Package 'forecastSNSTS'

July 22, 2025

Title Forecasting for Stationary and Non-Stationary Time Series

Version 1.3-0

Description Methods to compute linear h-step ahead prediction coefficients based on localised and iterated Yule-Walker estimates and empirical mean squared and absolute prediction errors for the resulting predictors. Also, functions to compute autocovariances for AR(p) processes, to simulate tvARMA(p,q) time series, and to verify an assumption from Kley et al. (2019), Electronic of Statistics, forthcoming. Preprint <doi:10.48550/arXiv.1611.04460>.

Depends R (>= 3.2.3)

License GPL (>= 2)

URL http://github.com/tobiaskley/forecastSNSTS

BugReports http://github.com/tobiaskley/forecastSNSTS/issues

Encoding UTF-8

LazyData true

LinkingTo Rcpp

Imports Rcpp

Collate 'RcppExports.R' 'acfARp.R' 'f.R' 'forecastSNSTS-package.R' 'measure-of-accuracy.R' 'models.R'

RoxygenNote 6.1.1

Suggests testthat

NeedsCompilation yes

Author Tobias Kley [aut, cre],
Philip Preuss [aut],
Piotr Fryzlewicz [aut]

Maintainer Tobias Kley <tobias.kley@bristol.ac.uk>

Repository CRAN

Date/Publication 2019-09-02 15:20:05 UTC

Contents

`forecastSNSTS-package` *Forecasting of Stationary and Non-Stationary Time Series*

Description

Methods to compute linear h-step ahead prediction coefficients based on localised and iterated Yule-Walker estimates and empirical mean squared and absolute prediction errors for the resulting predictors. Also, functions to compute autocovariances for AR(p) processes, to simulate tvARMA(p,q) time series, and to verify an assumption from Kley et al. (2019).

Details

Package:	forecastSNSTS
Type:	Package
Version:	1.3-0
Date:	2019-09-02
License:	GPL (>= 2)

Contents

The core functionality of this R package is accessable via the function `predCoef`, which is used to compute the linear prediction coefficients, and the functions `MSPE` and `MAPE`, which are used to compute the empirical mean squared or absolute prediction errors. Further, the function `f` can be used to verify condition (10) of Theorem 3.1 in Kley et al. (2019) for any given tvAR(p) model. The function `tvARMA` can be used to simulate time-varying ARMA(p,q) time series. The function `acfARp` computes the autocovariances of a AR(p) process from the coefficients and innovations standard deviation.

Author(s)

Tobias Kley

References

Kley, T., Preuss, P. & Fryzlewicz, P. (2019). Predictive, finite-sample model choice for time series under stationarity and non-stationarity. Electronic Journal of Statistics, forthcoming. [cf. https://arxiv.org/abs/1611.04460]

---

acfARp *Compute autocovariances of an AR(p) process*

Description

This functions returns the autocovariances $Cov(X_{t-k}, X_t)$ of a stationary time series (Y_t) that fulfills the following equation:

$$Y_t = \sum_{j=1}^{p} a_j Y_{t-j} + \sigma \varepsilon_t,$$

where $\sigma > 0$, ε_t is white noise and $a_1, \ldots, a_p$ are real numbers satisfying that the roots z_0 of the polynomial $1 - \sum_{j=1}^{p} a_j z^j$ lie strictly outside the unit circle.

Usage

```
acfARp(a = NULL, sigma, k)
```

Arguments

a	vector $(a_1, \ldots, a_p)$ of coefficients; default NULL, corresponding to p = 0, white noise with variance σ^2,
sigma	standard deviation of ε_t; default 1,
k	lag for which to compute the autocovariances.

Value

Returns autocovariance at lag k of the AR(p) process.

Examples

```
## Taken from Section 6 in Dahlhaus (1997, AoS)
a1 <- function(u) {1.8 * cos(1.5 - cos(4*pi*u))}
a2 <- function(u) {-0.81}
# local autocovariance for u === 1/2: lag 1
acfARp(a = c(a1(1/2), a2(1/2)), sigma = 1, k = 1)
# local autocovariance for u === 1/2: lag -2
acfARp(a = c(a1(1/2), a2(1/2)), sigma = 1, k = -1)
# local autocovariance for u === 1/2: the variance
acfARp(a = c(a1(1/2), a2(1/2)), sigma = 1, k = 0)
```

computeMSPEcpp *Mean Squared Prediction Errors, for a single h*

Description

This function computes the estimated mean squared prediction errors from a given time series and prediction coefficients

Arguments

`X`	the data
`coef`	the array of coefficients.
`h`	which lead time to compute the MSPE for
`t`	a vector of times from which backward the forecasts are computed
`type`	indicating what type of measure of accuracy is to be computed; 1: mspe, 2: msae
`trimLo`	percentage of lower observations to be trimmed away
`trimUp`	percentage of upper observations to be trimmed away

Details

The array of prediction coefficients `coef` is expected to be of dimension `P` x `P` x `H` x `length(N)` x `length(t)` and in the format as it is returned by the function `predCoef`. More precisely, for $p = 1, \ldots, P$ and the `j.N`th element of `N` element of `N` the coefficient of the `h`-step ahead predictor for X_{i+h} which is computed from the observations $X_i, \ldots, X_{i-p+1}$ has to be available via `coef[p, 1:p, h, j.N, t==i]`.

Note that `t` have to be the indices corresponding to the coefficients.

The resulting mean squared prediction error

$$\frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} (X_{t+h} - (X_t, \ldots, X_{t-p+1}) \hat{v}_{N[j.N],T}^{(p,h)}(t))^2$$

is then stored in the resulting matrix at position `(p, j.N)`.

Value

Returns a `P` x `length(N)` matrix with the results.

`f` *Compute $f(\delta)$ for a tvAR(p) process*

Description

This functions computes the quantity $f(\delta)$ defined in (24) of Kley et al. (2019) when the underlying process follows an tvAR(p) process. Recall that, to apply Theorem 3.1 in Kley et al. (2019), the function $f(\delta)$ is required to be positive, which can be verified with the numbers returned from this function. The function returns a vector with elements $f(\delta)$ for each δ in `which.deltas`, with $f(\delta)$ defined as

$$f(\delta) := \min_{p_1,p_2=0,\dots,p_{\max}} \min_{N \in \mathcal{N}} \Big| \mathrm{MSPE}^{(p_1,h)}_{s_1/T,m/T}(\frac{s_1}{T}) - (1+\delta) \cdot \mathrm{MSPE}^{(p_2,h)}_{N/T,m/T}(\frac{s_1}{T}) \Big|, \quad \delta \geq 0$$

where $T, m, p_{\max}, h$ are positive integers, $\mathcal{N} \subset \{p_{\max}+1, \dots, T-m-h\}$, and $s_1 := T-m-h+1$.

Usage

```
f(which.deltas, p_max, h, T, Ns, m, a, sigma)
```

Arguments

`which.deltas`	vector containing the δ's for which to to compute $f(\delta)$,
`p_max`	parameter $p_{\max}$,
`h`	parameter h,
`T`	parameter T,
`Ns`	a vector containing the elements of the set $\mathcal{N}$,
`m`	parameter m,
`a`	a list of real-valued functions, specifying the coefficients of the tvAR(p) process,
`sigma`	a positive-valued function, specifying the variance of the innovations of the tvAR(p) process,

Details

The function $\mathrm{MSPE}^{(p,h)}_{\Delta_1,\Delta_2}(u)$ is defined, for real-valued u and $\Delta_1, \Delta_2 \geq 0$, in terms of the second order properties of the process:

$$\mathrm{MSPE}^{(p,h)}_{\Delta_1,\Delta_2}(u) := \int_0^1 g^{(p,h)}_{\Delta_1}\Big(u + \Delta_2(1-x)\Big)\mathrm{d}x,$$

with $g^{(0,h)}_{\Delta}(u) := \gamma_0(u)$ and, for $p = 1, 2, \dots$,

$$g^{(p,h)}_{\Delta}(u) := \gamma_0(u) - 2\big(v^{(p,h)}_{\Delta}(u)\big)'\gamma^{(p,h)}_0(u) + \big(v^{(p,h)}_{\Delta}(u)\big)'\Gamma^{(p)}_0(u)v^{(p,h)}_{\Delta}(u)$$

$$\gamma^{(p,h)}_0(u) := \big(\gamma_h(u), \dots, \gamma_{h+p-1}(u)\big)',$$

where

$$v_{\Delta}^{(p,h)}(u) := e_1' \big(e_1 \big(a_{\Delta}^{(p)}(t)\big)' + H\big)^h,$$

with e_1 and H defined in the documentation of predCoef and, for every real-valued u and $\Delta \geq 0$,

$$a_{\Delta}^{(p)}(u) := \Gamma_{\Delta}^{(p)}(u)^{-1} \gamma_{\Delta}^{(p)}(u),$$

where

$$\gamma_{\Delta}^{(p)}(u) := \int_0^1 \gamma^{(p)}(u + \Delta(x-1)) \mathrm{d}x, \quad \gamma^{(p)}(u) := [\gamma_1(u) \ \ldots \ \gamma_p(u)]',$$

$$\Gamma_{\Delta}^{(p)}(u) := \int_0^1 \Gamma^{(p)}(u + \Delta(x-1)) \mathrm{d}x, \quad \Gamma^{(p)}(u) := (\gamma_{i-j}(u);\, i, j = 1, \ldots, p).$$

The local autocovariances $\gamma_k(u)$ are defined as the lag-k autocovariances of an AR(p) process which has coefficients $a_1(u), \ldots, a_p(u)$ and innovations with variance $\sigma(u)^2$, because the underlying model is assumed to be tvAR(p)

$$Y_{t,T} = \sum_{j=1}^{p} a_j(t/T) Y_{t-j,T} + \sigma(t/T) \varepsilon_t,$$

where $a_1, \ldots, a_p$ are real valued functions (defined on $[0, 1]$) and σ is a positive function (defined on $[0, 1]$).

Value

Returns a vector with the values $f(\delta)$, as defined in (24) of Kley et al. (2019), where it is now denoted by $q(\delta)$, for each δ in `which.delta`.

Examples

```
## Not run:
## because computation is quite time-consuming.
n <- 100
a <- list( function(u) {return(0.8+0.19*sin(4*pi*u))} )
sigma <- function (u) {return(1)}

Ns <- seq( floor((n/2)^(4/5)), floor(n^(4/5)),
           ceiling((floor(n^(4/5)) - floor((n/2)^(4/5)))/25) )
which.deltas <- c(0, 0.01, 0.05, 0.1, 0.15, 0.2, 0.4, 0.6)
P_max <- 7
H <- 1
m <- floor(n^(.85)/4)

# now replicate some results from Table 4 in Kley et al. (2019)
f( which.deltas, P_max, h = 1, n - m, Ns, m, a, sigma )
f( which.deltas, P_max, h = 5, n - m, Ns, m, a, sigma )

## End(Not run)
```

measure-of-accuracy *Mean squared or absolute h-step ahead prediction errors*

Description

The function MSPE computes the empirical mean squared prediction errors for a collection of h-step ahead, linear predictors ($h = 1, \dots, H$) of observations X_{t+h}, where $m_1 \leq t + h \leq m_2$, for two indices m_1 and m_2. The resulting array provides

$$\frac{1}{m_{\mathrm{lo}} - m_{\mathrm{up}} + 1} \sum_{t=m_{\mathrm{lo}}}^{m_{\mathrm{up}}} R_{(t)}^2,$$

with $R_{(t)}$ being the prediction errors

$$R_t := |X_{t+h} - (X_t, \dots, X_{t-p+1}) \hat{v}_{N,T}^{(p,h)}(t)|,$$

ordered by magnitude; i.e., they are such that $R_{(t)} \leq R_{(t+1)}$. The lower and upper limits of the indices are $m_{\mathrm{lo}} := m_1 - h + \lfloor (m_2 - m_1 + 1)\alpha_1 \rfloor$ and $m_{\mathrm{up}} := m_2 - h - \lfloor (m_2 - m_1 + 1)\alpha_2 \rfloor$. The function MAPE computes the empirical mean absolute prediction errors

$$\frac{1}{m_{\mathrm{lo}} - m_{\mathrm{up}} + 1} \sum_{t=m_{\mathrm{lo}}}^{m_{\mathrm{up}}} R_{(t)},$$

with m_{lo}, m_{up} and $R_{(t)}$ defined as before.

Usage

```
MSPE(X, predcoef, m1 = length(X)/10, m2 = length(X), P = 1, H = 1,
  N = c(0, seq(P + 1, m1 - H + 1)), trimLo = 0, trimUp = 0)

MAPE(X, predcoef, m1 = length(X)/10, m2 = length(X), P = 1, H = 1,
  N = c(0, seq(P + 1, m1 - H + 1)), trimLo = 0, trimUp = 0)
```

Arguments

X	the data $X_1, \dots, X_T$
predcoef	the prediction coefficients in form of a list of an array coef, and two integer vectors t and N. The two integer vectors provide the information for which indices t and segment lengths N the coefficients are to be interpreted; (m1-H):(m2-1) has to be a subset of predcoef$t. if not provided the necessary coefficients will be computed using predCoef.
m1	first index from the set in which the indices $t + h$ shall lie
m2	last index from the set in which the indices $t + h$ shall lie
P	maximum order of prediction coefficients to be used; must not be larger than dim(predcoef$coef)[1].

`H`	maximum lead time to be used; must not be larger than `dim(predcoef$coef)[3]`.
`N`	vector with the segment sizes to be used, 0 corresponds to using 1, ..., t; has to be a subset of predcoef$N.
`trimLo`	percentage α_1 of lower observations to be trimmed away
`trimUp`	percentage α_2 of upper observations to be trimmed away

Value

`MSPE` returns an object of type `MSPE` that has `mspe`, an array of size `H`×`P`×`length(N)`, as an attribute, as well as the parameters `N`, `m1`, `m2`, `P`, and `H`. `MAPE` analogously returns an object of type `MAPE` that has `mape` and the same parameters as attributes.

Examples

```
T <- 1000
X <- rnorm(T)
P <- 5
H <- 1
m <- 20
Nmin <- 20
pcoef <- predCoef(X, P, H, (T - m - H + 1):T, c(0, seq(Nmin, T - m - H, 1)))

mspe <- MSPE(X, pcoef, 991, 1000, 3, 1, c(0, Nmin:(T-m-H)))

plot(mspe, vr = 1, Nmin = Nmin)
```

---

plot.measure-of-accuracy

Plot a `MSPE` *or* `MAPE` *object*

Description

The function `plot.MSPE` plots a `MSPE` object that is returned by the `MSPE` function. The function `plot.MAPE` plots a `MAPE` object that is returned by the `MAPE` function.

Usage

```
## S3 method for class 'MSPE'
plot(x, vr = NULL, h = 1, N_min = 1, legend = TRUE,
  display.mins = TRUE, add.for.legend = 0, ...)

## S3 method for class 'MAPE'
plot(x, vr = NULL, h = 1, N_min = 1, legend = TRUE,
  display.mins = TRUE, add.for.legend = 0, ...)
```

Arguments

x	The MSPE or MAPE object to be plotted.
vr	parameter to plot a line at level vr. Intended to be used to plot the mean squared prediction error of the trivial, null predictor; optional.
h	Defines for which h-step predictor the mean squared prediction errors will be shown; default: 1.
N_min	If specified, the mean squared prediction errors with $N < N_{\min}$ will not be shown; integer and optional.
legend	Flag to specify if a legend, indicating which colour of the lines corresponds to which p, will be shown; default: TRUE.
display.mins	Flag to specify if the minima for each p, and the minimum accross $N = 0$ will be highlighted.
add.for.legend	add this much extra space for the legend, right of the lines.
...	Arguments to be passed to the underlying plot method

Value

Returns the plot, as specified.

See Also

MSPE, MAPE

---

predCoef *h-step Prediction coefficients*

Description

This function computes the localised and iterated Yule-Walker coefficients for h-step ahead forecasting of X_{t+h} from $X_t, ..., X_{t-p+1}$, where $h = 1, \ldots,$ H and $p = 1, \ldots,$ P.

Arguments

X	the data $X_1, \ldots, X_T$
P	the maximum order of coefficients to be computed; has to be a positive integer
H	the maximum lead time; has to be a positive integer
t	a vector of values t; the elements have to satisfy max(t) <= length(X) and min(t) >= min(max(N[N != 0]),p).
N	a vector of values N; the elements have to satisfy max(N[N != 0]) <= min(t) and min(N[N != 0]) >= 1 + P. $N = 0$ corresponds to the case where all data is taken into account.

Details

For every $t \in$ `t` and every $N \in$ `N` the (iterated) Yule-Walker estimates $\hat{v}_{N,T}^{(p,h)}(t)$ are computed. They are defined as

$$\hat{v}_{N,T}^{(p,h)}(t) := e_1' \big(e_1 \big(\hat{a}_{N,T}^{(p)}(t) \big)' + H \big)^h, \quad N \geq 1,$$

and

$$\hat{v}_{0,T}^{(p,h)}(t) := \hat{v}_{t,T}^{(p,h)}(t),$$

with

$$e_1 := \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad H := \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}$$

and

$$\hat{a}_{N,T}^{(p)}(t) := \big(\hat{\Gamma}_{N,T}^{(p)}(t) \big)^{-1} \hat{\gamma}_{N,T}^{(p)}(t),$$

where

$$\hat{\Gamma}_{N,T}^{(p)}(t) := \big[\hat{\gamma}_{i-j;N,T}(t) \big]_{i,j=1,\ldots,p}, \quad \hat{\gamma}_{N,T}^{(p)}(t) := \big(\hat{\gamma}_{1;N,T}(t), \ldots, \hat{\gamma}_{p;N,T}(t) \big)'$$

and

$$\hat{\gamma}_{k;N,T}(t) := \frac{1}{N} \sum_{\ell=t-N+|k|+1}^{t} X_{\ell-|k|,T} X_{\ell,T}$$

is the usual lag-k autocovariance estimator (without mean adjustment), computed from the observations $X_{t-N+1}, \ldots, X_t$.

The Durbin-Levinson Algorithm is used to successively compute the solutions to the Yule-Walker equations (cf. Brockwell/Davis (1991), Proposition 5.2.1). To compute the h-step ahead coefficients we use the recursive relationship

$$\hat{v}_{i,N,T}^{(p)}(t,h) = \hat{a}_{i,N,T}^{(p)}(t) \hat{v}_{1,N,T}^{(p,h-1)}(t) + \hat{v}_{i+1,N,T}^{(p,h-1)}(t) I\{i \leq p-1\},$$

(cf. Section 3.2, Step 3, in Kley et al. (2019)).

Value

Returns a named list with elements `coef`, `t`, and `N`, where `coef` is an array of dimension `P` $\times$ `P` $\times$ `H` $\times$ `length(t)` $\times$ `length(N)`, and `t`, and `N` are the parameters provided on the call of the function. See the example on how to access the vector $\hat{v}_{N,T}^{(p,h)}(t)$.

References

Brockwell, P. J. & Davis, R. A. (1991). Time Series: Theory and Methods. Springer, New York.

Examples

```
T <- 100
X <- rnorm(T)

P <- 5
H <- 1
m <- 20

Nmin <- 25
pcoef <- predCoef(X, P, H, (T - m - H + 1):T, c(0, seq(Nmin, T - m - H, 1)))

## Access the prediction vector for p = 2, h = 1, t = 95, N = 25
p <- 2
h <- 1
t <- 95
N <- 35
res <- pcoef$coef[p, 1:p, h, pcoef$t == t, pcoef$N == N]
```

`ts-models-tvARMA` *Simulation of an tvARMA(p,q) time series.*

Description

Returns a simulated time series $Y_{1,T}, ..., Y_{T,T}$ that fulfills the following equation:

$$Y_{t,T} = \sum_{j=1}^{p} a_j(t/T) Y_{t-j,T} + \sigma(t/T)\varepsilon_t + \sum_{k=1}^{q} \sigma((t-k)/T) b_k(t/T)\varepsilon_{t-k},$$

where $a_1, \ldots, a_p, b_0, b_1, \ldots, b_q$ are real-valued functions on $[0,1]$, σ is a positive function on $[0,1]$ and ε_t is white noise.

Usage

```
tvARMA(T = 128, a = list(), b = list(), sigma = function(u) {
  return(1) }, innov = function(n) {     rnorm(n, 0, 1) })
```

Arguments

T	length of the time series to be returned
a	list of p real-valued functions defined on $[0,1]$
b	list of q real-valued functions defined on $[0,1]$
sigma	function
innov	a function with one argument n that simulates a vector of the n residuals ε_t.

Value

Returns a tvARMA(p,q) time series with specified parameters.

Examples

```
## Taken from Section 6 in Dahlhaus (1997, AoS)
a1 <- function(u) {1.8 * cos(1.5 - cos(4 * pi * u))}
a2 <- function(u) {-0.81}
plot(tvARMA(128, a = list(a1, a2), b = list()), type = "l")
```

---

tvARMAcpp *Workhorse function for tvARMA time series generation*

Description

More explanation!

Arguments

`z`	a ...
`x_int`	a ...
`A`	...
`B`	a ...
`Sigma`	a ...

Value

Returns a ...

Index