Package: smoots (via r-universe)

October 13, 2026

Type Package

Title Nonparametric Estimation of the Trend and Its Derivatives in TS

Version 1.1.5

Description The nonparametric trend and its derivatives in equidistant time series (TS) with short-memory stationary errors can be estimated. The estimation is conducted via local polynomial regression using an automatically selected bandwidth obtained by a built-in iterative plug-in algorithm or a bandwidth fixed by the user. A Nadaraya-Watson kernel smoother is also built-in as a comparison. With version 1.1.0, a linearity test for the trend function, forecasting methods and backtesting approaches are implemented as well. The smoothing methods of the package are described in Feng, Y., Gries, T., and Fritz, M. (2020) <doi:10.1080/10485252.2020.1759598>. The software itself is introduced in Feng, Y., Gries, T., Letmathe, S., and Schulz, D. (2022) <doi:10.32614/RJ-2022-017>.

License GPL-3

Encoding UTF-8

LazyData true

Imports stats, utils, graphics, grDevices, Rcpp (>= 1.0.7), future (>= 1.22.1), future.apply (>= 1.8.1), progressr (>= 0.8.0), progress (>= 1.2.2)

Suggests knitr, rmarkdown, fGarch, RcppArmadillo (>= 0.10.6.0.0), testthat (>= 3.0.0)

LinkingTo Rcpp, RcppArmadillo

Depends R (>= 2.10)

Acknowledgments This work was supported by the German DFG project GZ-FE-1500-2-1.

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Yuanhua Feng [aut] (Paderborn University, Germany), Sebastian Letmathe [aut] (formerly Paderborn University, Germany), Dominik Schulz [aut, cre] (formerly Paderborn University, Germany), Thomas Gries [ctb] (Paderborn University, Germany), Marlon Fritz [ctb] (formerly Paderborn University, Germany)

Maintainer Dominik Schulz <dominik.schulz.r@gmail.com>

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

Date/Publication 2026-10-13 11:20:02 UTC

RemoteUrl https://github.com/cran/smoots

RemoteRef HEAD

RemoteSha 5d9dfbae0bc3bcba4668673993fae82164e0facf

Contents

smoots-package . . . 3
bootCast . . . 5
confBounds . . . 9
critMatrix . . . 12
dax . . . 14
dsmooth . . . 15
fitted.smoots . . . 19
gdpUS . . . 19
gsmooth . . . 20
knsmooth . . . 23
modelCast . . . 25
msmooth . . . 29
normCast . . . 35
optOrd . . . 38
plot.smoots . . . 40
print.smoots . . . 41
rescale . . . 41
residuals.smoots . . . 42
rollCast . . . 43
tempNH . . . 48
trendCast . . . 49
tsmooth . . . 51
vix . . . 57

Index **59**

smoots-package *smoots: A package for data-driven nonparametric estimation of the trend and its derivatives in equidistant time series.*

Description

The smoots package provides different applicable functions for the estimation of the trend or its derivatives in equidistant time series. The main functions include an automated bandwidth selection method for time series with short-memory errors. With package version 1.1.0 several functions for forecasting as well as linearity tests were added.

Functions (version 1.0.0)

The smoots functions are either meant for calculating nonparametric estimates of the trend of a time series or its derivatives.

msmooth is the central function of the package. It allows the user to conduct a local polynomial regression of the trend based on an optimal bandwidth that is obtained by an iterative plug-in algorithm. There are also different algorithms implemented concerning the inflation rate and other factors that can be chosen from (see also: msmooth).

dsmooth is a function that calculates the derivatives of the trend after obtaining the optimal bandwidth by an iterative plug-in algorithm. The estimates are obtained for rescaled time points on the interval $[0, 1]$ (see also: dsmooth).

tsmooth is similar to msmooth as it also calculates the trend of the series. Instead of using the name of a predefined algorithm that settles the inflation rate (and other factors), these factors can be manually and individually adjusted as arguments in the function (see also: tsmooth).

gsmooth is a standard smoothing function that applies the local polynomial regression method. A bandwidth can be chosen freely. The estimates are obtained for rescaled time points on the interval $[0, 1]$ (see also: gsmooth).

knsmooth is a standard smoothing function that applies the kernel regression method. A bandwidth can be chosen freely (see also: knsmooth).

Added Functions (version 1.1.0)

With the publication of the package version 1.1.0, new functions were added. These include functions for forecasting and functions for testing linearity of the deterministic trend.

rescale helps rescaling the estimates of the derivatives of the nonparametric trend function, because the estimates are obtained for rescaled time points on the interval $[0, 1]$ and not for the actual time points (see also: rescale).

critMatrix is a quick tool for the calculation of information criteria for ARMA(p, q) models with different order combinations p and q. The function returns a matrix with the obtained values of the selected criterion for the different combinations of p and q (see also: critMatrix).

optOrd is useful in combination with critMatrix. It reads a matrix equal in structure to the ones returned by critMatrix and returns the optimal orders p and q. Furthermore, additional restrictions for the selection can be imposed (see also: optOrd).

normCast provides means to obtain point forecasts as well as forecasting intervals for a given series under the assumption that it follows an ARMA(p, q) model and that its innovations are normally distributed (see also: normCast).

bootCast can also be used to calculate point forecasts and forecasting intervals, if the series of interest follows an ARMA(p, q) model. However, the main difference is that this function should be applied, if the distribution of the innovations is unknown or explicitly non-Gaussian, because the underlying bootstrap process does not need a distribution assumption. In spite of this advantage, it also needs significantly more computation time. Therefore, with version 1.1.1, the bootstrap can now be conducted in parallel for an improved performance (see also: bootCast).

trendCast uses a smoots object that is the output of a trend estimation and calculates point forecasts of the trend. Forecasting intervals are omitted for reasons that are explained within the function's documentation (see also: trendCast).

modelCast calculates the point forecasts and forecasting intervals of a trend-stationary series. Based on a smoots object that is the output of a trend estimation, trendCast is applied to the estimated trend, whereas either normCast or bootCast is applied to the residuals (see also: modelCast).

rollCast is a backtesting function for Semi-ARMA models. A given series is divided into in-sample and out-of-sample observations, where the in-sample is used to fit a Semi-ARMA model. One-step rolling forecasts and the corresponding forecasting intervals are then obtained for the out-of-sample observations. The quality of the model is then assessed via a comparison of the true out-of-sample observations with the point forecasts and forecasting intervals. Different quality criteria are calculated and returned (see also: modelCast).

confBounds calculates the confidence bounds of the estimated trend or of the estimated derivative of the trend at a predefined confidence level. A graphic of the confidence bounds alongside a previously chosen constant or parametric illustration of the estimated series is created. With this plot it can be tested graphically if the deterministic trend is constant or if it follows a parametric polynomial model. Furthermore, it can also be tested, if the derivatives of the trend are constant (see also: confBounds).

Datasets

The package includes four datasets: gdpUS (see also: gdpUS) that has data concerning the quarterly US GDP between Q1 1947-01 and Q2 2019, tempNH (see also: tempNH) with mean monthly Northern Hemisphere temperature changes from 1880 to 2018, dax (see also: dax) with daily financial time series data of the German stock index (DAX) from 1990 to July 2019 and vix (see also: vix) with daily financial time series data of the CBOE Volatility Index (VIX) from 1990 to July 2019.

License

The package is distributed under the General Public License v3 ([GPL-3](https://tldrlegal.com/license/gnu-general-public-license-v3-(gpl-3))).

Author(s)

- Yuanhua Feng (Department of Economics, Paderborn University),
  Author of the Algorithms

- Dominik Schulz (formerly Department of Economics, Paderborn University),
  Package Creator and Maintainer

References

Feng, Y., Gries, T. and Fritz, M. (2020). Data-driven local polynomial for the trend and its derivatives in economic time series. Journal of Nonparametric Statistics, 32:2, 510-533.

Feng, Y., Gries, T., Letmathe, S. and Schulz, D. (2022). The smoots package in R for semiparametric modeling of trend stationary time series. The R Journal, 14:1, 182-195.

`bootCast` *Forecasting Function for ARMA Models via Bootstrap*

Description

Point forecasts and the respective forecasting intervals for autoregressive-moving-average (ARMA) models can be calculated, the latter via bootstrap, by means of this function.

Usage

```
bootCast(
  X,
  p = NULL,
  q = NULL,
  include.mean = FALSE,
  n.start = 1000,
  h = 1,
  it = 10000,
  pb = TRUE,
  cores = max(future::availableCores() - 1, 1),
  alpha = 0.95,
  export.error = FALSE,
  plot = FALSE,
  ...
)
```

Arguments

`X`	a numeric vector that contains the time series that is assumed to follow an ARMA model ordered from past to present.
`p`	an integer value ≥ 0 that defines the AR order p of the underlying ARMA(p, q) model within `X`; is set to `NULL` by default; if no value is passed to `p` but one is passed to `q`, `p` is set to `0`; if both `p` and `q` are `NULL`, optimal orders following the BIC for $0 \leq p, q \leq 5$ are chosen; is set to `NULL` by default; decimal numbers will be rounded off to integers.
`q`	an integer value ≥ 0 that defines the MA order q of the underlying ARMA(p, q) model within `X`; is set to `NULL` by default; if no value is passed to `q` but one is passed to `p`, `q` is set to `0`; if both `p` and `q` are `NULL`, optimal orders following the BIC for $0 \leq p, q \leq 5$ are chosen; is set to `NULL` by default; decimal numbers will be rounded off to integers.

include.mean — a logical value; if set to TRUE, the mean of the series is also also estimated; if set to FALSE, $E(X_t) = 0$ is assumed; is set to FALSE by default.

n.start — an integer that defines the 'burn-in' number of observations for the simulated ARMA series via bootstrap; is set to 1000 by default; decimal numbers will be rounded off to integers.

h — an integer that represents the forecasting horizon; if n is the number of observations, point forecasts and forecasting intervals will be obtained for the time points $n+1$ to $n+h$; is set to h = 1 by default; decimal numbers will be rounded off to integers.

it — an integer that represents the total number of iterations, i.e., the number of simulated series; is set to 10000 by default; decimal numbers will be rounded off to integers.

pb — a logical value; for pb = TRUE, a progress bar will be shown in the console.

cores — an integer value >0 that states the number of (logical) cores to use in the bootstrap (or NULL); the default is the maximum number of available cores minus 1 (via future::availableCores); for cores = NULL, parallel computation is disabled.

alpha — a numeric vector of length 1 with $0 <$ alpha < 1; the forecasting intervals will be obtained based on the confidence level $(100$alpha$)$-percent; is set to alpha = 0.95 by default, i.e., a 95-percent confidence level.

export.error — a single logical value; if the argument is set to TRUE, a list is returned instead of a matrix (FALSE); the first element of the list is the usual forecasting matrix, whereas the second element is a matrix with h columns, where each column represents the calculated forecasting errors for the respective future time point $n+1, n+2, ..., n+h$; is set to FALSE by default.

plot — a logical value that controls the graphical output; for plot = TRUE, the original series with the obtained point forecasts as well as the forecasting intervals will be plotted; for the default plot = FALSE, no plot will be created.

... — additional arguments for the standard plot function, e.g., xlim, type, ... ; arguments with respect to plotted graphs, e.g., the argument col, only affect the original series X; please note that in accordance with the argument x (lower case) of the standard plot function, an additional numeric vector with time points can be implemented via the argument x (lower case). x should be valid for the sample observations only, i.e. length(x) == length(X) should be TRUE, as future time points will be calculated automatically.

Details

This function is part of the smoots package and was implemented under version 1.1.0. For a given time series X_t, $t = 1, 2, ..., n$, the point forecasts and the respective forecasting intervals will be calculated. It is assumed that the series follows an ARMA(p, q) model

$$X_t - \mu = \epsilon_t + \beta_1(X_{t-1} - \mu) + ... + \beta_p(X_{t-p} - \mu) + \alpha_1\epsilon_{t-1} + ... + \alpha_q\epsilon_{t-q},$$

where α_j and β_i are real numbers (for $i = 1, 2, .., p$ and $j = 1, 2, ..., q$) and ϵ_t are i.i.d. (identically and independently distributed) random variables with zero mean and constant variance. μ is equal to $E(X_t)$.

The point forecasts and forecasting intervals for the future periods $n+1, n+2, ..., n+h$ will be obtained. With respect to the point forecasts $\hat{X}_{n+k}$, where $k = 1, 2, ..., h$,

$$\hat{X}_{n+k} = \hat{\mu} + \sum_{i=1}^{p} \hat{\beta}_i (X_{n+k-i} - \hat{\mu}) + \sum_{j=1}^{q} \hat{\alpha}_j \hat{\epsilon}_{n+k-j}$$

with $X_{n+k-i} = \hat{X}_{n+k-i}$ for $n+k-i > n$ and $\hat{\epsilon}_{n+k-j} = E(\epsilon_t) = 0$ for $n+k-j > n$ will be applied.

The forecasting intervals on the other hand are obtained by a forward bootstrap method that was introduced by Pan and Politis (2016) for autoregressive models and extended by Lu and Wang (2020) for applications to autoregressive-moving-average models. For this purpose, let l be the number of the current bootstrap iteration. Based on the demeaned residuals of the initial ARMA estimation, different innovation series $\epsilon^s_{l,t}$ will be sampled. The initial coefficient estimates and the sampled innovation series are then used to simulate a variety of series $X^s_{l,t}$, from which again coefficient estimates will be obtained. With these newly obtained estimates, proxy residual series $\hat{\epsilon}^s_{l,t}$ are calculated for the original series X_t. Subsequently, point forecasts for the time points $n+1$ to $n+h$ are obtained for each iteration l based on the original series X_t, the newly obtained coefficient forecasts and the proxy residual series $\epsilon^s_{l,t}$. Simultaneously, "true" forecasts, i.e., true future observations, are simulated. Within each iteration, the difference between the simulated true forecast and the bootstrapped point forecast is calculated and saved for each future time point $n+1$ to $n+h$. The result for these time points are simulated empirical values of the forecasting error. Denote by $q_k(.)$ the quantile of the empirical distribution for the future time point $n+k$. Given a predefined confidence level `alpha`, define $\alpha_s = (1-$ `alpha`$)/2$. The bootstrapped forecasting interval is then

$$[\hat{X}_{n+k} + q_k(\alpha_s), \hat{X}_{n+k} + q_k(1-\alpha_s)],$$

i.e., the forecasting intervals are given by the sum of the respective point forecasts and quantiles of the respective bootstrapped forecasting error distributions.

The function `bootCast` allows for different adjustments to the forecasting progress. At first, a vector with the values of the observed time series ordered from past to present has to be passed to the argument `X`. Orders p and q of the underlying ARMA process can be defined via the arguments `p` and `q`. If only one of these orders is inserted by the user, the other order is automatically set to `0`. If none of these arguments are defined, the function will choose orders based on the Bayesian Information Criterion (BIC) for $0 \leq p, q \leq 5$. Via the logical argument `include.mean` the user can decide, whether to consider the mean of the series within the estimation process. By means of `n.start`, the number of "burn-in" observations for the simulated ARMA processes can be regulated. These observations are usually used for the processes to build up and then omitted. Furthermore, the argument `h` allows for the definition of the maximum future time point $n+h$. Point forecasts and forecasting intervals will be returned for the time points $n+1$ to $n+h$. `it` corresponds to the number of bootstrap iterations. We recommend a sufficiently high number of repetitions for maximum accuracy of the results. Another argument is `alpha`, which is the equivalent of the confidence level considered within the calculation of the forecasting intervals, i.e., the quantiles $(1-$ `alpha`$)/2$ and $1-(1-$ `alpha`$)/2$ of the bootstrapped forecasting error distribution will be obtained.

Since this bootstrap approach needs a lot of computation time, especially for series with high numbers of observations and when fitting models with many parameters, parallel computation of the bootstrap iterations is enabled. With `cores`, the number of cores can be defined with an integer. Nonetheless, for `cores = NULL`, no cluster is created and therefore the parallel computation is disabled. Note that the bootstrapped results are fully reproducible for all cluster sizes. The progress of

the bootstrap can be observed in the R console, where a progress bar and the estimated remaining time are displayed for pb = TRUE.

If the argument export.error is set to TRUE, the output of the function is a list instead of a matrix with additional information on the simulated forecasting errors. For more information see the section *Value*.

For simplicity, the function also incorporates the possibility to directly create a plot of the output, if the argument plot is set to TRUE. By the additional and optional arguments ..., further arguments of the standard plot function can be implemented to shape the returned plot.

NOTE:

Within this function, the arima function of the stats package with its method "CSS-ML" is used throughout for the estimation of ARMA models. Furthermore, to increase the performance, C++ code via the Rcpp and RcppArmadillo packages was implemented. Also, the future and future.apply packages are considered for parallel computation of bootstrap iterations. The progress of the bootstrap is shown via the progressr package.

Value

The function returns a 3 by h matrix with its columns representing the future time points and the point forecasts, the lower bounds of the forecasting intervals and the upper bounds of the forecasting intervals as the rows. If the argument plot is set to TRUE, a plot of the forecasting results is created.

If export.error = TRUE is selected, a list with the following elements is returned instead.

fcast the 3 by h matrix forecasting matrix with point forecasts and bounds of the forecasting intervals.

error a it by h matrix, where each column represents a future time point $n+1, n+2, ..., n+h$; in each column the respective it simulated forecasting errors are saved.

Author(s)

- Dominik Schulz (formerly Department of Economics, Paderborn University), Package Creator and Maintainer

References

Feng, Y., Gries, T. and Fritz, M. (2020). Data-driven local polynomial for the trend and its derivatives in economic time series. Journal of Nonparametric Statistics, 32:2, 510-533.

Feng, Y., Gries, T., Letmathe, S. and Schulz, D. (2022). The smoots package in R for semiparametric modeling of trend stationary time series. The R Journal, 14:1, 182-195.

Feng, Y., Gries, T., Fritz, M., Letmathe, S. and Schulz, D. (2026). Diagnosing the trend and bootstrapping the forecasting intervals using a semiparametric ARMA. CIE Working Paper Series (Paderborn University), 168.

Lu, X., and Wang, L. (2020). Bootstrap prediction interval for ARMA models with unknown orders. REVSTAT–Statistical Journal, 18:3, 375-396.

Pan, L. and Politis, D. N. (2016). Bootstrap prediction intervals for linear, nonlinear and nonparametric autoregressions. In: Journal of Statistical Planning and Inference 177, pp. 1-27.

Examples

```
### Example 1: Simulated ARMA process ###

# Function for drawing from a demeaned chi-squared distribution
rchisq0 <- function(n, df, npc = 0) {
 rchisq(n, df, npc) - df
}

# Simulation of the underlying process
n <- 2000
n.start = 1000
set.seed(23)
X <- arima.sim(model = list(ar = c(1.2, -0.7), ma = 0.63), n = n,
 rand.gen = rchisq0, n.start = n.start, df = 3) + 13.1

# Quick application with low number of iterations
# (not recommended in practice)
result <- bootCast(X = X, p = 2, q = 1, include.mean = TRUE,
 n.start = n.start, h = 5, it = 10, cores = 2, plot = TRUE,
 lty = 3, col = "forestgreen", xlim = c(1950, 2005), type = "b",
 main = "Exemplary title", pch = "*")
result

### Example 2: Application with more iterations ###
## Not run:
result2 <- bootCast(X = X, p = 2, q = 1, include.mean = TRUE,
 n.start = n.start, h = 5, it = 10000, cores = 2, plot = TRUE,
 lty = 3, col = "forestgreen", xlim = c(1950, 2005),
 main = "Exemplary title")
result2


## End(Not run)
```

---

confBounds *Asymptotically Unbiased Confidence Bounds*

Description

Asymptotically Unbiased Confidence Bounds

Usage

```
confBounds(
  obj,
  alpha = 0.95,
  p = c(0, 1, 2, 3),
  plot = TRUE,
  showPar = TRUE,
```

```
  rescale = TRUE,
  ...
)
```

Arguments

obj	an object returned by either msmooth, tsmooth or dsmooth.
alpha	the confidence level; a single numeric value between 0 and 1; 0.95 is the default.
p	the order of polynomial used for the parametric polynomial regression that is conducted as a benchmark for the trend function; must satisfy $0 \leq$ p ≤ 3; set to 1 by default; is irrelevant, if a derivative of the trend of order greater than zero is being analyzed.
plot	a logical value; for plot = TRUE, the default, a plot is created.
showPar	set to TRUE, if the parametric fitted values are to be shown against the unbiased estimates and the confidence bounds for plot = TRUE; the default is TRUE.
rescale	a single logical value; is set to TRUE by default; if the output of a derivative estimation process is passed to obj and if rescale = TRUE, the estimates and confidence bounds will be rescaled according to x for the plot (see also the details on the parameter ...); the numerical output stays unchanged.
...	further arguments that can be passed to the plot function; if an argument x with time points is not given by the user, x = 1:length(obj$ye) is used per default for the observation time points.

Details

This function is part of the smoots package and was implemented under version 1.1.0. The underlying theory is based on the additive nonparametric regression function

$$y_t = m(x_t) + \epsilon_t,$$

where y_t is the observed time series, x_t is the rescaled time on the interval $[0, 1]$, $m(x_t)$ is a smooth trend function and ϵ_t are stationary errors with $E(\epsilon_t) = 0$ and short-range dependence.

The purpose of this function is the estimation of reasonable confidence intervals for the nonparametric trend function and its derivatives. The optimal bandwidth minimizes the Asymptotic Mean Integrated Squared Error (AMISE) criterion, however, local polynomial estimates are (usually) biased. The bias is then (approximately)

$$\frac{h^{k-v} m^{(k)}(x) \beta_{(\nu,k)}}{k!},$$

where p is the order of the local polynomials, $k = p + 1$ is the order of the asymptotically equivalent kernel, ν is the order of the of the trend function's derivative, $m^(v)$ is the ν-th order derivative of the trend function and $\beta_{(\nu,k)} = \int_{-1}^{1} u^k K_{(\nu,k)}(u) du$. $K_{(\nu,k)}(u)$ is the k-th order asymptotically equivalent kernel function for estimating $m^{(\nu)}$. A renewed estimation with an adjusted bandwidth $h_{ub} = o(n^{-1/(2k+1)})$, i.e., a bandwidth with a smaller order than the optimal bandwidth, is conducted. $h = h_A^{(2k+1)/(2k)}$, where h_A is the optimal bandwidth, is implemented.

Following this idea, we have that

$$\sqrt{nh}[m^{(\nu)}(x) - \hat{m}^{(\nu)}(x)]$$

converges to

$$N(0, 2\pi c_f R(x))$$

in distribution, where $2\pi c_f$ is the sum of autocovariances. Consequently, the trend (or derivative) estimates are asymptotically unbiased and normally distributed.

To make use of this function, an object of class `smoots` can be given as input that was created by either `msmooth`, `tsmooth` or `dsmooth`. Based on the optimal bandwidth saved within `obj`, an adjustment to the bandwidth is made so that the estimates following the adjusted bandwidth are (relatively) unbiased.

Based on the input argument `alpha`, the level of confidence between `0` and `1`, the respective confidence bounds are calculated for each observation point.

From the input argument `obj`, the order of derivative is automatically obtained. By means of the argument `p`, an order of polynomial is selected for a parametric regression of the trend function. This is only meaningful, if the trend (and not its derivatives) is analyzed. Otherwise, the argument is automatically dropped by the function. Furthermore, if `plot = TRUE`, a plot of the unbiased trend (or derivative) estimates alongside the confidence bounds is created. If also `showPar = TRUE`, the estimated parametric trend (or parametric constant value for the derivatives) is added to the confidence bound plot for comparison.

NOTE:

The values that are returned by the function are obtained with respect to the rescaled time points on the interval $[0, 1]$. While the plot can be adjusted and rescaled by means of a given vector with the actual time points, the numeric output is not rescaled. For this purpose we refer the user to the `rescale` function of the `smoots` package.

This function implements C++ code by means of the `Rcpp` and `RcppArmadillo` packages for better performance.

Value

A plot is created in the plot window and a list with different components is returned.

alpha a numeric vector of length 1; the level of confidence; input argument.

b.ub a numeric vector with one element that represents the adjusted bandwidth for the unbiased trend estimation.

p.estim a numeric vector with the estimates following the parametric regression defined by `p` that is conducted as a benchmark for the trend function; for the trend's derivatives or for `p = 0`, a constant value is the benchmark; the values are obtained with respect to the rescaled time points on the interval $[0, 1]$.

n the number of observations.

np.estim a data frame with the three (numeric) columns **ye.ub**, **lower** and **upper**; in **ye.ub** the unbiased trend estimates, in **lower** the lower confidence bound and in **upper** the upper confidence bound can be found; the values are obtained with respect to the rescaled time points on the interval $[0, 1]$.

v the order of the trend's derivative considered for the test.

Author(s)

- Yuanhua Feng (Department of Economics, Paderborn University),
  Author of the Algorithms

- Dominik Schulz (formerly Department of Economics, Paderborn University),
  Package Creator and Maintainer

References

Beran, J. and Feng, Y. (2002). Local polynomial fitting with long-memory, short-memory and antipersistent errors. Annals of the Institute of Statistical Mathematics, 54(2), 291-311.

Feng, Y., Gries, T. and Fritz, M. (2020). Data-driven local polynomial for the trend and its derivatives in economic time series. Journal of Nonparametric Statistics, 32:2, 510-533.

Feng, Y., Gries, T., Letmathe, S. and Schulz, D. (2022). The smoots package in R for semiparametric modeling of trend stationary time series. The R Journal, 14:1, 182-195.

Feng, Y., Gries, T., Fritz, M., Letmathe, S. and Schulz, D. (2026). Diagnosing the trend and bootstrapping the forecasting intervals using a semiparametric ARMA. CIE Working Paper Series (Paderborn University), 168.

Examples

```
log_gdp <- log(smoots::gdpUS$GDP)
est <- msmooth(log_gdp)
confBounds(est)
```

`critMatrix` *ARMA Order Selection Matrix*

Description

An information criterion is calculated for different orders of an autoregressive-moving-average (ARMA) model.

Usage

```
critMatrix(
  X,
  p.max = 5,
  q.max = 5,
  criterion = c("bic", "aic"),
  include.mean = TRUE
)
```

Arguments

X	a numeric vector that contains the observed time series ordered from past to present; the series is assumed to follow an ARMA process.
p.max	an integer value $>= 0$ that defines the maximum autoregressive order to calculate the criterion for; is set to 5 by default; decimal numbers will be rounded off to integers.
q.max	an integer value $>= 0$ that defines the maximum moving-average order to to calculate the criterion for; is set to 5 by default; decimal numbers will be rounded off to integers.
criterion	a character value that defines the information criterion that will be calculated; the Bayesian Information Criterion ("bic") and Akaike Information Criterion ("aic") are the supported choices; is set to "bic" by default.
include.mean	a logical value; this argument regulates whether to estimate the mean of the series (TRUE) or not (FALSE); is set to TRUE by default.

Details

This function is part of the smoots package and was implemented under version 1.1.0. The series passed to X is assumed to follow an ARMA(p, q) model. A p.max + 1 by q.max + 1 matrix is calculated for this series. More precisely, the criterion chosen via the argument criterion is calculated for all combinations of orders $p = 0, 1, ..., p_{max}$ and $q = 0, 1, ..., q_{max}$.

Within the function, two information criteria are supported: the Bayesian Information Criterion (BIC) and Akaike's Information Criterion (AIC). The AIC is given by

$$AIC_{p,q} := \ln(\hat{\sigma}^2_{p,q}) + \frac{2(p+q)}{n},$$

where $\hat{sigma}^2_{p,q}$ is the estimated innovation variance, p and q are the ARMA orders and n is the number of observations.

The BIC, on the other hand, is defined by

$$BIC_{p,q} := k \ln(n) - 2 \ln(\hat{L})$$

with k being the number of estimated parameters and $\hat{L}$ being the estimated Log-Likelihood. Since the parameter k only differs with respect to the orders p and q for all estimated models, the term $k \ln(n)$ is reduced to $(p + q) \ln(n)$ within the function. Exemplarily, if the mean of the series is estimated as well, it is usually considered within the parameter k when calculating the BIC. However, since the mean is estimated for all models, not considering this estimated parameter within the calculation of the BIC will reduce all BIC values by the same amount of $\ln(n)$. Therefore, the selection via this simplified criterion is still valid, if the number of the estimated parameters only differs with respect to p and q between the models that the BIC is obtained for.

The optimal orders are considered to be the ones which minimize either the BIC or the AIC. The use of the BIC is however recommended, because the BIC is consistent, whereas the AIC is not.

NOTE:

Within this function, the arima function of the stats package with its method "CSS-ML" is used throughout for the estimation of ARMA models.

Value

The function returns a `p.max + 1` by `q.max + 1` matrix, where the rows represent the AR orders from $p = 0$ to $p = p_{max}$ and the columns represent the MA orders from $q = 0$ to $q = q_{max}$. The values within the matrix are the values of the previously selected information criterion for the different combinations of p and q.

Author(s)

- Dominik Schulz (formerly Department of Economics, Paderborn University), Package Creator and Maintainer

Examples

```
## Not run:
# Simulate an ARMA(2,1) process
set.seed(23)
X.sim <- stats::arima.sim(model = list(ar = c(1.2, -0.71), ma = 0.46),
 n = 1000) + 13.1
# Application of the function
critMatrix(X.sim)
# Result: Via the BIC, the orders p.opt = 2 and q.opt = 1 are selected.

## End(Not run)
```

---

dax *German Stock Market Index (DAX) Financial Time Series Data*

Description

A dataset that contains the daily financial data of the DAX from 1990 to July 2019 (currency in EUR).

Usage

```
dax
```

Format

A data frame with 7475 rows and 9 variables:

Year the observation year
Month the observation month
Day the observation day
Open the opening price of the day
High the highest price of the day
Low the lowest price of the day
Close the closing price of the day
AdjClose the adjusted closing price of the day
Volume the traded volume

Source

The data was obtained from Yahoo Finance (accessed: 2019-08-22).

https://query1.finance.yahoo.com/v7/finance/download/^GDAXI?period1=631148400&period2=1564524000&interval=1d&events=history&crumb=Iaq1EPZAQRb

---

`dsmooth` *Data-driven Local Polynomial for the Trend's Derivatives in Equidistant Time Series*

Description

This function runs through an iterative process in order to find the optimal bandwidth for the non-parametric estimation of the first or second derivative of the trend in an equidistant time series (with short-memory errors) and subsequently employs the obtained bandwidth via local polynomial regression.

Usage

```
dsmooth(
  y,
  d = c(1, 2),
  mu = c(0, 1, 2, 3),
  pp = c(1, 3),
  bStart.p = 0.15,
  bStart = 0.15
)
```

Arguments

`y`	a numeric vector that contains the time series ordered from past to present.
`d`	an integer `1` or `2` that defines the order of derivative; the default is `d = 1`.
`mu`	an integer `0`, ..., `3` that represents the smoothness parameter of the kernel weighting function and thus defines the kernel function that will be used within the local polynomial regression; is set to `1` by default.

Number	Kernel
`0`	Uniform Kernel
`1`	Epanechnikov Kernel
`2`	Bisquare Kernel
`3`	Triweight Kernel

`pp`	an integer `1` (local linear regression) or `3` (local cubic regression) that indicates the order of polynomial upon which c_f, i.e. the variance factor, will be calculated by `msmooth`; the default is `pp = 1`.
`bStart.p`	a numeric object that indicates the starting value of the bandwidth for the iterative process for the calculation of c_f; should be > 0; is set to `0.15` by default.
`bStart`	a numeric object that indicates the starting value of the bandwidth for the iterative process; should be > 0; is set to `0.15` by default.

Details

The trend's derivative is estimated based on the additive nonparametric regression model for an equidistant time series

$$y_t = m(x_t) + \epsilon_t,$$

where y_t is the observed time series, x_t is the rescaled time on the interval $[0, 1]$, $m(x_t)$ is a smooth and deterministic trend function and ϵ_t are stationary errors with $E(\epsilon_t) = 0$ and short-range dependence (see also Beran and Feng, 2002). With this function, the first or second derivative of $m(x_t)$ can be estimated without a parametric model assumption for the error series.

The iterative-plug-in (IPI) algorithm, which numerically minimizes the Asymptotic Mean Squared Error (AMISE), was proposed by Feng, Gries and Fritz (2020).

Define $I[m^{(k)}] = \int_{c_b}^{d_b} [m^{(k)}(x)]^2 dx$, $\beta_{(\nu,k)} = \int_{-1}^{1} u^k K_{(\nu,k)}(u) du$ and $R(K) = \int_{-1}^{1} K_{(\nu,k)}^2(u) du$, where p is the order of the polynomial, $k = p + 1$ is the order of the asymptotically equivalent kernel, ν is the order of the trend function's derivative, $0 \leq c_b < d_b \leq 1$, c_f is the variance factor and $K_{(\nu,k)}(u)$ the k-th order equivalent kernel obtained for the estimation of $m^{(\nu)}$ in the interior. $m^{(\nu)}$ is the ν-th order derivative ($\nu = 0, 1, 2, ...$) of the nonparametric trend.

Furthermore, we define

$$C_1 = \frac{I[m^{(k)}]\beta_{(\nu,k)}^2}{(k!)^2}$$

and

$$C_2 = \frac{2\pi c_f (d_b - c_b) R(K)}{n h^{2\nu+1}}$$

with h being the bandwidth and n being the number of observations. The AMISE is then

$$AMISE(h) = h^{2(k-\nu)} C_1 + C_2.$$

The variance factor c_f is first obtained from a pilot-estimation of the time series' nonparametric trend ($\nu = 0$) with polynomial order p_p. The estimate is then plugged into the iterative procedure for estimating the first or second derivative ($\nu = 1$ or $\nu = 2$). For further details on the asymptotic theory or the algorithm, we refer the user to Feng, Fritz and Gries (2020) and Feng et al. (2022).

The function itself is applicable in the following way: Based on a data input `y`, an order of polynomial `pp` for the variance factor estimation procedure, a starting value for the relative bandwidth `bStart.p` in the variance factor estimation procedure, a kernel function defined by the smoothness parameter `mu` and a starting value for the relative bandwidth `bStart` in the bandwidth estimation procedure, an optimal bandwidth is numerically calculated for the trend's derivative of order `d`. In fact, aside from the input vector `y`, every argument has a default setting that can be adjusted for the individual case. However, it is recommended to initially use the default values for the estimation of the first derivative and adjust the argument `d` to `d = 2` for the estimation of the second derivative. Following Feng, Gries and Fritz (2020), the initial bandwidth does not affect the resulting optimal bandwidth in theory. However in practice, local minima of the AMISE can influence the results. Therefore, the default starting bandwidth is set to `0.15`, the suggested starting bandwidth by Feng, Gries and Fritz (2020) for the data-driven estimation of the first derivative. The recommended initial bandwidth for the second derivative, however, is `0.2` and not `0.15`. Thus, if the algorithm does not give suitable results (especially for `d = 2`), the adjustment of the initial bandwidth might be a good starting point. Analogously, the default starting bandwidth for the trend estimation for the variance factor is `bStart.p = 0.15`, although according to Feng, Gries and Fritz (2020), `bStart.p = 0.1` is suggested for `pp = 1` and `bStart.p = 0.2` for `pp = 3`. The default is therefore a compromise

between the two suggested values. For more specific information on the input arguments consult the section *Arguments*.

After the bandwidth estimation, the nonparametric derivative of the series is calculated with respect to the obtained optimal bandwidth by means of a local polynomial regression. The output object is then a list that contains, among other components, the original time series, the estimates of the derivative and the estimated optimal bandwidth.

The default print method for this function delivers key numbers such as the iteration steps and the generated optimal bandwidth rounded to the fourth decimal. The exact numbers and results such as the estimated nonparametric trend series are saved within the output object and can be addressed via the `$` sign.

NOTE:

The estimates are obtained for the rescaled time points on the interval $[0, 1]$. Therefore, the estimated derivatives might not reflect the derivatives for the actual time points. To rescale them, we refer the user to the `rescale` function of the `smoots` package.

With package version 1.1.0, this function implements C++ code by means of the `Rcpp` and `RcppArmadillo` packages for better performance.

Value

The function returns a list with different components:

b0 the optimal bandwidth chosen by the IPI-algorithm.

bStart the starting bandwidth for the local polynomial regression based derivative estimation procedure; input argument.

bStart.p the starting bandwidth for the nonparametric trend estimation that leads to the variance factor estimate; input argument.

bvc indicates whether an enlarged bandwidth was used for the variance factor estimation or not; it is always set to `"Y"` (yes) for this function.

cf0 the estimated variance factor; in contrast to the definitions given in the *Details* section, this object actually contains an estimated value of $2\pi c_f$, i.e. it corresponds to the estimated sum of autocovariances.

InfR the inflation rate setting.

iterations the bandwidths of the single iterations steps

Mcf the estimation method for the variance factor estimation; it is always estimated nonparametrically (`"NP"`) within this function.

mu the smoothness parameter of the second order kernel; input argument.

n the number of observations.

niterations the total number of iterations until convergence.

orig the original input series; input argument.

p the order of polynomial for the local polynomial regression used within derivative estimation procedure.

pp the order of polynomial for the local polynomial regression used in the variance factor estimation; input argument.

v the considered order of the trend's derivative; input argument d.

ws the weighting system matrix used within the local polynomial regression; this matrix is a condensed version of a complete weighting system matrix; in each row of ws, the weights for conducting the smoothing procedure at a specific observation time point can be found; the first $[nb + 0.5]$ rows, where n corresponds to the number of observations, b is the bandwidth considered for smoothing and $[.]$ denotes the integer part, contain the weights at the $[nb + 0.5]$ left-hand boundary points; the weights in row $[nb + 0.5] + 1$ are representative for the estimation at all interior points and the remaining rows contain the weights for the right-hand boundary points; each row has exactly $2[nb + 0.5] + 1$ elements, more specifically the weights for observations of the nearest $2[nb + 0.5] + 1$ time points; moreover, the weights are normalized, i.e. the weights are obtained under consideration of the time points $x_t = t/n$, where $t = 1, 2, ..., n$.

ye the nonparametric estimates of the derivative for the rescaled time points on the interval $[0, 1]$.

Author(s)

- Yuanhua Feng (Department of Economics, Paderborn University),
  Author of the Algorithms
- Dominik Schulz (formerly Department of Economics, Paderborn University),
  Package Creator and Maintainer

References

Feng, Y., Gries, T. and Fritz, M. (2020). Data-driven local polynomial for the trend and its derivatives in economic time series. Journal of Nonparametric Statistics, 32:2, 510-533.

Feng, Y., Gries, T., Letmathe, S. and Schulz, D. (2022). The smoots package in R for semiparametric modeling of trend stationary time series. The R Journal, 14:1, 182-195.

Examples

```
# Logarithm of test data
test_data <- gdpUS
y <- log(test_data$GDP)
t <- seq(from = 1947, to = 2019.25, by = 0.25)

# Applied dsmooth function for the trend's first derivative
result_d <- dsmooth(y, d = 1, mu = 1, pp = 1, bStart.p = 0.1, bStart = 0.15)
estim <- result_d$ye

# Plot of the results
plot(t, estim, xlab = "Year", ylab = "First derivative", type = "l",
 main = paste0("Estimated first derivative of the trend for log-quarterly ",
 "US-GDP, Q1 1947 - Q2 2019"), cex.axis = 0.8, cex.main = 0.8,
 cex.lab = 0.8, bty = "n")

# Print result
result_d
```

fitted.smoots *Extract Model Fitted Values*

Description

Generic function which extracts fitted values from a `smoots` class object. Both `fitted` and `fitted.values` can be called.

Usage

```
## S3 method for class 'smoots'
fitted(object, ...)
```

Arguments

`object`	an object from the `smoots` class.
`...`	included for consistency with the generic function.

Value

Fitted values extracted from a `smoots` class object.

Author(s)

- Sebastian Letmathe (formerly Department of Economics, Paderborn University),

gdpUS *Quarterly US GDP, Q1 1947 to Q2 2019*

Description

A dataset that contains the (seasonally adjusted) Gross Domestic Product of the US from the first quarter of 1947 to the second quarter of 2019

Usage

```
gdpUS
```

Format

A data frame with 290 rows and 3 variables:

Year the observation year

Quarter the observation quarter in the given year

GDP the Gross Domestic Product of the US in billions of chained 2012 US Dollars (annual rate)

Source

The data was obtained from the Federal Reserve Bank of St. Louis (accessed: 2019-09-01).

https://fred.stlouisfed.org/series/GDPC1

`gsmooth` *Estimation of Trends and their Derivatives via Local Polynomial Regression*

Description

This function is an R function for estimating the trend function and its derivatives in an equidistant time series with local polynomial regression and a fixed bandwidth given beforehand.

Usage

```
gsmooth(y, v = 0, p = v + 1, mu = 1, b = 0.15, bb = c(0, 1))
```

Arguments

`y` a numeric vector that contains the time series data ordered from past to present.

`v` an integer `0`, `1`, ... that represents the order of derivative that will be estimated; is set to `v = 0` by default.

Number (`v`)	**Degree of derivative**
`0`	The function *f(x)* itself
`1`	The first derivative *f'(x)*
`2`	The second derivative *f''(x)*
...	...

`p` an integer $>= (\ v + 1)$ that represents the order of polynomial; `p - v` must be an odd number; is set to `v + 1` by default.

Exemplary for `v = 0`:

Number (`p`)	**Polynomial**	`p - v`	`p - v` **odd?**	`p` **usable?**
`1`	Linear	1	Yes	Yes
`2`	Quadratic	2	No	No
`3`	Cubic	3	Yes	Yes
...	...	...	...	...

`mu` an integer `0`, `1`, `2`, ... that represents the smoothness parameter of the kernel weighting function that will be used; is set to `1` by default.

Number (`mu`)	**Kernel**
`0`	Uniform Kernel
`1`	Epanechnikov Kernel

2	Bisquare Kernel
3	Triweight Kernel
...	...

b — a real number $0 < b < 0.5$; represents the relative bandwidth that will be used for the smoothing process; is set to `0.15` by default.

bb — can be set to `0` or `1`; the parameter controlling the bandwidth used at the boundary; is set to `1` by default.

Number (bb)	**Estimation procedure at boundary points**
0	Fixed bandwidth on one side with possible large bandwidth on the other side at the boundary
1	The k-nearest neighbor method will be used

Details

The trend or its derivatives are estimated based on the additive nonparametric regression model for an equidistant time series

$$y_t = m(x_t) + \epsilon_t,$$

where y_t is the observed time series, x_t is the rescaled time on the interval $[0, 1]$, $m(x_t)$ is a smooth and deterministic trend function and ϵ_t are stationary errors with $E(\epsilon_t) = 0$ (see also Beran and Feng, 2002).

This function is part of the package `smoots` and is used in the field of analyzing equidistant time series data. It applies the local polynomial regression method to the input data with an arbitrarily selectable bandwidth. By these means, the trend as well as its derivatives can be estimated non-parametrically, even though the result will strongly depend on the bandwidth given beforehand as an input.

NOTE:

The estimates are obtained with regard to the rescaled time points on the interval $[0, 1]$. Thus, if $\nu > 0$, the estimates might not reflect the values for the actual time points. To rescale the estimates, we refer the user to the `rescale` function of the `smoots` package.

With package version 1.1.0, this function implements C++ code by means of the `Rcpp` and `RcppArmadillo` packages for better performance.

Value

The output object is a list with different components:

b the chosen (relative) bandwidth; input argument.

bb the chosen bandwidth option at the boundaries; input argument.

mu the chosen smoothness parameter for the second order kernel; input argument.

n the number of observations.

orig the original input series; input argument.

p the chosen order of polynomial; input argument.

res a vector with the estimated residual series; is set to `NULL` for `v > 0`.

v the order of derivative; input argument.

ws the weighting system matrix used within the local polynomial regression; this matrix is a condensed version of a complete weighting system matrix; in each row of ws, the weights for conducting the smoothing procedure at a specific observation time point can be found; the first $[nb + 0.5]$ rows, where n corresponds to the number of observations, b is the bandwidth considered for smoothing and $[.]$ denotes the integer part, contain the weights at the $[nb + 0.5]$ left-hand boundary points; the weights in row $[nb + 0.5] + 1$ are representative for the estimation at all interior points and the remaining rows contain the weights for the right-hand boundary points; each row has exactly $2[nb + 0.5] + 1$ elements, more specifically the weights for observations of the nearest $2[nb + 0.5] + 1$ time points; moreover, the weights are normalized, i.e. the weights are obtained under consideration of the time points $x_t = t/n$, where $t = 1, 2, ..., n$.

ye a vector with the estimates of the selected nonparametric order of derivative on the rescaled time interval $[0, 1]$.

Author(s)

- Yuanhua Feng (Department of Economics, Paderborn University),
  Author of the Algorithms

- Dominik Schulz (formerly Department of Economics, Paderborn University),
  Package Creator and Maintainer

References

Beran, J. and Feng, Y. (2002). Local polynomial fitting with long-memory, short-memory and antipersistent errors. Annals of the Institute of Statistical Mathematics, 54(2), 291-311.

Feng, Y., Gries, T. and Fritz, M. (2020). Data-driven local polynomial for the trend and its derivatives in economic time series. Journal of Nonparametric Statistics, 32:2, 510-533.

Feng, Y., Gries, T., Letmathe, S. and Schulz, D. (2019). The smoots package in R for semiparametric modeling of trend stationary time series. Discussion Paper. Paderborn University. Unpublished.

Examples

```
# Logarithm of test data
test_data <- gdpUS
y <- log(test_data$GDP)

# Applied gsmooth function for the trend with two different bandwidths
results1 <- gsmooth(y, v = 0, p = 1, mu = 1, b = 0.28, bb = 1)
results2 <- gsmooth(y, v = 0, p = 1, mu = 1, b = 0.11, bb = 1)
trend1 <- results1$ye
trend2 <- results2$ye

# Plot of the results
t <- seq(from = 1947, to = 2019.25, by = 0.25)
plot(t, y, type = "l", xlab = "Year", ylab = "log(US-GDP)", bty = "n",
 lwd = 2,
 main = "Estimated trend for log-quarterly US-GDP, Q1 1947 - Q2 2019")
points(t, trend1, type = "l", col = "red", lwd = 1)
```

```
points(t, trend2, type = "l", col = "blue", lwd = 1)
legend("bottomright", legend = c("Trend (b = 0.28)", "Trend (b = 0.11)"),
 fill = c("red", "blue"), cex = 0.6)
title(sub = expression(italic("Figure 1")), col.sub = "gray47",
 cex.sub = 0.6, adj = 0)
```

---

knsmooth *Estimation of Nonparametric Trend Functions via Kernel Regression*

Description

This function estimates the nonparametric trend function in an equidistant time series with Nadaraya-Watson kernel regression.

Usage

```
knsmooth(y, mu = 1, b = 0.15, bb = c(0, 1))
```

Arguments

y	a numeric vector that contains the time series data ordered from past to present.
mu	an integer 0, 1, 2, ... that represents the smoothness parameter of the second order kernel function that will be used; is set to 1 by default.

Number (mu)	**Kernel**
0	Uniform Kernel
1	Epanechnikov Kernel
2	Bisquare Kernel
3	Triweight Kernel
...	...

b	a real number $0 < b < 0.5$; represents the relative bandwidth that will be used for the smoothing process; is set to 0.15 by default.
bb	can be set to 0 or 1; the parameter controlling the bandwidth used at the boundary; is set to 0 by default.

Number (bb)	**Estimation procedure at boundary points**
0	Fixed bandwidth on one side with possible large bandwidth on the other side at the boundary
1	The k-nearest neighbor method will be used

Details

The trend is estimated based on the additive nonparametric regression model for an equidistant time series

$$y_t = m(x_t) + \epsilon_t,$$

where y_t is the observed time series, x_t is the rescaled time on the interval $[0, 1]$, $m(x_t)$ is a smooth and deterministic trend function and ϵ_t are stationary errors with $E(\epsilon_t) = 0$.

This function is part of the package `smoots` and is used for the estimation of trends in equidistant time series. The applied method is a kernel regression with arbitrarily selectable second order kernel, relative bandwidth and boundary method. Especially the chosen bandwidth has a strong impact on the final result and has thus to be selected carefully. This approach is not recommended by the authors of this package.

Value

The output object is a list with different components:

b the chosen (relative) bandwidth; input argument.

bb the chosen bandwidth option at the boundaries; input argument.

mu the chosen smoothness parameter for the second order kernel; input argument.

n the number of observations.

orig the original input series; input argument.

res a vector with the estimated residual series.

ye a vector with the estimates of the nonparametric trend.

Author(s)

- Yuanhua Feng (Department of Economics, Paderborn University),
  Author of the Algorithms

- Dominik Schulz (formerly Department of Economics, Paderborn University),
  Package Creator and Maintainer

References

Feng, Y. (2009). Kernel and Locally Weighted Regression. Verlag für Wissenschaft und Forschung, Berlin.

Examples

```
# Logarithm of test data
test_data <- gdpUS
y <- log(test_data$GDP)

#Applied knmooth function for the trend with two different bandwidths
trend1 <- knsmooth(y, mu = 1, b = 0.28, bb = 1)$ye
trend2 <- knsmooth(y, mu = 1, b = 0.05, bb = 1)$ye
```

```
# Plot of the results
t <- seq(from = 1947, to = 2019.25, by = 0.25)
plot(t, y, type = "l", xlab = "Year", ylab = "log(US-GDP)", bty = "n",
 lwd = 2,
 main = "Estimated trend for log-quarterly US-GDP, Q1 1947 - Q2 2019")
points(t, trend1, type = "l", col = "red", lwd = 1)
points(t, trend2, type = "l", col = "blue", lwd = 1)
legend("bottomright", legend = c("Trend (b = 0.28)", "Trend (b = 0.05)"),
 fill = c("red", "blue"), cex = 0.6)
title(sub = expression(italic("Figure 1")), col.sub = "gray47",
 cex.sub = 0.6, adj = 0)
```

`modelCast` *Forecasting Function for Trend-Stationary Time Series*

Description

Point forecasts and the respective forecasting intervals for trend-stationary time series are calculated.

Usage

```
modelCast(
  obj,
  p = NULL,
  q = NULL,
  h = 1,
  method = c("norm", "boot"),
  alpha = 0.95,
  it = 10000,
  n.start = 1000,
  pb = TRUE,
  cores = max(future::availableCores() - 1, 1),
  np.fcast = c("lin", "const"),
  export.error = FALSE,
  plot = FALSE,
  ...
)
```

Arguments

`obj`	an object of class `smoots`; must be the output of a trend estimation process and not of a first or second derivative estimation process.
`p`	an integer value $>= 0$ that defines the AR order p of the underlying ARMA(p, q) model within the rest term (see the section *Details* for more information); is set to `NULL` by default; if no value is passed to `p` but one is passed to `q`, `p` is set to

0; if both p and q are NULL, optimal orders following the BIC for $0 \leq p, q \leq 5$ are chosen; is set to NULL by default; decimal numbers will be rounded off to integers.

q — an integer value ≥ 0 that defines the MA order q of the underlying ARMA(p, q) model within X; is set to NULL by default; if no value is passed to q but one is passed to p, q is set to 0; if both p and q are NULL, optimal orders following the BIC for $0 \leq p, q \leq 5$ are chosen; is set to NULL by default; decimal numbers will be rounded off to integers.

h — an integer that represents the forecasting horizon; if n is the number of observations, point forecasts and forecasting intervals will be obtained for the time points $n+1$ to $n+h$; is set to h = 1 by default; decimal numbers will be rounded off to integers.

method — a character object; defines the method used for the calculation of the forecasting intervals; with "norm" the intervals are obtained under the assumption of normally distributed innovations; with "boot" the intervals are obtained via a bootstrap; is set to "norm" by default.

alpha — a numeric vector of length 1 with $0 <$ alpha < 1; the forecasting intervals will be obtained based on the confidence level $(100$alpha$)$-percent; is set to alpha = 0.95 by default, i.e., a 95-percent confidence level.

it — an integer that represents the total number of iterations, i.e., the number of simulated series; is set to 10000 by default; only necessary, if method = "boot"; decimal numbers will be rounded off to integers.

n.start — an integer that defines the 'burn-in' number of observations for the simulated ARMA series via bootstrap; is set to 1000 by default; only necessary, if method = "boot";decimal numbers will be rounded off to integers.

pb — a logical value; for pb = TRUE, a progress bar will be shown in the console, if method = "boot".

cores — an integer value >0 that states the number of (logical) cores to use in the bootstrap (or NULL); the default is the maximum number of available cores minus 1 (via future::availableCores); for cores = NULL, parallel computation is disabled.

np.fcast — a character object; defines the forecasting method used for the nonparametric trend; for np.fcast = "lin" the trend is is extrapolated linearly based on the last two trend estimates; for np.fcast = "const", the last trend estimate is used as a constant estimate for future values; is set to *"lin"* by default.

export.error — a single logical value; if the argument is set to TRUE and if also method = "boot", a list is returned instead of a matrix (FALSE); the first element of the list is the usual forecasting matrix whereas the second element is a matrix with h columns, where each column represents the calculated forecasting errors for the respective future time point $n+1, n+2, ..., n+h$; is set to FALSE by default.

plot — a logical value that controls the graphical output; for plot = TRUE, the original series with the obtained point forecasts as well as the forecasting intervals will be plotted; for the default plot = FALSE, no plot will be created.

... — additional arguments for the standard plot function, e.g., xlim, type, ... ; arguments with respect to plotted graphs, e.g., the argument col, only affect the

original series X; please note that in accordance with the argument x (lower case) of the standard plot function, an additional numeric vector with time points can be implemented via the argument x (lower case). x should be valid for the sample observations only, i.e. `length(x) == length(obj$orig)` should be TRUE, as future time points will be calculated automatically.

Details

This function is part of the *smoots* package and was implemented under version 1.1.0. The point forecasts and forecasting intervals are obtained based on the additive nonparametric regression model

$$y_t = m(x_t) + \epsilon_t,$$

where y_t is the observed time series with equidistant design, x_t is the rescaled time on the interval $[0, 1]$, $m(x_t)$ is a smooth trend function and ϵ_t are stationary errors with $E(\epsilon_t) = 0$ and short-range dependence (see also Beran and Feng, 2002). Thus, we assume y_t to be a trend-stationary time series. Furthermore, we assume that the rest term ϵ_t follows an ARMA(p, q) model

$$\epsilon_t = \zeta_t + \beta_1\epsilon_{t-1} + ... + \beta_p\epsilon_{t-p} + \alpha_1\zeta_{t-1} + ... + \alpha_q\zeta_{t-q},$$

where α_j, $j = 1, 2, ..., q$, and β_i, $i = 1, 2, ..., p$, are real numbers and the random variables ζ_t are i.i.d. (identically and independently distributed) with zero mean and constant variance.

The point forecasts and forecasting intervals for the future periods $n + 1, n + 2, ..., n + h$ will be obtained. With respect to the point forecasts of ϵ_t, i.e., $\hat{\epsilon}_{n+k}$, where $k = 1, 2, ..., h$,

$$\hat{\epsilon}_{n+k} = \sum_{i=1}^{p} \hat{\beta}_i\epsilon_{n+k-i} + \sum_{j=1}^{q} \hat{\alpha}_j\hat{\zeta}_{n+k-j}$$

with $\epsilon_{n+k-i} = \hat{\epsilon}_{n+k-i}$ for $n + k - i > n$ and $\hat{\zeta}_{n+k-j} = E(\zeta_t) = 0$ for $n + k - j > n$ will be applied. In practice, this procedure will not be applied directly to ϵ_t but to $y_t - \hat{m}(x_t)$.

The point forecasts of the nonparametric trend are simply obtained following the proposal by Fritz et al. (forthcoming) by

$$\hat{m}(x_{n+k}) = \hat{m}(x_n) + Dk(\hat{m}(x_n) - \hat{m}(x_{n-1})),$$

where D is a dummy variable that is either equal to the constant value 1 or 0. Consequently, if $D = 0$, $\hat{m}(x_n)$, i.e., the last trend estimate, is used as a constant estimate for the future. However, if $D = 1$, the trend is extrapolated linearly. The point forecast for the whole component model is then given by

$$\hat{y}_{n+k} = \hat{m}(x_{n+k}) + \hat{\epsilon}_{n+k},$$

i.e., it is equal to the sum of the point forecasts of the individual components.

Equivalently to the point forecasts, the forecasting intervals are the sum of the forecasting intervals of the individual components. To simplify the process, the forecasting error in $\hat{m}(x_{n+k})$, which is of order $O(-2/5)$, is not considered (see Fritz et al. (forthcoming)), i.e., only the forecasting intervals with respect to the rest term ϵ_t will be calculated.

If the distribution of the innovations is non-normal or generally not further specified, bootstrapping the forecasting intervals is recommended. If they are however normally distributed or if it is at least assumed that they are, the forecasting errors are also approximately normally distributed with a

quickly obtainable variance. For further details on the bootstrapping method, we refer the readers to bootCast, whereas more information on the calculation under normality can be found at normCast.

In order to apply the function, a smoots object that was generated as the result of a trend estimation process needs to be passed to the argument obj. The arguments p and q represent the orders of the of the ARMA(p, q) model that the error term ϵ_t is assumed to follow. If both arguments are set to NULL, which is the default setting, orders will be selected according to the Bayesian Information Criterion (BIC) for all possible combinations of $p, q = 0, 1, ..., 5$. Furthermore, the forecasting horizon can be adjusted by means of the argument h, so that point forecasts and forecasting intervals will be obtained for all time points $n + 1, n + 2, ..., n + h$.

The function also allows for two calculation approaches for the forecasting intervals. Via the argument method, intervals can be obtained under the assumption that the ARMA innovations are normally distributed (method = "norm"). Alternatively, bootstrapped intervals can be obtained for unknown innovation distributions that are clearly non-Gaussian (method = "boot").

Another argument is alpha. By passing a value to this argument, the $(100$alpha$)$-percent confidence level for the forecasting intervals can be defined. If method = "boot" is selected, the additional arguments it and n.start can be adjusted. More specifically, it regulates the number of iterations of the bootstrap, whereas n.start sets the number of 'burn-in' observations in the simulated ARMA processes within the bootstrap that are omitted.

Since this bootstrap approach for method = "boot" generally needs a lot of computation time, especially for series with high numbers of observations and when fitting models with many parameters, parallel computation of the bootstrap iterations is enabled. With cores, the number of cores can be defined with an integer. Nonetheless, for cores = NULL, no cluster is created and therefore the parallel computation is disabled. Note that the bootstrapped results are fully reproducible for all cluster sizes. The progress of the bootstrap can be observed in the R console, where a progress bar and the estimated remaining time are displayed for pb = TRUE.

Moreover, the argument np.fcast allows to set the forecasting method for the nonparametric trend function. As previously discussed, the two options are a linear extrapolation of the trend (np.fcast = "lin") and a constant continuation of the last estimated value of the trend (np.fcast = "const").

The function also implements the option to automatically create a plot of the forecasting results for plot = TRUE. This includes the feature to pass additional arguments of the standard plot function to modelCast (see also the section 'Examples').

NOTE:

Within this function, the arima function of the stats package with its method "CSS-ML" is used throughout for the estimation of ARMA models. Furthermore, to increase the performance, C++ code via the Rcpp and RcppArmadillo packages was implemented. Also, the future and future.apply packages are considered for parallel computation of bootstrap iterations. The progress of the bootstrap is shown via the progressr package.

Value

The function returns a 3 by h matrix with its columns representing the future time points and the point forecasts, the lower bounds of the forecasting intervals and the upper bounds of the forecasting intervals as the rows. If the argument plot is set to TRUE, a plot of the forecasting results is created.

#'If export.error = TRUE is selected, a list with the following elements is returned instead.

fcast the 3 by h forecasting matrix with point forecasts and bounds of the forecasting intervals.

error an `it` by h matrix, where each column represents a future time point $n+1, n+2, ..., n+h$; in each column the respective `it` simulated forecasting errors are saved.

Author(s)

- Yuanhua Feng (Department of Economics, Paderborn University),
  Author of the Algorithms

- Dominik Schulz (formerly Department of Economics, Paderborn University),
  Package Creator and Maintainer

References

Beran, J. and Feng, Y. (2002). Local polynomial fitting with long-memory, short-memory and antipersistent errors. Annals of the Institute of Statistical Mathematics, 54(2), 291-311.

Feng, Y., Gries, T. and Fritz, M. (2020). Data-driven local polynomial for the trend and its derivatives in economic time series. Journal of Nonparametric Statistics, 32:2, 510-533.

Feng, Y., Gries, T., Letmathe, S. and Schulz, D. (2022). The smoots package in R for semiparametric modeling of trend stationary time series. The R Journal, 14:1, 182-195.

Feng, Y., Gries, T., Fritz, M., Letmathe, S. and Schulz, D. (2026). Diagnosing the trend and bootstrapping the forecasting intervals using a semiparametric ARMA. CIE Working Paper Series (Paderborn University), 168.

Fritz, M., Forstinger, S., Feng, Y., and Gries, T. (forthcoming). Forecasting economic growth processes for developing economies. Unpublished.

Examples

```
X <- log(smoots::gdpUS$GDP)
NPest <- smoots::msmooth(X)
modelCast(NPest, h = 5, plot = TRUE, xlim = c(261, 295), type = "b",
 col = "deepskyblue4", lty = 3, pch = 20, main = "Exemplary title")
```

`msmooth` *Data-driven Nonparametric Regression for the Trend in Equidistant Time Series*

Description

This function runs an iterative plug-in algorithm to find the optimal bandwidth for the estimation of the nonparametric trend in equidistant time series (with short memory errors) and then employs the resulting bandwidth via either local polynomial or kernel regression.

Usage

```
msmooth(
  y,
  p = c(1, 3),
  mu = c(0, 1, 2, 3),
  bStart = 0.15,
  alg = c("A", "B", "N", "NA", "NAM", "NM", "O", "OA", "OAM", "OM"),
  method = c("lpr", "kr"),
  nar_lim = c(0, 1),
  nma_lim = c(0, 1),
  arma_mean = TRUE
)
```

Arguments

y	a numeric vector that contains the input time series ordered from past to present.
p	an integer 1 (local linear regression) or 3 (local cubic regression); represents the order of polynomial within the local polynomial regression (see also the 'Details' section); is set to 1 by default; is automatically set to 1 if method = "kr".
mu	an integer 0, ..., 3 that represents the smoothness parameter of the kernel weighting function and thus defines the kernel function that will be used within the local polynomial regression; is set to 1 by default.

Number	Kernel
0	Uniform Kernel
1	Epanechnikov Kernel
2	Bisquare Kernel
3	Triweight Kernel

bStart	a numeric object that indicates the starting value of the bandwidth for the iterative process; should be > 0; is set to 0.15 by default.
alg	a control parameter (as character) that indicates the corresponding algorithm used (set to "A" by default for p = 1 and to "B" for p = 3).

Algorithm	Description
"A"	Nonparametric estimation of the variance factor with an enlarged bandwidth, optimal inflation rate
"B"	Nonparametric estimation of the variance factor with an enlarged bandwidth, naive inflation rate
"O"	Nonparametric estimation of the variance factor, optimal inflation rate
"N"	Nonparametric estimation of the variance factor, naive inflation rate
"OAM"	Estimation of the variance factor with ARMA(p, q)-models, optimal inflation rate
"NAM"	Estimation of the variance factor with ARMA(p, q)-models, naive inflation rate
"OA"	Estimation of the variance factor with AR(p)-models, optimal inflation rate
"NA"	Estimation of the variance factor with AR(p)-models, naive inflation rate
"OM"	Estimation of the variance factor with MA(q)-models, optimal inflation rate
"NM"	Estimation of the variance factor with MA(q)-models, naive inflation rate

It is proposed to use `alg = "A"` in combination with `p = 1`. If the user finds that the chosen bandwidth by algorithm `"A"` is too small, `alg = "B"` with preferably `p = 3` is suggested. For more information on the components of the different algorithms, please consult `tsmooth`.

`method` the smoothing approach; `"lpr"` represents the local polynomial regression, whereas `"kr"` implements a kernel regression; is set to `"lpr"` by default.

`nar_lim` a two-element numeric vector with the minimum and the maximum AR order to check via BIC in the fitting of ARMA models, when the variance factor is obtained via fitted parametric ARMA models and not nonparametrically.

`nma_lim` a two-element numeric vector with the minimum and the maximum MA order to check via BIC in the fitting of ARMA models, when the variance factor is obtained via fitted parametric ARMA models and not nonparametrically.

`arma_mean` logical; whether to include the mean in the fitting of ARMA models, only if the variance factor is estimated via parametric ARMA models and not nonparametrically.

Details

The trend is estimated based on the additive nonparametric regression model for an equidistant time series

$$y_t = m(x_t) + \epsilon_t,$$

where y_t is the observed time series, x_t is the rescaled time on the interval $[0, 1]$, $m(x_t)$ is a smooth and deterministic trend function and ϵ_t are stationary errors with $E(\epsilon_t) = 0$ and short-range dependence (see also Beran and Feng, 2002). With this function $m(x_t)$ can be estimated without a parametric model assumption for the error series. Thus, after estimating and removing the trend, any suitable parametric model, e.g. an ARMA(p, q) model, can be fitted to the residuals (see `arima`).

The iterative-plug-in (IPI) algorithm, which numerically minimizes the Asymptotic Mean Squared Error (AMISE), was proposed by Feng, Gries and Fritz (2020).

Define $I[m^{(k)}] = \int_{c_b}^{d_b} [m^{(k)}(x)]^2 dx$, $\beta_{(\nu,k)} = \int_{-1}^{1} u^k K_{(\nu,k)}(u) du$ and $R(K) = \int_{-1}^{1} K_{(\nu,k)}^2(u) du$, where p is the order of the polynomial, $k = p + 1$ is the order of the asymptotically equivalent kernel, ν is the order of the trend function's derivative, $0 \leq c_b < d_b \leq 1$, c_f is the variance factor and $K_{(\nu,k)}(u)$ the k-th order equivalent kernel obtained for the estimation of $m^{(\nu)}$ in the interior. $m^{(\nu)}$ is the ν-th order derivative ($\nu = 0, 1, 2, ...$) of the nonparametric trend.

Furthermore, we define

$$C_1 = \frac{I[m^{(k)}]\beta_{(\nu,k)}^2}{(k!)^2}$$

and

$$C_2 = \frac{2\pi c_f (d_b - c_b) R(K)}{n h^{2\nu+1}}$$

with h being the bandwidth and n being the number of observations. The AMISE is then

$$AMISE(h) = h^{2(k-\nu)} C_1 + C_2.$$

The function calculates suitable estimates for c_f, the variance factor, and $I[m^{(k)}]$ over different iterations. In each iteration, a bandwidth is obtained in accordance with the AMISE that once

more serves as an input for the following iteration. The process repeats until either convergence or the 40th iteration is reached. For further details on the asymptotic theory or the algorithm, please consult Feng, Gries and Fritz (2020) or Feng et al. (2022).

To apply the function, only few arguments are needed: a data input `y`, an order of polynomial `p`, a kernel function defined by the smoothness parameter `mu`, a starting value for the relative bandwidth `bStart` and a final smoothing method `method`. In fact, aside from the input vector `y`, every argument has a default setting that can be adjusted for the individual case. It is recommended to initially use the default values for `p`, `alg` and `bStart` and adjust them in the rare case of the resulting optimal bandwidth being either too small or too large. Theoretically, the initial bandwidth does not affect the selected optimal bandwidth. However, in practice local minima of the AMISE might exist and influence the selected bandwidth. Therefore, the default setting is `bStart = 0.15`, which is a compromise between the starting values `bStart = 0.1` for `p = 1` and `bStart = 0.2` for `p = 3` that were proposed by Feng, Gries and Fritz (2020). In the rare case of a clearly unsuitable optimal bandwidth, a starting bandwidth that differs from the default value is a first possible approach to obtain a better result. Other argument adjustments can be tried as well. For more specific information on the input arguments consult the section *Arguments*.

When applying the function, an optimal bandwidth is obtained based on the IPI algorithm proposed by Feng, Gries and Fritz (2020). In a second step, the nonparametric trend of the series is calculated with respect to the chosen bandwidth and the selected regression method (`lpf` or `kr`). It is notable that `p` is automatically set to 1 for `method = "kr"`. The output object is then a list that contains, among other components, the original time series, the estimated trend values and the series without the trend.

The default print method for this function delivers key numbers such as the iteration steps and the generated optimal bandwidth rounded to the fourth decimal. The exact numbers and results such as the estimated nonparametric trend series are saved within the output object and can be addressed via the `$` sign.

NOTE:

With package version 1.1.0, this function implements C++ code by means of the `Rcpp` and `RcppArmadillo` packages for better performance.

Value

The function returns a list with different components:

AR.BIC the Bayesian Information Criterion of the optimal AR(p) model when estimating the variance factor via autoregressive models (if calculated; calculated for `alg = "OA"` and `alg = "NA"`).

ARMA.BIC the Bayesian Information Criterion of the optimal ARMA(p, q) model when estimating the variance factor via autoregressive-moving-average models (if calculated; calculated for `alg = "OAM"` and `alg = "NAM"`).

cb the percentage of omitted observations on each side of the observation period; always equal to 0.05.

b0 the optimal bandwidth chosen by the IPI-algorithm.

bb the boundary bandwidth method used within the IPI; always equal to 1.

bStart the starting value of the (relative) bandwidth; input argument.

bvc indicates whether an enlarged bandwidth was used for the variance factor estimation or not; depends on the chosen algorithm.

cf0 the estimated variance factor; in contrast to the definitions given in the *Details* section, this object actually contains an estimated value of $2\pi c_f$, i.e. it corresponds to the estimated sum of autocovariances.

cf0.AR the estimated variance factor obtained by estimation of autoregressive models (if calculated; `alg = "OA"` or `"NA"`).

cf0.ARMA the estimated variance factor obtained by estimation of autoregressive-moving-average models (if calculated; calculated for `alg = "OAM"` and `alg = "NAM"`).

cf0.LW the estimated variance factor obtained by Lag-Window Spectral Density Estimation following Bühlmann (1996) (if calculated; calculated for algorithms `"A"`, `"B"`, `"O"` and `"N"`).

cf0.MA the estimated variance factor obtained by estimation of moving-average models (if calculated; calculated for `alg = "OM"` and `alg = "NM"`).

I2 the estimated value of $I[m^{(k)}]$.

InfR the setting for the inflation rate according to the chosen algorithm.

iterations the bandwidths of the single iterations steps

L0.opt the optimal bandwidth for the lag-window spectral density estimation (if calculated; calculated for algorithms `"A"`, `"B"`, `"O"` and `"N"`).

MA.BIC the Bayesian Information Criterion of the optimal MA(q) model when estimating the variance factor via moving-average models (if calculated; calculated for `alg = "OM"` and `alg = "NM"`).

Mcf the estimation method for the variance factor estimation; depends on the chosen algorithm.

mu the smoothness parameter of the second order kernel; input argument.

n the number of observations.

niterations the total number of iterations until convergence.

orig the original input series; input argument.

p.BIC the order p of the optimal AR(p) or ARMA(p, q) model when estimating the variance factor via autoregressive or autoregressive-moving average models (if calculated; calculated for `alg = "OA"`, `alg = "NA"`, `alg = "OAM"` and `alg = "NAM"`).

p the order of polynomial used in the IPI-algorithm; also used for the final smoothing, if `method = "lpr"`; input argument.

q.BIC the order q of the optimal MA(q) or ARMA(p, q) model when estimating the variance factor via moving-average or autoregressive-moving average models (if calculated; calculated for `alg = "OM"`, `alg = "NM"`, `alg = "OAM"` and `alg = "NAM"`).

res the estimated residual series.

v the considered order of derivative of the trend; is always zero for this function.

ws the weighting system matrix used within the local polynomial regression; this matrix is a condensed version of a complete weighting system matrix; in each row of `ws`, the weights for conducting the smoothing procedure at a specific observation time point can be found; the first $[nb + 0.5]$ rows, where n corresponds to the number of observations, b is the bandwidth considered for smoothing and $[.]$ denotes the integer part, contain the weights at the $[nb + 0.5]$

left-hand boundary points; the weights in row $[nb + 0.5] + 1$ are representative for the estimation at all interior points and the remaining rows contain the weights for the right-hand boundary points; each row has exactly $2[nb + 0.5] + 1$ elements, more specifically the weights for observations of the nearest $2[nb + 0.5] + 1$ time points; moreover, the weights are normalized, i.e. the weights are obtained under consideration of the time points $x_t = t/n$, where $t = 1, 2, ..., n$.

ye the nonparametric estimates of the trend.

Author(s)

- Yuanhua Feng (Department of Economics, Paderborn University),
  Author of the Algorithms

- Dominik Schulz (formerly Department of Economics, Paderborn University),
  Package Creator and Maintainer

References

Beran, J. and Feng, Y. (2002). Local polynomial fitting with long-memory, short-memory and antipersistent errors. Annals of the Institute of Statistical Mathematics, 54(2), 291-311.

Bühlmann, P. (1996). Locally adaptive lag-window spectral estimation. Journal of Time Series Analysis, 17(3), 247-270.

Feng, Y., Gries, T. and Fritz, M. (2020). Data-driven local polynomial for the trend and its derivatives in economic time series. Journal of Nonparametric Statistics, 32:2, 510-533.

Feng, Y., Gries, T., Letmathe, S. and Schulz, D. (2022). The smoots package in R for semiparametric modeling of trend stationary time series. The R Journal, 14:1, 182-195.

Examples

```
### Example 1: US-GDP ###

# Logarithm of test data
# -> the logarithm of the data is assumed to follow the additive model
test_data <- gdpUS
y <- log(test_data$GDP)

# Applied msmooth function for the trend
results <- msmooth(y, p = 1, mu = 1, bStart = 0.1, alg = "A", method = "lpr")
res <- results$res
ye <- results$ye

# Plot of the results
t <- seq(from = 1947, to = 2019.25, by = 0.25)
matplot(t, cbind(y, ye), type = "ll", lty = c(3, 1), col = c(1, "red"),
 xlab = "Years", ylab = "Log-Quartlery US-GDP",
 main = "Log-Quarterly US-GDP vs. Trend, Q1 1947 - Q2 2019")
legend("bottomright", legend = c("Original series", "Estimated trend"),
 fill = c(1, "red"), cex = 0.7)
results
```

```
## Not run:
### Example 2: German Stock Index ###

# The following procedure can be considered, if (log-)returns are assumed
# to follow a model from the general class of semiparametric GARCH-type
# models (including Semi-GARCH, Semi-Log-GARCH and Semi-APARCH models among
# others) with a slowly changing variance over time due to a deterministic,
# nonparametric scale function.

# Obtain the logarithm of the squared returns
returns <- diff(log(dax$Close))   # (log-)returns
rt <- returns - mean(returns)     # demeaned (log-)returns
yt <- log(rt^2)                   # logarithm of the squared returns

# Apply 'smoots' function to the log-data, because the logarithm of
# the squared returns follows an additive model with a nonparametric trend
# function, if the returns are assumed to follow a semiparametric GARCH-type
# model.

# In this case, the setting 'alg = "A"' is used in combination with p = 3, as
# the resulting estimates appear to be more suitable than for 'alg = "B"'.
est <- msmooth(yt, p = 3, alg = "A")
m_xt <- est$ye                    # estimated trend values

# Obtain the standardized returns 'eps' and the scale function 'scale.f'
res <- est$res                    # the detrended log-data
C <- -log(mean(exp(res)))         # an estimate of a constant value needed
                                  # for the retransformation
scale.f <- exp((m_xt - C) / 2)    # estimated values of the scale function in
                                  # the returns
eps <- rt / scale.f               # the estimated standardized returns

# -> 'eps' can now be analyzed by any suitable GARCH-type model.
#    The total volatilities are then the product of the conditional
#    volatilities obtained from 'eps' and the scale function 'scale.f'.

## End(Not run)
```

`normCast` *Forecasting Function for ARMA Models under Normally Distributed Innovations*

Description

Point forecasts and the respective forecasting intervals for autoregressive- moving-average (ARMA) models can be calculated, the latter under the assumption of normally distributed innovations, by means of this function.

Usage

```
normCast(
  X,
  p = NULL,
  q = NULL,
  include.mean = FALSE,
  h = 1,
  alpha = 0.95,
  plot = FALSE,
  ...
)
```

Arguments

X	a numeric vector that contains the time series that is assumed to follow an ARMA model ordered from past to present.
p	an integer value $>= 0$ that defines the AR order p of the underlying ARMA(p, q) model within X; is set to NULL by default; if no value is passed to p but one is passed to q, p is set to 0; if both p and q are NULL, optimal orders following the BIC for $0 \leq p, q \leq 5$ are chosen; is set to NULL by default; decimal numbers will be rounded off to integers.
q	an integer value $>= 0$ that defines the MA order q of the underlying ARMA(p, q) model within X; is set to NULL by default; if no value is passed to q but one is passed to p, q is set to 0; if both p and q are NULL, optimal orders following the BIC for $0 \leq p, q \leq 5$ are chosen; is set to NULL by default; decimal numbers will be rounded off to integers.
include.mean	a logical value; if set to TRUE, the mean of the series is also estimated; if set to FALSE, $E(X_t) = 0$ is assumed; is set to FALSE by default.
h	an integer that represents the forecasting horizon; if n is the number of observations, point forecasts and forecasting intervals will be obtained for the time points $n + 1$ to $n + h$; is set to 1 by default; decimal numbers will be rounded off to integers.
alpha	a numeric vector of length 1 with $0 <$ alpha < 1; the forecasting intervals will be obtained based on the confidence level $(100$alpha$)$-percent; is set to *alpha = 0.95* by default, i.e., a 95-percent confidence level.
plot	a logical value that controls the graphical output; for plot = TRUE, the original series with the obtained point forecasts as well as the forecasting intervals will be plotted; for the default plot = FALSE, no plot will be created.
...	additional arguments for the standard plot function, e.g., xlim, type, ... ; arguments with respect to plotted graphs, e.g., the argument col, only affect the original series X; please note that in accordance with the argument x (lower case) of the standard plot function, an additional numeric vector with time points can be implemented via the argument x (lower case). x should be valid for the sample observations only, i.e. length(x) == length(X) should be TRUE, as future time points will be calculated automatically.

Details

This function is part of the `smoots` package and was implemented under version 1.1.0. For a given time series $X_[t]$, $t = 1, 2, ..., n$, the point forecasts and the respective forecasting intervals will be calculated. It is assumed that the series follows an ARMA(p, q) model

$$X_t - \mu = \epsilon_t + \beta_1(X_{t-1} - \mu) + ... + \beta_p(X_{t-p} - \mu) + \alpha_1\epsilon_{t-1} + ... + \alpha_q\epsilon_{t-q},$$

where α_j and β_i are real numbers (for $i = 1, 2, .., p$ and $j = 1, 2, ..., q$) and ϵ_t are i.i.d. (identically and independently distributed) random variables with zero mean and constant variance. μ is equal to $E(X_t)$.

The point forecasts and forecasting intervals for the future periods $n + 1, n + 2, ..., n + h$ will be obtained. With respect to the point forecasts $\hat{X}_{n+k}$, where $k = 1, 2, ..., h$,

$$\hat{X}_{n+k} = \hat{\mu} + \sum_{i=1}^{p} \hat{\beta}_i(X_{n+k-i} - \hat{\mu}) + \sum_{j=1}^{q} \hat{\alpha}_j\hat{\epsilon}_{n+k-j}$$

with $X_{n+k-i} = \hat{X}_{n+k-i}$ for $n + k - i > n$ and $\hat{\epsilon}_{n+k-j} = E(\epsilon_t) = 0$ for $n + k - j > n$ will be applied.

The forecasting intervals on the other hand are obtained under the assumption of normally distributed innovations. Let $q(c)$ be the $100c$-percent quantile of the standard normal distribution. The $100a$-percent forecasting interval at a point $n + k$, where $k = 1, 2, ..., h$, is given by

$$[\hat{X}_{n+k} - q(a_r)s_k, \hat{X}_{n+k} + q(a_r)s_k]$$

with s_k being the standard deviation of the forecasting error at the time point $n + k$ and with $a_r = 1 - (1 - a)/2$. For ARMA models with normal innovations, the variance of the forecasting error can be derived from the MA(∞) representation of the model. It is

$$\sigma_\epsilon^2 \sum_{i=0}^{k-1} d_i^2,$$

where d_i are the coefficients of the MA(∞) representation and σ_ϵ^2 is the innovation variance.

The function `normCast` allows for different adjustments to the forecasting progress. At first, a vector with the values of the observed time series ordered from past to present has to be passed to the argument `X`. Orders p and q of the underlying ARMA process can be defined via the arguments `p` and `q`. If only one of these orders is inserted by the user, the other order is automatically set to `0`. If none of these arguments are defined, the function will choose orders based on the Bayesian Information Criterion (BIC) for $0 \leq p, q \leq 5$. Via the logical argument `include.mean` the user can decide, whether to consider the mean of the series within the estimation process. Furthermore, the argument `h` allows for the definition of the maximum future time point $n + h$. Point forecasts and forecasting intervals will be returned for the time points $n+1$ to $n+h$. Another argument is `alpha`, which is the equivalent of the confidence level considered within the calculation of the forecasting intervals, i.e., the quantiles $(1-$ `alpha`$)/2$ and $1 - (1-$ `alpha`$)/2$ of the forecasting intervals will be obtained.

For simplicity, the function also incorporates the possibility to directly create a plot of the output, if the argument `plot` is set to `TRUE`. By the additional and optional arguments `...`, further arguments of the standard plot function can be implemented to shape the returned plot.

NOTE: Within this function, the `arima` function of the `stats` package with its method `"CSS-ML"` is used throughout for the estimation of ARMA models.

Value

The function returns a 3 by h matrix with its columns representing the future time points and the point forecasts, the lower bounds of the forecasting intervals and the upper bounds of the forecasting intervals as the rows. If the argument `plot` is set to `TRUE`, a plot of the forecasting results is created.

Author(s)

- Dominik Schulz (formerly Department of Economics, Paderborn University), Package Creator and Maintainer

References

Feng, Y., Gries, T. and Fritz, M. (2020). Data-driven local polynomial for the trend and its derivatives in economic time series. Journal of Nonparametric Statistics, 32:2, 510-533.

Feng, Y., Gries, T., Letmathe, S. and Schulz, D. (2022). The smoots package in R for semiparametric modeling of trend stationary time series. The R Journal, 14:1, 182-195.

Feng, Y., Gries, T., Fritz, M., Letmathe, S. and Schulz, D. (2026). Diagnosing the trend and bootstrapping the forecasting intervals using a semiparametric ARMA. CIE Working Paper Series (Paderborn University), 168.

Fritz, M., Forstinger, S., Feng, Y., and Gries, T. (2020). Forecasting economic growth processes for developing economies. Unpublished.

Examples

```
### Example 1: Simulated ARMA process ###

# Simulation of the underlying process
n <- 2000
n.start = 1000
set.seed(21)
X <- arima.sim(model = list(ar = c(1.2, -0.7), ma = 0.63), n = n,
 rand.gen = rnorm, n.start = n.start) + 7.7

# Application of normCast()
result <- normCast(X = X, p = 2, q = 1, include.mean = TRUE, h = 5,
 plot = TRUE, xlim = c(1971, 2005), col = "deepskyblue4",
 type = "b", lty = 3, pch = 16, main = "Exemplary title")
result
```

`optOrd` *Optimal Order Selection*

Description

From a matrix with values of an information criterion for different orders p and q of an autoregressive-moving-average (ARMA) model, the optimal orders are selected.

Usage

```
optOrd(mat, restr = NULL, sFUN = min)
```

Arguments

mat	a numeric matrix, whose rows represent the AR orders $p = 0, 1, ..., p_{max}$ and whose columns represent the MA orders $q = 0, 1, q_{max}$; the elements of the matrix are then the values of an information criterion calculated for ARMA models with the different order combinations; a matrix returned by the function critMatrix of the smoots package shares these characteristics.
restr	a single expression (not a character object) that defines further restrictions; the standard logical operators, e.g. >=, & or ==, can be used; refer to the rows with p and to the columns with q; is set to NULL by default, i.e. no restrictions are imposed.
sFUN	the selection function; is set to min, i.e. the minimal value that meets the restrictions restr is selected and the corresponding orders p and q are returned.

Details

Given a matrix mat filled with the values of an information criterion for different estimated ARMA(p, q) models, where the rows represent different orders $p = 0, 1, ..., p_{max}$ and where the columns represent the orders $q = 0, 1, ..., q_{max}$, the function returns a vector with the optimal orders p and q. Further selection restrictions can be passed to the argument restr as an expression. To implement a restriction, the rows and columns are addressed via p and q, respectively. Moreover, standard boolean operators such as ==, >= or & can be used. See the Section *Examples* for examples of different restrictions. In many cases, the minimum value of a criterion is considered to indicate the best model. However, in some other cases a different selection approach might be appropriate. Therefore, a selection function can be considered by means of the argument sFUN. The default is sFUN = min, i.e. the function min is applied to select the optimal orders.

Value

The function returns a vector with two elements. The first element is the optimal order p, whereas the second element is the selected optimal order q.

Author(s)

- Sebastian Letmathe (formerly Department of Economics, Paderborn University),

Examples

```
## Not run:
set.seed(21)
Xt <- arima.sim(model = list(ar = c(1.2, -0.5), ma = 0.7), n = 1000) + 7
mat <- smoots::critMatrix(Xt)
optOrd(mat)  # without restrictions
optOrd(mat, p <= q)  # with one restriction
optOrd(mat, p >= 1 & q >= 4)  # with two restrictions
```

```
## End(Not run)
```

`plot.smoots` *Plot Method for the Package 'smoots'*

Description

This function regulates how objects created by the package `smoots` are plotted.

Usage

```
## S3 method for class 'smoots'
plot(x, t = NULL, rescale = TRUE, which = NULL, ...)
```

Arguments

`x`	an input object of class `smoots`.
`t`	an optional vector with time points that will be considered for the x-axis within the plot; is set to NULL by default and uses a vector `1:length(x$ye)` for time points.
`rescale`	a single logical value; is set to `TRUE` by default; if the output of a derivative estimation process is passed to `x` and if `rescale = TRUE`, the estimates will be rescaled according to `t`.
`which`	a selector for the plot type so that the interactive prompt is avoided; for the default, `which = NULL`, the user will be asked interactively via the console which plot to show; to avoid this behavior, set `which` to the corresponding number of the plot you would like to create (1: original series, 2: trend series, 3: residual series, 4: original series with trend series for trend estimation objects, 1: original series, 2: derivative series for trend derivative estimation object).
`...`	additional arguments of the standard plot method.

Value

None

Author(s)

- Dominik Schulz (formerly Department of Economics, Paderborn University), Package Creator and Maintainer

print.smoots *Print Method for the Package 'smoots'*

Description

This function regulates how objects created by the package `smoots` are printed.

Usage

```
## S3 method for class 'smoots'
print(x, ...)
```

Arguments

`x`	an input object of class `smoots`.
`...`	included for compatibility; additional arguments will however not affect the output.

Value

None

Author(s)

- Dominik Schulz (formerly Department of Economics, Paderborn University), Package Creator and Maintainer

rescale *Rescaling Derivative Estimates*

Description

The estimation functions of the `smoots` package estimate the nonparametric trend function or its derivatives on the rescaled time interval $[0, 1]$. With this function the derivative estimates can be rescaled in accordance with a given vector with time points.

Usage

```
rescale(y, x = seq_along(y), v = 1)
```

Arguments

`y`	a numeric vector or matrix with the derivative estimates obtained for time points on the interval $[0, 1]$; pass the list element `ye` of the output of the functions `dsmooth` or `gsmooth` (if the argument `v` > 0) to this argument.
`x`	a numeric vector of length `length(y)` with the actual (equidistant) time points ordered from past to present; the default is `seq_along(y)`.
`v`	the order of derivative that is implemented for `y`; the default is `1`.

Details

The derivative estimation process is based on the additive time series model

$$y_t = m(x_t) + \epsilon_t,$$

where y_t is the observed time series with equidistant design, x_t is the rescaled time on $[0, 1]$, $m(x_t)$ is a smooth and deterministic trend function and ϵ_t are stationary errors with E(eps_[t]) = 0 (see also Beran and Feng, 2002). Since the estimates of the main smoothing functions in `smoots` are obtained with regard to the rescaled time points x_t, the derivative estimates returned by these functions are valid for x_t only. Thus, by passing the returned estimates to the argument `y`, a vector with the actual time points to the argument `x` and the order of derivative of `y` to the argument `v`, a rescaled estimate series is calculated and returned. The function can also be combined with the numeric output of `confBounds`.

Note that the trend estimates, even though they are also obtained for the rescaled time points x_t, are still valid for the actual time points.

Value

A numeric vector with the rescaled derivative estimates is returned.

Author(s)

- Dominik Schulz (formerly Department of Economics, Paderborn University), Package Creator and Maintainer

Examples

```
data <- smoots::gdpUS
Xt <- log(data$GDP)
time <- seq(from = 1947.25, to = 2019.5, by = 0.25)
d_est <- smoots::dsmooth(Xt)
ye_rescale <- smoots::rescale(d_est$ye, x = time, v = 1)
plot(time, ye_rescale, type = "l", main = "", ylab = "", xlab = "Year")
```

`residuals.smoots` *Extract Model Residuals*

Description

Generic function which extracts model residuals from a `smoots` class object. Both `residuals` and its abbreviated form `resid` can be called.

Usage

```
## S3 method for class 'smoots'
residuals(object, ...)
```

Arguments

object	an object from the smoots class.
...	included for consistency with the generic function.

Value

Residuals extracted from a smoots class object.

Author(s)

- Sebastian Letmathe (formerly Department of Economics, Paderborn University),

---

rollCast *Backtesting Semi-ARMA Models with Rolling Forecasts*

Description

A simple backtest of Semi-ARMA models via rolling forecasts can be implemented.

Usage

```
rollCast(
  y,
  p = NULL,
  q = NULL,
  K = 5,
  method = c("norm", "boot"),
  alpha = 0.95,
  np.fcast = c("lin", "const"),
  it = 10000,
  n.start = 1000,
  pb = TRUE,
  cores = max(future::availableCores() - 1, 1),
  argsSmoots = list(),
  plot = TRUE,
  argsPlot = list()
)
```

Arguments

y	a numeric vector that represents the equidistant time series assumed to follow a Semi-ARMA model; must be ordered from past to present.
p	an integer value ≥ 0 that defines the AR order p of the underlying ARMA(p, q) model within X; is set to NULL by default; if no value is passed to p but one is passed to q, p is set to 0; if both p and q are NULL, optimal orders following the BIC for $0 \leq p, q \leq 5$ are chosen; is set to NULL by default; decimal numbers will be rounded off to integers.

q — an integer value ≥ 0 that defines the MA order q of the underlying ARMA(p, q) model within X; is set to NULL by default; if no value is passed to q but one is passed to p, q is set to 0; if both p and q are NULL, optimal orders following the BIC for $0 \leq p, q \leq 5$ are chosen; is set to NULL by default; decimal numbers will be rounded off to integers.

K — a single, positive integer value that defines the number of out-of-sample observations; the last K observations in y are treated as the out-of-sample observations, whereas the rest of the observations in y are the in-sample values.

method — a character object; defines the method used for the calculation of the forecasting intervals; with "norm" the intervals are obtained under the assumption of normally distributed innovations; with "boot" the intervals are obtained via a bootstrap; is set to "norm" by default.

alpha — a numeric vector of length 1 with $0 <$ alpha < 1; the forecasting intervals will be obtained based on the confidence level $(100$alpha$)$-percent; is set to alpha = 0.95 by default, i.e., a 95-percent confidence level.

np.fcast — a character object; defines the forecasting method used for the nonparametric trend; for np.fcast = "lin" the trend is is extrapolated linearly based on the last two trend estimates; for np.fcast = "const", the last trend estimate is used as a constant estimate for future values; is set to *"lin"* by default.

it — an integer that represents the total number of iterations, i.e., the number of simulated series; is set to 10000 by default; only necessary, if method = "boot"; decimal numbers will be rounded off to integers.

n.start — an integer that defines the 'burn-in' number of observations for the simulated ARMA series via bootstrap; is set to 1000 by default; only necessary, if method = "boot";decimal numbers will be rounded off to integers.

pb — a logical value; for pb = TRUE, a progress bar will be shown in the console, if method = "boot".

cores — an integer value >0 that states the number of (logical) cores to use in the bootstrap (or NULL); the default is the maximum number of available cores minus 1 (via future::availableCores); for cores = NULL, parallel computation is disabled.

argsSmoots — a list that contains arguments that will be passed to msmooth for the estimation of the nonparametric trend function; by default, the default values of msmooth are used.

plot — a logical value that controls the graphical output; for the default (plot = TRUE), the original series with the obtained point forecasts as well as the forecasting intervals will be plotted; for plot = FALSE, no plot will be created.

argsPlot — a list; additional arguments for the standard plot function, e.g., xlim, type, ..., can be passed to it; arguments with respect to plotted graphs, e.g., the argument col, only affect the original series y; please note that in accordance with the argument x (lower case) of the standard plot function, an additional numeric vector with time points can be implemented via the argument x (lower case).

Details

Define that an observed, equidistant time series y_t, with $t = 1, 2, ..., n$, follows

$$y_t = m(x_t) + \epsilon_t,$$

where $x_t = t/n$ is the rescaled time on the closed interval $[0, 1]$ and $m(x_t)$ is a nonparametric and deterministic trend function (see Beran and Feng, 2002, and Feng, Gries and Fritz, 2020). ϵ_t, on the other hand, is a stationary process with $E(\epsilon_t) = 0$ and short-range dependence. For the purpose of this function, ϵ_t is assumed to follow an autoregressive-moving-average (ARMA) model with

$$\epsilon_t = \zeta_t + \beta_1\epsilon_{t-1} + ... + \beta_p\epsilon_{t-p} + \alpha_1\zeta_{t-1} + ... + \alpha_q\zeta_{t-q}.$$

Here, the random variables ζ_t are identically and independently distributed (i.i.d.) with zero-mean and a constant variance and the coefficients α_j and β_i, $i = 1, 2, ..., p$ and $j = 1, 2, ..., q$, are real numbers. The combination of both previous formulas will be called a semiparametric ARMA (Semi-ARMA) model.

An explicit forecasting method of Semi-ARMA models is described in `modelCast`. To backtest a selected model, a slightly adjusted procedure is used. The data is divided into in-sample and an out-of-sample values (usually the last $K = 5$ observations in the data are reserved for the out-of-sample observations). A model is fitted to the in-sample data, whereas one-step rolling point forecasts and forecasting intervals are obtained for the out-of-sample time points. The proposed forecasts of the trend are either a linear or a constant extrapolation of the trend with negligible forecasting intervals, whereas the point forecasts of the stationary rest term are obtained via the selected ARMA(p, q) model (see Fritz et al., 2020). The corresponding forecasting intervals are calculated under the assumption that the innovations ζ_t are either normally distributed (see e.g. pp. 93-94 in Brockwell and Davis, 2016) or via a forward bootstrap (see Lu and Wang, 2020). For a one-step forecast for time point t, all observations until time point $t - 1$ are assumed to be known.

The function calculates three important values for backtesting: the number of breaches, i.e. the number of true observations that lie outside of the forecasting intervals, the mean absolute scaled error (MASE, see Hyndman and Koehler, 2006) and the root mean squared scaled error (RMSSE, see Hyndman and Koehler, 2006) are obtained. For the MASE, a value < 1 indicates a better average forecasting potential than a naive forecasting approach. Furthermore, it is independent from the scale of the data and can thus be used to compare forecasts of different datasets. Closely related is the RMSSE, however here, the mean of the squared forecasting errors is computed and scaled by the mean of the squared naive forecasting approach. Then the root of that value is the RMSSE. Due to the close relation, the interpretation of the RMSSE is similarly but not identically to the interpretation of the MASE. Of course, a value close to zero is preferred in both cases.

To make use of the function, a numeric vector with the values of a time series that is assumed to follow a Semi-ARMA model needs to be passed to the argument `y`. Moreover, the arguments `p` and `q` represent the AR and MA orders, respectively, of the underlying ARMA process in the parametric part of the model. If both values are set to `NULL`, an optimal order in accordance with the Bayesian Information Criterion (BIC) will be selected. If only one of the values is `NULL`, it will be changed to zero instead. `K` defines the number of the out-of-sample observations; these will be cut off the end of `y`, while the remaining observations are treated as the in-sample observations. For the K out-of-sample time points, rolling forecasts will be obtained. `method` describes the method to use for the computation of the prediction intervals. Under the normality assumption for the innovations ζ_t, intervals can be obtained via *method* = "norm". However, if the assumption does not hold, a bootstrap can be implemented as well (*method* = *"boot"*). Both approaches are explained in more

detail in normCast and bootCast, respectively. With alpha, the confidence level of the forecasting intervals can be adjusted, as the (100alpha)-percent forecasting intervals will be computed. By means of the argument np.fcast, the forecasting method for the nonparametric trend function can be defined. Selectable are a linear (np.fcast = "lin") and a constant (np.fcast = "const") extrapolation. For more information on these methods, we refer the reader to trendCast.

it, n.start, pb and cores are only relevant for method = "boot". With it the total number of bootstrap iterations is defined, whereas n.start regulates, how many 'burn-in' observations are generated for each simulated ARMA process in the bootstrap. Since a bootstrap may take a longer computation time, with the argument cores the number of cores for parallel computation of the bootstrap iterations can be defined. Nonetheless, for cores = NULL, no cluster is created and therefore the parallel computation is disabled. Note that the bootstrapped results are fully reproducible for all cluster sizes. Moreover, for pb = TRUE, the progress of the bootstrap approach can be observed in the R console via a progress bar. Additional information on these four function arguments can be found in bootCast.

The argument argsSmoots is a list. In this list, different arguments of the function msmooth can be implemented to adjust the estimation of the nonparametric part of the complete model. The arguments of the smoothing function are described in msmooth.

rollCast allows for a quick plot of the results. If the logical argument plot is set to TRUE, a graphic with default settings is created. Nevertheless, users are allowed to implement further arguments of the standard plot function in the list argsPlot. For example, the limits of the plot can be adjusted by xlim and ylim. Furthermore, an argument x can be included in argsPlot with the actual equidistant time points of the whole series (in-sample & out-of-sample observations). Otherwise, simply 1:n is used as the in-sample time points by default.

NOTE:

Within this function, the arima function of the stats package with its method "CSS-ML" is used throughout for the estimation of ARMA models. Furthermore, to increase the performance, C++ code via the Rcpp and RcppArmadillo packages was implemented. Also, the future and future.apply packages are considered for parallel computation of bootstrap iterations. The progress of the bootstrap is shown via the progressr package.

Value

A list with different elements is returned. The elements are as follows.

alpha a single numeric value; it describes, what confidence level (100alpha)-percent has been considered for the forecasting intervals.

breach a logical vector that states whether the K true out-of-sample observations lie outside of the forecasting intervals, respectively; a breach is denoted by TRUE.

breach.val a numeric vector that contains the margin of the breaches (in absolute terms) for the K out-of-sample time points; if a breach did not occur, the respective element is set to zero.

error a numeric vector that contains the simulated empirical values of the forecasting error for method = "boot"; otherwise, it is set to NULL.

fcast.rest a numeric vector that contains the K point forecasts of the parametric part of the model.

fcast.roll a numeric matrix that contains the K rolling point forecasts as well as the values of the bounds of the respective forecasting intervals for the complete model; the first row contains the point forecasts, the lower bounds of the forecasting intervals are in the second row and the upper bounds can be found in the third row.

fcast.trend a numeric vector that contains the K obtained trend forecasts.

K a positive integer; states the number of out-of-sample observations as well as the number of forecasts for the out-of-sample time points.

MASE the obtained value of the mean average scaled error for the selected model.

method a character object that states, whether the forecasting intervals were obtained via a bootstrap (`method = "boot"`) or under the normality assumption for the innovations (`method = "norm"`).

model.nonpar the output (usually a list) of the nonparametric trend estimation via `msmooth`.

model.par the output (usually a list) of the parametric ARMA estimation of the detrended series via `arima`.

n the number of observations (in-sample & out-of-sample observations).

n.in the number of in-sample observations (`n - n.out`).

n.out the number of out-of-sample observations (equals `K`).

np.fcast a character object that states the applied forecasting method for the nonparametric trend function; either a linear (`np.fcast = "lin"`) or a constant `np.fcast = "const"` are possible.

quants a numeric vector of length 2 with the $[100(1-$ `alpha`$)/2]$-percent and $\{100[1-(1-$ `alpha`$)/2]\}$-percent quantiles of the forecasting error distribution.

RMSSE the obtained value of the root mean squared scaled error for the selected model.

y a numeric vector that contains all true observations (in-sample & out-of-sample observations).

y.in a numeric vector that contains all in-sample observations.

y.out a numeric vector that contains the K out-of-sample observations.

Author(s)

- Yuanhua Feng (Department of Economics, Paderborn University),
  Author of the Algorithms

- Dominik Schulz (formerly Department of Economics, Paderborn University),
  Package Creator and Maintainer

References

Beran, J., and Feng, Y. (2002). Local polynomial fitting with long-memory, short-memory and antipersistent errors. Annals of the Institute of Statistical Mathematics, 54, 291-311.

Brockwell, P. J., and Davis, R. A. (2016). Introduction to time series and forecasting, 3rd edition. Springer.

Fritz, M., Forstinger, S., Feng, Y., and Gries, T. (2020). Forecasting economic growth processes for developing economies. Unpublished.

Feng, Y., Gries, T. and Fritz, M. (2020). Data-driven local polynomial for the trend and its derivatives in economic time series. Journal of Nonparametric Statistics, 32:2, 510-533.

Hyndman, R. J., and Koehler, A. B. (2006). Another look at measures of forecast accuracy. International Journal of Forecasting, 22:4, 679-688.

Lu, X., and Wang, L. (2020). Bootstrap prediction interval for ARMA models with unknown orders. REVSTAT–Statistical Journal, 18:3, 375-396.

Examples

```
lgdp <- log(smoots::gdpUS$GDP)
time <- seq(from = 1947.25, to = 2019.5, by = 0.25)
backtest <- rollCast(lgdp, K = 5,
 argsPlot = list(x = time, xlim = c(2012, 2019.5), col = "forestgreen",
 type = "b", pch = 20, lty = 2, main = "Example"))
backtest
```

---

tempNH *Mean Monthly Northern Hemisphere Temperature Changes*

Description

A dataset that contains mean monthly Northern Hemisphere temperature changes from 1880 to 2018. The base period is 1951 to 1980.

Usage

```
tempNH
```

Format

A data frame with 1668 rows and 3 variables:

Year the observation year

Month the observation month

Change the observed mean monthly temperature changes in degrees Celsius

Source

The data was obtained from the Goddard Institute for Space Studies (part of the National Aeronautics and Space Administration [NASA]) (accessed: 2019-09-25).

https://data.giss.nasa.gov/gistemp/tabledata_v4/NH.Ts+dSST.txt

trendCast *Forecasting Function for Nonparametric Trend Functions*

Description

Forecasting Function for Nonparametric Trend Functions

Usage

```
trendCast(object, h = 1, np.fcast = c("lin", "const"), plot = FALSE, ...)
```

Arguments

object	an object returned by either msmooth, tsmooth, gsmooth (with v = 0) or knsmooth.
h	the forecasting horizon; the values $m(n+1)$ to $m(n+h)$ will be predicted; is set to h = 1 by default; decimal numbers will be rounded off to integers.
np.fcast	the forecasting method; np.fcast = "lin" uses a linear extrapolation, whereas np.fcast = "const" uses the last fitted value of $m(x_t)$ as a forecast; is set to "lin" by default.
plot	a logical value; if set to TRUE, a simple plot of the original time series, the local polynomial trend estimates as well as the predicted values is generated.
...	additional arguments for the standard plot function, e.g., xlim, type, ... ; arguments with respect to plotted graphs, e.g., the argument col, only affect the original series X; please note that in accordance with the argument x (lower case) of the standard plot function, an additional numeric vector with time points can be implemented via the argument x (lower case). x should be valid for the sample observations only, i.e. length(x) == length(obj$orig) should be TRUE, as future time points will be calculated automatically.

Details

This function is part of the smoots package and was implemented under version 1.1.0. The underlying theory is based on the additive nonparametric regression function

$$y_t = m(x_t) + \epsilon_t,$$

where y_t is the observed time series with equidistant design, x_t is the rescaled time on the interval $[0, 1]$, $m(x_t)$ is a smooth and deterministic trend function and ϵ_t are stationary errors with $E(\epsilon_t) = 0$.

The purpose of this function is the forecasting of future values based on a nonparametric regression model. Following the proposition in Fritz et al. (2020), point predictions can be conducted separately for the nonparametric trend function $m(x_t)$ and the stationary part ϵ_t. The sum of both forecasts is then the forecast of y_t. With this function, only the forecast with respect to $m(x_t)$ is computable. Now assume that the variance of the error in the local polynomial forecasts is negligible when calculating the forecasting intervals. We define the forecast for time point $n + k$, $k = 1, 2, ..., h$, by

$$\hat{m}(x_{n+k}) = \hat{m}(x_n) + Dk\delta_m,$$

where δ_m is equal to $\hat{m}(x_n) - \hat{m}(x_{n-1})$ and D is a dummy variable. If $D = 1$, a linear extrapolation is applied. For $D = 0$, $\hat{m}(x_n)$ is the predicted value.

To make use of this function, an object of class `smoots` can be given as input. However, since the discussed approach is only valid for the estimated trend function, only objects created by `msmooth`, `tsmooth`, `knsmooth` and `link{gsmooth}`, if the trend was estimated, will be appropriate input objects.

With the input argument `h`, a positive integer can be given to the function that represents the forecasting horizon, i.e. how many future values are to be estimated. Via the argument `np.fcast` the value of the dummy variable D can be specified and thus the forecasting method. For `np.fcast = "lin"`, $D = 1$ is applied, whereas for `np.fcast = "const"`, D is set to 0.

By means of the argument `plot` that can be either set to the logical values `TRUE` or `FALSE`, a simple plot of the original series alongside the local polynomial estimates as well as the forecasted values can be either generated or suppressed.

The function always returns a vector of forecasted values ordered from $n + 1$ to $n + h$. Depending on the setting of the argument `plot`, a generic plot of the results may be generated. Furthermore, additional arguments of the standard plot function can be passed to this function as well to adjust the generated plot.

Value

A numeric vector is always returned with the forecasted values. Depending on the setting for the argument *plot*, a generic plot might be created.

Author(s)

- Yuanhua Feng (Department of Economics, Paderborn University),
  Author of the Algorithms

- Dominik Schulz (formerly Department of Economics, Paderborn University),
  Package Creator and Maintainer

References

Feng, Y., Gries, T. and Fritz, M. (2020). Data-driven local polynomial for the trend and its derivatives in economic time series. Journal of Nonparametric Statistics, 32:2, 510-533.

Feng, Y., Gries, T., Letmathe, S. and Schulz, D. (2019). The smoots package in R for semiparametric modeling of trend stationary time series. Discussion Paper. Paderborn University. Unpublished.

Feng, Y., Gries, T., Fritz, M., Letmathe, S. and Schulz, D. (2020). Diagnosing the trend and bootstrapping the forecasting intervals using a semiparametric ARMA. Discussion Paper. Paderborn University. Unpublished.

Fritz, M., Forstinger, S., Feng, Y., and Gries, T. (2020). Forecasting economic growth processes for developing economies. Unpublished.

Examples

```
log_gdp <- log(smoots::gdpUS$GDP)
est <- msmooth(log_gdp)
```

```
forecasts <- trendCast(est, h = 5, plot = TRUE)
forecasts
```

`tsmooth` *Advanced Data-driven Nonparametric Regression for the Trend in Equidistant Time Series*

Description

This function runs an iterative plug-in algorithm to find the optimal bandwidth for the estimation of the nonparametric trend in equidistant time series (with short-memory errors) and then employs the resulting bandwidth via either local polynomial or kernel regression. This function allows for more flexibility in its arguments than *msmooth*.

Usage

```
tsmooth(
  y,
  p = c(1, 3),
  mu = c(0, 1, 2, 3),
  Mcf = c("NP", "ARMA", "AR", "MA"),
  InfR = c("Opt", "Nai", "Var"),
  bStart = 0.15,
  bvc = c("Y", "N"),
  bb = c(0, 1),
  cb = 0.05,
  method = c("lpr", "kr"),
  nar_lim = c(0, 1),
  nma_lim = c(0, 1),
  arma_mean = TRUE
)
```

Arguments

`y`	a numeric vector that contains the time series ordered from past to present.
`p`	an integer `1` (local linear regression) or `3` (local cubic regression); represents the order of polynomial within the local polynomial regression (see also the 'Details' section); is set to `1` by default; is automatically set to `1` if `method = "kr"`.
`mu`	an integer `0`, ..., `3` that represents the smoothness parameter of the kernel weighting function and thus defines the kernel function that will be used within the local polynomial regression; is set to `1` by default.

Number	Kernel
`0`	Uniform Kernel

1	Epanechnikov Kernel
2	Bisquare Kernel
3	Triweight Kernel

`Mcf` method for estimating the variance factor c_f by the IPI (see also the 'Details' section); is set to `NP` by default.

Method	Explanation
`"NP"`	Nonparametric estimation using the Bartlett window
`"ARMA"`	Estimation on the assumption that the residuals follow an ARMA model
`"AR"`	Estimation on the assumption that the residuals follow an AR model
`"MA"`	Estimation on the assumption that the residuals follow an MA model

`InfR` a character object that represents the inflation rate in the form $h_d = h^a$ for the bandwidth in the estimation of $I[m^{(k)}]$ (see also the 'Details' section); is set to `"Opt"` by default.

Inflation rate	Description
`"Opt"`	Optimal inflation rate $a_{p,O}$ ($5/7$ for `p = 1`; $9/11$ for `p = 3`)
`"Nai"`	Naive inflation rate $a_{p,N}$ ($5/9$ for `p = 1`; $9/13$ for `p = 3`)
`"Var"`	Stable inflation rate $a_{p,S}$ ($1/2$ for `p = 1` and `p = 3`)

`bStart` a numeric object that indicates the starting value of the bandwidth for the iterative process; should be > 0; is set to `0.15` by default.

`bvc` a character object that indicates whether an enlarged bandwidth is being used for the estimation of the variance factor c_f (see also the 'Details' section) or not; is set to `"Y"` by default.

Bandwidth	Description
"Y"	Using an enlarged bandwidth
"N"	Using a bandwidth without enlargement

`bb` can be set to `0` or `1`; the parameter controlling the bandwidth used at the boundary; is set to `1` by default.

Number (`bb`)	Estimation procedure at boundary points
`0`	Fixed bandwidth on one side with possible large bandwidth on the other side at the boundary
`1`	The k-nearest neighbor method will be used

`cb` a numeric value that indicates the percentage of omitted observations on each side of the observation period for the automated bandwidth selection; is set to `0.05` by default.

`method` the final smoothing approach; `"lpr"` represents the local polynomial regression, whereas `"kr"` implements a kernel regression; is set to `"lpr"` by default.

`nar_lim` a two-element numeric vector with the minimum and the maximum AR order to check via BIC in the fitting of ARMA models, when the variance factor is obtained via fitted parametric ARMA models and not nonparametrically.

`nma_lim` a two-element numeric vector with the minimum and the maximum MA order to check via BIC in the fitting of ARMA models, when the variance factor is obtained via fitted parametric ARMA models and not nonparametrically.

`arma_mean` logical; whether to include the mean in the fitting of ARMA models, only if the variance factor is estimated via parametric ARMA models and not nonparametrically.

Details

The trend is estimated based on the additive nonparametric regression model for an equidistant time series

$$y_t = m(x_t) + \epsilon_t,$$

where y_t is the observed time series, x_t is the rescaled time on the interval $[0, 1]$, $m(x_t)$ is a smooth and deterministic trend function and ϵ_t are stationary errors with $E(\epsilon_t) = 0$ and short-range dependence (see also Beran and Feng, 2002). With this function $m(x_t)$ can be estimated without a parametric model assumption for the error series. Thus, after estimating and removing the trend, any suitable parametric model, e.g. an ARMA(p, q) model, can be fitted to the residuals (see `arima`).

The iterative-plug-in (IPI) algorithm, which numerically minimizes the Asymptotic Mean Squared Error (AMISE), was proposed by Feng, Gries and Fritz (2020).

Define $I[m^{(k)}] = \int_{c_b}^{d_b} [m^{(k)}(x)]^2 dx$, $\beta_{(\nu,k)} = \int_{-1}^{1} u^k K_{(\nu,k)}(u) du$ and $R(K) = \int_{-1}^{1} K_{(\nu,k)}^2(u) du$, where p is the order of the polynomial, $k = p + 1$ is the order of the asymptotically equivalent kernel, ν is the order of the trend function's derivative, $0 \leq c_b < d_b \leq 1$, c_f is the variance factor and $K_{(\nu,k)}(u)$ the k-th order equivalent kernel obtained for the estimation of $m^{(\nu)}$ in the interior. $m^{(\nu)}$ is the ν-th order derivative ($\nu = 0, 1, 2, ...$) of the nonparametric trend.

Furthermore, we define

$$C_1 = \frac{I[m^{(k)}]\beta_{(\nu,k)}^2}{(k!)^2}$$

and

$$C_2 = \frac{2\pi c_f (d_b - c_b) R(K)}{n h^{2\nu+1}}$$

with h being the bandwidth and n being the number of observations. The AMISE is then

$$AMISE(h) = h^{2(k-\nu)} C_1 + C_2.$$

The function calculates suitable estimates for c_f, the variance factor, and $I[m^{(k)}]$ over different iterations. In each iteration, a bandwidth is obtained in accordance with the AMISE that once more serves as an input for the following iteration. The process repeats until either convergence or the 40th iteration is reached. For further details on the asymptotic theory or the algorithm, please consult Feng, Gries and Fritz (2020) or Feng et al. (2022).

To apply the function, more arguments are needed compared to the similar function `msmooth`: a data input `y`, an order of polynomial `p`, a kernel weighting function defined by the smoothness parameter `mu`, a variance factor estimation method `Mcf`, an inflation rate setting `InfR` (see also Beran and Feng, 2002), a starting value for the relative bandwidth `bStart`, an inflation setting for the variance factor estimation `bvc`, a boundary method `bb`, a boundary cut-off percentage `cb` and a final smoothing method `method`. In fact, aside from the input vector `y`, every argument has a default setting that can be adjusted for the individual case. Theoretically, the initial bandwidth does not affect the selected

optimal bandwidth. However, in practice local minima of the AMISE might exist and influence the selected bandwidth. Therefore, the default setting is `bStart = 0.15`, which is a compromise between the starting values `bStart = 0.1` for `p = 1` and `bStart = 0.2` for `p = 3` that were proposed by Feng, Gries and Fritz (2020). In the rare case of a clearly unsuitable optimal bandwidth, a starting bandwidth that differs from the default value is a first possible approach to obtain a better result. Other argument adjustments can be tried as well. For more specific information on the input arguments consult the section *Arguments*.

When applying the function, an optimal bandwidth is obtained based on the IPI algorithm proposed by Feng, Gries and Fritz (2020). In a second step, the nonparametric trend of the series is calculated with respect to the chosen bandwidth and the selected regression method (`lpf` or `kr`). Please note that `method = "lpf"` is strongly recommended by the authors. Moreover, it is notable that `p` is automatically set to `1` for `method = "kr"`. The output object is then a list that contains, among other components, the original time series, the estimated trend values and the series without the trend.

The default print method for this function delivers only key numbers such as the iteration steps and the generated optimal bandwidth rounded to the fourth decimal. The exact numbers and results such as the estimated nonparametric trend series are saved within the output object and can be addressed via the `$` sign.

NOTE:

With package version 1.1.0, this function implements C++ code by means of the `Rcpp` and `RcppArmadillo` packages for better performance.

Value

The function returns a list with different components:

AR.BIC the Bayesian Information Criterion of the optimal AR(p) model when estimating the variance factor via autoregressive models (if calculated; calculated for `alg = "OA"` and `alg = "NA"`).

ARMA.BIC the Bayesian Information Criterion of the optimal ARMA(p, q) model when estimating the variance factor via autoregressive-moving-average models (if calculated; calculated for `alg = "OAM"` and `alg = "NAM"`).

cb the percentage of omitted observations on each side of the observation period.

b0 the optimal bandwidth chosen by the IPI-algorithm.

bb the boundary bandwidth method used within the IPI; always equal to 1.

bStart the starting value of the (relative) bandwidth; input argument.

bvc indicates whether an enlarged bandwidth was used for the variance factor estimation or not; depends on the chosen algorithm.

cf0 the estimated variance factor; in contrast to the definitions given in the *Details* section, this object actually contains an estimated value of $2\pi c_f$, i.e. it corresponds to the estimated sum of autocovariances.

cf0.AR the estimated variance factor obtained by estimation of autoregressive models (if calculated; `alg = "OA"` or `"NA"`).

cf0.ARMA the estimated variance factor obtained by estimation of autoregressive-moving-average models (if calculated; calculated for `alg = "OAM"` and `alg = "NAM"`).

cf0.LW the estimated variance factor obtained by Lag-Window Spectral Density Estimation following Bühlmann (1996) (if calculated; calculated for algorithms `"A"`, `"B"`, `"O"` and `"N"`).

cf0.MA the estimated variance factor obtained by estimation of moving-average models (if calculated; calculated for `alg = "OM"` and `alg = "NM"`).

I2 the estimated value of $I[m^{(k)}]$.

InfR the setting for the inflation rate according to the chosen algorithm.

iterations the bandwidths of the single iterations steps

L0.opt the optimal bandwidth for the lag-window spectral density estimation (if calculated; calculated for algorithms `"A"`, `"B"`, `"O"` and `"N"`).

MA.BIC the Bayesian Information Criterion of the optimal MA(q) model when estimating the variance factor via moving-average models (if calculated; calculated for `alg = "OM"` and `alg = "NM"`).

Mcf the estimation method for the variance factor estimation; depends on the chosen algorithm.

mu the smoothness parameter of the second order kernel; input argument.

n the number of observations.

niterations the total number of iterations until convergence.

orig the original input series; input argument.

p.BIC the order p of the optimal AR(p) or ARMA(p, q) model when estimating the variance factor via autoregressive or autoregressive-moving average models (if calculated; calculated for `alg = "OA"`, `alg = "NA"`, `alg = "OAM"` and `alg = "NAM"`).

p the order of polynomial used in the IPI-algorithm; also used for the final smoothing, if `method = "lpr"`; input argument.

q.BIC the order q of the optimal MA(q) or ARMA(p, q) model when estimating the variance factor via moving-average or autoregressive-moving average models (if calculated; calculated for `alg = "OM"`, `alg = "NM"`, `alg = "OAM"` and `alg = "NAM"`).

res the estimated residual series.

v the considered order of derivative of the trend; is always zero for this function.

ws the weighting system matrix used within the local polynomial regression; this matrix is a condensed version of a complete weighting system matrix; in each row of `ws`, the weights for conducting the smoothing procedure at a specific observation time point can be found; the first $[nb + 0.5]$ rows, where n corresponds to the number of observations, b is the bandwidth considered for smoothing and $[.]$ denotes the integer part, contain the weights at the $[nb + 0.5]$ left-hand boundary points; the weights in row $[nb + 0.5] + 1$ are representative for the estimation at all interior points and the remaining rows contain the weights for the right-hand boundary points; each row has exactly $2[nb + 0.5] + 1$ elements, more specifically the weights for observations of the nearest $2[nb + 0.5] + 1$ time points; moreover, the weights are normalized, i.e. the weights are obtained under consideration of the time points $x_t = t/n$, where $t = 1, 2, ..., n$.

ye the nonparametric estimates of the trend.

Author(s)

- Yuanhua Feng (Department of Economics, Paderborn University),
  Author of the Algorithms

- Dominik Schulz (formerly Department of Economics, Paderborn University),
  Package Creator and Maintainer

References

Beran, J. and Feng, Y. (2002). Local polynomial fitting with long-memory, short-memory and antipersistent errors. Annals of the Institute of Statistical Mathematics, 54(2), 291-311.

Bühlmann, P. (1996). Locally adaptive lag-window spectral estimation. Journal of Time Series Analysis, 17(3), 247-270.

Feng, Y., Gries, T. and Fritz, M. (2020). Data-driven local polynomial for the trend and its derivatives in economic time series. Journal of Nonparametric Statistics, 32:2, 510-533.

Feng, Y., Gries, T., Letmathe, S. and Schulz, D. (2022). The smoots package in R for semiparametric modeling of trend stationary time series. The R Journal, 14:1, 182-195.

Examples

```
### Example 1: US-GDP ###

# Logarithm of test data
# -> the logarithm of the data is assumed to follow the additive model
test_data <- gdpUS
y <- log(test_data$GDP)

# Applied tsmooth function for the trend
result <- tsmooth(y, p = 1, mu = 1, Mcf = "NP", InfR = "Opt",
 bStart = 0.1, bvc = "Y")
trend1 <- result$ye

# Plot of the results
t <- seq(from = 1947, to = 2019.25, by = 0.25)
plot(t, y, type = "l", xlab = "Year", ylab = "log(US-GDP)", bty = "n",
 lwd = 1, lty = 3,
 main = "Estimated trend for log-quarterly US-GDP, Q1 1947 - Q2 2019")
points(t, trend1, type = "l", col = "red", lwd = 1)
title(sub = expression(italic("Figure 1")), col.sub = "gray47",
 cex.sub = 0.6, adj = 0)
result

## Not run:
### Example 2: German Stock Index ###

# The following procedure can be considered, if (log-)returns are assumed
# to follow a model from the general class of semiparametric GARCH-type
# models (including Semi-GARCH, Semi-Log-GARCH and Semi-APARCH models among
# others) with a slowly changing variance over time due to a deterministic,
# nonparametric scale function.
```

```
# Obtain the logarithm of the squared returns
returns <- diff(log(dax$Close))   # (log-)returns
rt <- returns - mean(returns)     # demeaned (log-)returns
yt <- log(rt^2)                   # logarithm of the squared returns

# Apply 'smoots' function to the log-data, because the logarithm of
# the squared returns follows an additive model with a nonparametric trend
# function, if the returns are assumed to follow a semiparametric GARCH-type
# model.

# In this case, the optimal inflation rate is used for p = 3.
est <- tsmooth(yt, p = 3, InfR = "Opt")
m_xt <- est$ye                    # estimated trend values

# Obtain the standardized returns 'eps' and the scale function 'scale.f'
res <- est$res                    # the detrended log-data
C <- -log(mean(exp(res)))         # an estimate of a constant value needed
                                  # for the retransformation
scale.f <- exp((m_xt - C) / 2)    # estimated values of the scale function in
                                  # the returns
eps <- rt / scale.f               # the estimated standardized returns

# -> 'eps' can now be analyzed by any suitable GARCH-type model.
#    The total volatilities are then the product of the conditional
#    volatilities obtained from 'eps' and the scale function 'scale.f'.

## End(Not run)
```

---

vix *CBOE Volatility Index (VIX) Financial Time Series Data*

Description

A dataset that contains the daily financial data of the VIX from 1990 to July 2019 (currency in USD).

Usage

```
vix
```

Format

A data frame with 7452 rows and 9 variables:

Year the observation year

Month the observation month

Day the observation day

Open the opening price of the day

High the highest price of the day

Low the lowest price of the day

Close the closing price of the day

AdjClose the adjusted closing price of the day

Volume the traded volume

Source

The data was obtained from Yahoo Finance (accessed: 2019-08-22).

https://query1.finance.yahoo.com/v7/finance/download/^VIX?period1=631148400&period2=1564524000&interval=1d&events=history&crumb=Iaq1EPZAQRb

Index

∗ **datasets**
dax, 14
gdpUS, 19
tempNH, 48
vix, 57

arima, *8*, *13*, *28*, *31*, *37*, *46*, *47*, *53*

bootCast, *4*, 5, *28*, *46*

confBounds, *4*, 9, *42*
critMatrix, *3*, 12, *39*

dax, *4*, 14
dsmooth, *3*, *10*, *11*, 15, *41*

fitted.smoots, 19
future, *8*, *28*, *46*
future.apply, *8*, *28*, *46*
future::availableCores, *6*, *26*, *44*

gdpUS, *4*, 19
gsmooth, *3*, 20, *41*, *49*

knsmooth, *3*, 23, *49*, *50*

min, *39*
modelCast, *4*, 25, *45*
msmooth, *3*, *10*, *11*, *15*, 29, *44*, *46*, *47*, *49*, *50*, *53*

normCast, *4*, *28*, 35, *46*

optOrd, *3*, 38

plot.smoots, 40
print.smoots, 41
progressr, *8*, *28*, *46*

Rcpp, *8*, *11*, *17*, *21*, *28*, *32*, *46*, *54*
RcppArmadillo, *8*, *11*, *17*, *21*, *28*, *32*, *46*, *54*
rescale, *3*, *11*, *17*, *21*, 41
residuals.smoots, 42
rollCast, 43

smoots *(*smoots-package*)*, 3
smoots-package, 3

tempNH, *4*, 48
trendCast, *4*, *46*, 49
tsmooth, *3*, *10*, *11*, *31*, *49*, *50*, 51

vix, *4*, 57