analysis on the given numeric data matrix. Choices of rotation techniques and shrinkage operators are available.

**Usage**

```
sca(
  x,
  k = min(5, dim(x)),
  gamma = NULL,
  is.cov = FALSE,
  rotate = c("varimax", "absmin"),
  shrink = c("soft", "hard"),
  center = TRUE,
  scale = FALSE,
  normalize = FALSE,
  order = TRUE,
  flip = TRUE,
  max.iter = 1000,
  epsilon = 1e-05,
  quiet = TRUE
)
```

**Arguments**

<code>x</code>	matrix or Matrix to be analyzed.
<code>k</code>	integer, rank of approximation.
<code>gamma</code>	numeric(1), sparsity parameter, default to $\sqrt{pk}$ , where $n \times p$ is the dimension of <code>x</code> .
<code>is.cov</code>	logical, default to FALSE, whether the <code>x</code> is a covariance matrix (or Gram matrix, i.e., <code>crossprod()</code> of some design matrix). If TRUE, both <code>center</code> and <code>scale</code> will be ignored/skipped.
<code>rotate</code>	character(1), rotation method. Two options are currently available: "varimax" (default) or "absmin" (see details).
<code>shrink</code>	character(1), shrinkage method, either "soft"- (default) or "hard"-thresholding (see details).
<code>center</code>	logical, whether to center columns of <code>x</code> (see <a href="#">scale()</a> ).
<code>scale</code>	logical, whether to scale columns of <code>x</code> (see <a href="#">scale()</a> ).
<code>normalize</code>	logical, whether to rows normalization should be done before and undone afterward the rotation (see details).
<code>order</code>	logical, whether to re-order the columns of the estimates (see Details below).
<code>flip</code>	logical, whether to flip the signs of the columns of estimates such that all columns are positive-skewed (see details).
<code>max.iter</code>	integer, maximum number of iteration (default to 1,000).
<code>epsilon</code>	numeric, tolerance of convergence precision (default to 0.00001).
<code>quiet</code>	logical, whether to mute the process report (default to TRUE)

## Details

**rotate:** The `rotate` option specifies the rotation technique to use. Currently, there are two build-in options—“`varimax`” and “`absmin`”. The “`varimax`” rotation maximizes the element-wise L4 norm of the rotated matrix. It is faster and computationally more stable. The “`absmin`” rotation minimizes the absolute sum of the rotated matrix. It is sharper (as it directly minimizes the L1 norm) but slower and computationally less stable.

**shrink:** The `shrink` option specifies the shrinkage operator to use. Currently, there are two build-in options—“`soft`”- and “`hard`”-thresholding. The “`soft`”-thresholding universally reduce all elements and sets the small elements to zeros. The “`hard`”-thresholding only sets the small elements to zeros.

**normalize:** The argument `normalize` gives an indication of if and how any normalization should be done before rotation, and then undone after rotation. If `normalize` is `FALSE` (the default) no normalization is done. If `normalize` is `TRUE` then Kaiser normalization is done. (So squared row entries of normalized  $x$  sum to 1.0. This is sometimes called Horst normalization.) For `rotate="absmin"`, if `normalize` is a vector of length equal to the number of indicators (i.e., the number of rows of  $x$ ), then the columns are divided by `normalize` before rotation and multiplied by `normalize` after rotation. Also, If `normalize` is a function then it should take  $x$  as an argument and return a vector which is used like the vector above.

**order:** In PCA (and SVD), the principal components (and the singular vectors) are ordered. For this, we order the sparse components (i.e., the columns of  $z$  or  $y$ ) by their explained variance in the data, which is defined as  $\sum(x \% \% y)^2$ , where  $y$  is a column of the sparse component. Note: not to be confused with the cumulative proportion of variance explained by  $y$  (and  $z$ ), particularly when  $y$  (and  $z$ ) is may not be strictly orthogonal.

**flip:** The argument `flip` gives an indication of if and the columns of estimated sparse component should be flipped. Note that the estimated (sparse) loadings, i.e., the weights on original variables, are column-wise invariant to a sign flipping. This is because flipping of a principal direction does not influence the amount of the explained variance by the component. If `flip=TRUE`, then the columns of loadings will be flip accordingly, such that each column is positive-skewed. This means that for each column, the sum of cubic elements (i.e.,  $\sum(x^3)$ ) are non-negative.

## Value

an `sca` object that contains:

<code>loadings</code>	matrix, sparse loadings of PCs.
<code>scores</code>	an $n \times k$ matrix, the component scores, calculated using centered (and/or scaled) $x$ . This will only be available when <code>is.cov = FALSE</code> .
<code>cpve</code>	a numeric vector of length $k$ , cumulative proportion of variance in $x$ explained by the top PCs (after center and/or scale).
<code>center</code>	logical, this records the center parameter.
<code>scale</code>	logical, this records the scale parameter.
<code>n.iter</code>	integer, number of iteration taken.
<code>n.obs</code>	integer, sample size, that is, <code>nrow(x)</code> .

## References

Chen, F. and Rohe, K. (2020) "A New Basis for Sparse Principal Component Analysis."

**See Also**[sma](#), [prs](#)**Examples**

```
## ----- example 1 -----
## simulate a low-rank data matrix with some additive Gaussian noise
n <- 300
p <- 50
k <- 5 ## rank
z <- shrinkage(polar(matrix(runif(n * k), n, k)), sqrt(n))
b <- diag(5) * 3
y <- shrinkage(polar(matrix(runif(p * k), p, k)), sqrt(p))
e <- matrix(rnorm(n * p, sd = .01), n, p)
x <- scale(z %*% b %*% t(y) + e)

## perform sparse PCA
s.sca <- sca(x, k)
s.sca

## ----- example 2 -----
## use the `pitprops` data from the `elasticnet` package
data(pitprops)

## find 6 sparse PCs
s.sca <- sca(pitprops, 6, gamma = 6, is.cov = TRUE)
print(s.sca, verbose = TRUE)
```

shrinkage

*Shrinkage***Description**

Shrink a matrix using soft-thresholding or hard-thresholding.

**Usage**

```
shrinkage(x, gamma, shrink = c("soft", "hard"), epsilon = 1e-11)
```

**Arguments**

x	matrix or Matrix, to be threshold.
gamma	numeric, the constraint of Lp norm, i.e. $\ x\  \leq \gamma$ .
shrink	character(1), shrinkage method, either "soft"- (default) or "hard"-thresholding (see details).
epsilon	numeric, precision tolerance. This should be greater than <code>.Machine\$double.eps</code> .

## Details

A binary search to find the cut-off value.

`shrink`: The `shrink` option specifies the shrinkage operator to use. Currently, there are two build-in options—“soft”- and “hard”-thresholding. The “soft”-thresholding universally reduce all elements and sets the small elements to zeros. The “hard”-thresholding only sets the small elements to zeros.

## Value

a list with two components:

<code>matrix</code>	matrix, the matrix that results from soft-thresholding
<code>norm</code>	numeric, the norm of the matrix after soft-thresholding. This value is close to constraint if using the second option.

## References

Chen, F. and Rohe, K. (2020) "A New Basis for Sparse Principal Component Analysis."

## See Also

[prs](#)

## Examples

```
x <- matrix(1:6, nrow = 3)
shrink_x <- shrinkage(x, 1)
```

---

sma

*Sparse Matrix Approximation*

---

## Description

Perform the sparse matrix approximation (SMA) of a data matrix  $x$  as three multiplicative components:  $z$ ,  $b$ , and  $t(y)$ , where  $z$  and  $y$  are sparse, and  $b$  is low-rank but not necessarily diagonal.

## Usage

```
sma(
  x,
  k = min(5, dim(x)),
  gamma = NULL,
  rotate = c("varimax", "absmin"),
  shrink = c("soft", "hard"),
  center = FALSE,
  scale = FALSE,
  normalize = FALSE,
```

```

order = FALSE,
flip = FALSE,
max.iter = 1000,
epsilon = 1e-05,
quiet = TRUE
)

```

## Arguments

x	matrix or Matrix to be analyzed.
k	integer, rank of approximation.
gamma	numeric(2), sparsity parameters. If gamma is numeric(1), it is used for both left and right sparsity component (i.e. z and y). If absent, the two parameters are set as (default): $\sqrt{nk}$ and $\sqrt{pk}$ for z and y respectively, where n x p is the dimension of x.
rotate	character(1), rotation method. Two options are currently available: "varimax" (default) or "absmin" (see details).
shrink	character(1), shrinkage method, either "soft"- (default) or "hard"-thresholding (see details).
center	logical, whether to center columns of x (see <a href="#">scale()</a> ).
scale	logical, whether to scale columns of x (see <a href="#">scale()</a> ).
normalize	logical, whether to rows normalization should be done before and undone afterward the rotation (see details).
order	logical, whether to re-order the columns of the estimates (see Details below).
flip	logical, whether to flip the signs of the columns of estimates such that all columns are positive-skewed (see details).
max.iter	integer, maximum number of iteration (default to 1,000).
epsilon	numeric, tolerance of convergence precision (default to 0.00001).
quiet	logical, whether to mute the process report (default to TRUE)

## Details

**rotate:** The rotate option specifies the rotation technique to use. Currently, there are two build-in options—"varimax" and "absmin". The "varimax" rotation maximizes the element-wise L4 norm of the rotated matrix. It is faster and computationally more stable. The "absmin" rotation minimizes the absolute sum of the rotated matrix. It is sharper (as it directly minimizes the L1 norm) but slower and computationally less stable.

**shrink:** The shrink option specifies the shrinkage operator to use. Currently, there are two build-in options—"soft"- and "hard"-thresholding. The "soft"-thresholding universally reduce all elements and sets the small elements to zeros. The "hard"-thresholding only sets the small elements to zeros.

**normalize:** The argument normalize gives an indication of if and how any normalization should be done before rotation, and then undone after rotation. If normalize is FALSE (the default) no normalization is done. If normalize is TRUE then Kaiser normalization is done. (So squared row entries of normalized x sum to 1.0. This is sometimes called Horst normalization.) For rotate="absmin", if normalize is a vector of length equal to the number of indicators (i.e., the number of rows of

x), then the columns are divided by `normalize` before rotation and multiplied by `normalize` after rotation. Also, if `normalize` is a function then it should take `x` as an argument and return a vector which is used like the vector above.

`order`: In PCA (and SVD), the principal components (and the singular vectors) are ordered. For this, we order the sparse components (i.e., the columns of `z` or `y`) by their explained variance in the data, which is defined as  $\sum(x \% \% y)^2$ , where `y` is a column of the sparse component. Note: not to be confused with the cumulative proportion of variance explained by `y` (and `z`), particularly when `y` (and `z`) is may not be strictly orthogonal.

`flip`: The argument `flip` gives an indication of if and the columns of estimated sparse component should be flipped. Note that the estimated (sparse) loadings, i.e., the weights on original variables, are column-wise invariant to a sign flipping. This is because flipping of a principal direction does not influence the amount of the explained variance by the component. If `flip=TRUE`, then the columns of loadings will be flip accordingly, such that each column is positive-skewed. This means that for each column, the sum of cubic elements (i.e.,  $\sum(x^3)$ ) are non-negative.

## Value

an `sma` object that contains:

`z`, `b`, `t(y)`      the three parts in the SMA. `z` is a sparse  $n \times k$  matrix that contains the row components (loadings). The row names of `z` inherit the row names of `x`. `b` is a  $k \times k$  matrix that contains the scores of SMA; the Frobenius norm of `b` equals to the total variance explained by the SMA. `y` is a sparse  $n \times k$  matrix that contains the column components (loadings).

The row names of `y` inherit the column names of `x`.

`score`              the total variance explained by the SMA. This is the optimal objective value obtained.

`n.iter`             integer, the number of iteration taken.

## References

Chen, F. and Rohe, K. (2020) "A New Basis for Sparse Principal Component Analysis."

## See Also

[sca](#), [prs](#)

## Examples

```
## simulate a rank-5 data matrix with some additive Gaussian noise
n <- 300
p <- 50
k <- 5 ## rank
z <- shrinkage(polar(matrix(runif(n * k), n, k)), sqrt(n))
b <- diag(5) * 3
y <- shrinkage(polar(matrix(runif(p * k), p, k)), sqrt(p))
e <- matrix(rnorm(n * p, sd = .01), n, p)
x <- scale(z \% \% b \% \% t(y) + e)
```

```
## perform sparse matrix approximation
s.sma <- sma(x, k)
s.sma
```

---

soft	<i>Soft-thresholding</i>
------	--------------------------

---

### Description

Perform soft-thresholding given the cut-off value.

### Usage

```
soft(x, t)
```

### Arguments

x	any numerical matrix or vector.
t	numeric, the amount to soft-threshold, i.e., $sgn(x_{ij})( x_{ij} - t )_+$ .

---

varimax	<i>Varimax Rotation</i>
---------	-------------------------

---

### Description

This is a re-implementation of `stats::varimax`, which (1) adds a parameter for the maximum number of iterations, (2) sets the default normalize parameter to FALSE, (3) outputs the number of iteration taken, and (4) returns regular matrix rather than in loadings class.

### Usage

```
varimax(x, normalize = FALSE, eps = 1e-05, maxit = 1000L)
```

### Arguments

x	A loadings matrix, with $p$ rows and $k < p$ columns
normalize	logical. Should Kaiser normalization be performed? If so the rows of x are re-scaled to unit length before rotation, and scaled back afterwards.
eps	The tolerance for stopping: the relative change in the sum of singular values.
maxit	integer, maximum number of iteration (default to 1,000).

**Value**

A list with three elements:

rotated	the rotated matrix.
rotmat	the (orthogonal) rotation matrix.
n.iter	the number of iterations taken.

**See Also**

[stats::varimax](#)

---

varimax.criteria	<i>The varimax criterion</i>
------------------	------------------------------

---

**Description**

Calculate the varimax criterion

**Usage**

```
varimax.criteria(x)
```

**Arguments**

x a matrix or Matrix.

**Value**

a numeric of evaluated varimax criterion.

**References**

[Varimax rotation \(Wikipedia\)](#)

**Examples**

```
## use the "swiss" data
fa <- factanal( ~., 2, data = swiss, rotation = "none")
lds <- loadings(fa)

## compute varimax criterion:
varimax.criteria(lds)

## compute varimax criterion (after the varimax rotation):
rlds <- rotation(lds, rotate = "varimax")
varimax.criteria(rlds)
```

---

`vgQ.absmin`*Gradient of Absmin Criterion*

---

**Description**

This is a helper function for `absmin` and is not to be used directly by users.

**Usage**

```
vgQ.absmin(x)
```

**Arguments**

`x` a `matrix` or `Matrix`, initial factor loadings matrix for which the rotation criterion is to be optimized.

**Value**

a list required by `GPArotation::GPForth` for the `absmin` rotation.

**Examples**

```
## Not run:  
## NOT RUN  
## NOT for users to call.  
  
## End(Not run)
```

# Index

- \* **datasets**
  - pitprops, 10
- \* **package**
  - epca-package, 2
- absmin, 3, 3, 25
- absmin.criteria, 3, 3
- base::rank(), 8
- clue::solve\_LSAP, 5
- clue::solve\_LSAP(), 5
- cpve, 4
- dist.matrix, 5
- distance, 6
- epca-package, 2
- exp.frac, 6
- hard, 7
- inner, 7
- labelCluster, 8
- misClustRate, 8
- norm.Lp, 9
- permColumn, 10
- pitprops, 10
- polar, 11, 14
- print, 12
- print.sca, 11
- print.sma, 12
- prs, 12, 16, 19, 20, 22
- prs(), 11
- pve, 4, 14
- rootmatrix, 15
- rotation, 14, 15
- sca, 14, 16, 22
- scale(), 17, 21
- shrinkage, 14, 19
- shrinkage(), 9
- sma, 14, 19, 20
- sma(), 12
- soft, 23
- stats::varimax, 23, 24
- varimax, 16, 23
- varimax.criteria, 24
- vgQ.absmin, 25