Package ‘robslopes’

April 27, 2023

Type Package
Title Fast Algorithms for Robust Slopes
Version 1.1.3
Date 2023-04-27
Author Jakob Raymaekers
Maintainer Jakob Raymaekers <j.raymaekers@maastrichtuniversity.nl>
License GPL (>= 2)
Imports Rcpp (>= 1.0.5)
LinkingTo Rcpp, RcppArmadillo
NeedsCompilation yes
Repository CRAN
Date/Publication 2023-04-27 08:30:06 UTC

R topics documented:

- PassingBablok .................................................. 2
- RepeatedMedian .................................................. 3
- robslope .......................................................... 5
- robslope.fit ....................................................... 7
- TheilSen .......................................................... 9

Index 12
PassingBablok  

Passing-Bablok slope and intercept estimator.

Description

Computes the equivariant Passing-Bablok regression. The implemented algorithm was proposed by Raymaekers and Dufey (2022) and runs in an expected $O(n \log n)$ time while requiring $O(n)$ storage.

Usage

```r
PassingBablok(x, y, alpha = NULL, verbose = TRUE)
```

Arguments

- `x` A vector of predictor values.
- `y` A vector of response values.
- `alpha` Determines the order statistic of the target slope, which is equal to $\lfloor alpha \times n \times (n - 1) \rfloor$, where $n$ denotes the sample size. Defaults to `NULL`, which corresponds with the (upper) median.
- `verbose` Whether or not to print out the progress of the algorithm. Defaults to `TRUE`.

Details

Given two input vectors $x$ and $y$ of length $n$, the equivariant Passing-Bablok estimator is computed as $\text{med}_{ij}[(y_i - y_j)/(x_i - x_j)]$. By default, the median in this expression is the upper median, defined as $\lfloor (n + 2)/2 \rfloor$. By changing `alpha`, other order statistics of the slopes can be computed.

Value

A list with elements:

- `intercept` The estimate of the intercept.
- `slope` The Theil-Sen estimate of the slope.

Author(s)

Jakob Raymaekers

References


Raymaekers J., Dufey F. (2022). Equivariant Passing-Bablok regression in quasilinear time. (link to open access pdf)

**Examples**

```r
# We compare the implemented algorithm against a naive brute-force approach.

bruteForcePB <- function(x, y) {
  n <- length(x)
  medind1 <- floor(((n * (n - 1)) / 2 + 2) / 2) # upper median
  medind2 <- floor((n + 2) / 2)
  temp <- t(sapply(1:n, function(z) apply(cbind(x, y), 1,
                function(k) (k[2] - y[z]) / (k[1] - x[z])))
  PBslope <- sort(abs(as.vector(temp[lower.tri(temp)])))[medind1]
  PBintercept <- sort(y - x * PBslope)[medind2]
  return(list(intercept = PBintercept, slope = PBslope))
}

n = 100
set.seed(2)
x = rnorm(n)
y = x + rnorm(n)

t0 <- proc.time()
PB.fast <- PassingBablok(x, y, NULL, FALSE)
t1 <- proc.time()
t1 - t0

t0 <- proc.time()
PB.naive <- bruteForcePB(x, y)
t1 <- proc.time()
t1 - t0

PB.fast$slope - PB.naive$slope
```

---

**RepeatedMedian**

Siegel's repeated median slope and intercept estimator.

**Description**

Computes the repeated median slope proposed by Siegel (1982) using the algorithm by Matousek et. al (1998). The algorithm runs in an expected $O(n \log n)^2$ time, which is typically significantly faster than the $O(n^2)$ computational cost of the naive algorithm, and requires $O(n)$ storage.
Usage

RepeatedMedian(x, y, alpha = NULL, beta = NULL, verbose = TRUE)

Arguments

- **x**: A vector of predictor values.
- **y**: A vector of response values.
- **alpha**: Determines the outer order statistic, which is equal to \([alpha \times n]\), where \(n\) denotes the sample size. Defaults to NULL, which corresponds with the (upper) median.
- **beta**: Determines the inner order statistic, which is equal to \([beta \times (n - 1)]\), where \(n\) denotes the sample size. Defaults to NULL, which corresponds with the (upper) median.
- **verbose**: Whether or not to print out the progress of the algorithm. Defaults to TRUE.

Details

Given two input vectors \(x\) and \(y\) of length \(n\), the repeated median is computed as \(med_i med_j (y_i - y_j) / (x_i - x_j)\). The default "outer" median is the \(\lfloor (n + 2)/2 \rfloor\) largest element in the ordered median slopes. The inner median, which for each observation is calculated as the median of the slopes connected to this observation, is the \(\lfloor (n + 1)/2 \rfloor\) largest element in the ordered slopes. By changing \(alpha\) and \(beta\), other repeated order statistics of the slopes can be computed.

Value

A list with elements:

- **intercept**: The estimate of the intercept.
- **slope**: The Theil-Sen estimate of the slope.

Author(s)

Jakob Raymaekers

References


See Also

TheilSen
Examples

# We compare the implemented algorithm against a naive brute-force approach.

bruteForceRM <- function(x, y) {
  n <- length(x)
  medind1 <- floor((n+2) / 2)
  medind2 <- floor((n+1) / 2)
  temp <- t(sapply(1:n, function(z) sort(apply(cbind(x, y), 1 ,
    function(k) (k[2] - y[z]) /
    (k[1] - x[z])))))
  RMslope <- sort(temp[, medind2])[medind1]
  RMintercept <- sort(y - x * RMslope)[medind1]
  return(list(intercept = RMintercept, slope = RMslope))
}

n = 100
set.seed(2)
x = rnorm(n)
y = x + rnorm(n)

# We compare the implemented algorithm against a naive brute-force approach.

t0 <- proc.time()
RM.fast <- RepeatedMedian(x, y, NULL, NULL, FALSE)
t1 <- proc.time()
t1 - t0

t0 <- proc.time()
RM.naive <- bruteForceRM(x, y)
t1 <- proc.time()
t1 - t0

RM.fast$slope - RM.naive$slope

---

robslope  
Robust slope estimator

Description

Computes the Theil-Sen median slope, Siegel’s repeated median slope or te equivariant Passing-Bablok slope. The algorithms run in an expected linearithmic time while requiring $O(n)$ storage. They are based on Dillencourt et. al (1992), Matousek et. al (1998) and Raymaekers and Dufey (2022).

Usage

robslope(formula, data, subset, weights, na.action,  
  type = c("TheilSen", "RepeatedMedian","PassingBablok"),  
  alpha = NULL, beta = NULL, verbose = TRUE)
### Arguments

- **formula**: an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under ‘Details’.
- **data**: an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, typically the environment from which `robslope` is called.
- **subset**: an optional vector specifying a subset of observations to be used in the fitting process.
- **weights**: an optional vector of weights to be used in the fitting process. Currently not supported.
- **na.action**: a function which indicates what should happen when the data contain NAs. The default `na.exclude` is applied and an informative message is given in case NAs were removed.
- **type**: the type of robust slope estimator. Should be one of "TheilSen" (default), "RepeatedMedian" or "PassingBablok".
- **alpha**: Determines the order statistic of the target slope. Defaults to the upper median. See below for details.
- **beta**: Determines the inner order statistic. Only used when `type = "RepeatedMedian"`. See below for details.
- **verbose**: Whether or not to print out the progress of the algorithm. Defaults to TRUE.

### Details

This function provides a wrapper around `robslope.fit`, which in turn calls the individual functions `TheilSen`, `RepeatedMedian` or `PassingBablok`. The details on changing the parameters alpha and beta can be found in the documentation of those respective functions.

### Value

`robslope` returns an object of class "lm".

The generic accessor functions `coefficients`, `fitted.values` and `residuals` extract various useful features of the value returned by `lm`.

### Author(s)

Jakob Raymaekers

### References


Raymaekers J., Dufey F. (2022). Equivariant Passing-Bablok regression in quasilinear time. (link to open access pdf)


See Also

robslope.fit TheilSen RepeatedMedian PassingBablok

Examples

```r
set.seed(123)
df <- data.frame(cbind(rnorm(20), rnorm(20)))
colnames(df) <- c("x", "y")
robslope.out <- robslope(y~x, data = df, type = "RepeatedMedian", verbose = TRUE)
coef(robslope.out)
plot(fitted.values(robslope.out))

robslope.out <- robslope(y~x, data = df, type = "TheilSen", verbose = TRUE)
plot(residuals(robslope.out))
```

---

**robslope.fit**

**Robust slope estimator**

**Description**

This is the underlying computing engine called by robslope used to fit robust slopes. It wraps around the individual functions TheilSen, RepeatedMedian or PassingBablok. These should usually *not* be used directly unless by experienced users.
Usage

robslope.fit(x, y, weights, type, alpha = NULL, beta = NULL, verbose = TRUE)

Arguments

- **x**: design matrix of dimension \( n \times p \).
- **y**: vector of observations of length \( n \), or a matrix with \( n \) rows.
- **type**: the type of robust slope estimator. Should be one of "TheilSen" (default), "RepeatedMedian" or "PassingBablok".
- **weights**: vector of weights. Currently not in use.
- **alpha**: Determines the order statistic of the target slope. Defaults to the upper median. See below for details.
- **beta**: Determines the inner order statistic. Only used when \( \text{type} = \text{"RepeatedMedian"} \). See below for details.
- **verbose**: Whether or not to print out the progress of the algorithm. Defaults to \( \text{TRUE} \).

Details

This function provides a wrapper around the individual functions TheilSen, RepeatedMedian or PassingBablok. The details on changing the parameters alpha and beta can be found in the documentation of those respective functions.

Value

- **list** with components
  - **coefficients**: \( p \) vector
  - **residuals**: \( n \) vector or matrix
  - **fitted.values**: \( n \) vector or matrix

Author(s)

Jakob Raymaekers

References


TheilSen


Raymaekers J., Dufey F. (2022). Equivariant Passing-Bablok regression in quasilinear time. (link to open access pdf)


See Also

robslope TheilSen RepeatedMedian PassingBablok

Examples

```r
set.seed(123)
x <- rnorm(20)
y <- rnorm(20)
robslope.out <- robslope.fit(x, y, type = "RepeatedMedian", verbose = TRUE)
coef(robslope.out)
plot(fitted.values(robslope.out))

robslope.out <- robslope.fit(x, y, type = "TheilSen", verbose = TRUE)
plot(residuals(robslope.out))
```

---

TheilSen

*Theil-Sen slope and intercept estimator.*

Description

Computes the Theil-Sen median slope estimator by Theil (1950) and Sen (1968). The implemented algorithm was proposed by Dillencourt et. al (1992) and runs in an expected $O(n\log n)$ time while requiring $O(n)$ storage.

Usage

```r
TheilSen(x, y, alpha = NULL, verbose = TRUE)
```
TheilSen

Arguments

- **x**: A vector of predictor values.
- **y**: A vector of response values.
- **alpha**: Determines the order statistic of the target slope, which is equal to \( \alpha n (n - 1) \), where \( n \) denotes the sample size. Defaults to NULL, which corresponds with the (upper) median.
- **verbose**: Whether or not to print out the progress of the algorithm. Defaults to TRUE.

Details

Given two input vectors \( x \) and \( y \) of length \( n \), the Theil-Sen estimator is computed as \( \text{med}_{ij}(y_i - y_j)/(x_i - x_j) \). By default, the median in this expression is the upper median, defined as \( \lfloor (n+2)/2 \rfloor \).

By changing \( \alpha \), other order statistics of the slopes can be computed.

Value

A list with elements:

- **intercept**: The estimate of the intercept.
- **slope**: The Theil-Sen estimate of the slope.

Author(s)

Jakob Raymaekers

References


Examples

```r
# We compare the implemented algorithm against a naive brute-force approach.

bruteForceTS <- function(x, y) {
  n <- length(x)
  medind1 <- floor(((n * (n - 1)) / 2 + 2) / 2)
  medind2 <- floor((n + 2) / 2)
  temp <- t(sapply(1:n, function(z) apply(cbind(x, y), 1
    function(k) (k[2] - y[z]) / (k[1] - x[z]))))
  # ... (continue to compare the results of 'TheilSen' and 'bruteForceTS')
}
TheilSen

```r
TSslope <- sort(as.vector(temp[lower.tri(temp)]))[medind1]
TSintercept <- sort(y - x * TSslope)[medind2]
return(list(intercept = TSintercept, slope = TSslope))
}

n = 100
set.seed(2)
x = rnorm(n)
y = x + rnorm(n)

t0 <- proc.time()
TS.fast <- TheilSen(x, y, NULL, FALSE)
t1 <- proc.time()
t1 - t0

t0 <- proc.time()
TS.naive <- bruteForceTS(x, y)
t1 <- proc.time()
t1 - t0

TS.fast$slope - TS.naive$slope
```
Index

as.data.frame, 6
class, 6
formula, 6
list, 8
na.exclude, 6
PassingBablok, 2, 6–9
RepeatedMedian, 3, 6–9
robslope, 5, 7, 9
robslope.fit, 6, 7, 7
TheilSen, 4, 6–9, 9