Package ‘NPE’

August 24, 2020

Type Package

Title Non-Parametric Estimators

Version 1.0

Date 2020-08-03

Author Sumit Sethi

Maintainer Sumit Sethi <sumit.sethi@nyu.edu>

Description
Functions for calculating a variety of nonparametric estimators, including estimators of location (median, Hodges-Lehmann), of dispersion (Median Absolute Deviation), and ofdependency-covariance (Theil-Sen).
It also implements the nonparametric statistics of the Wilcoxon Signed Rank test, the Mann-Whitney-Wilcoxon
Rank Sum test, and the Kruskal-Wallis test. It also implements PCA.

License MPL-2.0

Imports Rcpp (>= 1.0.4.6),
RcppParallel

LinkingTo Rcpp,
RcppArmadillo,
RcppParallel,
BH

Suggests knitr,
rmarkdown

VignetteBuilder knitr

LazyData true

ByteCompile true

Repository GitHub

URL https://github.com/marvic24/Non-Parametric-Estimators

SystemRequirements GNU make

RoxygenNote 7.1.1

Encoding UTF-8

R topics documented:

`calc_mad` *Calculate the Median Absolute Deviations* MAD *of the columns of a* time series *or a* matrix *using* `RcppArmadillo`.

Description

Calculate the Median Absolute Deviations *MAD* of the columns of a *time series* or a *matrix* using `RcppArmadillo`.

Usage

```
calc_mad(t_series)
```

Arguments

`t_series`	A *time series* or a *matrix* of data.

Details

The function `calc_mad()` calculates the Median Absolute Deviations *MAD* of the columns of a *time series* or a *matrix* of data using `RcppArmadillo` `C++` code.

The function `calc_mad()` performs the same calculation as the function `stats::mad()`, but it's much faster because it uses `RcppArmadillo` `C++` code.

Value

A row vector with the Median Absolute Deviations *MAD* of the columns of `t_series` matrix.

Examples

```
## Not run:
# Calculate VTI returns
re_turns <- na.omit(rutils::etf_env$re_turns[ ,"VTI", drop=FALSE])
# Compare calc_mad() with stats::mad()
all.equal(drop(NPE::calc_mad(re_turns)),
  mad(re_turns)/1.4826)
# Compare the speed of RcppArmadillo with stats::mad()
library(microbenchmark)
summary(microbenchmark(
  Rcpp=NPE::calc_mad(re_turns),
  Rcode=mad(re_turns),
  times=10))[, c(1, 4, 5)]  # end microbenchmark summary

## End(Not run)
```

`calc_pca` *Performs a principal component analysis on given* matrix *or* time series *using* `RcppArmadillo`.

Description

Performs a principal component analysis on given *matrix* or *time series* using `RcppArmadillo`.

Usage

```
calc_pca(mat_rix)
```

Arguments

`mat_rix`	A *matrix* or a *time series*.

Details

The function `calc_pca()` performs a principal component analysis on a *matrix* using `RcppArmadillo`.

Value

A *matrix* of variable loadings (i.e. a matrix whose columns contain the eigenvectors).

Examples

```
## Not run:
# Create a matrix of random returns
re_turns <- matrix(rnorm(5e6), nc=5)
# Compare calc_pca() with standard prcomp()
all.equal(drop(NPE::calc_pca(re_turns)),
  prcomp(re_turns))
# Compare the speed of RcppArmadillo with R code
library(microbenchmark)
summary(microbenchmark(
  rcpp=NPE::calc_pca(re_turns),
```

```
  rcode=prcomp(re_turns),
  times=10))[, c(1, 4, 5)]  # end microbenchmark summary

## End(Not run)
```

calc_ranksWithTies *Calculate the ranks of the elements of a* vector *or a single-column* time series *using* RcppArmadillo *and* boost.

Description

Calculate the ranks of the elements of a *vector* or a single-column *time series* using RcppArmadillo and boost.

Usage

```
calc_ranksWithTies(vec_tor)
```

Arguments

vec_tor	A *vector* or a single-column *time series*.

Details

The function calc_ranks() calculates the ranks of the elements of a *vector* or a single-column *time series*. It *averages* the ranks in case fo ties. It uses the boost function boost::sort::parallel_stable_sort for sorting array in parallel fashion.

Value

A *double vector* with the ranks of the elements of the *vector*.

Examples

```
## Not run:
# Create a vector of random data
da_ta <- round(runif(7), 2)
# Calculate the ranks of the elements in two ways
all.equal(rank(da_ta), drop(NPE::calc_ranksWithTies(da_ta)))
# Create a time series of random data
da_ta <- xts::xts(runif(7), seq.Date(Sys.Date(), by=1, length.out=7))
# Calculate the ranks of the elements in two ways
all.equal(rank(coredata(da_ta)), drop(NPE::calc_ranksWithTies(da_ta)))
# Compare the speed of this function with RcppArmadillo and R code
da_ta <- runif(7)
library(microbenchmark)
summary(microbenchmark(
  rcpp=calc_ranks(da_ta),
  rcode=rank(da_ta),
  boost=calc_ranksWithTies(da_ta)
  times=10))[, c(1, 4, 5)]  # end microbenchmark summary

## End(Not run)
```

calc_skew — *Calculate the skewness of the columns of a* time series *or a* matrix *using* RcppArmadillo.

Description

Calculate the skewness of the columns of a *time series* or a *matrix* using RcppArmadillo.

Usage

```
calc_skew(t_series, typ_e = "pearson", al_pha = 0.25)
```

Arguments

t_series	A *time series* or a *matrix* of data.
typ_e	A *string* specifying the objective for calculating the weights (see Details). (The default is the typ_e = "pearson".)
al_pha	The confidence level for calculating the quantiles. (the default is 0.25).

Details

The function calc_skew() calculates the skewness of the columns of a *time series* or a *matrix* of data using RcppArmadillo C++ code.

If typ_e = "pearson" (the default) then calc_skew() calculates the Pearson skewness using the third moment of the data.

If typ_e = "quantile" then it calculates the skewness using the differences between the quantiles of the data.

If typ_e = "nonparametric" then it calculates the skewness as the difference between the mean of the data minus its median, divided by the standard deviation.

The code examples below compare the function calc_skew() with the skewness calculated using R code.

Value

A row vector equal to the skewness of the columns of t_series.

Examples

```
## Not run:
# Calculate VTI returns
re_turns <- na.omit(rutils::etf_env$re_turns[ ,"VTI", drop=FALSE])
# Calculate the Pearson skewness
NPE::calc_skew(re_turns)
# Compare NPE::calc_skew() with Pearson skewness
calc_skewr <- function(x) {
  x <- (x-mean(x)); nr <- NROW(x);
  nr*sum(x^3)/(var(x))^1.5/(nr-1)/(nr-2)
}  # end calc_skewr
all.equal(NPE::calc_skew(re_turns),
  calc_skewr(re_turns), check.attributes=FALSE)
# Compare the speed of RcppArmadillo with R code
```

```
library(microbenchmark)
summary(microbenchmark(
  Rcpp=NPE::calc_skew(re_turns),
  Rcode=calc_skewr(re_turns),
  times=10))[, c(1, 4, 5)]  # end microbenchmark summary
# Calculate the quantile skewness
NPE::calc_skew(re_turns, typ_e = "quantile", al_pha = 0.1)
# Compare NPE::calc_skew() with quantile skewness
calc_skewq <- function(x) {
   quantile_s <- quantile(x, c(0.25, 0.5, 0.75), type=5)
   (quantile_s[3] + quantile_s[1] - 2*quantile_s[2])/(quantile_s[3] - quantile_s[1])
}  # end calc_skewq
all.equal(drop(NPE::calc_skew(re_turns, typ_e = "quantile")),
  calc_skewq(re_turns), check.attributes=FALSE)
# Compare the speed of RcppArmadillo with R code
summary(microbenchmark(
  Rcpp=NPE::calc_skew(re_turns, typ_e = "quantile"),
  Rcode=calc_skewq(re_turns),
  times=10))[, c(1, 4, 5)]  # end microbenchmark summary
# Calculate the nonparametric skewness
NPE::calc_skew(re_turns, typ_e = "nonparametric")
# Compare NPE::calc_skew() with R nonparametric skewness
all.equal(drop(NPE::calc_skew(re_turns, typ_e = "nonparametric")),
  (mean(re_turns)-median(re_turns))/sd(re_turns),
  check.attributes=FALSE)
# Compare the speed of RcppArmadillo with R code
summary(microbenchmark(
  Rcpp=NPE::calc_skew(re_turns, typ_e = "nonparametric"),
  Rcode=(mean(re_turns)-median(re_turns))/sd(re_turns),
  times=10))[, c(1, 4, 5)]  # end microbenchmark summary

## End(Not run)
```

`data` *The dataset called* `data` *contains a single environment called etf_env, which includes daily* `OHLC` *time series data for a portfolio of symbols. All the prices are already adjusted.*

Description

The etf_env environment includes daily `OHLC` time series data for a portfolio of symbols, and reference data:

sym_bols a `vector` of `strings` with the portfolio symbols.

price_s a single `xts` time series containing daily closing prices for all the `sym_bols`.

re_turns a single `xts` time series containing daily returns for all the `sym_bols`.

Individual time series "VTI", "VEU", etc., containing daily `OHLC` prices for the `sym_bols`.

Usage

```
data(data)  # not required - data is lazy load
```

Format

Each xts time series contains the following columns with adjusted prices and trading volume:

Open Open prices

High High prices

Low Low prices

Close Close prices

Volume daily trading volume

Examples

```
# Loading is not not needed - data is lazy load
# data(data)
# Get first six rows of OHLC prices
head(etf_env$VTI)
chart_Series(x=etf_env$VTI["2009-11"])
```

---

hle *Calculate the nonparametric Hodges-Lehmann estimator of location for a* vector *or a single-column* time series *using* RcppArmadillo *and* RcppParallel.

Description

Calculate the nonparametric Hodges-Lehmann estimator of location for a *vector* or a single-column *time series* using RcppArmadillo and RcppParallel.

Usage

```
hle(vec_tor)
```

Arguments

vec_tor A *vector* or a single-column *time series*.

Details

The function hle() calculates the Hodges-Lehmann estimator of the *vector*, using RcppArmadillo and RcppParallel. The function hle() is very much faster than function wilcox.test() in R.

Value

A single *double* value representing Hodges-Lehmann estimator of the vector.

Examples

```
## Not run:
# Create a vector of random returns
re_turns <- rnorm(1e6)
# Compare hle() with wilcox.test()
all.equal(drop(NPE::hle(re_turns)),
  wilcox.test(re_turns, conf.int = TRUE))
# Compare the speed of RcppParallel with R code
library(microbenchmark)
summary(microbenchmark(
  rcpp=NPE::hle(re_turns),
  rcode=wilcox.test(re_turns, conf.int = TRUE),
  times=10))[, c(1, 4, 5)]  # end microbenchmark summary

## End(Not run)
```

---

kruskalWalliceTest — *Performs a Kruskal-Wallis rank sum test. using* Rcpp *and* boost.

Description

Performs a Kruskal-Wallis rank sum test. using Rcpp and boost.

Usage

```
kruskalWalliceTest(x)
```

Arguments

x	A *List* of numeric data vectors

Details

The function kruskalWalliceTest() performs a Kruskal-Wallis rank sum test of the null hypothesis that the location parameters of the distribution of x are the same in each group. The alternative is that they differ in at least in one.

Value

A *double* indicating p-value of the test.

Examples

```
## Not run:
x <- c(2.9, 3.0, 2.5, 2.6, 3.2) # normal subjects
y <- c(3.8, 2.7, 4.0, 2.4)      # with obstructive airway disease
z <- c(2.8, 3.4, 3.7, 2.2, 2.0) # with asbestosis

# Carry out Kruskal wallice rank sum test on the elements in two ways
all.equal(kruskal.test(list(x, y, z))$p.value, drop(NPE::kruskalWalliceTest(list(x, y, z))))
# Compare the speed of Rcpp and R code
library(microbenchmark)
```

```
summary(microbenchmark(
  rcpp=kruskalWalliceTest(list(x, y, z)),
  rcode=kruskal.test(list(x, y, z))$p.value,
  times=10))[, c(1, 4, 5)]  # end microbenchmark summary

## End(Not run)
```

`med_couple` *Calculate the medcouple of a* vector *or a single-column* time series *using* Rcpp.

Description

Calculate the medcouple of a *vector* or a single-column *time series* using Rcpp.

Usage

```
med_couple(x, eps1 = 1e-14, eps2 = 1e-15)
```

Arguments

vec_tor	A *vector* or a single-column *time series*.
eps1	A *double* Tolerance of the algorithm.
eps2	A *couble* Tolerance of the algorithm..

Details

The function med_couple() calculates the medcouple of the *vector*, using Rcpp. The function med_couple() is several times faster than mc() in package robustbase.

Value

A single *double* value representing medcouple of the vector.

Examples

```
## Not run:
# Create a vector of random returns
re_turns <- rnorm(1e6)
# Compare med_couple() with mc()
all.equal(drop(NPE::med_couple(re_turns)),
  robustbase::mc(re_turns))
# Compare the speed of NPE with Robustbase code
library(microbenchmark)
summary(microbenchmark(
  rcpp=NPE::med_couple(re_turns),
  robustbase=robustbase::mc(re_turns),
  times=10))[, c(1, 4, 5)]  # end microbenchmark summary

## End(Not run)
```

med_ian *Calculate the median of a* vector *or a single-column* time series *using* RcppArmadillo.

Description

Calculate the median of a *vector* or a single-column *time series* using RcppArmadillo.

Usage

```
med_ian(vec_tor)
```

Arguments

vec_tor	A *vector* or a single-column *time series*.

Details

The function med_ian() calculates the median of the *vector*, using RcppArmadillo. The function med_ian() is several times faster than median() in R.

Value

A single *double* value representing median of the vector.

Examples

```
## Not run:
# Create a vector of random returns
re_turns <- rnorm(1e6)
# Compare med_ian() with median()
all.equal(drop(NPE::med_ian(re_turns)),
  median(re_turns))
# Compare the speed of RcppArmadillo with R code
library(microbenchmark)
summary(microbenchmark(
  rcpp=NPE::med_ian(re_turns),
  rcode=median(re_turns),
  times=10))[, c(1, 4, 5)]  # end microbenchmark summary

## End(Not run)
```

rolling_mad — *Calculate the rolling median absolute deviation over a* vector *or a single-column* time series *using* RcppArmadillo *and* RcppParallel.

Description

Calculate the rolling median absolute deviation over a *vector* or a single-column *time series* using RcppArmadillo and RcppParallel.

Usage

```
rolling_mad(vec_tor, look_back)
```

Arguments

vec_tor	A *vector* or a single-column *time series*.
look_back	The length of look back interval, equal to the number of elements of data used for calculating the median.

Details

The function rolling_mad() calculates a vector of rolling medians, over a *vector* of data, using *RcppArmadillo* and *RcppParallel*.

Value

A column *vector* of the same length as the argument vect_tor.

Examples

```
## Not run:
# Create a vector of random returns
re_turns <- rnorm(1e6)
rolling_mad(re_turns)

## End(Not run)
```

rolling_median — *Calculate the rolling median over a* vector *or a single-column* time series *using* RcppArmadillo *and* RcppParallel.

Description

Calculate the rolling median over a *vector* or a single-column *time series* using RcppArmadillo and RcppParallel.

Usage

```
rolling_median(vec_tor, look_back)
```

Arguments

vec_tor A *vector* or a single-column *time series*.

look_back The length of look back interval, equal to the number of elements of data used for calculating the median.

Details

The function rolling_median() calculates a vector of rolling medians, over a *vector* of data, using *RcppArmadillo* and *RcppParallel*. The function rolling_median() is faster than roll::roll_median() which uses Rcpp.

Value

A column *vector* of the same length as the argument vect_tor.

Examples

```
## Not run:
# Create a vector of random returns
re_turns <- rnorm(1e6)
# Compare rolling_median() with roll::roll_median()
all.equal(drop(NPE::rolling_median(re_turns, look_back=11)),
  roll::roll_median(re_turns, width=11))
# Compare the speed of RcppArmadillo with R code
library(microbenchmark)
summary(microbenchmark(
  parallel_rcpp=NPE::rolling_median(re_turns),
  rcpp=roll::roll_median(re_turns),
  times=10))[, c(1, 4, 5)]  # end microbenchmark summary

## End(Not run)
```

---

theilSenEstimator *Calculate the nonparametric Theil-Sen estimator of dependency-covariance for two* vectors *using* RcppArmadillo

---

Description

Calculate the nonparametric Theil-Sen estimator of dependency-covariance for two *vectors* using RcppArmadillo

Usage

```
theilSenEstimator(x, y)
```

Arguments

vector_x A *vector* independent (explanatory) data.

vector_y A *vector* dependent data.

Details

The function `theilSenEstimator()` calculates the Theil-Sen estimator of the *vector*, using `RcppArmadillo` . The function `theilSenEstimator()` is significantly faster than function `WRS::tsreg()` in `R`.

Value

A column *vector* containing two values i.e intercept and slope

Examples

```
## Not run:
# Create a vector of random returns
vector_x <- rnorm(10)
vactor_y <- rnorm(10)
# Compare theilSenEstimator() with tsreg()
# Compare the speed of RcppParallel with R code
library(microbenchmark)
summary(microbenchmark(
  rcpp=NPE::theilSenEstimator(vector_x, vector_y),
  rcode=WRS(vector_x, vector_y),
  times=10))[, c(1, 4, 5)]  # end microbenchmark summary

## End(Not run)
```

Index