Package 'funtimes'

July 22, 2025

Type Package

Title Functions for Time Series Analysis

Version 9.1

Date 2023-03-21

Depends R (>= 3.5.0)

License GPL (>= 2)

Imports dbscan, Kendall, lmtest, mlVAR, parallel, Rdpack, sandwich, vars

Suggests covid19us, Ecdat, ggplot2, gridExtra, knitr, patchwork, randomcoloR, readxl, reshape2, rmarkdown

Description Nonparametric estimators and tests for time series analysis. The functions use bootstrap techniques and robust nonparametric difference-based estimators to test for the presence of possibly non-monotonic trends and for synchronicity of trends in multiple time series.

RdMacros Rdpack

RoxygenNote 7.2.3

Encoding UTF-8

VignetteBuilder knitr, rmarkdown

NeedsCompilation no

Author Vyacheslav Lyubchich [aut, cre] (ORCID: <https://orcid.org/0000-0001-7936-4285>), Yulia R. Gel [aut], Alexander Brenning [ctb], Calvin Chu [ctb], Xin Huang [ctb], Umar Islambekov [ctb], Palina Niamkova [ctb], Dorcas Ofori-Boateng [ctb], Ethan D. Schaeffer [ctb], Srishti Vishwakarma [aut], Xingyu Wang [ctb]

Maintainer Vyacheslav Lyubchich <lyubchich@umces.edu>

Repository CRAN

Date/Publication 2023-03-21 23:40:02 UTC

Contents

`funtimes-package` *funtimes: Functions for Time Series Analysis*

Description

Advances in multiple aspects of time-series analysis are documented in this package. See available vignettes using
`browseVignettes(package = "funtimes")`

Tests for trends applicable to autocorrelated data, see
`vignette("trendtests", package = "funtimes")`
include bootstrapped versions of t-test and Mann–Kendall test (Noguchi et al. 2011) and bootstrapped version of WAVK test for possibly non-monotonic trends (Lyubchich et al. 2013). The WAVK test is further applied in testing synchronicity of trends (Lyubchich and Gel 2016); see an implementation to climate data in Lyubchich (2016). With iterative testing, the synchronicity test is also applied for identifying clusters of multiple time series (Ghahari et al. 2017).

Additional clustering methods are implemented using functions `BICC` (Schaeffer et al. 2016) and `DR` (Huang et al. 2018); function `purity` can be used to assess the accuracy of clustering if true classes are known.

Changepoint detection methods include modified CUSUM-based bootstrapped test (Lyubchich et al. 2020).

Additional functions include implementation of the Beale's ratio estimator, see
`vignette("beales", package = "funtimes")`
Nonparametric comparison of tails of distributions is implemented using small bins defined based on quantiles (Soliman et al. 2015) or intervals in the units in which the data are recorded (Lyubchich and Gel 2017).

For a list of currently deprecated functions, use `?'funtimes-deprecated'`

For a list of defunct (removed) functions, use `?'funtimes-defunct'`

Author(s)

Maintainer: Vyacheslav Lyubchich <lyubchich@umces.edu> (ORCID)

Authors:

- Yulia R. Gel
- Srishti Vishwakarma

Other contributors:

- Alexander Brenning [contributor]
- Calvin Chu [contributor]
- Xin Huang [contributor]
- Umar Islambekov [contributor]
- Palina Niamkova [contributor]
- Dorcas Ofori-Boateng [contributor]
- Ethan D. Schaeffer [contributor]
- Xingyu Wang [contributor]

References

Ghahari A, Gel YR, Lyubchich V, Chun Y, Uribe D (2017). "On employing multi-resolution weather data in crop insurance." In *Proceedings of the SIAM International Conference on Data Mining (SDM17) Workshop on Mining Big Data in Climate and Environment (MBDCE 2017)*.

Huang X, Iliev IR, Lyubchich V, Gel YR (2018). "Riding down the bay: space-time clustering of ecological trends." *Environmetrics*, **29**(5–6), e2455. doi:10.1002/env.2455.

Lyubchich V (2016). "Detecting time series trends and their synchronization in climate data." *Intelligence. Innovations. Investments*, **12**, 132–137.

Lyubchich V, Gel YR (2016). "A local factor nonparametric test for trend synchronism in multiple time series." *Journal of Multivariate Analysis*, **150**, 91–104. doi:10.1016/j.jmva.2016.05.004.

Lyubchich V, Gel YR (2017). "Can we weather proof our insurance?" *Environmetrics*, **28**(2), e2433. doi:10.1002/env.2433.

Lyubchich V, Gel YR, El-Shaarawi A (2013). "On detecting non-monotonic trends in environmental time series: a fusion of local regression and bootstrap." *Environmetrics*, **24**(4), 209–226. doi:10.1002/env.2212.

Lyubchich V, Lebedeva TV, Testa JM (2020). "A data-driven approach to detecting change points in linear regression models." *Environmetrics*, **31**(1), e2591. doi:10.1002/env.2591.

Noguchi K, Gel YR, Duguay CR (2011). "Bootstrap-based tests for trends in hydrological time series, with application to ice phenology data." *Journal of Hydrology*, **410**(3), 150–161. doi:10.1016/j.jhydrol.2011.09.008.

Schaeffer ED, Testa JM, Gel YR, Lyubchich V (2016). "On information criteria for dynamic spatio-temporal clustering." In Banerjee A, Ding W, Dy JG, Lyubchich V, Rhines A (eds.), *The 6th International Workshop on Climate Informatics: CI2016*, 5–8. doi:10.5065/D6K072N6.

Soliman M, Lyubchich V, Gel YR, Naser D, Esterby S (2015). "Evaluating the impact of climate change on dynamics of house insurance claims." In Lakshmanan V, Gilleland E, McGovern A, Tingley M (eds.), *Machine Learning and Data Mining Approaches to Climate Science*, chapter 16, 175–183. Springer, Switzerland. doi:10.1007/9783319172200_16.

`ARest` *Estimation of Autoregressive (AR) Parameters*

Description

Estimate parameters ϕ of autoregressive time series model

$$X_t = \sum_{i=1}^{p} \phi_i X_{t-i} + e_t,$$

by default using robust difference-based estimator and Bayesian information criterion (BIC) to select the order p. This function is employed for time series filtering in the functions `notrend_test`, `sync_test`, and `wavk_test`.

Usage

```
ARest(x, ar.order = NULL, ar.method = "HVK", ic = c("BIC", "AIC", "none"))
```

Arguments

`x`	a vector containing a univariate time series. Missing values are not allowed.
`ar.order`	order of the autoregressive model when `ic = "none"`, or the maximal order for IC-based filtering. Default is `round(10*log10(length(x)))`, where `x` is the time series.

`ar.method` method of estimating autoregression coefficients. Default `"HVK"` delivers robust difference-based estimates by Hall and Van Keilegom (2003). Alternatively, options of `ar` function can be used, such as `"burg"`, `"ols"`, `"mle"`, and `"yw"`.

`ic` information criterion used to select the order of autoregressive filter (AIC of BIC), considering models of orders p = 0,1,...,`ar.order`. If `ic = "none"`, the AR(p) model with p = `ar.order` is used, without order selection.

Details

The formula for information criteria used consistently for all methods:

$$IC = n \ln(\hat{\sigma}^2) + (p+1)k,$$

where n = `length(x)`, p is the autoregressive order ($p+1$ is the number of model parameters), and k is the penalty ($k = \ln(n)$ in BIC, and $k = 2$ in AIC).

Value

A vector of estimated AR coefficients. Returns `numeric(0)` if the final $p = 0$.

Author(s)

Vyacheslav Lyubchich

References

Hall P, Van Keilegom I (2003). "Using difference-based methods for inference in nonparametric regression with time series errors." *Journal of the Royal Statistical Society, Series B (Statistical Methodology)*, **65**(2), 443–456. doi:10.1111/14679868.00395.

See Also

`ar`, `HVK`, `notrend_test`, `sync_test`, `wavk_test`

Examples

```
# Simulate a time series Y:
Y <- arima.sim(n = 200, list(order = c(2, 0, 0), ar = c(-0.7, -0.1)))
plot.ts(Y)

# Estimate the coefficients:
ARest(Y) # HVK, by default
ARest(Y, ar.method = "yw") # Yule--Walker
ARest(Y, ar.method = "burg") # Burg
```

AuePolyReg_test *Testing for Change Points in Time Series via Polynomial Regression*

Description

The function uses a nonlinear polynomial regression model in which it tests for the null hypothesis of structural stability in the regression parameters against the alternative of a break at an unknown time. The method is based on the extreme value distribution of a maximum-type test statistic which is asymptotically equivalent to the maximally selected likelihood ratio. The resulting testing approach is easily tractable and delivers accurate size and power of the test, even in small samples (Aue et al. 2008).

Usage

```
AuePolyReg_test(
  y,
  a.order,
  alpha = 0.05,
  crit.type = c("asymptotic", "bootstrap"),
  bootstrap.method = c("nonparametric", "parametric"),
  num.bootstrap = 1000
)
```

Arguments

y	a vector that contains univariate time series observations. Missing values are not allowed.
a.order	order of the autoregressive model which must be a non-negative integer number.
alpha	significance level for testing hypothesis of no change point. Default value is 0.05.
crit.type	method of obtaining critical values: "asymptotic" (default) or "bootstrap".
bootstrap.method	type of bootstrap if crit.type = "bootstrap": "nonparametric" (default) or "parametric".
num.bootstrap	number of bootstrap replications if crit.type = "bootstrap". Default number is 1000.

Value

A list with the following components:

index	time point where the change point has occurred.
stat	test statistic.
crit.val	critical region value (CV(alpha, n)).
p.value	p-value of the change point test.

Author(s)

Palina Niamkova, Dorcas Ofori-Boateng, Yulia R. Gel

References

Aue A, Horvath L, Huskova M, Kokoszka P (2008). "Testing for changes in polynomial regression." *Bernoulli*, **14**(3), 637–660. doi:10.3150/08BEJ122.

See Also

`mcusum.test` change point test for regression

Examples

```
## Not run:
#Example 1:

#Simulate some time series:
set.seed(23450)
series_1 = rnorm(137, 3, 5)
series_2 = rnorm(213, 0, 1)
series_val = c(series_1, series_2)
AuePolyReg_test(series_1, 1) # no change (asymptotic)
AuePolyReg_test(series_val,1) # one change (asymptotic)

#Example 2:

#Consider a time series with annual number of world terrorism incidents from 1970 till 2016:
c.data = Ecdat::terrorism["incidents"]
incidents.ts <- ts(c.data, start = 1970, end = 2016)

#Run a test for change points:
AuePolyReg_test(incidents.ts, 2) # one change (asymptotic)
AuePolyReg_test(incidents.ts, 2, 0.05,"bootstrap", "parametric", 200)
# one change (bootstrap)
incidents.ts[44] #number of victims at the value of change point
year <- 197 + 44 - 1  # year when the change point occurred
plot(incidents.ts) # see the visualized data

#The structural change point occurred at the 44th value which corresponds to 2013,
#with 11,990 identified incidents in that year. These findings can be explained with
#a recent rise of nationalism and  extremism due to appearance of the social media,
#Fisher (2019): White Terrorism Shows 'Stunning' Parallels to Islamic State's Rise.
#The New York Times.

## End(Not run)
```

`beales` *Beale's Estimator and Sample Size*

Description

Beale's ratio estimator (Beale 1962) for estimating population total and confidence intervals, with an option of calculating sample size for a required relative error (`p`) or margin of error (`d`).

Usage

```
beales(x, y, level = 0.95, N = NULL, p = NULL, d = NULL, verbose = TRUE)
```

Arguments

`x`	a numeric vector with quantities of interest, such as river discharge per month. Missing values (`NA`) are allowed.
`y`	a numeric vector with quantities of interest for which the total shall be estimated, such as total nutrient loads per month. Missing values (`NA`) are allowed. Lengths of `x` and `y` mush be the same.
`level`	confidence level, from 0 to 1. Default is `0.95`, that is, 95% confidence.
`N`	population size for which the estimate of the total `y` is required. By default, `length(x)` is used.
`p`	optional argument specifying the required relative error, from 0 to 1, for computing the corresponding sample size. For example, `p = 0.15` defines a 15% relative error.
`d`	optional argument specifying the required margin of error for computing the corresponding sample size. If both `p` and `d` are specified, only `p` is used.
`verbose`	logical value defining whether the output should be printed out in words. Default is set to `TRUE` to give such output.

Value

A list with the following components:

`estimate`	Beale's estimate of the population total for the variable `y`.
`se`	standard error of the estimate.
`CI`	a vector of length 2 with a confidence interval (lower and upper value) for the estimate.
`level`	confidence level for the interval.
`N`	population size.
`n`	the actual sample size.
`p`	the relative error used for sample size calculations. Reported only if `p` was specified in the input.
`d`	the margin of error used for sample size calculations. Reported only if `d` was specified and `p` was not specified in the input.
`nhat`	estimated sample size for the given `level` and error (`p` or `d`).

Author(s)

Vyacheslav Lyubchich, thanks to Dave Lorenz for pointing out an error in version 7 and below of the package

References

Beale EML (1962). "Some uses of computers in operational research." *Industrielle Organisation*, **31**(1), 27–28.

See Also

`vignette("beales", package = "funtimes")`

Examples

```
#Some hypothetical data for monthly river discharge
#and corresponding nutrient loads:
discharge <- c(NA, 50, 90, 100, 80, 90, 100, 90, 80, 70, NA, NA)
loads <- c(33, 22, 44, 48, NA, 44, 49, NA, NA, 36, NA, NA)

#Example 1:
#Estimate total annual load (12 months),
#with 90% confidence intervals
beales(discharge, loads, level = 0.9)

#Example 2:
#Calculate sample size required for 90% confidence intervals
#with a margin of error 30 units
beales(discharge, loads, level = 0.9, d = 30)
```

BICC *BIC-Based Spatio-Temporal Clustering*

Description

Apply the algorithm of unsupervised spatio-temporal clustering, TRUST (Ciampi et al. 2010), with automatic selection of its tuning parameters `Delta` and `Epsilon` based on Bayesian information criterion, BIC (Schaeffer et al. 2016).

Usage

```
BICC(X, Alpha = NULL, Beta = NULL, Theta = 0.8, p, w, s)
```

Arguments

X	a matrix of time series observed within a slide (time series in columns).
Alpha	lower limit of the time-series domain, passed to CSlideCluster.
Beta	upper limit of the time-series domain passed to CSlideCluster.
Theta	connectivity parameter passed to CSlideCluster.
p	number of layers (time-series observations) in each slide.
w	number of slides in each window.
s	step to shift a window, calculated in the number of slides. The recommended values are 1 (overlapping windows) or equal to w (non-overlapping windows).

Details

This is the upper-level function for time series clustering. It exploits the functions CWindowCluster and CSlideCluster to cluster time series based on closeness and homogeneity measures. Clustering is performed multiple times with a range of equidistant values for the parameters Delta and Epsilon, then optimal parameters Delta and Epsilon along with the corresponding clustering results are shown (see Schaeffer et al. 2016, for more details).

The total length of time series (number of levels, i.e., nrow(X)) should be divisible by p.

Value

A list with the following elements:

delta.opt	optimal value for the clustering parameter Delta.
epsilon.opt	optimal value for the clustering parameter Epsilon.
clusters	vector of length ncol(X) with cluster labels.
IC	values of the information criterion (BIC) for each considered combination of Delta (rows) and Epsilon (columns).
delta.all	vector of considered values for Delta.
epsilon.all	vector of considered values for Epsilon.

Author(s)

Ethan Schaeffer, Vyacheslav Lyubchich

References

Ciampi A, Appice A, Malerba D (2010). "Discovering trend-based clusters in spatially distributed data streams." In *International Workshop of Mining Ubiquitous and Social Environments*, 107–122.

Schaeffer ED, Testa JM, Gel YR, Lyubchich V (2016). "On information criteria for dynamic spatio-temporal clustering." In Banerjee A, Ding W, Dy JG, Lyubchich V, Rhines A (eds.), *The 6th International Workshop on Climate Informatics: CI2016*, 5–8. doi:10.5065/D6K072N6.

See Also

CSlideCluster, CWindowCluster, purity

Examples

```
# Fix seed for reproducible simulations:
set.seed(1)

##### Example 1
# Similar to Schaeffer et al. (2016), simulate 3 years of monthly data
#for 10 locations and apply clustering:
# 1.1 Simulation
T <- 36 #total months
N <- 10 #locations
phi <- c(0.5) #parameter of autoregression
burn <- 300 #burn-in period for simulations
X <- sapply(1:N, function(x)
    arima.sim(n = T + burn,
              list(order = c(length(phi), 0, 0), ar = phi)))[(burn + 1):(T + burn),]
colnames(X) <- paste("TS", c(1:dim(X)[2]), sep = "")

# 1.2 Clustering
# Assume that information arrives in year-long slides or data chunks
p <- 12 #number of time layers (months) in a slide
# Let the upper level of clustering (window) be the whole period of 3 years, so
w <- 3 #number of slides in a window
s <- w #step to shift a window, but it does not matter much here as we have only one window of data
tmp <- BICC(X, p = p, w = w, s = s)

# 1.3 Evaluate clustering
# In these simulations, it is known that all time series belong to one class,
#since they were all simulated the same way:
classes <- rep(1, 10)
# Use the information on the classes to calculate clustering purity:
purity(classes, tmp$clusters[1,])

##### Example 2
# 2.1 Modify time series and update classes accordingly:
# Add a mean shift to a half of the time series:
X2 <- X
X2[, 1:(N/2)] <- X2[, 1:(N/2)] + 3
classes2 <- rep(c(1, 2), each = N/2)

# 2.2 Re-apply clustering procedure and evaluate clustering purity:
tmp2 <- BICC(X2, p = p, w = w, s = s)
tmp2$clusters
purity(classes2, tmp2$clusters[1,])
```

causality_pred *Out-of-sample Tests of Granger Causality*

Description

Test for Granger causality using out-of-sample prediction errors from an autoregression (AR) model, where some of the near-contemporaneous lags can be removed:

$$Y_t = \sum_{i=1}^{p1} \alpha_i Y_{t-i} + \sum_{i=lag.restrict+1}^{p2} \beta_i X_{t-i} + e_t,$$

where Y_t is the dependent variable, X_t is the cause variable, $p1$ and $p2$ are the AR orders (if `p.free = FALSE`, $p1 = p2$), $lag.restrict$ is the number of restricted first lags (see the argument `lag.restrict`).

Usage

```
causality_pred(
  y,
  cause = NULL,
  p = NULL,
  p.free = FALSE,
  lag.restrict = 0L,
  lag.max = NULL,
  k = 2,
  B = 500L,
  test = 0.3,
  cl = 1L
)
```

Arguments

`y`	matrix, data frame, or `ts` object with two columns (a dependent and an explanatory time-series variable). Missing values are not allowed.
`cause`	name of the cause variable. If not specified, the first variable in `y` is treated as the dependent variable and the second is treated as the cause.
`p`	a vector of one or two positive integers specifying the order p of autoregressive dependence. The input of length one is recycled, then `p[1]` is used for the dependent variable and `p[2]` is used for the cause variable. The user must specify `p` or `lag.max`. If `lag.max` is specified, the argument `p` is ignored.
`p.free`	logical value indicating whether the autoregressive orders for the dependent and cause variables should be selected independently. The default `p.free = FALSE` means the same autoregressive order is selected for both variables. Note that if `p.free = TRUE` and `lag.max` is specified, then `lag.max[1] * (lag.max[2] - lag.restrict)` models are compared, which might be slow depending on the maximal lags and sample size.

`lag.restrict` integer for the number of short-term lags in the cause variable to remove from consideration (default is zero, meaning no lags are removed). This setting does not affect the dependent variable lags that are always present.

`lag.max` a vector of one or two positive integers for the highest lag orders to explore. The input of length one is recycled, then `lag.max[1]` used for the dependent variable and `lag.max[2]` is used for the cause variable. The order is then selected using the Akaike information criterion (AIC; default), see the argument `k` to change the criterion. `lag.max` of length 2 automatically sets `p.free = TRUE`.

`k` numeric scalar specifying the weight of the equivalent degrees of freedom part in the AIC formula. Default `k = 2` corresponds to the traditional AIC. Use `k = log(n)` to use the Bayesian information criterion instead (see `extractAIC`).

`B` number of bootstrap replications. Default is 500.

`test` a numeric value specifying the size of the testing set. If `test < 1`, the value is treated as a proportion of the sample size to be used as the testing set. Otherwise, `test` is treated as the number of the most recent values to be used as the testing set. Default is 0.3, which means that 30% of the sample is used for calculating out-of-sample errors. The testing set is always at the end of the time series.

`cl` parameter to specify computer cluster for bootstrapping passed to the package `parallel` (default `cl = 1`, means no cluster is used). Possible values are:

- cluster object (list) produced by makeCluster. In this case, a new cluster is not started nor stopped;
- `NULL`. In this case, the function will detect available cores (see detectCores) and, if there are multiple cores (> 1), a cluster will be started with makeCluster. If started, the cluster will be stopped after the computations are finished;
- positive integer defining the number of cores to start a cluster. If `cl = 1` (default), no attempt to create a cluster will be made. If `cl > 1`, a cluster will be started (using makeCluster) and stopped afterward (using stopCluster).

Details

The tests include the MSE-t approach (McCracken 2007) and MSE-correlation test as in Chapter 9.3 of Granger and Newbold (1986). The bootstrap is used to empirically derive distributions of the statistics.

In the implemented bootstrapping, residuals of the restricted model under the null hypothesis of no Granger causality are bootstrapped to generate new data under the null hypothesis. Then, the full and restricted models are re-estimated on the bootstrapped data to obtain new (bootstrapped) forecast errors.

In the current implementation, the bootstrapped p-value is calculated using Equation 4.10 in Davison and Hinkley (1997): `p.value = (1 + n) / (B + 1)`, where n is the number of bootstrapped statistics smaller or equal to the observed statistic.

This function tests the Granger causation of X to Y or from Y to X (to test in both directions, need to run the function twice, with different argument `cause`). To use the symmetric vector autoregression (VAR), use the function `causality_predVAR`.

Value

A list containing the following elements:

`stat`	a table with the observed values of the test statistics and p-values.
`cause`	the cause variable.
`p`	the AR orders used for the dependent variable (`p[1]`) and for the cause variable (`p[2]`).

Author(s)

Vyacheslav Lyubchich

References

Davison AC, Hinkley DV (1997). *Bootstrap Methods and Their Application*. Cambridge University Press, Cambridge.

Granger CWJ, Newbold P (1986). *Forecasting economic time series*, 2 edition. Academic Press.

McCracken MW (2007). "Asymptotics for out of sample tests of Granger causality." *Journal of Econometrics*, **140**(2), 719–752. doi:10.1016/j.jeconom.2006.07.020.

See Also

`causality_predVAR`

Examples

```
## Not run:
# Example 1: Canada time series (ts object)
Canada <- vars::Canada
causality_pred(Canada[,1:2], cause = "e", lag.max = 5, p.free = TRUE)
causality_pred(Canada[,1:2], cause = "e", lag.restrict = 3, lag.max = 15, p.free = TRUE)

# Example 2 (run in parallel, initiate the cluster automatically)
# Box & Jenkins time series
# of sales and a leading indicator, see ?BJsales

D <- cbind(BJsales.lead, BJsales)
causality_pred(D, cause = "BJsales.lead", lag.max = 5, B = 1000, cl = NULL)

# Example 3 (run in parallel, initiate the cluster manually)

# Initiate a local cluster
cores <- parallel::detectCores()
cl <- parallel::makeCluster(cores)
parallel::clusterSetRNGStream(cl, 123) # to make parallel computations reproducible

causality_pred(D, cause = "BJsales.lead", lag.max = 5, B = 1000, cl = cl)
causality_pred(D, cause = "BJsales.lead", lag.restrict = 3, p = 5, B = 1000, cl = cl)
```

```
parallel::stopCluster(cl)

## End(Not run)
```

`causality_predVAR` *Out-of-sample Tests of Granger Causality using (Restricted) Vector Autoregression*

Description

Test for Granger causality using out-of-sample prediction errors from a vector autoregression (VAR), where the original VAR can be restricted (see Details). The tests include the MSE-t approach (McCracken 2007) and MSE-correlation test as in Chapter 9.3 of Granger and Newbold (1986). The bootstrap is used to empirically derive distributions of the statistics.

Usage

```
causality_predVAR(
  y,
  p = NULL,
  cause = NULL,
  B = 500L,
  test = 0.3,
  cl = 1L,
  ...
)
```

Arguments

`y`	data frame or `ts` object for estimating VAR(p).
`p`	an integer specifying the order p in VAR. By default (if `p` is not specified), p is selected based on the information criterion `ic` (see `...` arguments; default `ic` is AIC).
`cause`	name of the cause variable. If not specified, the first variable in `y` is treated as the dependent variable and the second is treated as the cause.
`B`	number of bootstrap replications. Default is 500.
`test`	a numeric value specifying the size of the testing set. If `test` < 1, the value is treated as a proportion of the sample size to be used as the testing set. Otherwise, `test` is treated as the number of the most recent values to be used as the testing set. Default is 0.3, which means that 30% of the sample is used for calculating out-of-sample errors. The testing set is always at the end of the time series.
`cl`	parameter to specify computer cluster for bootstrapping passed to the package `parallel` (default `cl` = 1, means no cluster is used). Possible values are: • cluster object (list) produced by makeCluster. In this case, a new cluster is not started nor stopped;

- NULL. In this case, the function will detect available cores (see detectCores) and, if there are multiple cores (> 1), a cluster will be started with makeCluster. If started, the cluster will be stopped after the computations are finished;
- positive integer defining the number of cores to start a cluster. If cl = 1 (default), no attempt to create a cluster will be made. If cl > 1, a cluster will be started (using makeCluster) and stopped afterward (using stopCluster).

... other arguments passed to the function for VAR estimation. The arguments include lag.restrict that is used to remove the first lags in the cause variable from consideration (use restricted VAR to avoid testing for short-term causality); default lag.restrict = 0L, i.e., no restrictions. Other possible arguments are as in the VAR function. Also, see Details and Examples.

Details

The arguments specified in ... are passed to the VAR function. Additionally, lag.restrict can be specified to remove short-term lags from consideration (lag.restrict is not an option in the original package vars). Note that if p is specified, lag.restrict must be smaller than p otherwise the default lag.restrict = 0 will be used. If lag.max is specified instead of p, VAR orders lag.restrict + 1, ..., lag.max will be considered using the training data and the order p will be automatically selected according to the information criterion (by default, AIC).

In the current implementation, the bootstrapped p-value is calculated using equation 4.10 of Davison and Hinkley (1997): p.value = (1 + n) / (B + 1), where n is the number of bootstrapped statistics smaller or equal to the observed statistic. In the fast bootstrap, n is the number of bootstrapped statistics greater or equal to 0.

This function uses symmetric VAR with the same orders p for modeling both Y to X. To select these orders more independently, consider using the function causality_pred.

Value

Two lists (one for the fast bootstrap, another for the bootstrap under the null hypothesis) each containing the following elements:

result	a table with the observed values of the test statistics and p-values.
cause	the cause variable.
p	the AR order used.

Author(s)

Vyacheslav Lyubchich

References

Davison AC, Hinkley DV (1997). *Bootstrap Methods and Their Application*. Cambridge University Press, Cambridge.

Granger CWJ, Newbold P (1986). *Forecasting economic time series*, 2 edition. Academic Press.

McCracken MW (2007). "Asymptotics for out of sample tests of Granger causality." *Journal of Econometrics*, **140**(2), 719–752. doi:10.1016/j.jeconom.2006.07.020.

See Also

causality_pred

Examples

```
## Not run:
# Example 1: Canada time series (ts object)
Canada <- vars::Canada
causality_predVAR(Canada[,1:2], cause = "e", lag.max = 5)
causality_predVAR(Canada[,1:2], cause = "e", lag.restrict = 3, lag.max = 15)

# Example 2 (run in parallel, initiate the cluster manually):
# Box & Jenkins time series
# of sales and a leading indicator, see ?BJsales

# Initiate a local cluster
cores <- parallel::detectCores()
cl <- parallel::makeCluster(cores)
parallel::clusterSetRNGStream(cl, 123) # to make parallel computations reproducible

D <- cbind(BJsales.lead, BJsales)
causality_predVAR(D, cause = "BJsales.lead", lag.max = 5, B = 1000, cl = cl)
causality_predVAR(D, cause = "BJsales.lead", lag.restrict = 3, p = 5, B = 1000, cl = cl)
parallel::stopCluster(cl)

## End(Not run)
```

---

ccf_boot *Cross-Correlation of Autocorrelated Time Series*

Description

Account for possible autocorrelation of time series when assessing the statistical significance of their cross-correlation. A sieve bootstrap approach is used to generate multiple copies of the time series with the same autoregressive dependence, under the null hypothesis of the two time series under investigation being uncorrelated. The significance of cross-correlation coefficients is assessed based on the distribution of their bootstrapped counterparts. Both Pearson and Spearman types of coefficients are obtained, but a plot is provided for only one type, with significant correlations shown using filled circles (see Examples).

Usage

```
ccf_boot(
  x,
  y,
  lag.max = NULL,
  plot = c("Pearson", "Spearman", "none"),
```

```
  level = 0.95,
  B = 1000,
  smooth = FALSE,
  cl = 1L,
  ...
)
```

Arguments

x, y	univariate numeric time-series objects or numeric vectors for which to compute cross-correlation. Different time attributes in `ts` objects are acknowledged, see Example 2 below.
lag.max	maximum lag at which to calculate the cross-correlation. Will be automatically limited as in `ccf`.
plot	choose whether to plot results for Pearson correlation (default, or use `plot = "Pearson"`), Spearman correlation (use `plot = "Spearman"`), or suppress plotting (use `plot = "none"`). Both Pearson's and Spearman's results are given in the output, regardless of the `plot` setting.
level	confidence level, from 0 to 1. Default is 0.95, that is, 95% confidence.
B	number of bootstrap simulations to obtain empirical critical values. Default is 1000.
smooth	logical value indicating whether the bootstrap confidence bands should be smoothed across lags. Default is `FALSE` meaning no smoothing.
cl	parameter to specify computer cluster for bootstrapping passed to the package `parallel` (default `cl = 1`, means no cluster is used). Possible values are: • cluster object (list) produced by makeCluster. In this case, a new cluster is not started nor stopped; • `NULL`. In this case, the function will detect available cores (see detectCores) and, if there are multiple cores (> 1), a cluster will be started with makeCluster. If started, the cluster will be stopped after the computations are finished; • positive integer defining the number of cores to start a cluster. If `cl = 1` (default), no attempt to create a cluster will be made. If `cl > 1`, a cluster will be started (using makeCluster) and stopped afterward (using stopCluster).
...	other parameters passed to the function `ARest` to control how autoregressive dependencies are estimated. The same set of parameters is used separately on `x` and `y`.

Details

Note that the smoothing of confidence bands is implemented purely for the look. This smoothing is different from the smoothing methods that can be applied to adjust bootstrap performance (De Angelis and Young 1992). For correlations close to the significance bounds, the setting of `smooth` might affect the decision on the statistical significance. In this case, it is recommended to keep `smooth = FALSE` and set a higher `B`.

Value

A data frame with the following columns:

`Lag`	lags for which the following values were obtained.
`r_P`	observed Pearson correlations.
`lower_P`, `upper_P`	lower and upper confidence bounds (for the confidence level set by `level`) for Pearson correlations.
`r_S`	observed Spearman correlations.
`lower_S`, `upper_S`	lower and upper confidence bounds (for the confidence level set by `level`) for Spearman correlations.

Author(s)

Vyacheslav Lyubchich

See Also

`ARest`, `ar`, `ccf`, `HVK`

Examples

```
## Not run:
# Fix seed for reproducible simulations:
set.seed(1)

# Example 1
# Simulate independent normal time series of same lengths
x <- rnorm(100)
y <- rnorm(100)
# Default CCF with parametric confidence band
ccf(x, y)
# CCF with bootstrap
tmp <- ccf_boot(x, y)
# One can extract results for both Pearson and Spearman correlations
tmp$rP
tmp$rS

# Example 2
# Simulated ts objects of different lengths and starts (incomplete overlap)
x <- arima.sim(list(order = c(1, 0, 0), ar = 0.5), n = 30)
x <- ts(x, start = 2001)
y <- arima.sim(list(order = c(2, 0, 0), ar = c(0.5, 0.2)), n = 40)
y <- ts(y, start = 2020)
# Show how x and y are aligned
ts.plot(x, y, col = 1:2, lty = 1:2)
# The usual CCF
ccf(x, y)
# CCF with bootstrap confidence intervals
```

```
ccf_boot(x, y, plot = "Spearman")
# Notice that only +-7 lags can be calculated in both cases because of the small
# overlap of the time series. If we save these time series as plain vectors, the time
# information would be lost, and the time series will be misaligned.
ccf(as.numeric(x), as.numeric(y))

# Example 3
# Box & Jenkins time series of sales and a leading indicator, see ?BJsales
plot.ts(cbind(BJsales.lead, BJsales))
# Each of the BJ time series looks as having a stochastic linear trend, so apply differences
plot.ts(cbind(diff(BJsales.lead), diff(BJsales)))
# Get cross-correlation of the differenced series
ccf_boot(diff(BJsales.lead), diff(BJsales), plot = "Spearman")
# The leading indicator "stands out" with significant correlations at negative lags,
# showing it can be used to predict the sales 2-3 time steps ahead (that is,
# diff(BJsales.lead) at times t-2 and t-3 is strongly correlated with diff(BJsales) at
# current time t).

## End(Not run)
```

`CSlideCluster` *Slide-Level Time Series Clustering*

Description

Cluster time series at a slide level, based on Algorithm 1 of Ciampi et al. (2010).

Usage

```
CSlideCluster(X, Alpha = NULL, Beta = NULL, Delta = NULL, Theta = 0.8)
```

Arguments

`X`	a matrix of time series observed within a slide (time series in columns).
`Alpha`	lower limit of the time series domain. Default is `quantile(X)[2] - 1.5*(quantile(X)[4] - quantile(X)[2])`.
`Beta`	upper limit of the time series domain. Default is `quantile(X)[2] + 1.5*(quantile(X)[4] - quantile(X)[2])`.
`Delta`	closeness parameter, a real value in $[0, 1]$. Default is `0.1*(Beta - Alpha)`.
`Theta`	connectivity parameter, a real value in $[0, 1]$. Default is 0.8.

Value

A vector of length `ncol(X)` with cluster labels.

Author(s)

Vyacheslav Lyubchich

References

Ciampi A, Appice A, Malerba D (2010). "Discovering trend-based clusters in spatially distributed data streams." In *International Workshop of Mining Ubiquitous and Social Environments*, 107–122.

See Also

CSlideCluster, CWindowCluster, and BICC

Examples

```
set.seed(123)
X <- matrix(rnorm(50), 10, 5)
CSlideCluster(X)
```

---

cumsumCPA_test *Change Point Detection in Time Series via a Linear Regression with Temporally Correlated Errors*

Description

The function tests for a change point in parameters of a linear regression model with errors exhibiting a general weakly dependent structure. The approach extends earlier methods based on cumulative sums derived under the assumption of independent errors. The approach applies smoothing when the time series is dominated by high frequencies. To detect multiple changes, it is recommended to employ a binary or wild segmentation (Gombay 2010).

Usage

```
cumsumCPA_test(
  y,
  a.order,
  crit.type = c("asymptotic", "bootstrap"),
  bootstrap.method = c("nonparametric", "parametric"),
  num.bootstrap = 1000
)
```

Arguments

y	a numeric time series vector. Missing values are not allowed.
a.order	order of the autoregressive model which must be a non-negative integer number.
crit.type	a string parameter allowing to choose "asymptotic" or "bootstrap" options.
bootstrap.method	a string parameter allowing to choose "nonparametric" or "parametric" method of bootstrapping. "nonparametric" – resampling of the estimated residuals (with replacement); "parametric" – sampling innovations from a normal distribution.
num.bootstrap	number of bootstrap replications if crit.type = "bootstrap". The default number is 1000.

Value

A list with the following components:

index	time point where the change has occurred.
stat	test statistic.
p.value	p-value of the change point test.

Author(s)

Palina Niamkova, Dorcas Ofori-Boateng, Yulia R. Gel

References

Gombay E (2010). "Change detection in linear regression with time series errors." *Canadian Journal of Statistics*, **38**(1), 65–79.

See Also

mcusum.test for change point test for regression

Examples

```
## Not run:
#Example 1:

#Simulate some time series:
series_1 = rnorm(157, 2, 1)
series_2 = rnorm(43, 7, 10)
main_val = c(series_1, series_2)

#Now perform a change point detection:
cumsumCPA_test(series_1, 1) # no change
cumsumCPA_test(main_val, 1) # one change, asymptotic critical region
cumsumCPA_test(main_val, 1, "bootstrap", "parametric") # one change, parametric bootstrap
cumsumCPA_test(main_val, 1, "bootstrap", "nonparametric") # one change, nonparametric
#bootstrap

#Example 2:

#Consider time series with ratio of real GDP per family to the median income. This is a
#skewness and income inequality measure for the US families from 1947 till 2012.
e.data = (Ecdat::incomeInequality['mean.median'])
incomeInequality.ts = ts(e.data, start = 1947, end = 2012, frequency = 1)

#Now perform a change point detection:
cumsumCPA_test(incomeInequality.ts, 0)
cumsumCPA_test(incomeInequality.ts, 0, "bootstrap", "parametric")
cumsumCPA_test(incomeInequality.ts, 0, "bootstrap", "nonparametric")
incomeInequality.ts[13] # median income
Ecdat::incomeInequality$Year[13] + 1 # year of change point
```

```
#The first change point occurs at the 13th time point, that is 1960, where the ratio of real
#GDP per family to the median income is 1.940126. This ratio shows that in 1960 the national
#wealth was not distributed equally between all the population and that most people earn
#almost twice less than the equal share of the all produced goods and services by the nation.

#Note: To look for the other possible change points, run the same function for the
#segment of time series after value 13.

## End(Not run)
```

`CWindowCluster` *Window-Level Time Series Clustering*

Description

Cluster time series at a window level, based on Algorithm 2 of Ciampi et al. (2010).

Usage

```
CWindowCluster(
  X,
  Alpha = NULL,
  Beta = NULL,
  Delta = NULL,
  Theta = 0.8,
  p,
  w,
  s,
  Epsilon = 1
)
```

Arguments

`X`	a matrix of time series observed within a slide (time series in columns).
`Alpha`	lower limit of the time-series domain, passed to `CSlideCluster`.
`Beta`	upper limit of the time-series domain passed to `CSlideCluster`.
`Delta`	closeness parameter passed to `CSlideCluster`.
`Theta`	connectivity parameter passed to `CSlideCluster`.
`p`	number of layers (time-series observations) in each slide.
`w`	number of slides in each window.
`s`	step to shift a window, calculated in the number of slides. The recommended values are 1 (overlapping windows) or equal to `w` (non-overlapping windows).
`Epsilon`	a real value in $[0, 1]$ used to identify each pair of time series that are clustered together over at least `w*Epsilon` slides within a window; see Definition 7 by Ciampi et al. (2010). Default is 1.

Details

This is the upper-level function for time series clustering. It exploits the function CSlideCluster to cluster time series within each slide based on closeness and homogeneity measures. Then, it uses slide-level cluster assignments to cluster time series within each window.

The total length of time series (number of levels, i.e., nrow(X)) should be divisible by p.

Value

A vector (if X contains only one window) or matrix with cluster labels for each time series (columns) and window (rows).

Author(s)

Vyacheslav Lyubchich

References

Ciampi A, Appice A, Malerba D (2010). "Discovering trend-based clusters in spatially distributed data streams." In *International Workshop of Mining Ubiquitous and Social Environments*, 107–122.

See Also

CSlideCluster, CWindowCluster, and BICC

Examples

```
#For example, weekly data come in slides of 4 weeks
p <- 4 #number of layers in each slide (data come in a slide)

#We want to analyze the trend clusters within a window of 1 year
w <- 13 #number of slides in each window
s <- w  #step to shift a window

#Simulate 26 autoregressive time series with two years of weekly data (52*2 weeks),
#with a 'burn-in' period of 300.
N <- 26
T <- 2*p*w

set.seed(123)
phi <- c(0.5) #parameter of autoregression
X <- sapply(1:N, function(x) arima.sim(n = T + 300,
     list(order = c(length(phi), 0, 0), ar = phi)))[301:(T + 300),]
colnames(X) <- paste("TS", c(1:dim(X)[2]), sep = "")

tmp <- CWindowCluster(X, Delta = NULL, Theta = 0.8, p = p, w = w, s = s, Epsilon = 1)

#Time series were simulated with the same parameters, but based on the clustering parameters,
#not all time series join the same cluster. We can plot the main cluster for each window, and
#time series out of the cluster:
par(mfrow = c(2, 2))
ts.plot(X[c(1:(p*w)), tmp[1,] == 1], ylim = c(-4, 4),
```

```
        main = "Time series cluster 1 in window 1")
ts.plot(X[c(1:(p*w)), tmp[1,] != 1], ylim = c(-4, 4),
        main = "The rest of the time series in window 1")
ts.plot(X[c(1:(p*w)) + s*p, tmp[2,] == 1], ylim = c(-4, 4),
        main = "Time series cluster 1 in window 2")
ts.plot(X[c(1:(p*w)) + s*p, tmp[2,] != 1], ylim = c(-4, 4),
        main = "The rest of the time series in window 2")
```

DR *Downhill Riding (DR) Procedure*

Description

Downhill riding procedure for selecting optimal tuning parameters in clustering algorithms, using an (in)stability probe.

Usage

```
DR(X, method, minPts = 3, theta = 0.9, B = 500, lb = -30, ub = 10)
```

Arguments

X	an $n \times k$ matrix where columns are k objects to be clustered, and each object contains n observations (objects could be a set of time series).
method	the clustering method to be used – currently either "TRUST" (Ciampi et al. 2010) or "DBSCAN" (Ester et al. 1996). If the method is DBSCAN, then set MinPts and optimal ϵ is selected using DR. If the method is TRUST, then set theta, and optimal δ is selected using DR.
minPts	the minimum number of samples in an ϵ-neighborhood of a point to be considered as a core point. The minPts is to be used only with the DBSCAN method. The default value is 3.
theta	connectivity parameter $\theta \in (0, 1)$, which is to be used only with the TRUST method. The default value is 0.9.
B	number of random splits in calculating the Average Cluster Deviation (ACD). The default value is 500.
lb, ub	endpoints for a range of search for the optimal parameter.

Details

Parameters lb,ub are endpoints for the search for the optimal parameter. The parameter candidates are calculated in a way such that $P := 1.1^x, x \in lb, lb + 0.5, lb + 1.0, ..., ub$. Although the default range of search is sufficiently wide, in some cases lb,ub can be further extended if a warning message is given.

For more discussion on properties of the considered clustering algorithms and the DR procedure see Huang et al. (2016) and Huang et al. (2018).

Value

A list containing the following components:

P_opt	the value of the optimal parameter. If the method is DBSCAN, then P_opt is optimal ϵ. If the method is TRUST, then P_opt is optimal δ.
ACD_matrix	a matrix that returns ACD for different values of a tuning parameter. If the method is DBSCAN, then the tuning parameter is ϵ. If the method is TRUST, then the tuning parameter is δ.

Author(s)

Xin Huang, Yulia R. Gel

References

Ciampi A, Appice A, Malerba D (2010). "Discovering trend-based clusters in spatially distributed data streams." In *International Workshop of Mining Ubiquitous and Social Environments*, 107–122.

Ester M, Kriegel H, Sander J, Xu X (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise." In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, volume 96(34), 226–231.

Huang X, Iliev IR, Brenning A, Gel YR (2016). "Space-time clustering with stability probe while riding downhill." In *Proceedings of the 2nd SIGKDD Workshop on Mining and Learning from Time Series (MiLeTS)*.

Huang X, Iliev IR, Lyubchich V, Gel YR (2018). "Riding down the bay: space-time clustering of ecological trends." *Environmetrics*, **29**(5–6), e2455. doi:10.1002/env.2455.

See Also

BICC, dbscan

Examples

```
## Not run:
## example 1
## use iris data to test DR procedure

data(iris)
require(clue)  # calculate NMI to compare the clustering result with the ground truth
require(scatterplot3d)

Data <- scale(iris[,-5])
ground_truth_label <- iris[,5]

# perform DR procedure to select optimal eps for DBSCAN
# and save it in variable eps_opt
eps_opt <- DR(t(Data), method="DBSCAN", minPts = 5)$P_opt
```

```
# apply DBSCAN with the optimal eps on iris data
# and save the clustering result in variable res
res <- dbscan(Data, eps = eps_opt, minPts =5)$cluster

# calculate NMI to compare the clustering result with the ground truth label
clue::cl_agreement(as.cl_partition(ground_truth_label),
                   as.cl_partition(as.numeric(res)), method = "NMI")
# visualize the clustering result and compare it with the ground truth result
# 3D visualization of clustering result using variables Sepal.Width, Sepal.Length,
# and Petal.Length
scatterplot3d(Data[,-4],color = res)
# 3D visualization of ground truth result using variables Sepal.Width, Sepal.Length,
# and Petal.Length
scatterplot3d(Data[,-4],color = as.numeric(ground_truth_label))


## example 2
## use synthetic time series data to test DR procedure

require(funtimes)
require(clue)
require(zoo)

# simulate 16 time series for 4 clusters, each cluster contains 4 time series
set.seed(114)
samp_Ind <- sample(12,replace=F)
time_points <- 30
X <- matrix(0,nrow=time_points,ncol = 12)
cluster1 <- sapply(1:4,function(x) arima.sim(list(order = c(1, 0, 0), ar = c(0.2)),
                                             n = time_points, mean = 0, sd = 1))
cluster2 <- sapply(1:4,function(x) arima.sim(list(order = c(2 ,0, 0), ar = c(0.1, -0.2)),
                                             n = time_points, mean = 2, sd = 1))
cluster3 <- sapply(1:4,function(x) arima.sim(list(order = c(1, 0, 1), ar = c(0.3), ma = c(0.1)),
                                             n = time_points, mean = 6, sd = 1))

X[,samp_Ind[1:4]] <- t(round(cluster1, 4))
X[,samp_Ind[5:8]] <- t(round(cluster2, 4))
X[,samp_Ind[9:12]] <- t(round(cluster3, 4))


# create ground truth label of the synthetic data
ground_truth_label = matrix(1, nrow = 12, ncol = 1)
for(k in 1:3){
    ground_truth_label[samp_Ind[(4*k - 4 + 1):(4*k)]] = k
}

# perform DR procedure to select optimal delta for TRUST
# and save it in variable delta_opt
delta_opt <- DR(X, method = "TRUST")$P_opt

# apply TRUST with the optimal delta on the synthetic data
# and save the clustering result in variable res
res <- CSlideCluster(X, Delta = delta_opt, Theta = 0.9)
```

```
# calculate NMI to compare the clustering result with the ground truth label
clue::cl_agreement(as.cl_partition(as.numeric(ground_truth_label)),
                   as.cl_partition(as.numeric(res)), method = "NMI")

# visualize the clustering result and compare it with the ground truth result
# visualization of the clustering result obtained by TRUST
plot.zoo(X, type = "l", plot.type = "single", col = res, xlab = "Time index", ylab = "")
# visualization of the ground truth result
plot.zoo(X, type = "l", plot.type = "single", col = ground_truth_label,
         xlab = "Time index", ylab = "")

## End(Not run)
```

GombayCPA_test *Change Point Detection in Autoregressive Time Series*

Description

The function detects change points in autoregressive (AR) models for time series. Changes can be detected in any of p + 2 (mean, var, phi) autoregressive parameters where p is the order of the AR model. The test statistic is based on the efficient score vector (Gombay 2008).

Usage

```
GombayCPA_test(
  y,
  a.order,
  alternatives = c("two-sided", "greater", "lesser", "temporary"),
  crit.type = c("asymptotic", "bootstrap"),
  num.bootstrap = 1000
)
```

Arguments

y	a vector that contains univariate time-series observations. Missing values are not allowed.
a.order	order of the autoregressive model which must be a non-negative integer number.
alternatives	a string parameter that specifies a type of the test (i.e., "two-sided", "greater", "lesser", and "temporary"). The option "temporary" examines the temporary change in one of the parameters (Gombay 2008).
crit.type	method of obtaining critical values: "asymptotic" (default) or "bootstrap".
num.bootstrap	number of bootstrap replications if crit.type = "bootstrap". The default number is 1000.

Details

The function tests for a temporary change and a change in specific model parameters. Critical values can be estimated via asymptotic distribution `"asymptotic"` (i.e., the default option) or sieve bootstrap `"bootstrap"`. The function employs internal function `change.point` and sieve bootstrap `change.point.sieve` function.

Value

A list with the following components:

`index`	points of change for each parameter. The value of the `"alternatives"` determines the return: "temporary" – returns max, min, and abs.max points; "greater" – returns max points; "lesser" – returns min points; "two-sided" – returns abs.max.
`stats`	test statistic values for change points in mean, var, phi.
`p.values`	p-value of the change point test.

Author(s)

Palina Niamkova, Dorcas Ofori-Boateng, Yulia R. Gel

References

Gombay E (2008). "Change detection in autoregressive time series." *Journal of Multivariate Analysis*, **99**(3), 451–464. doi:10.1016/j.jmva.2007.01.003.

See Also

`mcusum.test` change point test for regression and `terrorism` dataset used in the Example 2

Examples

```
## Not run:
#Example 1:

#Simulate some time series:
series_1 = arima.sim(n = 100, list(order = c(2,0,0), ar = c(-0.7, -0.1)))
series_2 = arima.sim(n = 200, list(order = c(2,0,0), ar = c(0.1, -0.6)))
main_series = c(series_1, series_2)

result11 = GombayCPA_test(series_1, 2, "two-sided")
result11 #== No change point ===#

result12 = GombayCPA_test(main_series, 2, "two-sided")
result12  #=== One change at phi values ===#

result13 = GombayCPA_test(main_series, 2, "two-sided", "bootstrap")
result13  #=== One change at phi values ===#
```

```
#Example 2:

#From the package 'Ecdat' consider a time series with annual world number of victims of
#terrorism in the US from 1970 till 2016:
c.data = Ecdat::terrorism['nkill.us']
nkill.us.ts <- ts(c.data, start = 1970, end = 2016)

#Now perform a change point detection with one sided tests:
GombayCPA_test(nkill.us.ts, 0, "lesser")
GombayCPA_test(nkill.us.ts, 0, "greater")
nkill.us.ts[32]
year = 1970 + 31
print(year)
plot(nkill.us.ts)

#In both cases we find that the change point is located at the position 31 or 32. We can
# examine it further by checking the value of this position (using: nkill.us.ts[32]) as well as
# by plotting the graph (using: plot(nkill.us.ts)). The detected change point corresponds to
#the year of 2001, when the 9/11 attack happened.

## End(Not run)
```

HVK *HVK Estimator*

Description

Estimate coefficients in nonparametric autoregression using the difference-based approach by Hall and Van Keilegom (2003).

Usage

```
HVK(X, m1 = NULL, m2 = NULL, ar.order = 1)
```

Arguments

X	univariate time series. Missing values are not allowed.
m1, m2	subsidiary smoothing parameters. Default m1 = round(length(X)^(0.1)), m2 = round(length(X)^(0.5)).
ar.order	order of the nonparametric autoregression (specified by user).

Details

First, autocovariances are estimated using formula (2.6) by Hall and Van Keilegom (2003):

$$\hat{\gamma}(0) = \frac{1}{m_2 - m_1 + 1} \sum_{m=m_1}^{m_2} \frac{1}{2(n-m)} \sum_{i=m+1}^{n} \{(D_m X)_i\}^2,$$

$$\hat{\gamma}(j) = \hat{\gamma}(0) - \frac{1}{2(n-j)} \sum_{i=j+1}^{n} \{(D_j X)_i\}^2,$$

where n = `length(X)` is sample size, D_j is a difference operator such that $(D_j X)_i = X_i - X_{i-j}$. Then, Yule–Walker method is used to derive autoregression coefficients.

Value

Vector of length `ar.order` with estimated autoregression coefficients.

Author(s)

Yulia R. Gel, Vyacheslav Lyubchich, Xingyu Wang

References

Hall P, Van Keilegom I (2003). "Using difference-based methods for inference in nonparametric regression with time series errors." *Journal of the Royal Statistical Society, Series B (Statistical Methodology)*, **65**(2), 443–456. doi:10.1111/14679868.00395.

See Also

`ar`, `ARest`

Examples

```
X <- arima.sim(n = 300, list(order = c(1, 0, 0), ar = c(0.6)))
HVK(as.vector(X), ar.order = 1)
```

---

mcusum_test *Change Point Test for Regression*

Description

Apply change point test by Horvath et al. (2017) for detecting at-most-m changes in regression coefficients, where test statistic is a modified cumulative sum (CUSUM), and critical values are obtained with sieve bootstrap (Lyubchich et al. 2020).

Usage

```
mcusum_test(
  e,
  k,
  m = length(k),
  B = 1000,
  shortboot = FALSE,
  ksm = FALSE,
```

```
  ksm.arg = list(kernel = "gaussian", bw = "sj"),
  ...
)
```

Arguments

e	vector of regression residuals (a stationary time series).
k	an integer vector or scalar with hypothesized change point location(s) to test.
m	an integer specifying the maximum number of change points being confirmed as statistically significant (from those specified in k) would be $\leq m$. Thus, m must be in 1,...,k.
B	number of bootstrap simulations to obtain empirical critical values. Default is 1000.
shortboot	if TRUE, then a heuristic is used to perform the test with a reduced number of bootstrap replicates. Specifically, B/4 replicates are used, which may reduce computing time by up to 75% when the number of retained null hypotheses is large. A p-value of 999 is reported whenever a null hypothesis is retained as a result of this mechanism.
ksm	logical value indicating whether a kernel smoothing to innovations in sieve bootstrap shall be applied (default is FALSE, that is, the original estimated innovations are bootstrapped, without the smoothing).
ksm.arg	used only if ksm = TRUE. A list of arguments for kernel smoothing to be passed to density function. Default settings specify the use of the Gaussian kernel and the "sj" rule to choose the bandwidth.
...	additional arguments passed to ARest (for example, ar.method).

Details

The sieve bootstrap is applied by approximating regression residuals e with an AR(p) model using function ARest, where the autoregressive coefficients are estimated with ar.method, and order p is selected based on ar.order and BIC settings (see ARest). At the next step, B autoregressive processes are simulated under the null hypothesis of no change points. The distribution of test statistics M_T computed on each of those bootstrapped series is used to obtain bootstrap-based p-values for the test (Lyubchich et al. 2020).

In the current implementation, the bootstrapped p-value is calculated using equation 4.10 of Davison and Hinkley (1997): p.value = (1 + n) / (B + 1), where n is number of bootstrapped statistics greater or equal to the observed statistic.

The test statistic corresponds to the maximal value of the modified CUSUM over up to m combinations of hypothesized change points specified in k. The change points that correspond to that maximum are reported in estimate$khat, and their number is reported as the parameter.

Value

A list of class "htest" containing the following components:

method	name of the method.

`data.name`	name of the data.
`statistic`	obseved value of the test statistic.
`parameter`	`mhat` is the final number of change points, from those specified in the input `k`, for which the test statistic is reported. See the corresponding locations, `khat`, in the `estimate`.
`p.value`	bootstrapped p-value of the test.
`alternative`	alternative hypothesis.
`estimate`	list with elements: `AR_order` and `AR_coefficients` (the autoregressive order and estimated autoregressive coefficients used in sieve bootstrap procedure), `khat` (final change points, from those specified in the input `k` for which the test statistic is reported), and `B` (the number of bootstrap replications).

Author(s)

Vyacheslav Lyubchich

References

Davison AC, Hinkley DV (1997). *Bootstrap Methods and Their Application*. Cambridge University Press, Cambridge.

Horvath L, Pouliot W, Wang S (2017). "Detecting at-most-m changes in linear regression models." *Journal of Time Series Analysis*, **38**, 552–590. doi:10.1111/jtsa.12228.

Lyubchich V, Lebedeva TV, Testa JM (2020). "A data-driven approach to detecting change points in linear regression models." *Environmetrics*, **31**(1), e2591. doi:10.1002/env.2591.

Examples

```
##### Model 1 with normal errors, by Horvath et al. (2017)
T <- 100 #length of time series
X <- rnorm(T, mean = 1, sd = 1)
E <- rnorm(T, mean = 0, sd = 1)
SizeOfChange <- 1
TimeOfChange <- 50
Y <- c(1 * X[1:TimeOfChange] + E[1:TimeOfChange],
      (1 + SizeOfChange)*X[(TimeOfChange + 1):T] + E[(TimeOfChange + 1):T])
ehat <- lm(Y ~ X)$resid
mcusum_test(ehat, k = c(30, 50, 70))

#Same, but with bootstrapped innovations obtained from a kernel smoothed distribution:
mcusum_test(ehat, k = c(30, 50, 70), ksm = TRUE)
```

notrend_test *Sieve Bootstrap Based Test for the Null Hypothesis of no Trend*

Description

A combination of time series trend tests for testing the null hypothesis of no trend, versus the alternative hypothesis of a linear trend (Student's t-test), or monotonic trend (Mann–Kendall test), or more general, possibly non-monotonic trend (WAVK test).

Usage

```
notrend_test(
  x,
  B = 1000,
  test = c("t", "MK", "WAVK"),
  ar.method = "HVK",
  ar.order = NULL,
  ic = "BIC",
  factor.length = c("user.defined", "adaptive.selection"),
  Window = NULL,
  q = 3/4,
  j = c(8:11)
)
```

Arguments

x	a vector containing a univariate time series. Missing values are not allowed.
B	number of bootstrap simulations to obtain empirical critical values. Default is 1000.
test	trend test to implement: Student's t-test ("t", default), Mann–Kendall test ("MK"), or WAVK test ("WAVK", see WAVK).
ar.method	method of estimating autoregression coefficients. Default "HVK" delivers robust difference-based estimates by Hall and Van Keilegom (2003). Alternatively, options of ar function can be used, such as "burg", "ols", "mle", and "yw".
ar.order	order of the autoregressive model when ic = "none", or the maximal order for IC-based filtering. Default is round(10*log10(length(x))), where x is the time series.
ic	information criterion used to select the order of autoregressive filter (AIC of BIC), considering models of orders p = 0,1,...,ar.order. If ic = "none", the AR(p) model with p = ar.order is used, without order selection.
factor.length	method to define the length of local windows (factors). Used only if test = "WAVK". Default option "user.defined" allows to set only one value of the argument Window. The option "adaptive.selection" sets method = "boot" and employs heuristic m-out-of-n subsampling algorithm (Bickel and Sakov 2008) to select an optimal window from the set of possible windows length(x)*q^j

whose values are mapped to the largest previous integer and greater than 2. Vector x is the time series tested.

`Window` length of the local window (factor), default is `round(0.1*length(x))`. Used only if `test = "WAVK"`. This argument is ignored if `factor.length = "adaptive.selection"`.

`q` scalar from 0 to 1 to define the set of possible windows when `factor.length = "adaptive.selection"`. Used only if `test = "WAVK"`. Default is $3/4$. This argument is ignored if `factor.length = "user.defined"`.

`j` numeric vector to define the set of possible windows when `factor.length = "adaptive.selection"`. Used only if `test = "WAVK"`. Default is `c(8:11)`. This argument is ignored if `factor.length = "user.defined"`.

Details

This function tests the null hypothesis of no trend versus different alternatives. To set some other shape of trend as the null hypothesis, use `wavk_test`. Note that `wavk_test` employs hybrid bootstrap, which is an alternative to the sieve bootstrap employed by the current function.

Value

A list with class `"htest"` containing the following components:

`method` name of the method.

`data.name` name of the data.

`statistic` value of the test statistic.

`p.value` p-value of the test.

`alternative` alternative hypothesis.

`estimate` list with the following elements: employed AR order and estimated AR coefficients.

`parameter` window that was used in WAVK test, included in the output only if `test = "WAVK"`.

Author(s)

Vyacheslav Lyubchich

References

Bickel PJ, Sakov A (2008). "On the choice of m in the m out of n bootstrap and confidence bounds for extrema." *Statistica Sinica*, **18**(3), 967–985.

Hall P, Van Keilegom I (2003). "Using difference-based methods for inference in nonparametric regression with time series errors." *Journal of the Royal Statistical Society, Series B (Statistical Methodology)*, **65**(2), 443–456. doi:10.1111/14679868.00395.

See Also

`ar`, `HVK`, `WAVK`, `wavk_test`, `vignette("trendtests", package = "funtimes")`

Examples

```
## Not run:
# Fix seed for reproducible simulations:
set.seed(1)

#Simulate autoregressive time series of length n with smooth linear trend:
n <- 200
tsTrend <- 1 + 2*(1:n/n)
tsNoise <- arima.sim(n = n, list(order = c(2, 0, 0), ar = c(0.5, -0.1)))
U <- tsTrend + tsNoise
plot.ts(U)

#Use t-test
notrend_test(U)

#Use Mann--Kendall test and Yule-Walker estimates of the AR parameters
notrend_test(U, test = "MK", ar.method = "yw")

#Use WAVK test for the H0 of no trend, with m-out-of-n selection of the local window:
notrend_test(U, test = "WAVK", factor.length = "adaptive.selection")
# Sample output:
## Sieve-bootstrap WAVK trend test
##
##data:  U
##WAVK test statistic = 21.654, moving window = 15, p-value < 2.2e-16
##alternative hypothesis: (non-)monotonic trend.
##sample estimates:
##$AR_order
##[1] 1
##
##$AR_coefficients
##    phi_1
##0.4041848

## End(Not run)
```

`purity` *Clustering Purity*

Description

Calculate the purity of the clustering results. For example, see Schaeffer et al. (2016).

Usage

```
purity(classes, clusters)
```

Arguments

`classes`	a vector with labels of true classes.
`clusters`	a vector with labels of assigned clusters for which purity is to be tested. Should be of the same length as `classes`.

Details

Following Manning et al. (2008), each cluster is assigned to the class which is most frequent in the cluster, then

$$Purity(\Omega, C) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j|,$$

where $\Omega = \{\omega_1, \ldots, \omega_K\}$ is the set of identified clusters and $C = \{c_1, \ldots, c_J\}$ is the set of classes. That is, within each class $j = 1, \ldots, J$ find the size of the most populous cluster from the $K - j$ unassigned clusters. Then, sum together the $\min(K, J)$ sizes found and divide by N, where N = `length(classes)` = `length(clusters)`.

If $\max_j |\omega_k \cap c_j|$ is not unique for some j, it is assigned to the class which the second maximum is the smallest, to maximize the $Purity$ (see 'Examples').

The number of unique elements in `classes` and `clusters` may differ.

Value

A list with two elements:

`pur`	purity value.
`out`	table with $\min(K, J)$ = `min(length(unique(classes)),length(unique(clusters)))` rows and the following columns: `ClassLabels`, `ClusterLabels`, and `ClusterSize`.

Author(s)

Vyacheslav Lyubchich

References

Manning CD, Raghavan P, Schutze H (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York.

Schaeffer ED, Testa JM, Gel YR, Lyubchich V (2016). "On information criteria for dynamic spatio-temporal clustering." In Banerjee A, Ding W, Dy JG, Lyubchich V, Rhines A (eds.), *The 6th International Workshop on Climate Informatics: CI2016*, 5–8. doi:10.5065/D6K072N6.

Examples

```
# Fix seed for reproducible simulations:
# RNGkind(sample.kind = "Rounding") #run this line to have same seed across R versions > R 3.6.0
set.seed(1)

##### Example 1
#Create some classes and cluster labels:
```

```
classes <- rep(LETTERS[1:3], each = 5)
clusters <- sample(letters[1:5], length(classes), replace = TRUE)

#From the table below:
# - cluster 'b' corresponds to class A;
# - either of the clusters 'd' and 'e' can correspond to class B,
#   however, 'e' should be chosen, because cluster 'd' also highly
#   intersects with Class C. Thus,
# - cluster 'd' corresponds to class C.
table(classes, clusters)
##       clusters
##classes a b c d e
##      A 0 3 1 0 1
##      B 1 0 0 2 2
##      C 1 2 0 2 0

#The function does this choice automatically:
purity(classes, clusters)

#Sample output:
##$pur
##[1] 0.4666667
##
##$out
##  ClassLabels ClusterLabels ClusterSize
##1           A             b           3
##2           B             e           2
##3           C             d           2


##### Example 2
#The labels can be also numeric:
classes <- rep(1:5, each = 3)
clusters <- sample(1:3, length(classes), replace = TRUE)
purity(classes, clusters)
```

---

sync_cluster *Time Series Clustering based on Trend Synchronism*

Description

Cluster time series with a common parametric trend using the sync_test function (Lyubchich and Gel 2016; Ghahari et al. 2017).

Usage

```
sync_cluster(formula, rate = 1, alpha = 0.05, ...)
```

Arguments

formula	an object of class "formula", specifying the type of common trend for clustering the time series in a T by N matrix of time series (time series in columns) which is passed to sync_test. Variable t should be used to specify the form of the trend, where t is specified within the function automatically as a regular sequence of length T on the interval (0,1]. See Examples.
rate	rate of removal of time series. Default is 1 (i.e., if the hypothesis of synchronism is rejected one time series is removed at a time to re-test the remaining time series). Integer values above 1 are treated as the number of time series to be removed. Values from 0 to 1 are treated as a fraction of the time series to be removed.
alpha	significance level for testing the hypothesis of a common trend (using sync_test) of the parametric form specified in the formula.
...	arguments to be passed to sync_test, for example, number of bootstrap replications (B).

Details

The sync_cluster function recursively clusters time series having a pre-specified common parametric trend until there is no time series left. Starting with the given N time series, the sync_test function is used to test for a common trend. If the null hypothesis of common trend is not rejected by sync_test, the time series are grouped (i.e., assigned to a cluster). Otherwise, the time series with the largest contribution to the test statistics are temporarily removed (the number of time series to remove depends on the rate of removal), and sync_test is applied again. The contribution to the test statistic is assessed by the WAVK test statistic calculated for each time series.

Value

A list with the elements:

cluster	an integer vector indicating the cluster to which each time series is allocated. A label '0' is assigned to time series which do not have a common trend with other time series (that is, all time series labeled with '0' are separate one-element clusters).
elements	a list with names of the time series in each cluster.

The further elements combine results of sync_test for each cluster with at least two elements (that is, single-element clusters labeled with '0' are excluded):

estimate	a list with common parametric trend estimates obtained by sync_test for each cluster. The length of this list is max(cluster).
pval	a list of p-values of sync_test for each cluster. The length of this list is max(cluster).
statistic	a list with values of sync_test test statistic for each cluster. The length of this list is max(cluster).
ar_order	a list of AR filter orders used in sync_test for each time series. The results are grouped by cluster in the list of length max(cluster).

`window_used` a list of local windows used in `sync_test` for each time series. The results are grouped by cluster in the list of length `max(cluster)`.

`all_considered_windows` a list of all windows considered in `sync_test` and corresponding test results, for each cluster. The length of this list is `max(cluster)`.

`WAVK_obs` a list of WAVK test statistics obtained in `sync_test` for each time series. The results are grouped by cluster in the list of length `max(cluster)`.

Author(s)

Srishti Vishwakarma, Vyacheslav Lyubchich

References

Ghahari A, Gel YR, Lyubchich V, Chun Y, Uribe D (2017). "On employing multi-resolution weather data in crop insurance." In *Proceedings of the SIAM International Conference on Data Mining (SDM17) Workshop on Mining Big Data in Climate and Environment (MBDCE 2017)*.

Lyubchich V, Gel YR (2016). "A local factor nonparametric test for trend synchronism in multiple time series." *Journal of Multivariate Analysis*, **150**, 91–104. doi:10.1016/j.jmva.2016.05.004.

See Also

`BICC`, `DR`, `sync_test`

Examples

```
## Not run:
## Simulate 4 autoregressive time series,
## 3 having a linear trend and 1 without a trend:
set.seed(123)
T = 100 #length of time series
N = 4 #number of time series
X = sapply(1:N, function(x) arima.sim(n = T,
          list(order = c(1, 0, 0), ar = c(0.6))))
X[,1] <- 5 * (1:T)/T + X[,1]
plot.ts(X)

# Finding clusters with common linear trends:
LinTrend <- sync_cluster(X ~ t)

## Sample Output:
##[1] "Cluster labels:"
##[1] 0 1 1 1
##[1] "Number of single-element clusters (labeled with '0'): 1"

## plotting the time series of the cluster obtained
for(i in 1:max(LinTrend$cluster)) {
    plot.ts(X[, LinTrend$cluster == i],
            main = paste("Cluster", i))
}
```

```
## Simulating 7 autoregressive time series,
## where first 4 time series have a linear trend added
set.seed(234)
T = 100 #length of time series
a <- sapply(1:4, function(x) -10 + 0.1 * (1:T) +
            arima.sim(n = T, list(order = c(1, 0, 0), ar = c(0.6))))
b <- sapply(1:3, function(x) arima.sim(n = T,
            list(order = c(1, 0, 0), ar = c(0.6))))
Y <- cbind(a, b)
plot.ts(Y)

## Clustering based on linear trend with rate of removal = 2
# and confidence level for the synchronism test 90%
LinTrend7 <- sync_cluster(Y ~ t, rate = 2, alpha = 0.1, B = 99)

## Sample output:
##[1] "Cluster labels:"
##[1] 1 1 1 0 2 0 2
##[1] "Number of single-element clusters (labeled with '0'): 2"

## End(Not run)
```

---

sync_test *Time Series Trend Synchronicity Test*

Description

Nonparametric test for synchronicity of parametric trends in multiple time series (Lyubchich and Gel 2016). The method tests whether N observed time series exhibit the same trend of some pre-specified smooth parametric form.

Usage

```
sync_test(
  formula,
  B = 1000,
  Window = NULL,
  q = NULL,
  j = NULL,
  ar.order = NULL,
  ar.method = "HVK",
  ic = "BIC"
)
```

Arguments

formula	an object of class "formula", specifying the form of the common parametric time trend to be tested in a T by N matrix of time series (time series in columns). Variable t should be used to specify the form of the trend, where t is specified within the function as a regular sequence on the interval (0,1]. See 'Examples'.
B	number of bootstrap simulations to obtain empirical critical values. Default is 1000.
Window	scalar or N-vector with lengths of the local windows (factors). If only one value is set, the same Window is applied to each time series. An N-vector gives a specific window for each time series. If Window is not specified, an automatic algorithm for optimal window selection is applied as a default option (see 'Details').
q	scalar from 0 to 1 to define the set of possible windows T*q^j and to automatically select an optimal window for each time series. Default is $3/4$. This argument is ignored if the Window is set by the user.
j	numeric vector to define the set of possible windows T*q^j and to automatically select an optimal window for each time series. Default is c(8:11). This argument is ignored if the Window is set by the user.
ar.order	order of the autoregressive filter when ic = "none", or the maximal order for IC-based filtering. Default is round(10*log10(T)). The ar.order can be a scalar or N-vector. If scalar, the same ar.order is applied to each time series. An N-vector specifies a separate ar.order for each time series.
ar.method	method of estimating autoregression coefficients. Default "HVK" delivers robust difference-based estimates by Hall and Van Keilegom (2003). Alternatively, options of ar function can be used, such as "burg", "ols", "mle", and "yw".
ic	information criterion used to select the order of autoregressive filter (AIC of BIC), considering models of orders p = 0,1,...,ar.order. If ic = "none", the AR(p) model with p = ar.order is used, without order selection.

Details

Arguments Window, j, and q are used to set windows for the local regression. Current version of the function assumes two options: (1) user specifies one fixed window for each time series using the argument Window (if Window is set, j and q are ignored), and (2) user specifies a set of windows by j and q to apply this set to each time series and to select an optimal window using a heuristic m-out-of-n subsampling algorithm (Bickel and Sakov 2008). The option of selecting windows automatically for some of the time series, while for other time series the window is fixed, is not available yet. If none of these three arguments is set, default j and q are used. Values T*q^j are mapped to the largest previous integer, then only those greater than 2 are used.

See more details in Lyubchich and Gel (2016) and Lyubchich (2016).

Value

A list of class "htest" containing the following components:

method	name of the method.

`data.name`	name of the data.
`statistic`	value of the test statistic.
`p.value`	p-value of the test.
`alternative`	alternative hypothesis.
`estimate`	list with elements `common_trend_estimates`, `ar_order_used`, `Window_used`, `wavk_obs`, and `all_considered_windows`. The latter is a table with bootstrap and asymptotic test results for all considered windows, that is, without adaptive selection of the local window.

Author(s)

Yulia R. Gel, Vyacheslav Lyubchich, Ethan Schaeffer, Xingyu Wang

References

Bickel PJ, Sakov A (2008). "On the choice of m in the m out of n bootstrap and confidence bounds for extrema." *Statistica Sinica*, **18**(3), 967–985.

Hall P, Van Keilegom I (2003). "Using difference-based methods for inference in nonparametric regression with time series errors." *Journal of the Royal Statistical Society, Series B (Statistical Methodology)*, **65**(2), 443–456. doi:10.1111/14679868.00395.

Lyubchich V (2016). "Detecting time series trends and their synchronization in climate data." *Intelligence. Innovations. Investments*, **12**, 132–137.

Lyubchich V, Gel YR (2016). "A local factor nonparametric test for trend synchronism in multiple time series." *Journal of Multivariate Analysis*, **150**, 91–104. doi:10.1016/j.jmva.2016.05.004.

See Also

`ar`, `HVK`, `WAVK`, `wavk_test`

Examples

```
#Fix seed for reproducible simulations:
set.seed(1)

# Simulate two autoregressive time series of length n without trend
#(i.e., with zero or constant trend)
# and arrange the series into a matrix:
n <- 200
y1 <- arima.sim(n = n, list(order = c(1, 0, 0), ar = c(0.6)))
y2 <- arima.sim(n = n, list(order = c(1, 0, 0), ar = c(-0.2)))
Y <- cbind(y1, y2)
plot.ts(Y)


#Test H0 of a common linear trend:
## Not run:
```

```
    sync_test(Y ~ t, B = 500)

## End(Not run)
# Sample output:
## Nonparametric test for synchronism of parametric trends
##
##data:  Y
##Test statistic = -0.0028999, p-value = 0.7
##alternative hypothesis: common trend is not of the form Y ~ t.
##sample estimates:
##$common_trend_estimates
##               Estimate Std. Error    t value  Pr(>|t|)
##(Intercept) -0.02472566  0.1014069 -0.2438261 0.8076179
##t            0.04920529  0.1749859  0.2811958 0.7788539
##
##$ar.order_used
##         y1 y2
##ar.order  1  1
##
##$Window_used
##       y1 y2
##Window 15  8
##
##$all_considered_windows
## Window    Statistic p-value Asympt. p-value
##      8 -0.000384583   0.728       0.9967082
##     11 -0.024994408   0.860       0.7886005
##     15 -0.047030164   0.976       0.6138976
##     20 -0.015078579   0.668       0.8714980
##
##$wavk_obs
##[1]  0.05827148 -0.06117136

# Add a time series y3 with a different linear trend and re-apply the test:
y3 <- 1 + 3*((1:n)/n) + arima.sim(n = n, list(order = c(1, 0, 0), ar = c(-0.2)))
Y2 <- cbind(Y, y3)
plot.ts(Y2)
## Not run:
    sync_test(Y2 ~ t, B = 500)
## End(Not run)
# Sample output:
## Nonparametric test for synchronism of parametric trends
##
##data:  Y2
##Test statistic = 0.48579, p-value < 2.2e-16
##alternative hypothesis: common trend is not of the form Y2 ~ t.
##sample estimates:
##$common_trend_estimates
##              Estimate Std. Error  t value     Pr(>|t|)
##(Intercept) -0.3632963 0.07932649 -4.57976 8.219360e-06
##t            0.7229777 0.13688429  5.28167 3.356552e-07
##
##$ar.order_used
```

```
##         Y.y1 Y.y2 y3
##ar.order    1    1  0
##
##$Window_used
##       Y.y1 Y.y2 y3
##Window    8   11  8
##
##$all_considered_windows
## Window Statistic p-value Asympt. p-value
##      8 0.4930069       0    1.207378e-05
##     11 0.5637067       0    5.620248e-07
##     15 0.6369703       0    1.566057e-08
##     20 0.7431621       0    4.201484e-11
##
##$wavk_obs
##[1]  0.08941797 -0.07985614  0.34672734

#Other hypothesized trend forms can be specified, for example:
## Not run:
    sync_test(Y ~ 1) #constant trend
    sync_test(Y ~ poly(t, 2)) #quadratic trend
    sync_test(Y ~ poly(t, 3)) #cubic trend

## End(Not run)
```

`tails_i` *Interval-Based Tails Comparison*

Description

Compare right tails of two sample distributions using an interval-based approach (IBA); see Chu et al. (2015) and Lyubchich and Gel (2017).

Usage

```
tails_i(x0, x1, d = NULL)
```

Arguments

`x0`, `x1`	vectors of the same length (preferably). Tail in `x1` is compared against the tail in `x0`.
`d`	a threshold defining the tail. The threshold is the same for both `x0` and `x1`. Default is `quantile(x0, probs = 0.99)`.

Details

Sturges' formula is used to calculate the number of intervals (k) for `x0` $\geq$ `d`, then interval width is derived. The tails, `x0` $\geq$ `d` and `x1` $\geq$ `d`, are divided into intervals. The number of `x1`-values within each interval is compared with the number of `x0`-values within the same interval (this difference is reported as `Nk`).

Value

A list with two elements:

`Nk`	vector that tells how many more `x1`-values compared with `x0`-values there are within each interval.
`Ck`	vector of the intervals' centers.

Author(s)

Calvin Chu, Yulia R. Gel, Vyacheslav Lyubchich

References

Chu C, Gel YR, Lyubchich V (2015). "Climate change from an insurance perspective: a case study of Norway." In Dy JG, Emile-Geay J, Lakshmanan V, Liu Y (eds.), *The 5th International Workshop on Climate Informatics: CI2015*.

Lyubchich V, Gel YR (2017). "Can we weather proof our insurance?" *Environmetrics*, **28**(2), e2433. doi:10.1002/env.2433.

See Also

`q.tails`

Examples

```
x0 <- rnorm(1000)
x1 <- rt(1000, 5)
tails_i(x0, x1)
```

---

tails_q *Quantile-Based Tails Comparison*

Description

Compare right tails of two sample distributions using a quantile-based approach (QBA); see Soliman et al. (2014), Soliman et al. (2015), and Lyubchich and Gel (2017).

Usage

```
tails_q(x0, x1, q = 0.99)
```

Arguments

x0, x1	vectors of the same length (preferably). Tail in x1 is compared against the tail in x0.
q	a quantile defining the right tail for both x0 and x1. Values above the thresholds quantile(x0, probs = q) and quantile(x1, probs = q) are considered as the respective right tails.

Details

Sturges' formula is used to calculate the number of intervals (k) to split the upper $100(1-q)\backslash$ (the right tails). Then, each tail is divided into equally-filled intervals with a quantile step $d = (1-q)/k$. Pk reports the difference between corresponding intervals' centers obtained from x0 and x1.

Value

A list with two elements:

d	the step in probabilities for defining the quantiles.
Pk	vector of differences of the intervals' centers.

Author(s)

Vyacheslav Lyubchich, Yulia R. Gel

References

Lyubchich V, Gel YR (2017). "Can we weather proof our insurance?" *Environmetrics*, **28**(2), e2433. doi:10.1002/env.2433.

Soliman M, Lyubchich V, Gel YR, Naser D, Esterby S (2015). "Evaluating the impact of climate change on dynamics of house insurance claims." In Lakshmanan V, Gilleland E, McGovern A, Tingley M (eds.), *Machine Learning and Data Mining Approaches to Climate Science*, chapter 16, 175–183. Springer, Switzerland. doi:10.1007/9783319172200_16.

Soliman M, Naser D, Lyubchich V, Gel YR, Esterby S (2014). "Evaluating the impact of climate change on dynamics of house insurance claims." In Ebert-Uphoff I (ed.), *The 4th International Workshop on Climate Informatics: CI2014*.

See Also

i.tails

Examples

```
x0 <- rnorm(1000)
x1 <- rt(1000, 5)
tails_q(x0, x1)
```

WAVK *WAVK Statistic*

Description

Statistic for testing the parametric form of a regression function, suggested by Wang et al. (2008).

Usage

```
WAVK(z, kn = NULL)
```

Arguments

z	filtered univariate time series (see formula (2.1) by Wang and Van Keilegom 2007): $$Z_i = \left(Y_{i+p} - \sum_{j=1}^{p} \hat{\phi}_{j,n} Y_{i+p-j} \right) - \left(f(\hat{\theta}, t_{i+p}) - \sum_{j=1}^{p} \hat{\phi}_{j,n} f(\hat{\theta}, t_{i+p-j}) \right),$$ where Y_i is observed time series of length n, $\hat{\theta}$ is an estimator of hypothesized parametric trend $f(\theta, t)$, and $\hat{\phi}_p = (\hat{\phi}_{1,n}, \ldots, \hat{\phi}_{p,n})'$ are estimated coefficients of an autoregressive filter of order p. Missing values are not allowed.
kn	length of the local window.

Value

A list with following components:

Tn	test statistic based on artificial ANOVA and defined by Wang and Van Keilegom (2007) as a difference of mean square for treatments (MST) and mean square for errors (MSE): $$T_n = MST - MSE = \frac{k_n}{n-1} \sum_{t=1}^{T} \left(\overline{V}_{t.} - \overline{V}_{..} \right)^2 - \frac{1}{n(k_n - 1)} \sum_{t=1}^{n} \sum_{j=1}^{k_n} \left(V_{tj} - \overline{V}_{t.} \right)^2,$$ where $\{V_{t1}, \ldots, V_{tk_n}\} = \{Z_j : j \in W_t\}$, W_t is a local window, $\overline{V}_{t.}$ and $\overline{V}_{..}$ are the mean of the tth group and the grand mean, respectively.
Tns	standardized version of Tn according to Theorem 3.1 by Wang and Van Keilegom (2007): $$T_{ns} = \left(\frac{n}{k_n} \right)^{\frac{1}{2}} T_n \Big/ \left(\frac{4}{3} \right)^{\frac{1}{2}} \sigma^2,$$ where n is the length and σ^2 is the variance of the time series. Robust difference-based Rice's estimator (Rice 1984) is used to estimate σ^2.
p.value	p-value for Tns based on its asymptotic $N(0, 1)$ distribution.

Author(s)

Yulia R. Gel, Vyacheslav Lyubchich

References

Rice J (1984). "Bandwidth choice for nonparametric regression." *The Annals of Statistics*, **12**(4), 1215–1230. doi:10.1214/aos/1176346788.

Wang L, Akritas MG, Van Keilegom I (2008). "An ANOVA-type nonparametric diagnostic test for heteroscedastic regression models." *Journal of Nonparametric Statistics*, **20**(5), 365–382.

Wang L, Van Keilegom I (2007). "Nonparametric test for the form of parametric regression with time series errors." *Statistica Sinica*, **17**, 369–386.

See Also

wavk_test

Examples

```
z <- rnorm(300)
WAVK(z, kn = 7)
```

---

wavk_test *WAVK Trend Test*

Description

Nonparametric test to detect (non-)monotonic parametric trends in time series (based on Lyubchich et al. 2013).

Usage

```
wavk_test(
  formula,
  factor.length = c("user.defined", "adaptive.selection"),
  Window = NULL,
  q = 3/4,
  j = c(8:11),
  B = 1000,
  method = c("boot", "asympt"),
  ar.order = NULL,
  ar.method = "HVK",
  ic = "BIC",
  out = FALSE
)
```

Arguments

formula	an object of class "formula", specifying the form of the parametric time trend to be tested. Variable t should be used to specify the form, where t is specified within the function as a regular sequence on the interval (0,1]. See Examples.
factor.length	method to define the length of local windows (factors). Default option "user.defined" allows setting only one value of the argument Window. The option "adaptive.selection" sets method = "boot" and employs heuristic m-out-of-n subsampling algorithm (Bickel and Sakov 2008) to select an optimal window from the set of possible windows length(x)*q^j whose values are mapped to the largest previous integer and greater than 2. Vector x is the time series tested.
Window	length of the local window (factor), default is round(0.1*length(x)), where x is the time series tested. This argument is ignored if factor.length = "adaptive.selection".
q	scalar from 0 to 1 to define the set of possible windows when factor.length = "adaptive.selection". Default is $3/4$. This argument is ignored if factor.length = "user.defined".
j	numeric vector to define the set of possible windows when factor.length = "adaptive.selection". Default is c(8:11). This argument is ignored if factor.length = "user.defined".
B	number of bootstrap simulations to obtain empirical critical values. Default is 1000.
method	method of obtaining critical values: from asymptotical ("asympt") or bootstrap ("boot") distribution. If factor.length = "adaptive.selection" the option "boot" is used.
ar.order	order of the autoregressive model when ic = "none", or the maximal order for IC-based filtering. Default is round(10*log10(length(x))), where x is the time series.
ar.method	method of estimating autoregression coefficients. Default "HVK" delivers robust difference-based estimates by Hall and Van Keilegom (2003). Alternatively, options of ar function can be used, such as "burg", "ols", "mle", and "yw".
ic	information criterion used to select the order of autoregressive filter (AIC of BIC), considering models of orders p = 0,1,...,ar.order. If ic = "none", the AR(p) model with p = ar.order is used, without order selection.
out	logical value indicates whether the full output should be shown. Default is FALSE.

Details

See more details in Lyubchich and Gel (2016) and Lyubchich (2016).

Value

A list with class "htest" containing the following components:

method	name of the method.

`data.name`	name of the data.
`statistic`	value of the test statistic.
`p.value`	p-value of the test.
`alternative`	alternative hypothesis.
`parameter`	window that was used.
`estimate`	list with the following elements: estimated trend coefficients; user-defined or IC-selected AR order; estimated AR coefficients; and, if `factor.length = "adaptive.selection"`, test results for all considered windows.

Author(s)

Yulia R. Gel, Vyacheslav Lyubchich, Ethan Schaeffer

References

Bickel PJ, Sakov A (2008). "On the choice of m in the m out of n bootstrap and confidence bounds for extrema." *Statistica Sinica*, **18**(3), 967–985.

Hall P, Van Keilegom I (2003). "Using difference-based methods for inference in nonparametric regression with time series errors." *Journal of the Royal Statistical Society, Series B (Statistical Methodology)*, **65**(2), 443–456. doi:10.1111/14679868.00395.

Lyubchich V (2016). "Detecting time series trends and their synchronization in climate data." *Intelligence. Innovations. Investments*, **12**, 132–137.

Lyubchich V, Gel YR (2016). "A local factor nonparametric test for trend synchronism in multiple time series." *Journal of Multivariate Analysis*, **150**, 91–104. doi:10.1016/j.jmva.2016.05.004.

Lyubchich V, Gel YR, El-Shaarawi A (2013). "On detecting non-monotonic trends in environmental time series: a fusion of local regression and bootstrap." *Environmetrics*, **24**(4), 209–226. doi:10.1002/env.2212.

See Also

`ar`, `HVK`, `WAVK`, `sync_test`, `vignette("trendtests", package = "funtimes")`

Examples

```
# Fix seed for reproducible simulations:
set.seed(1)

#Simulate autoregressive time series of length n with smooth quadratic trend:
n <- 100
tsTrend <- 1 + 2*(1:n/n) + 4*(1:n/n)^2
tsNoise <- arima.sim(n = n, list(order = c(2, 0, 0), ar = c(-0.7, -0.1)))
U <- tsTrend + tsNoise
plot.ts(U)

#Test H0 of a linear trend, with m-out-of-n selection of the local window:
```

```
## Not run:
    wavk_test(U ~ t, factor.length = "adaptive.selection")
## End(Not run)
# Sample output:
## Trend test by Wang, Akritas, and Van Keilegom (bootstrap p-values)
##
##data:  U
##WAVK test statistic = 5.3964, adaptively selected window = 4, p-value < 2.2e-16
##alternative hypothesis: trend is not of the form U ~ t.

#Test H0 of a quadratic trend, with m-out-of-n selection of the local window
#and output of all results:
## Not run:
    wavk_test(U ~ poly(t, 2), factor.length = "adaptive.selection", out = TRUE)
## End(Not run)
# Sample output:
## Trend test by Wang, Akritas, and Van Keilegom (bootstrap p-values)
##
##data:  U
##WAVK test statistic = 0.40083, adaptively selected window = 4, p-value = 0.576
##alternative hypothesis: trend is not of the form U ~ poly(t, 2).
##sample estimates:
##$trend_coefficients
##(Intercept) poly(t, 2)1 poly(t, 2)2
##   3.408530   17.681422    2.597213
##
##$AR_order
##[1] 1
##
##$AR_coefficients
##         phi_1
##[1] -0.7406163
##
##$all_considered_windows
## Window WAVK-statistic p-value
##      4     0.40083181   0.576
##      5     0.06098625   0.760
##      7    -0.57115451   0.738
##     10    -1.02982929   0.360

# Test H0 of no trend (constant trend) using asymptotic distribution of statistic.
wavk_test(U ~ 1, method = "asympt")
# Sample output:
## Trend test by Wang, Akritas, and Van Keilegom (asymptotic p-values)
##
##data:  U
##WAVK test statistic = 25.999, user-defined window = 10, p-value < 2.2e-16
##alternative hypothesis: trend is not of the form U ~ 1.
```

Index