Package ‘pcaPP’

April 23, 2021

Version 1.9-74

Date 2021-04-22

VersionNote Released 1.9-73 on 2018-01-04 on CRAN

Title Robust PCA by Projection Pursuit

Author Peter Filzmoser, Heinrich Fritz, Klaudius Kalcher

Maintainer Valentin Todorov <valentin.todorov@chello.at>

Imports mvtnorm

Suggests robustbase

Description Provides functions for robust PCA by projection pursuit.
The methods are described in Croux et al. (2006) <doi:10.2139/ssrn.968376>,
Croux et al. (2013) <doi:10.1080/00401706.2012.727746>,
Todorov and Filzmoser (2013) <doi:10.1007/978-3-642-33042-1_31>.

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-04-23 04:40:02 UTC

R topics documented:

`cor.fk` *Fast estimation of Kendall's tau rank correlation coefficient*

Description

Calculates Kendall's tau rank correlation coefficient in O (n log (n)) rather than O (n\^2) as in the current R implementation.

Usage

```
cor.fk (x, y = NULL)
```

Arguments

`x`	A vector, a matrix or a data frame of data.
`y`	A vector of data.

Details

The code of this implementation of the fast Kendall's tau correlation algorithm has originally been published by David Simcha. Due to it's runtime (`O(n log n)`) it's essentially faster than the current R implementation (`O (n^2)`), especially for large numbers of observations. The algorithm goes back to Knight (1966) and has been described more detailed by Abrevaya (1999) and Christensen (2005).

Value

The estimated correlation coefficient.

Author(s)

David Simcha, Heinrich Fritz, Christophe Croux, Peter Filzmoser <`<P.Filzmoser@tuwien.ac.at>`>

References

Knight, W. R. (1966). A Computer Method for Calculating Kendall's Tau with Ungrouped Data. Journal of the American Statistical Association, **314**(61) Part 1, 436-439.
Christensen D. (2005). Fast algorithms for the calculation of Kendall's Tau. Journal of Computational Statistics **20**, 51-62.
Abrevaya J. (1999). Computation of the Maximum Rank Correlation Estimator. Economic Letters **62**, 279-285.

See Also

cor

Examples

```
  set.seed (100) ## creating test data
  n <- 1000
  x <- rnorm (n)
  y <- x+  rnorm (n)

  tim <- proc.time ()[1] ## applying cor.fk
  cor.fk (x, y)
  cat ("cor.fk runtime [s]:", proc.time ()[1] - tim, "(n =", length (x), ")\n")

  tim <- proc.time ()[1] ## applying cor (standard R implementation)
  cor (x, y, method = "kendall")
  cat ("cor runtime [s]:", proc.time ()[1] - tim, "(n =", length (x), ")\n")

## applying cor and cor.fk on data containing

  Xt <- cbind (c (x, as.integer (x)), c (y, as.integer (y)))

  tim <- proc.time ()[1] ## applying cor.fk
  cor.fk (Xt)
  cat ("cor.fk runtime [s]:", proc.time ()[1] - tim, "(n =", nrow (Xt), ")\n")

  tim <- proc.time ()[1] ## applying cor (standard R implementation)
  cor (Xt, method = "kendall")
  cat ("cor runtime [s]:", proc.time ()[1] - tim, "(n =", nrow (Xt), ")\n")
```

---

covPC *Covariance Matrix Estimation from princomp Object*

Description

computes the covariance matrix from a princomp object. The number of components k can be given as input.

Usage

```
covPC(x, k, method)
```

Arguments

x	an object of class princomp.
k	number of PCs to use for covariance estimation (optional).
method	method how the PCs have been estimated (optional).

Details

There are several possibilities to estimate the principal components (PCs) from an input data matrix, including the functions PCAproj and PCAgrid. This function uses the estimated PCs to reconstruct the covariance matrix. Not all PCs have to be used, the number k of PCs (first k PCs) can be given as input to the function.

Value

cov	the estimated covariance matrix
center	the center of the data, as provided from the princomp object.
method	a string describing the method that was used to calculate the PCs.

Author(s)

Heinrich Fritz, Peter Filzmoser <<P.Filzmoser@tuwien.ac.at>>

References

C. Croux, P. Filzmoser, M. Oliveira, (2007). Algorithms for Projection-Pursuit Robust Principal Component Analysis, *Chemometrics and Intelligent Laboratory Systems*, Vol. 87, pp. 218-225.

See Also

PCAgrid, PCAproj, princomp

Examples

```
# multivariate data with outliers
library(mvtnorm)
x <- rbind(rmvnorm(200, rep(0, 6), diag(c(5, rep(1,5)))),
           rmvnorm( 15, c(0, rep(20, 5)), diag(rep(1, 6))))
pc <- princomp(x)
covPC(pc, k=2)
```

---

covPCA *Robust Covariance Matrix Estimation*

Description

computes the robust covariance matrix using the PCAgrid and PCAproj functions.

Usage

```
covPCAproj(x, control)
covPCAgrid(x, control)
```

Arguments

x — a numeric matrix or data frame which provides the data.

control — a list whose elements must be the same as (or a subset of) the parameters of the appropriate PCA function (PCAgrid or PCAproj).

Details

The functions covPCAproj and covPCAgrid use the functions PCAproj and PCAgrid respectively to estimate the covariance matrix of the data matrix x.

Value

cov — the actual covariance matrix estimated from x

center — the center of the data x that was substracted from them before the PCA algorithms were run.

method — a string describing the method that was used to calculate the covariance matrix estimation

Author(s)

Heinrich Fritz, Peter Filzmoser <<P.Filzmoser@tuwien.ac.at>>

References

C. Croux, P. Filzmoser, M. Oliveira, (2007). Algorithms for Projection-Pursuit Robust Principal Component Analysis, *Chemometrics and Intelligent Laboratory Systems*, Vol. 87, pp. 218-225.

See Also

PCAgrid, ScaleAdv, princomp

Examples

```
# multivariate data with outliers
library(mvtnorm)
x <- rbind(rmvnorm(200, rep(0, 6), diag(c(5, rep(1,5)))),
           rmvnorm( 15, c(0, rep(20, 5)), diag(rep(1, 6))))
covPCAproj(x)
# compare with classical covariance matrix:
cov(x)
```

`data.Zou` *Test Data Generation for Sparse PCA examples*

Description

Draws a sample data set, as introduced by Zou et al. (2006).

Usage

```
data.Zou (n = 250, p =  c(4, 4, 2), ...)
```

Arguments

`n`	The required number of observations.
`p`	A vector of length 3, specifying how many variables shall be constructed using the three factors V1, V2 and V3.
`...`	Further arguments passed to or from other functions.

Details

This data set has been introduced by Zou et al. (2006), and then been referred to several times, e.g. by Farcomeni (2009), Guo et al. (2010) and Croux et al. (2011).

The data set contains two latent factors V1 ~ N(0, 290) and V2 ~ N(0, 300) and a third mixed component V3 = -0.3 V1 + 0.925V2 + e; e ~ N(0, 1).
The ten variables Xi of the original data set are constructed the following way:
Xi = V1 + ei; i = 1, 2, 3, 4
Xi = V2 + ei; i = 5, 6, 7, 8
Xi = V3 + ei; i = 9, 10
whereas ei ~ N(0, 1) is indepependent for i = 1 , ..., 10

Value

A matrix of dimension `n x sum (p)` containing the generated sample data set.

Author(s)

Heinrich Fritz, Peter Filzmoser <`P.Filzmoser@tuwien.ac.at`>

References

C. Croux, P. Filzmoser, H. Fritz (2011). Robust Sparse Principal Component Analysis Based on Projection-Pursuit, *??* To appear.

A. Farcomeni (2009). An exact approach to sparse principal component analysis, *Computational Statistics*, Vol. 24(4), pp. 583-604.

J. Guo, G. James, E. Levina, F. Michailidis, and J. Zhu (2010). Principal component analysis with sparse fused loadings, *Journal of Computational and Graphical Statistics.* To appear.

H. Zou, T. Hastie, R. Tibshirani (2006). Sparse principal component analysis, *Journal of Computational and Graphical Statistics*, Vol. 15(2), pp. 265-286.

See Also

sPCAgrid, princomp

Examples

```
                 ##  data generation
set.seed (0)
x <- data.Zou ()

                 ##  applying PCA
pc <-  princomp (x)
                 ##  the corresponding non-sparse loadings
unclass (pc$load[,1:3])
pc$sdev[1:3]

                 ##  lambda as calculated in the opt.TPO - example
lambda <- c (0.23, 0.34, 0.005)
                 ##  applying sparse PCA
spc <- sPCAgrid (x, k = 3, lambda = lambda, method = "sd")
unclass (spc$load)
spc$sdev[1:3]

                 ## comparing the non-sparse and sparse biplot
par (mfrow = 1:2)
biplot (pc, main = "non-sparse PCs")
biplot (spc, main = "sparse PCs")
```

---

l1median   *Multivariate L1 Median*

Description

Computes the multivariate L1 median (also called spatial median) of a data matrix.

Usage

```
l1median(X, MaxStep = 200, ItTol = 10^-8, trace = 0, m.init = .colMedians (X))
```

Arguments

X	A matrix containing the values whose multivariate L1 median is to be computed.
MaxStep	The maximum number of iterations.
ItTol	Tolerance for convergence of the algorithm.
trace	The tracing level.
m.init	An initial estimate.

Value

returns the vector of the coordinates of the L1 median.

Author(s)

Heinrich Fritz, Peter Filzmoser <<P.Filzmoser@tuwien.ac.at>>

References

C. Croux, P. Filzmoser, M. Oliveira, (2007). Algorithms for Projection-Pursuit Robust Principal Component Analysis, *Chemometrics and Intelligent Laboratory Systems*, Vol. 87, pp. 218-225.

See Also

median

Examples

```
l1median(rnorm(100), trace = -1) # this returns the median of the sample

# multivariate data with outliers
library(mvtnorm)
x <- rbind(rmvnorm(200, rep(0, 4), diag(c(1, 1, 2, 2))),
           rmvnorm( 50, rep(3, 4), diag(rep(2, 4))))
l1median(x, trace = -1)
# compare with coordinate-wise median:
apply(x,2,median)
```

---

l1median_NLM *Multivariate L1 Median*

Description

Computes the multivariate L1 median (also called spatial median) of a data matrix X.

Usage

```
l1median_NM (X, maxit = 200, tol = 10^-8, trace = 0,
             m.init = .colMedians (X), ...)
l1median_CG (X, maxit = 200, tol = 10^-8, trace = 0,
             m.init = .colMedians (X), ...)
l1median_BFGS (X, maxit = 200, tol = 10^-8, trace = 0,
               m.init = .colMedians (X), REPORT = 10, ...)
l1median_NLM (X, maxit = 200, tol = 10^-8, trace = 0,
              m.init = .colMedians (X), ...)
l1median_HoCr (X, maxit = 200, tol = 10^-8, zero.tol = 1e-15, trace = 0,
               m.init = .colMedians (X), ...)
l1median_VaZh (X, maxit = 200, tol = 10^-8, zero.tol = 1e-15, trace = 0,
       m.init = .colMedians (X), ...)
```

Arguments

X	a matrix of dimension n x p.
maxit	The maximum number of iterations to be performed.
tol	The convergence tolerance.
trace	The tracing level. Set trace > 0 to retrieve additional information on the single iterations.
m.init	A vector of length p containing the initial value of the L1-median.
REPORT	A parameter internally passed to optim.
zero.tol	The zero-tolerance level used in l1median_VaZh and l1median_HoCr for determining the equality of two observations (i.e. an observation and a current center estimate).
...	Further parameters passed from other functions.

Details

The L1-median is computed using the built-in functions optim and nlm. These functions are a transcript of the L1median method of package robustX, porting as much code as possible into C++.

Value

par	A vector of length p containing the L1-median.
value	The value of the objective function ||X -l1median|| which is minimized for finding the L1-median.
code	The return code of the optimization algorithm. See optim and nlm for further information.
iterations	The number of iterations performed.
iterations_gr	When using a gradient function this value holds the number of times the gradient had to be computed.
time	The algorithms runtime in milliseconds.

Note

See the vignette "Compiling pcaPP for Matlab" which comes with this package to compile and use some of these functions in Matlab.

Author(s)

Heinrich Fritz, Peter Filzmoser <<P.Filzmoser@tuwien.ac.at>>

See Also

median

Examples

```
# multivariate data with outliers
library(mvtnorm)
x <- rbind(rmvnorm(200, rep(0, 4), diag(c(1, 1, 2, 2))),
           rmvnorm( 50, rep(3, 4), diag(rep(2, 4))))

l1median_NM (x)$par
l1median_CG (x)$par
l1median_BFGS (x)$par
l1median_NLM (x)$par
l1median_HoCr (x)$par
l1median_VaZh (x)$par

# compare with coordinate-wise median:
apply(x,2,median)
```

---

objplot *Objective Function Plot for Sparse PCs*

Description

Plots an objective function (TPO or BIC) of one or more sparse PCs for a series of lambdas.

Usage

```
objplot (x, k, ...)
```

Arguments

x	An opt.TPO or opt.BIC object.
k	This function displays the objective function's values of the k-th component for opt.TPO-objects, or the first k components for opt.BIC-objects.
...	Further arguments passed to or from other functions.

Details

This function operates on the return object of function opt.TPO or opt.BIC. The model (lambda) selected by the minimization of the corresponding criterion is highlighted by a dashed vertical line.

The component the argument k refers to, corresponds to the $pc.noord item of argument x. For more info on the order of sparse PCs see the details section of opt.TPO.

Author(s)

Heinrich Fritz, Peter Filzmoser <<P.Filzmoser@tuwien.ac.at>>

References

C. Croux, P. Filzmoser, H. Fritz (2011). Robust Sparse Principal Component Analysis Based on Projection-Pursuit, *??* To appear.

See Also

sPCAgrid, princomp

Examples

```
set.seed (0)
                    ##  generate test data
x <- data.Zou (n = 250)

k.max <- 3          ##  max number of considered sparse PCs

                    ##  arguments for the sPCAgrid algorithm
maxiter <- 25       ##    the maximum number of iterations
method <- "sd"      ##    using classical estimations

                    ##  Optimizing the TPO criterion
oTPO <- opt.TPO (x, k.max = k.max, method = method, maxiter = maxiter)

                    ##  Optimizing the BIC criterion
oBIC <- opt.BIC (x, k.max = k.max, method = method, maxiter = maxiter)

        ##  Objective function vs. lambda
par (mfrow = c (2, k.max))
for (i in 1:k.max)        objplot (oTPO, k = i)
for (i in 1:k.max)        objplot (oBIC, k = i)
```

---

opt.TPO *Model Selection for Sparse (Robust) Principal Components*

Description

These functions compute a suggestion for the sparseness parameter lambda which is required by function sPCAgrid. A range of different values for lambda is tested and according to an objective function, the best solution is selected. Two different approaches (TPO and BIC) are available, which is further discussed in the details section. A graphical summary of the optimization can be obtained by plotting the function's return value (plot.opt.TPO, plot.opt.BIC for tradeoff curves or objplot for an objective function plot).

Usage

```
opt.TPO (x, k.max = ncol (x), n.lambda = 30, lambda.max, ...)
opt.BIC (x, k.max = ncol (x), n.lambda = 30, lambda.max, ...)
```

Arguments

- `x` — a numerical matrix or data frame of dimension (`n x p`), which provides the data for the principal components analysis.
- `k.max` — the maximum number of components which shall be considered for optimizing an objective function (optional).
- `n.lambda` — the number of lambdas to be checked for each component (optional).
- `lambda.max` — the maximum value of lambda to be checked (optional). If omitted, the lambda which yields "full sparseness" (i.e. loadings of only zeros and ones) is computed and used as default value.
- `...` — further arguments passed to `sPCAgrid`

Details

The choice for a particular lambda is done by optimizing an objective function, which is calculated for a set of `n.lambda` models with different lambdas, ranging from 0 to `lambda.max`. If `lambda.max` is not specified, the minimum lambda yielding "full sparseness" is used. "Full sparseness" refers to a model with minimum possible absolute sum of loadings, which in general implies only zeros and ones in the loadings matrix.

The user can choose between two optimization methods: TPO (Tradeoff Product Optimization; see below), or the BIC (see Guo et al., 2011; Croux et al., 2011). The main difference is, that optimization based on the BIC always chooses the same lambda for all PCs, and refers to a particular choice of `k`, the number of considered components. TPO however is optimized separately for each component, and so delivers different lambdas within a model and does not depend on a decision on `k`.
This characteristic can be noticed in the return value of the function: `opt.TPO` returns a single model with `k.max` PCs and different values of `lambda` for each PC. On the contrary `opt.BIC` returns `k.max` distinct models with `k.max` different lambdas, whereas for each model a different number of components `k` has been considered for the optimization. Applying the latter method, the user finally has to select one of these `k.max` models manually, e.g. by considering the cumulated explained variance, whereas the TPO method does not require any further decisions.

The tradeoff made in the context of sparse PCA refers to the loss of explained variance vs. the gain of sparseness. TPO (Tradeoff Product Optimization) maximizes the area under the tradeoff curve (see `plot.opt.TPO`), in particular it maximizes the explained variance multiplied by the number of zero loadings of a particular component. As in this context the according criterion is minimized, the negative product is considered.

Note that in the context of sparse PCA, there are two sorting orders of PCs, which must be considered: Either according to the objective function's value, (item `$pc.noord`)or the variance of each PC(item `$pc`). As in none-sparse PCA the objective function is identical to the PCs' variance, this is not an issue there.
The sPCAgrid algorithm delivers the components in decreasing order, according to the objective function (which apart from the variance also includes sparseness terms), whereas the method `sPCAgrid` subsequently re-orders the components according to their explained variance.

Value

The functions return an S3 object of type "opt.TPO" or "opt.BIC" respectively, containing the following items:

pc	An S3 object of type princomp (opt.TPO), or a list of such objects of length k.max (opt.BIC), as returned by sPCAgrid.
pc.noord	An S3 object of type princomp (opt.TPO), or a list of such objects of length k.max (opt.BIC), as returned by sPCAgrid. Here the PCs have not been re-ordered according to their variance, but are still ordered according to their objective function's value as returned by the sPCAgrid - algorithm. This information is used for according tradeoff curves and the objective function plot.
x	The input data matrix as provided by the user.
k.ini, opt	These items contain optimization information, as used in functions plot.opt.TPO, plot.opt.BIC and objplot.

Author(s)

Heinrich Fritz, Peter Filzmoser <<P.Filzmoser@tuwien.ac.at>>

References

C. Croux, P. Filzmoser, H. Fritz (2011). Robust Sparse Principal Component Analysis Based on Projection-Pursuit, *??* To appear.

See Also

sPCAgrid, princomp

Examples

```
set.seed (0)
                    ##  generate test data
x <- data.Zou (n = 250)

k.max <- 3          ##  max number of considered sparse PCs

                    ##  arguments for the sPCAgrid algorithm
maxiter <- 25       ##    the maximum number of iterations
method <- "sd"      ##    using classical estimations

                    ##  Optimizing the TPO criterion
oTPO <- opt.TPO (x, k.max = k.max, method = method, maxiter = maxiter)

oTPO$pc             ##  the model selected by opt.TPO
oTPO$pc$load        ##  and the according sparse loadings.

                    ##  Optimizing the BIC criterion
oBIC <- opt.BIC (x, k.max = k.max, method = method, maxiter = maxiter)

oBIC$pc[[1]]        ##  the first model selected by opt.BIC (k = 1)

        ##  Tradeoff Curves: Explained Variance vs. sparseness
```

```
par (mfrow = c (2, k.max))
for (i in 1:k.max)       plot (oTPO, k = i)
for (i in 1:k.max)       plot (oBIC, k = i)

        ##  Tradeoff Curves: Explained Variance vs. lambda
par (mfrow = c (2, k.max))
for (i in 1:k.max)       plot (oTPO, k = i, f.x = "lambda")
for (i in 1:k.max)       plot (oBIC, k = i, f.x = "lambda")

        ##  Objective function vs. lambda
par (mfrow = c (2, k.max))
for (i in 1:k.max)       objplot (oTPO, k = i)
for (i in 1:k.max)       objplot (oBIC, k = i)
```

PCAgrid *(Sparse) Robust Principal Components using the Grid search algorithm*

Description

Computes a desired number of (sparse) (robust) principal components using the grid search algorithm in the plane. The global optimum of the objective function is searched in planes, not in the p-dimensional space, using regular grids in these planes.

Usage

```
PCAgrid (x, k = 2, method = c ("mad", "sd", "qn"),
         maxiter = 10, splitcircle = 25, scores = TRUE, zero.tol = 1e-16,
 center = l1median, scale, trace = 0, store.call = TRUE, control, ...)

sPCAgrid (x, k = 2, method = c ("mad", "sd", "qn"), lambda = 1,
          maxiter = 10, splitcircle = 25, scores = TRUE, zero.tol = 1e-16,
  center = l1median, scale, trace = 0, store.call = TRUE, control, ...)
```

Arguments

`x`	a numerical matrix or data frame of dimension (n x p)which provides the data for the principal components analysis.
`k`	the desired number of components to compute
`method`	the scale estimator used to detect the direction with the largest variance. Possible values are `"sd"`, `"mad"` and `"qn"`, the latter can be called `"Qn"` too. `"mad"` is the default value.
`lambda`	the sparseness constraint's strength(`sPCAgrid` only). A single value for all components, or a vector of length `k` with different values for each component can be specified. See `opt.TPO` for the choice of this argument.
`maxiter`	the maximum number of iterations.

`splitcircle`	the number of directions in which the algorithm should search for the largest variance. The direction with the largest variance is searched for in the directions defined by a number of equally spaced points on the unit circle. This argument determines, how many such points are used to split the unit circle.
`scores`	A logical value indicating whether the scores of the principal component should be calculated.
`zero.tol`	the zero tolerance used internally for checking convergence, etc.
`center`	this argument indicates how the data is to be centered. It can be a function like `mean` or `median` or a vector of length `ncol(x)` containing the center value of each column.
`scale`	this argument indicates how the data is to be rescaled. It can be a function like `sd` or `mad` or a vector of length `ncol(x)` containing the scale value of each column.
`trace`	an integer value >= 0, specifying the tracing level.
`store.call`	a logical variable, specifying whether the function call shall be stored in the result structure.
`control`	a list which elements must be the same as (or a subset of) the parameters above. If the control object is supplied, the parameters from it will be used and any other given parameters are overridden.
`...`	further arguments passed to or from other functions.

Details

In contrast to `PCAgrid`, the function `sPCAgrid` computes sparse principal components. The strength of the applied sparseness constraint is specified by argument `lambda`.

Similar to the function `princomp`, there is a `print` method for the these objects that prints the results in a nice format and the `plot` method produces a scree plot (`screeplot`). There is also a `biplot` method.

Angle halving is an extension of the original algorithm. In the original algorithm, the search directions are determined by a number of points on the unit circle in the interval [-pi/2 ; pi/2). Angle halving means this angle is halved in each iteration, eg. for the first approximation, the above mentioned angle is used, for the second approximation, the angle is halved to [-pi/4 ; pi/4) and so on. This usually gives better results with less iterations needed.
NOTE: in previous implementations angle halving could be suppressed by the former argument `"anglehalving"`. This still can be done by setting argument `maxiter = 0`.

Value

The function returns an object of class `"princomp"`, i.e. a list similar to the output of the function `princomp`.

`sdev`	the (robust) standard deviations of the principal components.
`loadings`	the matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors). This is of class `"loadings"`: see `loadings` for its `print` method.
`center`	the means that were subtracted.
`scale`	the scalings applied to each variable.

n.obs	the number of observations.
scores	if scores = TRUE, the scores of the supplied data on the principal components.
call	the matched call.
obj	A vector containing the objective functions values. For function PCAgrid this is the same as sdev.
lambda	The lambda each component has been calculated with (sPCAgrid only).

Note

See the vignette "Compiling pcaPP for Matlab" which comes with this package to compile and use these functions in Matlab.

Author(s)

Heinrich Fritz, Peter Filzmoser <<P.Filzmoser@tuwien.ac.at>>

References

C. Croux, P. Filzmoser, M. Oliveira, (2007). Algorithms for Projection-Pursuit Robust Principal Component Analysis, *Chemometrics and Intelligent Laboratory Systems*, Vol. 87, pp. 218-225.

C. Croux, P. Filzmoser, H. Fritz (2011). Robust Sparse Principal Component Analysis Based on Projection-Pursuit, *??* To appear.

See Also

PCAproj, princomp

Examples

```
# multivariate data with outliers
library(mvtnorm)
x <- rbind(rmvnorm(200, rep(0, 6), diag(c(5, rep(1,5)))),
           rmvnorm( 15, c(0, rep(20, 5)), diag(rep(1, 6))))
# Here we calculate the principal components with PCAgrid
pc <- PCAgrid(x)
# we could draw a biplot too:
biplot(pc)
# now we want to compare the results with the non-robust principal components
pc <- princomp(x)
# again, a biplot for comparison:
biplot(pc)

##  Sparse loadings
set.seed (0)
x <- data.Zou ()

                 ##  applying PCA
pc <-  princomp (x)
                 ##  the corresponding non-sparse loadings
unclass (pc$load[,1:3])
```

```
pc$sdev[1:3]

                 ##  lambda as calculated in the opt.TPO - example
lambda <- c (0.23, 0.34, 0.005)
                 ##  applying sparse PCA
spc <- sPCAgrid (x, k = 3, lambda = lambda, method = "sd")
unclass (spc$load)
spc$sdev[1:3]

                 ## comparing the non-sparse and sparse biplot
par (mfrow = 1:2)
biplot (pc, main = "non-sparse PCs")
biplot (spc, main = "sparse PCs")
```

PCAproj — *Robust Principal Components using the algorithm of Croux and Ruiz-Gazen (2005)*

Description

Computes a desired number of (robust) principal components using the algorithm of Croux and Ruiz-Gazen (JMVA, 2005).

Usage

```
PCAproj(x, k = 2, method = c("mad", "sd", "qn"), CalcMethod = c("eachobs",
"lincomb", "sphere"), nmax = 1000, update = TRUE, scores = TRUE, maxit = 5,
maxhalf = 5, scale = NULL, center = l1median_NLM, zero.tol = 1e-16, control)
```

Arguments

`x`	a numeric matrix or data frame which provides the data for the principal components analysis.
`k`	desired number of components to compute
`method`	scale estimator used to detect the direction with the largest variance. Possible values are `"sd"`, `"mad"` and `"qn"`, the latter can be called `"Qn"` too. `"mad"` is the default value.
`CalcMethod`	the variant of the algorithm to be used. Possible values are `"eachobs"`, `"lincomb"` and `"sphere"`, with `"eachobs"` being the default.
`nmax`	maximum number of directions to search in each step (only when using `"sphere"` or `"lincomb"` as the `CalcMethod`).
`update`	a logical value indicating whether an update algorithm should be used.
`scores`	a logical value indicating whether the scores of the principal component should be calculated.
`maxit`	maximim number of iterations.
`maxhalf`	maximum number of steps for angle halving.

`scale`	this argument indicates how the data is to be rescaled. It can be a function like `sd` or `mad` or a vector of length `ncol(x)` containing the scale value of each column.
`center`	this argument indicates how the data is to be centered. It can be a function like `mean` or `median` or a vector of length `ncol(x)` containing the center value of each column.
`zero.tol`	the zero tolerance used internally for checking convergence, etc.
`control`	a list which elements must be the same as (or a subset of) the parameters above. If the control object is supplied, the parameters from it will be used and any other given parameters are overridden.

Details

Basically, this algrithm considers the directions of each observation through the origin of the centered data as possible projection directions. As this algorithm has some drawbacks, especially if `ncol(x) > nrow(x)` in the data matrix, there are several improvements that can be used with this algorithm.

- updateAn updating step basing on the algorithm for finding the eigenvectors is added to the algorithm. This can be used with any `CalcMethod`
- sphereAdditional search directions are added using random directions. The random directions are determined using random data points generated from a p-dimensional multivariate standard normal distribution. These new data points are projected to the unit sphere, giving the new search directions.
- lincombAdditional search directions are added using linear combinations of the observations. It is similar to the `"sphere"`-algorithm, but the new data points are generated using linear combinations of the original data `b_1*x_1 + ... + b_n*x_n` where the coefficients `b_i` come from a uniform distribution in the interval `[0,1]`.

Similar to the function `princomp`, there is a `print` method for the these objects that prints the results in a nice format and the `plot` method produces a scree plot (`screeplot`). There is also a `biplot` method.

Value

The function returns a list of class `"princomp"`, i.e. a list similar to the output of the function `princomp`.

`sdev`	the (robust) standard deviations of the principal components.
`loadings`	the matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors). This is of class `"loadings"`: see `loadings` for its `print` method.
`center`	the means that were subtracted.
`scale`	the scalings applied to each variable.
`n.obs`	the number of observations.
`scores`	if `scores = TRUE`, the scores of the supplied data on the principal components.
`call`	the matched call.

Author(s)

Heinrich Fritz, Peter Filzmoser <<P.Filzmoser@tuwien.ac.at>>

References

C. Croux, P. Filzmoser, M. Oliveira, (2007). Algorithms for Projection-Pursuit Robust Principal Component Analysis, *Chemometrics and Intelligent Laboratory Systems*, Vol. 87, pp. 218-225.

See Also

PCAgrid, ScaleAdv, princomp

Examples

```
# multivariate data with outliers
library(mvtnorm)
x <- rbind(rmvnorm(200, rep(0, 6), diag(c(5, rep(1,5)))),
           rmvnorm( 15, c(0, rep(20, 5)), diag(rep(1, 6))))
# Here we calculate the principal components with PCAgrid
pc <- PCAproj(x, 6)
# we could draw a biplot too:
biplot(pc)

# we could use another calculation method and another objective function, and
# maybe only calculate the first three principal components:
pc <- PCAproj(x, 3, "qn", "sphere")
biplot(pc)

# now we want to compare the results with the non-robust principal components
pc <- princomp(x)
# again, a biplot for comparision:
biplot(pc)
```

PCdiagplot *Diagnostic plot for principal components*

Description

Computes Orthogonal Distances (OD) and Score Distances (SD) for already computed principal components using the projection pursuit technique.

Usage

```
PCdiagplot(x, PCobj, crit = c(0.975, 0.99, 0.999), ksel = NULL, plot = TRUE,
           plotbw = TRUE, raw = FALSE, colgrid = "black", ...)
```

Arguments

`x`	a numeric matrix or data frame which provides the data for the principal components analysis.
`PCobj`	a PCA object resulting from `PCAproj` or `PCAgrid`
`crit`	quantile(s) used for the critical value(s) for OD and SD
`ksel`	range for the number of PCs to be used in the plot; if NULL all PCs provided are used
`plot`	if TRUE a plot is generated, otherwise only the values are returned
`plotbw`	if TRUE the plot uses gray, otherwise color representation
`raw`	if FALSE, the distribution of the SD will be transformed to approach chisquare distribution, otherwise the raw values are reported and used for plotting
`colgrid`	the color used for the grid lines in the plot
`...`	additional graphics parameters as used in `par`

Details

Based on (robust) principal components, a diagnostics plot is made using Orthogonal Distance (OD) and Score Distance (SD). This plot can provide important information about the multivariate data structure.

Value

`ODist`	matrix with OD for each observation (rows) and each selected PC (cols)
`SDist`	matrix with SD for each observation (rows) and each selected PC (cols)
`critOD`	matrix with critical values for OD for each selected PC (rows) and each critical value (cols)
`critSD`	matrix with critical values for SD for each selected PC (rows) and each critical value (cols)

Author(s)

Peter Filzmoser <`<P.Filzmoser@tuwien.ac.at>`>

References

P. Filzmoser and H. Fritz (2007). Exploring high-dimensional data with robust principal components. In S. Aivazian, P. Filzmoser, and Yu. Kharin, editors, Proceedings of the Eighth International Conference on Computer Data Analysis and Modeling, volume 1, pp. 43-50, Belarusian State University, Minsk.

M. Hubert, P.J. Rousseeuwm, K. Vanden Branden (2005). ROBCA: a new approach to robust principal component analysis Technometrics 47, pp. 64-79.

See Also

`PCAproj`, `PCAgrid`

Examples

```
# multivariate data with outliers
library(mvtnorm)
x <- rbind(rmvnorm(85, rep(0, 6), diag(c(5, rep(1,5)))),
           rmvnorm( 15, c(0, rep(20, 5)), diag(rep(1, 6))))
# Here we calculate the principal components with PCAgrid
pcrob <- PCAgrid(x, k=6)
resrob <- PCdiagplot(x,pcrob,plotbw=FALSE)

# compare with classical method:
pcclass <- PCAgrid(x, k=6, method="sd")
resclass <- PCdiagplot(x,pcclass,plotbw=FALSE)
```

---

plot.opt.TPO *Tradeoff Curves for Sparse PCs*

Description

Tradeoff curves of one or more sparse PCs for a series of lambdas, which contrast the loss of explained variance and the gain of sparseness.

Usage

```
## S3 method for class 'opt.TPO'
 plot(x, k, f.x = c ("l0", "pl0", "l1", "pl1", "lambda"),
                         f.y = c ("var", "pvar"), ...)
## S3 method for class 'opt.BIC'
 plot(x, k, f.x = c ("l0", "pl0", "l1", "pl1", "lambda"),
                         f.y = c ("var", "pvar"), ...)
```

Arguments

x	An opt.TPO or opt.BIC object.
k	This function plots the tradeoff curve of the k-th component for opt.TPO-objects, or the first k components for opt.BIC-objects.
f.x, f.y	A string, specifying which information shall be plotted on the x and y - axis. See the details section for more information.
...	Further arguments passed to or from other functions.

Details

The argument f.x can obtain the following values:

- "l0": l0 - sparseness, which corresponds to the number of zero loadings of the considered component(s).
- "pl0": l0 - sparseness in percent (l0 - sparseness ranges from 0 to p-1 for each component).

- "l1": l1 - sparseness, which corresponds to the negative sum of absolute loadings of the considered component(s).
  (The exact value displayed for a single component is sqrt (p) -S, with S as the the absolute sum of loadings.)
  As this value is a part of the objective function which selects the candidate directions within the sPCAgrid function, this option is provided here.
- "pl1" The "l1 - sparseness" in percent (l1 - sparseness ranges from 0 to sqrt (p-1) for each component).
- "lambda": The lambda used for computing a particular model.

The argument f.y can obtain the following values:

- "var": The (cumulated) explained variance of the considered component(s). The value shown here is calculated using the variance estimator specified via the method argument of function sPCAgrid.
- "pvar": The (cumulated) explained variance of the considered component(s) in percent. The 100%-level is assumed as the sum of variances of all columns of argument x.
  Again the same variance estimator is used as specified via the method argument of function sPCAgrid.

The subtitle summarizes the result of the applied criterion for selecting a value of lambda:

- The name of the applied method (TPO/BIC).
- The selected value of lambda for the k-th component (opt.TPO) or all computed components (opt.BIC).
- The empirical cumulated variance (ECV) of the first k components in percent.
- The obtained l0-sparseness of the first k components.

This function operates on the return object of function opt.TPO or opt.BIC. The model (lambda) selected by the minimization of the corresponding criterion is highlighted by a dashed vertical line.

The component the argument k refers to, corresponds to the $pc.noord item of argument x. For more info on the order of sparse PCs see the details section of opt.TPO.

Author(s)

Heinrich Fritz, Peter Filzmoser <<P.Filzmoser@tuwien.ac.at>>

References

C. Croux, P. Filzmoser, H. Fritz (2011). Robust Sparse Principal Component Analysis Based on Projection-Pursuit, *??* To appear.

See Also

sPCAgrid, princomp

Examples

```
set.seed (0)
                    ##  generate test data
x <- data.Zou (n = 250)

k.max <- 3          ##  max number of considered sparse PCs

                    ##  arguments for the sPCAgrid algorithm
maxiter <- 25       ##    the maximum number of iterations
method <- "sd"      ##    using classical estimations

                    ##  Optimizing the TPO criterion
oTPO <- opt.TPO (x, k.max = k.max, method = method, maxiter = maxiter)

                    ##  Optimizing the BIC criterion
oBIC <- opt.BIC (x, k.max = k.max, method = method, maxiter = maxiter)

        ##  Tradeoff Curves: Explained Variance vs. sparseness
par (mfrow = c (2, k.max))
for (i in 1:k.max)       plot (oTPO, k = i)
for (i in 1:k.max)       plot (oBIC, k = i)

        ##  Explained Variance vs. lambda
par (mfrow = c (2, k.max))
for (i in 1:k.max)       plot (oTPO, k = i, f.x = "lambda")
for (i in 1:k.max)       plot (oBIC, k = i, f.x = "lambda")
```

---

plotcov — *Compare two Covariance Matrices in Plots*

Description

allows a direct comparison of two estimations of the covariance matrix (e.g. resulting from covPC) in a plot.

Usage

```
plotcov(cov1, cov2, method1, labels1, method2, labels2, ndigits, ...)
```

Arguments

`cov1`	a covariance matrix (from cov, covMcd, covPC, covPCAgrid, covPCAproj, etc.
`cov2`	a covariance matrix (from cov, covMcd, covPC, covPCAgrid, covPCAproj, etc.
`method1`	legend for ellipses of estimation method1
`method2`	legend for ellipses of estimation method2
`labels1`	legend for numbers of estimation method1

`labels2`	legend for numbers of estimation method2
`ndigits`	number of digits to use for printing covariances, by default ndigits=4
`...`	additional arguments for text or plot

Details

Since (robust) PCA can be used to re-compute the (robust) covariance matrix, one might be interested to compare two different methods of covariance estimation visually. This routine takes as input objects for the covariances to compare the output of `cov`, but also the return objects from `covPCAgrid`, `covPCAproj`, `covPC`, and `covMcd`. The comparison of the two covariance matrices is done by numbers (the covariances) and by ellipses.

Value

only the plot is generated

Author(s)

Heinrich Fritz, Peter Filzmoser <`<P.Filzmoser@tuwien.ac.at>`>

References

C. Croux, P. Filzmoser, M. Oliveira, (2007). Algorithms for Projection-Pursuit Robust Principal Component Analysis, *Chemometrics and Intelligent Laboratory Systems*, Vol. 87, pp. 218-225.

See Also

`PCAgrid`, `PCAproj`, `princomp`

Examples

```
# multivariate data with outliers
library(mvtnorm)
x <- rbind(rmvnorm(200, rep(0, 6), diag(c(5, rep(1,5)))),
           rmvnorm( 15, c(0, rep(20, 5)), diag(rep(1, 6))))
plotcov(covPCAproj(x),covPCAgrid(x))
```

---

qn *scale estimation using the robust Qn estimator*

Description

Returns a scale estimation as calculated by the (robust) Qn estimator.

Usage

```
qn(x, corrFact)
```

Arguments

`x`	a vector of data
`corrFact`	the finite sample bias correction factor. By default a value of ~ 2.219144 is used (assuming normality).

Details

The Qn estimator computes the first quartile of the pairwise absolute differences of all data values.

Value

The estimated scale of the data.

Warning

Earlier implementations used a wrong correction factor for the final result. Thus qn estimations computed with package pcaPP version > 1.8-1 differ about 0.12% from earlier estimations (version <= 1.8-1).

Note

See the vignette "Compiling pcaPP for Matlab" which comes with this package to compile and use this function in Matlab.

Author(s)

Heinrich Fritz, Peter Filzmoser <`<P.Filzmoser@tuwien.ac.at>`>

References

P.J. Rousseeuw, C. Croux (1993) Alternatives to the Median Absolute Deviation, *JASA*, **88**, 1273-1283.

See Also

`mad`

Examples

```
# data with outliers
x <- c(rnorm(100), rnorm(10, 10))
qn(x)
```

ScaleAdv *centers and rescales data*

Description

Data is centered and rescaled (to have mean 0 and a standard deviation of 1).

Usage

```
ScaleAdv(x, center = mean, scale = sd)
```

Arguments

`x`	matrix containing the observations. If this is not a matrix, but a data frame, it is automatically converted into a matrix using the function `as.matrix`. In any other case, (eg. a vector) it is converted into a matrix with one single column.
`center`	this argument indicates how the data is to be centered. It can be a function like `mean` or `median` or a vector of length `ncol(x)` containing the center value of each column.
`scale`	this argument indicates how the data is to be rescaled. It can be a function like `sd` or `mad` or a vector of length `ncol(x)` containing the scale value of each column.

Details

The default `scale` being `NULL` means that no rescaling is done.

Value

The function returns a list containing

`x`	centered and rescaled data matrix.
`center`	a vector of the centers of each column x. If you add to each column of `x` the appropriate value from `center`, you will obtain the data with the original location of the observations.
`scale`	a vector of the scale factors of each column x. If you multiply each column of `x` by the appropriate value from `scale`, you will obtain the data with the original scales.

Author(s)

Heinrich Fritz, Peter Filzmoser <`P.Filzmoser@tuwien.ac.at`>

References

C. Croux, P. Filzmoser, M. Oliveira, (2007). Algorithms for Projection-Pursuit Robust Principal Component Analysis, *Chemometrics and Intelligent Laboratory Systems*, Vol. 87, pp. 218-225.

Examples

```
x <- rnorm(100, 10, 5)
x <- ScaleAdv(x)$x

# can be used with multivariate data too
library(mvtnorm)
x <- rmvnorm(100, 3:7, diag((7:3)^2))
res <- ScaleAdv(x, center = l1median, scale = mad)
res

# instead of using an estimator, you could specify the center and scale yourself too
x <- rmvnorm(100, 3:7, diag((7:3)^2))
res <- ScaleAdv(x, 3:7, 7:3)
res
```

Index