Package: timevarcorr (via r-universe)

October 18, 2026

Title Time Varying Correlation

Version 0.1.1.9000

Description Computes how the correlation between 2 time-series changes over time. To do so, the package follows the method from Choi & Shin (2021) <doi:10.1007/s42952-020-00073-6>. It performs a non-parametric kernel smoothing (using a common bandwidth) of all underlying components required for the computation of a correlation coefficient (i.e., x, y, x^2, y^2, xy). An automatic selection procedure for the bandwidth parameter is implemented. Alternative kernels can be used (Epanechnikov, box and normal). Both Pearson and Spearman correlation coefficients can be estimated and change in correlation over time can be tested.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Depends R (>= 2.10)

Imports lpridge

LazyData true

Suggests dplyr, ggplot2, spelling, testthat (>= 3.0.0)

Language en-US

URL https://courtiol.github.io/timevarcorr/, https://github.com/courtiol/timevarcorr

BugReports https://github.com/courtiol/timevarcorr/issues

Config/testthat/edition 3

Repository https://courtiol.r-universe.dev

Date/Publication 2023-11-07 20:02:24 UTC

RemoteUrl https://github.com/courtiol/timevarcorr

RemoteRef HEAD

RemoteSha 074663dbd83ddeb873480dd266bce51989712ae7

Contents

.onAttach 2
CI 2
in_pkgdown 5
kern_smooth 6
stockprice 8
tcor 9
test_equality 17
test_ref 19

Index **21**

`.onAttach` *Display welcome message*

Description

This function should not be called by the user. It displays a message when the package is being loaded.

Usage

```
.onAttach(libname, pkgname)
```

Arguments

`libname`	argument needed but automatically defined.
`pkgname`	argument needed but automatically defined.

Value

nothing (invisible NULL).

`CI` *Internal functions for the computation of confidence intervals*

Description

These functions compute the different terms required for `tcor()` to compute the confidence interval around the time-varying correlation coefficient. These terms are defined in Choi & Shin (2021).

Usage

```
calc_H(smoothed_obj)

calc_e(smoothed_obj, H)

calc_Gamma(e, l)

calc_GammaINF(e, L)

calc_L_And(e, AR.method = c("yule-walker", "burg", "ols", "mle", "yw"))

calc_D(smoothed_obj)

calc_SE(
  smoothed_obj,
  h,
  AR.method = c("yule-walker", "burg", "ols", "mle", "yw")
)
```

Arguments

`smoothed_obj`	an object created with `calc_rho`.
`H`	an object created with `calc_H`.
`e`	an object created with `calc_e`.
`l`	a scalar indicating a number of time points.
`L`	a scalar indicating a bandwidth parameter.
`AR.method`	character string specifying the method to fit the autoregressive model used to compute $\hat{\gamma}_1$ in L_{And} (see `stats::ar` for details).
`h`	a scalar indicating the bandwidth used by the smoothing function.

Value

- `calc_H()` returns a 5 x 5 x t array of elements of class numeric, which corresponds to $\hat{H}_t$ in Choi & Shin (2021).
- `calc_e()` returns a t x 5 matrix of elements of class numeric storing the residuals, which corresponds to $\hat{e}_t$ in Choi & Shin (2021).
- `calc_Gamma()` returns a 5 x 5 matrix of elements of class numeric, which corresponds to $\hat{\Gamma}_l$ in Choi & Shin (2021).
- `calc_GammaINF()` returns a 5 x 5 matrix of elements of class numeric, which corresponds to $\hat{\Gamma}^{\infty}$ in Choi & Shin (2021).
- `calc_L_And()` returns a scalar of class numeric, which corresponds to L_{And} in Choi & Shin (2021).
- `calc_D()` returns a t x 5 matrix of elements of class numeric storing the residuals, which corresponds to D_t in Choi & Shin (2021).
- `calc_SE()` returns a vector of length t of elements of class numeric, which corresponds to $se(\hat{\rho}_t(h))$ in Choi & Shin (2021).

Functions

- `calc_H()`: computes the $\hat{H}_t$ array.
  $\hat{H}_t$ is a component needed to compute confidence intervals; H_t is defined in eq. 6 from Choi & Shin (2021).
- `calc_e()`: computes $\hat{e}_t$.
  $\hat{e}_t$ is defined in eq. 9 from Choi & Shin (2021).
- `calc_Gamma()`: computes $\hat{\Gamma}_l$.
  $\hat{\Gamma}_l$ is defined in eq. 9 from Choi & Shin (2021).
- `calc_GammaINF()`: computes $\hat{\Gamma}^{\infty}$.
  $\hat{\Gamma}^{\infty}$ is the long run variance estimator, defined in eq. 9 from Choi & Shin (2021).
- `calc_L_And()`: computes L_{And}.
  L_{And} is defined in Choi & Shin (2021, p 342). It also corresponds to S_T^*, eq 5.3 in Andrews (1991).
- `calc_D()`: computes D_t.
  D_t is defined in Choi & Shin (2021, p 338).
- `calc_SE()`: computes $se(\hat{\rho}_t(h))$.
  The standard deviation of the time-varying correlation ($se(\hat{\rho}_t(h))$) is defined in eq. 8 from Choi & Shin (2021). It depends on D_{Lt}, D_{Mt} & D_{Ut}, themselves defined in Choi & Shin (2021, p 337 & 339). The D_{Xt} terms are all computed within the function since they all rely on the same components.

References

Choi, JE., Shin, D.W. Nonparametric estimation of time varying correlation coefficient. J. Korean Stat. Soc. 50, 333–353 (2021). doi:10.1007/s42952020000736

Andrews, D. W. K. Heteroskedasticity and autocorrelation consistent covariance matrix estimation. Econometrica: Journal of the Econometric Society, 817-858 (1991).

See Also

`tcor()`

Examples

```
rho_obj <- with(na.omit(stockprice),
                calc_rho(x = SP500, y = FTSE100, t = DateID, h = 20, kernel = "box"))
head(rho_obj)

## Computing \eqn{\hat{H_t}}

H <- calc_H(smoothed_obj = rho_obj)
H[, , 1:2] # H array for the first two time points

## Computing \eqn{\hat{e}_t}

e <- calc_e(smoothed_obj = rho_obj, H = H)
```

```
head(e) # e matrix for the first six time points

## Computing \eqn{\hat{\Gamma}_l}

calc_Gamma(e = e, l = 3)

## Computing \eqn{\hat{\Gamma}^\infty}

calc_GammaINF(e = e, L = 2)

## Computing \eqn{L_{And}}

calc_L_And(e = e)
sapply(c("yule-walker", "burg", "ols", "mle", "yw"),
       function(m) calc_L_And(e = e, AR.method = m)) ## comparing AR.methods

## Computing \eqn{D_t}

D <- calc_D(smoothed_obj = rho_obj)
head(D) # D matrix for the first six time points

## Computing \eqn{se(\hat{\rho}_t(h))}
# nb: takes a few seconds to run

run <- FALSE ## change to TRUE to run the example
if (in_pkgdown() || run) {

SE <- calc_SE(smoothed_obj = rho_obj, h = 50)
head(SE) # SE vector for the first six time points

}
```

---

in_pkgdown *Determine if the package is being used by pkgdown*

Description

This function should not be called by the user. It allows to run some examples conditionally to being used by pkgdown. Code copied from `pkgdown::in_pkgdown()`.

Usage

```
in_pkgdown()
```

Value

a logical value (`TRUE` or `FALSE`).

kern_smooth *Smoothing by kernel regression*

Description

The function perform the smoothing of a time-series by non-parametric kernel regression.

Usage

```
kern_smooth(
  x,
  t = seq_along(x),
  h,
  t.for.pred = t,
  kernel = c("epanechnikov", "box", "normal"),
  param_smoother = list(),
  output = c("dataframe", "list")
)
```

Arguments

x	a numeric vector of the series to be smoothed.
t	a (numeric or Date) vector of time points. If missing, observations are considered to correspond to sequential time steps (i.e., 1, 2 ...).
h	a scalar indicating the bandwidth used by the smoothing function.
t.for.pred	a (numeric or Date) vector of time points at which to evaluate the smoothed fit. If missing, t is used.
kernel	a character string indicating which kernel to use: "epanechnikov" (the default), "box", or "normal" (abbreviations also work).
param_smoother	a list of additional parameters to provide to the internal smoothing function (see **Details**).
output	a character string indicating if the output should be a "dataframe" (default) or a list (for faster computation when the function is called repeatedly).

Details

The function is essentially a wrapper that calls different underlying functions depending on the kernel that is selected:

- lpridge::lpepa() for "epanechnikov".
- stats::ksmooth() for "normal" and "box". The argument param_smoother can be used to pass additional arguments to these functions.

Value

a dataframe of time points (t.for.pred) and corresponding fitted values.

References

A short post we found useful: http://users.stat.umn.edu/~helwig/notes/smooth-notes.html

See Also

tcor

Examples

```
## Smooth 10 first values of a vector

kern_smooth(stockprice$DAX[1:20], h = 5)


## Prediction at time step 2 and 3

kern_smooth(stockprice$DAX, h = 1, t.for.pred = c(2, 3))


## Smoothing using a vector of dates for time

kern_smooth(x = stockprice$DAX[1:10], t = stockprice$DateID[1:10], h = 5)


## Smoothing conserves original order

kern_smooth(x = stockprice$DAX[10:1], t = stockprice$DateID[10:1], h = 5)


## Effect of the bandwidth

plot(stockprice$DAX[1:100] ~ stockprice$DateID[1:100],
     las = 1, ylab = "DAX index", xlab = "Date")
points(kern_smooth(stockprice$DAX[1:100], stockprice$DateID[1:100], h = 1),
       type = "l", col = "grey")
points(kern_smooth(stockprice$DAX[1:100], stockprice$DateID[1:100], h = 3),
       type = "l", col = "blue")
points(kern_smooth(stockprice$DAX[1:100], stockprice$DateID[1:100], h = 10),
       type = "l", col = "red")
legend("topright", fill = c("grey", "blue", "red"),
       legend = c("1", "3", "10"), bty = "n", title = "Bandwidth (h)")


## Effect of the kernel

plot(stockprice$DAX[1:100] ~ stockprice$DateID[1:100],
     las = 1, ylab = "DAX index", xlab = "Date")
points(kern_smooth(stockprice$DAX[1:100], stockprice$DateID[1:100], h = 10),
       type = "l", col = "orange")
points(kern_smooth(stockprice$DAX[1:100], stockprice$DateID[1:100], h = 10, kernel = "box"),
       type = "l", col = "blue")
```

```
points(kern_smooth(stockprice$DAX[1:100], stockprice$DateID[1:100], h = 10, kernel = "norm"),
       type = "l", col = "red")
legend("topright", fill = c("orange", "blue", "red"),
       legend = c("epanechnikov", "box", "normal"), bty = "n", title = "Kernel method")
```

`stockprice` *Daily Closing Prices of Major European Stock Indices, April 2000–December 2017*

Description

A dataset containing the stockmarket returns between 2000-04-03 and 2017-12-05. This dataset is very close to the one used by Choi & Shin (2021), although not strictly identical. It has been produced by the Oxford-Man Institute of Quantitative Finance.

Usage

```
stockprice
```

Format

A data frame with 4618 rows and 7 variables:

DateID a vector of `Date`.

SP500 a numeric vector of the stockmarket return for the S&P 500 Index.

FTSE100 a numeric vector of the stockmarket return for the FTSE 100.

Nikkei a numeric vector of the stockmarket return for the Nikkei 225.

DAX a numeric vector of the stockmarket return for the German stock index.

NASDAQ a numeric vector of the stockmarket return for the Nasdaq Stock Market.

Event a character string of particular events that have impacted the stockmarket, as in Choi & Shin (2021).

Source

The file was downloaded from the "Oxford-Man Institute's realized library", which no longer exists. At the time, the raw data file was named "oxfordmanrealizedvolatilityindices-0.2-final.zip".

References

Heber, Gerd, Asger Lunde, Neil Shephard and Kevin Sheppard (2009) "Oxford-Man Institute's realized library", Oxford-Man Institute, University of Oxford.

Choi, JE., Shin, D.W. Nonparametric estimation of time varying correlation coefficient. J. Korean Stat. Soc. 50, 333–353 (2021). doi:10.1007/s42952020000736

See Also

`datasets::EuStockMarkets` for a similar dataset, albeit formatted differently.

tcor *Compute time varying correlation coefficients*

Description

The function `tcor()` implements (together with its helper function `calc_rho()`) the nonparametric estimation of the time varying correlation coefficient proposed by Choi & Shin (2021). The general idea is to compute a (Pearson) correlation coefficient ($r(x,y) = \frac{\hat{xy}-\hat{x}\times\hat{y}}{\sqrt{\hat{x^2}-\hat{x}^2}\times\sqrt{\hat{y^2}-\hat{y}^2}}$), but instead of using the means required for such a computation, each component (i.e., x, y, x^2, y^2, $x \times y$) is smoothed and the smoothed terms are considered in place the original means. The intensity of the smoothing depends on a unique parameter: the bandwidth (`h`). If `h = Inf`, the method produces the original (i.e., time-invariant) correlation value. The smaller the parameter `h`, the more variation in time is being captured. The parameter `h` can be provided by the user; otherwise it is automatically estimated by the internal helper functions `select_h()` and `calc_RMSE()` (see **Details**).

Usage

```
tcor(
  x,
  y,
  t = seq_along(x),
  h = NULL,
  cor.method = c("pearson", "spearman"),
  kernel = c("epanechnikov", "box", "normal"),
  CI = FALSE,
  CI.level = 0.95,
  param_smoother = list(),
  keep.missing = FALSE,
  verbose = FALSE
)

calc_rho(
  x,
  y,
  t = seq_along(x),
  t.for.pred = t,
  h,
  cor.method = c("pearson", "spearman"),
  kernel = c("epanechnikov", "box", "normal"),
  param_smoother = list()
)

calc_RMSE(
  h,
  x,
  y,
```

```
  t = seq_along(x),
  cor.method = c("pearson", "spearman"),
  kernel = c("epanechnikov", "box", "normal"),
  param_smoother = list(),
  verbose = FALSE
)

select_h(
  x,
  y,
  t = seq_along(x),
  cor.method = c("pearson", "spearman"),
  kernel = c("epanechnikov", "box", "normal"),
  param_smoother = list(),
  verbose = FALSE
)
```

Arguments

x	a numeric vector.
y	a numeric vector of to be correlated with x.
t	a (numeric or Date) vector of time points. If missing, observations are considered to correspond to sequential time steps (i.e., 1, 2 ...).
h	a scalar indicating the bandwidth used by the smoothing function.
cor.method	a character string indicating which correlation coefficient is to be computed ("pearson", the default; or "spearman").
kernel	a character string indicating which kernel to use: "epanechnikov" (the default), "box", or "normal" (abbreviations also work).
CI	a logical specifying if a confidence interval should be computed or not (default = FALSE).
CI.level	a scalar defining the level for CI (default = 0.95 for 95% CI).
param_smoother	a list of additional parameters to provide to the internal smoothing function (see **Details**).
keep.missing	a logical specifying if time points associated with missing information should be kept in the output (default = FALSE to facilitate plotting).
verbose	a logical specifying if information should be displayed to monitor the progress of the cross validation (default = FALSE).
t.for.pred	a (numeric or Date) vector of time points at which to evaluate the smoothed fit. If missing, t is used.

Details

- **Smoothing**: the smoothing of each component is performed by kernel regression. The default is to use the Epanechnikov kernel following Choi & Shin (2021), but other kernels have also been implemented and can thus alternatively be used (see kern_smooth() for details). The normal kernel seems to sometimes lead to very small bandwidth being selected, but the default

kernel can lead to numerical issues (see next point). We thus recommend always comparing the results from different kernel methods.

- **Numerical issues**: some numerical issues can happen because the smoothing is performed independently on each component of the correlation coefficient. As a consequence, some relationship between components may become violated for some time points. For instance, if the square of the smoothed x term gets larger than the smoothed x^2 term, the variance of x would become negative. In such cases, coefficient values returned are `NA`.
- **Bandwidth selection**: when the value used to define the bandwidth (`h`) in `tcor()` is set to `NULL` (the default), the internal function `select_h()` is used to to select the optimal value for `h`. It is first estimated by leave-one-out cross validation (using internally `calc_RMSE()`). If the cross validation error (RMSE) is minimal for the maximal value of `h` considered ($8\sqrt{N}$), rather than taking this as the optimal `h` value, the bandwidth becomes estimated using the so-called elbow criterion. This latter method identifies the value `h` after which the cross validation error decreasing very little. The procedure is detailed in section 2.1 in Choi & Shin (2021).
- **Parallel computation**: if `h` is not provided, an automatic bandwidth selection occurs (see above). For large datasets, this step can be computationally demanding. The current implementation thus relies on `parallel::mclapply()` and is thus only effective for Linux and MacOS. Relying on parallel processing also implies that you call `options("mc.cores" = XX)` beforehand, replacing `XX` by the relevant number of CPU cores you want to use (see **Examples**). For debugging, do use `options("mc.cores" = 1)`, otherwise you may not be able to see the error messages generated in child nodes.
- **Confidence interval**: if `CI` is set to `TRUE`, a confidence interval is calculated as described in Choi & Shin (2021). This is also necessary for using `test_equality()` to test differences between correlations at two time points. The computation of the confidence intervals involves multiple internal functions (see `CI` for details).

Value

—Output for `tcor()`—

A 2 x t dataframe containing:

- the time points (`t`).
- the estimates of the correlation value (`r`).

Or, if `CI = TRUE`, a 5 x t dataframe containing:

- the time points (`t`).
- the estimates of the correlation value (`r`).
- the Standard Error (`SE`).
- the lower boundary of the confidence intervals (`lwr`).
- the upper boundary of the confidence intervals (`upr`).

Some metadata are also attached to the dataframe (as attributes):

- the call to the function (`call`).
- the argument `CI`.
- the bandwidth parameter (`h`).

- the method used to select h (h_selection).
- the minimal root mean square error when h is selected (RMSE).
- the computing time (in seconds) spent to select the bandwidth parameter (h_selection_duration) if h automatically selected.

—Output for `calc_rho()`**—**

A 14 x t dataframe with:

- the six raw components of correlation (x, y, x2, y2, xy).
- the time points (t).
- the six raw components of correlation after smoothing (x_smoothed, y_smoothed, x2_smoothed, y2_smoothed, xy_smoothed).
- the standard deviation around x and y (sd_x_smoothed, sd_y_smoothed).
- the smoothed correlation coefficient (rho_smoothed).

—Output for `calc_RMSE()`**—**

A scalar of class numeric corresponding to the RMSE.

—Output for `select_h()`**—**

A list with the following components:

- the selected bandwidth parameter (h).
- the method used to select h (h_selection).
- the minimal root mean square error when h is selected (RMSE).
- the computing time (in seconds) spent to select the bandwidth parameter (time).

Functions

- `tcor()`: **the user-level function to be used**.
- `calc_rho()`: computes the correlation for a given bandwidth.
  The function calls the kernel smoothing procedure on each component required to compute the time-varying correlation.
- `calc_RMSE()`: Internal function computing the root mean square error (RMSE) for a given bandwidth.
  The function removes each time point one by one and predicts the correlation at the missing time point based on the other time points. It then computes and returns the RMSE between this predicted correlation and the one predicted using the full dataset. See also *Bandwidth selection* and *Parallel computation* in **Details**.
- `select_h()`: Internal function selecting the optimal bandwidth parameter h.
  The function selects and returns the optimal bandwidth parameter h using an optimizer (`stats::optimize()`) which searches the h value associated with the smallest RMSE returned by `calc_RMSE()`. See also *Bandwidth selection* in **Details**.

References

Choi, JE., Shin, D.W. Nonparametric estimation of time varying correlation coefficient. J. Korean Stat. Soc. 50, 333–353 (2021). doi:10.1007/s42952020000736

See Also

test_equality, kern_smooth, CI

Examples

```
#####################################################
## Examples for the user-level function to be used ##
#####################################################

## Effect of the bandwidth

res_h50   <- with(stockprice, tcor(x = SP500, y = FTSE100, t = DateID, h = 50))
res_h100  <- with(stockprice, tcor(x = SP500, y = FTSE100, t = DateID, h = 100))
res_h200  <- with(stockprice, tcor(x = SP500, y = FTSE100, t = DateID, h = 200))
plot(res_h50, type = "l", ylab = "Cor", xlab = "Time", las = 1, col = "grey")
points(res_h100, type = "l", col = "blue")
points(res_h200, type = "l", col = "red")
legend("bottom", horiz = TRUE, fill = c("grey", "blue", "red"),
       legend = c("50", "100", "200"), bty = "n", title = "Bandwidth (h)")


## Effect of the correlation method

res_pearson  <- with(stockprice, tcor(x = SP500, y = FTSE100, t = DateID, h = 150))
res_spearman <- with(stockprice, tcor(x = SP500, y = FTSE100, t = DateID, h = 150,
                                      cor.method = "spearman"))
plot(res_pearson, type = "l", ylab = "Cor", xlab = "Time", las = 1)
points(res_spearman, type = "l", col = "blue")
legend("bottom", horiz = TRUE, fill = c("black", "blue"),
       legend = c("pearson", "spearman"), bty = "n", title = "cor.method")


## Infinite bandwidth should match fixed correlation coefficients
## nb: `h = Inf` is not supported by default kernel (`kernel = 'epanechnikov'`)

res_pearson_hInf  <- with(stockprice, tcor(x = SP500, y = FTSE100, t = DateID, h = Inf,
                                           kernel = "normal"))
res_spearman_hInf <- with(stockprice, tcor(x = SP500, y = FTSE100, t = DateID, h = Inf,
                                           kernel = "normal", cor.method = "spearman"))
r <- cor(stockprice$SP500, stockprice$FTSE100, use = "pairwise.complete.obs")
rho <- cor(stockprice$SP500, stockprice$FTSE100, method = "spearman", use = "pairwise.complete.obs")
round(unique(res_pearson_hInf$r) - r, digits = 3) ## 0 indicates near equality
round(unique(res_spearman_hInf$r) - rho, digits = 3) ## 0 indicates near equality


## Computing and plotting the confidence interval

res_withCI <- with(stockprice, tcor(x = SP500, y = FTSE100, t = DateID, h = 200, CI = TRUE))
with(res_withCI, {
     plot(r ~ t, type = "l", ylab = "Cor", xlab = "Time", las = 1, ylim = c(0, 1))
     points(lwr ~ t, type = "l", lty = 2)
     points(upr ~ t, type = "l", lty = 2)})
```

```
## Same using tidyverse packages (dplyr and ggplot2 must be installed)
## see https://github.com/courtiol/timevarcorr for more examples of this kind

if (require("dplyr", warn.conflicts = FALSE)) {

  stockprice |>
    reframe(tcor(x = SP500, y = FTSE100, t = DateID,
                 h = 200, CI = TRUE)) -> res_tidy
  res_tidy
}

if (require("ggplot2")) {

  ggplot(res_tidy) +
     aes(x = t, y = r, ymin = lwr, ymax = upr) +
     geom_ribbon(fill = "grey") +
     geom_line() +
     labs(title = "SP500 vs FTSE100", x = "Time", y = "Correlation") +
     theme_classic()

}


## Automatic selection of the bandwidth using parallel processing and comparison
## of the 3 alternative kernels on the first 500 time points of the dataset
# nb: takes a few seconds to run, so not run by default

run <- in_pkgdown() || FALSE ## change to TRUE to run the example
if (run) {

options("mc.cores" = 2L) ## CPU cores to be used for parallel processing

res_hauto_epanech <- with(stockprice[1:500, ],
         tcor(x = SP500, y = FTSE100, t = DateID, kernel = "epanechnikov")
         )

res_hauto_box <- with(stockprice[1:500, ],
          tcor(x = SP500, y = FTSE100, t = DateID, kernel = "box")
          )

res_hauto_norm <- with(stockprice[1:500, ],
          tcor(x = SP500, y = FTSE100, t = DateID, kernel = "norm")
          )

plot(res_hauto_epanech, type = "l", col = "red",
     ylab = "Cor", xlab = "Time", las = 1, ylim = c(0, 1))
points(res_hauto_box, type = "l", col = "grey")
points(res_hauto_norm, type = "l", col = "orange")
legend("top", horiz = TRUE, fill = c("red", "grey", "orange"),
       legend = c("epanechnikov", "box", "normal"), bty = "n",
       title = "Kernel")
```

```
}


## Comparison of the 3 alternative kernels under same bandwidth
## nb: it requires to have run the previous example

if (run) {

res_epanech <- with(stockprice[1:500, ],
          tcor(x = SP500, y = FTSE100, t = DateID,
          kernel = "epanechnikov", h = attr(res_hauto_epanech, "h"))
          )

res_box <- with(stockprice[1:500, ],
           tcor(x = SP500, y = FTSE100, t = DateID,
           kernel = "box", h = attr(res_hauto_epanech, "h"))
           )

res_norm <- with(stockprice[1:500, ],
          tcor(x = SP500, y = FTSE100, t = DateID,
          kernel = "norm", h = attr(res_hauto_epanech, "h"))
          )

plot(res_epanech, type = "l", col = "red", ylab = "Cor", xlab = "Time",
     las = 1, ylim = c(0, 1))
points(res_box, type = "l", col = "grey")
points(res_norm, type = "l", col = "orange")
legend("top", horiz = TRUE, fill = c("red", "grey", "orange"),
       legend = c("epanechnikov", "box", "normal"), bty = "n",
       title = "Kernel")

}

## Automatic selection of the bandwidth using parallel processing with CI
# nb: takes a few seconds to run, so not run by default

run <- in_pkgdown() || FALSE ## change to TRUE to run the example
if (run) {

res_hauto_epanechCI <- with(stockprice[1:500, ],
          tcor(x = SP500, y = FTSE100, t = DateID, CI = TRUE)
          )

plot(res_hauto_epanechCI[, c("t", "r")], type = "l", col = "red",
     ylab = "Cor", xlab = "Time", las = 1, ylim = c(0, 1))
points(res_hauto_epanechCI[, c("t", "lwr")], type = "l", col = "red", lty = 2)
points(res_hauto_epanechCI[, c("t", "upr")], type = "l", col = "red", lty = 2)

}


## Not all kernels work well in all situations
```

```
## Here the default kernell estimation leads to issues for last time points
## nb1: EuStockMarkets is a time-series object provided with R
## nb2: takes a few minutes to run, so not run by default

run <- in_pkgdown() || FALSE ## change to TRUE to run the example
if (run) {

EuStock_epanech <- tcor(EuStockMarkets[1:500, "DAX"], EuStockMarkets[1:500, "SMI"])
EuStock_norm <- tcor(EuStockMarkets[1:500, "DAX"], EuStockMarkets[1:500, "SMI"], kernel = "normal")

plot(EuStock_epanech, type = "l", col = "red", las = 1, ylim = c(-1, 1))
points(EuStock_norm, type = "l", col = "orange", lty = 2)
legend("bottom", horiz = TRUE, fill = c("red", "orange"),
       legend = c("epanechnikov", "normal"), bty = "n",
       title = "Kernel")
}



#####################################################################
## Examples for the internal function computing the correlation ##
#####################################################################

## Computing the correlation and its component for the first six time points

with(head(stockprice), calc_rho(x = SP500, y = FTSE100, t = DateID, h = 20))


## Predicting the correlation and its component at a specific time point

with(head(stockprice), calc_rho(x = SP500, y = FTSE100, t = DateID, h = 20,
     t.for.pred = DateID[1]))


## The function can handle non consecutive time points

set.seed(1)
calc_rho(x = rnorm(10), y = rnorm(10), t = c(1:5, 26:30), h = 3, kernel = "box")


## The function can handle non-ordered time series

with(head(stockprice)[c(1, 3, 6, 2, 4, 5), ], calc_rho(x = SP500, y = FTSE100, t = DateID, h = 20))


## Note: the function does not handle missing data (by design)

# calc_rho(x = c(NA, rnorm(9)), y = rnorm(10), t = c(1:2, 23:30), h = 2) ## should err (if ran)



#############################################################
```

```
## Examples for the internal function computing the RMSE ##
###########################################################

## Compute the RMSE on the correlation estimate
# nb: takes a few seconds to run, so not run by default

run <- in_pkgdown() || FALSE ## change to TRUE to run the example
if (run) {

small_clean_dataset <- head(na.omit(stockprice), n = 200)
with(small_clean_dataset, calc_RMSE(x = SP500, y = FTSE100, t = DateID, h = 10))

}




#################################################################
## Examples for the internal function selecting the bandwidth ##
#################################################################

## Automatic selection of the bandwidth using parallel processing
# nb: takes a few seconds to run, so not run by default

run <- in_pkgdown() || FALSE ## change to TRUE to run the example
if (run) {

small_clean_dataset <- head(na.omit(stockprice), n = 200)
with(small_clean_dataset, select_h(x = SP500, y = FTSE100, t = DateID))

}
```

`test_equality` *Compute equality test between correlation coefficient estimates at two time points*

Description

This function tests whether smoothed correlation values at two time points are equal (H0) or not. The test is described page 341 in Choi & Shin (2021).

Usage

```
test_equality(
  tcor_obj,
  t1 = 1,
  t2 = nrow(tcor_obj),
  test = c("student", "chi2")
)
```

Arguments

tcor_obj	the output of a call to tcor() with CI = TRUE.
t1	the first time point used by the test (by default, the first time point in the time series).
t2	the second time point used by the test (by default, the last time point in the time series).
test	a character string indicating which test to use ("student", the default; or "chi2").

Details

Two different test statistics can be used, one is asymptotically Student-t distributed under H0 and one is chi-square distributed. In practice, it seems to give very similar results.

Value

a data.frame with the result of the test, including the effect size (delta_r = r[t2] - r[t1]).

See Also

test_ref(), tcor()

Examples

```
## Simple example

res <- with(stockprice, tcor(x = SP500, y = FTSE100, t = DateID, h = 50, CI = TRUE))
test_equality(res)

## Chi2 instead of Student's t-test

test_equality(res, test = "chi2")


## Time point can be dates or indices (mixing possible) but output as in input data

test_equality(res, t1 = "2000-04-04", t2 = 1000)
res[1000, "t"] ## t2 matches with date in `res`
stockprice[1000, "DateID"] ## t2 does not match with date `stockprice` due to missing values


## It could be useful to use `keep.missing = TRUE` for index to match original data despite NAs

res2 <- with(stockprice, tcor(x = SP500, y = FTSE100, t = DateID,
                              h = 50, CI = TRUE, keep.missing = TRUE))
test_equality(res2, t1 = "2000-04-04", t2 = 1000)
res[1000, "t"] ## t2 matches with date in `res`
stockprice[1000, "DateID"] ## t2 does match with date `stockprice` despite missing values
```

test_ref *Test difference between correlation coefficient estimates and a value of reference*

Description

This function tests whether smoothed correlation values are equal (H0) or not to a reference value (default = 0). The test is not described in Choi & Shin, 2021, but it is based on the idea behind test_equality().

Usage

```
test_ref(
  tcor_obj,
  t = tcor_obj$t,
  r_ref = 0,
  test = c("student", "chi2"),
 p.adjust.methods = c("none", "bonferroni", "holm", "hochberg", "hommel", "BH", "BY",
    "fdr")
)
```

Arguments

tcor_obj	the output of a call to tcor() with CI = TRUE.
t	a vector of time point(s) used by the test (by default, all time points are considered).
r_ref	a scalar indicating the reference value for the correlation coefficient to be used in the test (default = 0).
test	a character string indicating which test to use ("student", the default; or "chi2").
p.adjust.methods	a character string indicating the method used to adjust p-values for multiple testing (see p.adjust(); default = "none").

Details

Two different test statistics can be used, one is asymptotically Student-t distributed under H0 and one is chi-square distributed. In practice, it seems to give very similar results.

Value

a data.frame with the result of the test, including the effect size (delta_r = r[t] - r_ref).

See Also

test_equality(), tcor()

Examples

```
## Comparison of all correlation values to reference of 0.5

res <- with(stockprice, tcor(x = SP500, y = FTSE100, t = DateID, h = 300, CI = TRUE))
ref <- 0.5
test_against_ref <- test_ref(res, r_ref = ref)
head(test_against_ref)


## Plot to illustrate the correspondance with confidence intervals

plot(res$r ~ res$t, type = "l", ylim = c(0, 1), col = NULL)
abline(v = test_against_ref$t[test_against_ref$p > 0.05], col = "lightgreen")
abline(v = test_against_ref$t[test_against_ref$p < 0.05], col = "red")
points(res$r ~ res$t, type = "l")
points(res$upr ~ res$t, type = "l", lty = 2)
points(res$lwr ~ res$t, type = "l", lty = 2)
abline(h = ref, col = "blue")


## Test correlation of 0 a specific time points (using index or dates)

test_ref(res, t = c(100, 150))
test_ref(res, t = c("2000-08-18", "2000-10-27"))
```

Index

∗ **datasets**
    stockprice, 8
.onAttach, 2

calc_D *(*CI*)*, 2
calc_e *(*CI*)*, 2
calc_Gamma *(*CI*)*, 2
calc_GammaINF *(*CI*)*, 2
calc_H *(*CI*)*, 2
calc_L_And *(*CI*)*, 2
calc_rho, *3*
calc_rho *(*tcor*)*, 9
calc_RMSE *(*tcor*)*, 9
calc_RMSE(), *12*
calc_SE *(*CI*)*, 2
CI, 2, *11*, *13*

datasets::EuStockMarkets, *8*

in_pkgdown, 5

kern_smooth, 6, *13*
kern_smooth(), *10*

lpridge::lpepa(), *6*

p.adjust(), *19*
parallel::mclapply(), *11*
pkgdown::in_pkgdown(), *5*

select_h *(*tcor*)*, 9
stats::ar, *3*
stats::ksmooth(), *6*
stats::optimize(), *12*
stockprice, 8

tcor, *7*, 9
tcor(), *2*, *4*, *18, 19*
test_equality, *13*, 17
test_equality(), *11*, *19*
test_ref, 19
test_ref(), *18*