Package ‘forecast’

May 12, 2015

Version 6.1
Date 2015-05-11
Title Forecasting Functions for Time Series and Linear Models
Description Methods and tools for displaying and analysing
univariate time series forecasts including exponential smoothing
via state space models and automatic ARIMA modelling.
Depends R (>= 3.0.2), stats, graphics, zoo, timeDate
Imports tseries, fracdiff, Rcpp (>= 0.11.0), nnet, colorspace,
parallel
Suggests testthat, fpp
LinkingTo Rcpp (>= 0.11.0), RcppArmadillo (>= 0.2.35)
LazyData yes
ByteCompile TRUE
Author Rob J Hyndman <Rob.Hyndman@monash.edu> with contributions from
George Athanasopoulos, Slava Razbash, Drew Schmidt, Zhenyu Zhou, Yousaf Khan,
Christoph Bergmeir, Earo Wang
Maintainer Rob J Hyndman <Rob.Hyndman@monash.edu>
BugReports https://github.com/robjhyndman/forecast/issues
License GPL (>= 2)
URL http://github.com/robjhyndman/forecast
NeedsCompilation yes
Repository CRAN
Date/Publication 2015-05-12 12:37:47

R topics documented:

accuracy .......................................................... 3
Acf .............................................................. 5
arfima .......................................................... 6
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Arima</td>
<td>8</td>
</tr>
<tr>
<td>arima.errors</td>
<td>10</td>
</tr>
<tr>
<td>arimaorder</td>
<td>11</td>
</tr>
<tr>
<td>auto.arima</td>
<td>12</td>
</tr>
<tr>
<td>bats</td>
<td>14</td>
</tr>
<tr>
<td>bizdays</td>
<td>15</td>
</tr>
<tr>
<td>BoxCox</td>
<td>16</td>
</tr>
<tr>
<td>BoxCox.lambda</td>
<td>17</td>
</tr>
<tr>
<td>croston</td>
<td>18</td>
</tr>
<tr>
<td>CV</td>
<td>20</td>
</tr>
<tr>
<td>dm.test</td>
<td>20</td>
</tr>
<tr>
<td>dshw</td>
<td>22</td>
</tr>
<tr>
<td>easter</td>
<td>24</td>
</tr>
<tr>
<td>ets</td>
<td>24</td>
</tr>
<tr>
<td>findfrequency</td>
<td>27</td>
</tr>
<tr>
<td>fitted.Arima</td>
<td>28</td>
</tr>
<tr>
<td>forecast</td>
<td>28</td>
</tr>
<tr>
<td>forecast.Arima</td>
<td>30</td>
</tr>
<tr>
<td>forecast.bats</td>
<td>32</td>
</tr>
<tr>
<td>forecast.ets</td>
<td>33</td>
</tr>
<tr>
<td>forecast.HoltWinters</td>
<td>35</td>
</tr>
<tr>
<td>forecast.lm</td>
<td>36</td>
</tr>
<tr>
<td>forecast.stl</td>
<td>38</td>
</tr>
<tr>
<td>forecast.StructTS</td>
<td>40</td>
</tr>
<tr>
<td>gas</td>
<td>42</td>
</tr>
<tr>
<td>getResponse</td>
<td>42</td>
</tr>
<tr>
<td>gold</td>
<td>43</td>
</tr>
<tr>
<td>logLik.ets</td>
<td>43</td>
</tr>
<tr>
<td>ma</td>
<td>44</td>
</tr>
<tr>
<td>meanf</td>
<td>45</td>
</tr>
<tr>
<td>monthdays</td>
<td>46</td>
</tr>
<tr>
<td>msts</td>
<td>47</td>
</tr>
<tr>
<td>na.interp</td>
<td>48</td>
</tr>
<tr>
<td>naive</td>
<td>49</td>
</tr>
<tr>
<td>ndiffs</td>
<td>50</td>
</tr>
<tr>
<td>nnetar</td>
<td>52</td>
</tr>
<tr>
<td>plot.Arima</td>
<td>53</td>
</tr>
<tr>
<td>plot.bats</td>
<td>55</td>
</tr>
<tr>
<td>plot.ets</td>
<td>56</td>
</tr>
<tr>
<td>plot.forecast</td>
<td>56</td>
</tr>
<tr>
<td>rwf</td>
<td>58</td>
</tr>
<tr>
<td>seasadj</td>
<td>59</td>
</tr>
<tr>
<td>seasonaldummy</td>
<td>60</td>
</tr>
<tr>
<td>seasonplot</td>
<td>62</td>
</tr>
<tr>
<td>ses</td>
<td>63</td>
</tr>
<tr>
<td>simulate.ets</td>
<td>65</td>
</tr>
<tr>
<td>sindexf</td>
<td>66</td>
</tr>
<tr>
<td>splinef</td>
<td>67</td>
</tr>
</tbody>
</table>
**accuracy**

Accuracy measures for forecast model

**Description**

Returns range of summary measures of the forecast accuracy. If x is provided, the function measures out-of-sample (test set) forecast accuracy based on x-f. If x is not provided, the function only produces in-sample (training set) accuracy measures of the forecasts based on f("x")-fitted(f). All measures are defined and discussed in Hyndman and Koehler (2006).

**Usage**

accuracy(f, x, test=NULL, d=NULL, D=NULL)

**Arguments**

- **f**: An object of class "forecast", or a numerical vector containing forecasts. It will also work with Arima, ets and lm objects if x is omitted – in which case in-sample accuracy measures are returned.

- **x**: An optional numerical vector containing actual values of the same length as object, or a time series overlapping with the times of f.

- **test**: Indicator of which elements of x and f to test. If test is NULL, all elements are used. Otherwise test is a numeric vector containing the indices of the elements to use in the test.

- **d**: An integer indicating the number of lag-1 differences to be used for the denominator in MASE calculation. Default value is 1 for non-seasonal series and 0 for seasonal series.

- **D**: An integer indicating the number of seasonal differences to be used for the denominator in MASE calculation. Default value is 0 for non-seasonal series and 1 for seasonal series.
Details

The measures calculated are:

- ME: Mean Error
- RMSE: Root Mean Squared Error
- MAE: Mean Absolute Error
- MPE: Mean Percentage Error
- MAPE: Mean Absolute Percentage Error
- MASE: Mean Absolute Scaled Error
- ACF1: Autocorrelation of errors at lag 1.

By default, the MASE calculation is scaled using MAE of in-sample naive forecasts for non-seasonal time series, in-sample seasonal naive forecasts for seasonal time series and in-sample mean forecasts for non-time series data.

See Hyndman and Koehler (2006) and Hyndman and Athanasopoulos (2014, Section 2.5) for further details.

Value

Matrix giving forecast accuracy measures.

Author(s)

Rob J Hyndman

References


Examples

```r
fit1 <- rwf(EuStockMarkets[1:200,1],h=100)
fit2 <- meanf(EuStockMarkets[1:200,1],h=100)
accuracy(fit1)
accuracy(fit2)
accuracy(fit1,EuStockMarkets[201:300,1])
accuracy(fit2,EuStockMarkets[201:300,1])
plot(fit1)
lines(EuStockMarkets[1:300,1])
```
(Partial) Autocorrelation Function Estimation

Description

The function `acf` computes (and by default plots) an estimate of the autocorrelation function of a univariate time series. Function `pacf` computes (and by default plots) an estimate of the partial autocorrelation function of a univariate time series.

Usage

```r
acf(x, lag.max=NULL, type=c("correlation", "partial"),
    plot=TRUE, main=NULL, xlim=NULL, ylim=NULL, xlab="Lag", ylab=NULL,
    na.action=na.contiguous, ...)
pacf(x, main=NULL, ...)
taperedacf(x, lag.max=NULL, type=c("correlation", "partial"),
    plot=TRUE, calc.ci=TRUE, level=95, nsim=100,
    xlim=NULL, ylim=NULL, xlab="Lag", ylab=NULL, ...)
taperedpacf(x, ...)
```

Arguments

- `x` a univariate time series
- `lag.max` maximum lag at which to calculate the acf. Default is 10*log10(N/m) where N is the number of observations and m the number of series. Will be automatically limited to one less than the number of observations in the series.
- `type` character string giving the type of acf to be computed. Allowed values are "correlation" (the default) or "partial".
- `plot` logical. If TRUE (the default) the acf is plotted.
- `main` Title for plot
- `xlim` The x limits of the plot
- `ylim` The y limits of the plot
- `xlab` The label on the x-axis of the plot
- `ylab` The label on the y-axis of the plot
- `na.action` function to handle missing values. Default is `na.contiguous`. Useful alternatives are `na.pass` and `na.interp`.
- `calc.ci` If TRUE, confidence intervals for the ACF/PACF estimates are calculated.
- `level` Percentage level used for the confidence intervals.
- `nsim` The number of bootstrap samples used in estimating the confidence intervals.
- `...` Additional arguments passed to `acf` or to the plotting function.
Details

The functions improve the \texttt{acf} and \texttt{pacf} functions when applied to univariate time series. The main differences are that \texttt{acf} does not plot a spike at lag 0 (which is redundant) and the horizontal axes show lags in time units rather than seasonal units.

The tapered versions implement the ACF and PACF estimates and plots described in Hyndman (2015), based on the banded and tapered estimates of autocovariance proposed by McMurry and Politis (2010).

Value

The \texttt{acf} and \texttt{pacf} functions return objects of class "acf" as described in \texttt{acf} from the stats package. The \texttt{taperedacf} and \texttt{taperedpacf} functions return objects of class "mpacf".

Author(s)

Rob J Hyndman

References


See Also

\texttt{acf, pacf, tsdisplay}

Examples

\begin{verbatim}
Acf(wineind)
Pacf(wineind)
## Not run:
taperedacf(wineind, nsim=50)
taperedpacf(wineind, nsim=50)

## End(Not run)
\end{verbatim}

\textit{arfima} \hspace{1cm} \textit{Fit a fractionally differenced ARFIMA model}

Description

An ARFIMA(p,d,q) model is selected and estimated automatically using the Hyndman-Khandakar (2008) algorithm to select p and q and the Haslett and Raftery (1989) algorithm to estimate the parameters including d.
arfima

Usage
arfima(x, drange=c(0, 0.5), estim=c("mle","ls"), lambda=NULL, ...)

Arguments
x a univariate time series (numeric vector).
drange Allowable values of d to be considered. Default of c(0,0.5) ensures a stationary model is returned.
estim If estim="ls", then the ARMA parameters are calculated using the Haslett-Raftery algorithm. If estim="mle", then the ARMA parameters are calculated using full MLE via the arima function.
lambda Box-Cox transformation parameter. Ignored if NULL. Otherwise, data transformed before model is estimated.
... Other arguments passed to auto.arima when selecting p and q.

Details
This function combines fracdiff and auto.arima to automatically select and estimate an ARFIMA model. The fractional differencing parameter is chosen first assuming an ARFIMA(2,d,0) model. Then the data are fractionally differenced using the estimated d and an ARMA model is selected for the resulting time series using auto.arima. Finally, the full ARFIMA(p,d,q) model is re-estimated using fracdiff. If estim="mle", the ARMA coefficients are refined using arima.

Value
A list object of S3 class "fracdiff", which is described in the fracdiff documentation. A few additional objects are added to the list including x (the original time series), and the residuals and fitted values.

Author(s)
Rob J Hyndman and Farah Yasmeen

References


See Also
fracdiff, auto.arima, forecast.fracdiff.
Examples

library(fracdiff)
x <- fracdiff.sim(100, ma=-.4, d=.3)$series
fit <- arfima(x)
tsdisplay(residuals(fit))

---

Arima

Fit ARIMA model to univariate time series

Description

Largely a wrapper for the `arima` function in the stats package. The main difference is that this function allows a drift term. It is also possible to take an ARIMA model from a previous call to `arima` and re-apply it to the data \( x \).

Usage

\[
\text{Arima}(x, \text{order}=c(p, q, d), \text{seasonal}=c(P, Q, s), \text{xreg}=NULL, \text{include.mean}=TRUE, \text{include.drift}=FALSE, \text{include.constant}, \text{lambda}=\text{model}$lambda, \text{transform.pars}=TRUE, \text{fixed}=NULL, \text{init}=NULL, \text{method}=c("CSS-ML","ML","CSS"), \text{n.cond}, \text{optim.control}=\text{list}(\), \text{kappa}=1e6, \text{model}=\text{NULL})
\]

Arguments

- **x**: a univariate time series of class `ts`.
- **order**: A specification of the non-seasonal part of the ARIMA model: the three components \( (p, d, q) \) are the AR order, the degree of differencing, and the MA order.
- **seasonal**: A specification of the seasonal part of the ARIMA model, plus the period (which defaults to frequency(\( x \))). This should be a list with components order and period, but a specification of just a numeric vector of length 3 will be turned into a suitable list with the specification as the order.
- **xreg**: Optionally, a vector or matrix of external regressors, which must have the same number of rows as \( x \).
- **include.mean**: Should the ARIMA model include a mean term? The default is TRUE for undifferenced series, FALSE for differenced ones (where a mean would not affect the fit nor predictions).
- **include.drift**: Should the ARIMA model include a linear drift term? (i.e., a linear regression with ARIMA errors is fitted.) The default is FALSE.
- **include.constant**: If TRUE, then `include.mean` is set to be TRUE for undifferenced series and `include.drift` is set to be TRUE for differenced series. Note that if there is more than one difference taken, no constant is included regardless of the value of this argument. This is deliberate as otherwise quadratic and higher order polynomial trends would be induced.
lambda

Box-Cox transformation parameter. Ignored if NULL. Otherwise, data transformed before model is estimated.

transform.pars

Logical. If true, the AR parameters are transformed to ensure that they remain in the region of stationarity. Not used for method="CSS".

fixed

optional numeric vector of the same length as the total number of parameters. If supplied, only NA entries in fixed will be varied. transform.pars=TRUE will be overridden (with a warning) if any AR parameters are fixed. It may be wise to set transform.pars=FALSE when fixing MA parameters, especially near non-invertibility.

init

optional numeric vector of initial parameter values. Missing values will be filled in, by zeroes except for regression coefficients. Values already specified in fixed will be ignored.

method

Fitting method: maximum likelihood or minimize conditional sum-of-squares. The default (unless there are missing values) is to use conditional-sum-of-squares to find starting values, then maximum likelihood.

n.cond

Only used if fitting by conditional-sum-of-squares: the number of initial observations to ignore. It will be ignored if less than the maximum lag of an AR term.

optim.control

List of control parameters for optim.

kappa

the prior variance (as a multiple of the innovations variance) for the past observations in a differenced model. Do not reduce this.

model

Output from a previous call to Arima. If model is passed, this same model is fitted to x without re-estimating any parameters.

Details

See the arima function in the stats package.

Value

See the arima function in the stats package. The additional objects returned are

x

The time series data

xreg

The regressors used in fitting (when relevant).

Author(s)

Rob J Hyndman

See Also

arima, forecast.Arima.
Examples

```r
fit <- Arima(WWWusage,order=c(3,1,0))
plot(forecast(fit,h=20))

# Fit model to first few years of AirPassengers data
air.model <- Arima(window(AirPassengers,end=1956+11/12),order=c(0,1,1),
                      seasonal=list(order=c(0,1,1),period=12),lambda=0)
plot(forecast(air.model,h=48))
lines(AirPassengers)

# Apply fitted model to later data
air.model2 <- Arima(window(AirPassengers,start=1957),model=air.model)

# Forecast accuracy measures on the log scale.
# in-sample one-step forecasts.
accuracy(air.model)
# out-of-sample one-step forecasts.
accuracy(air.model2)
# out-of-sample multi-step forecasts
accuracy(forecast(air.model,h=48,lambd=NULL),
         log(window(AirPassengers,start=1957)))
```

arima.errors  ARIMA errors

Description

Returns original time series after adjusting for regression variables. These are not the same as the residuals. If there are no regression variables in the ARIMA model, then the errors will be identical to the original series. If there are regression variables in the ARIMA model, then the errors will be equal to the original series minus the effect of the regression variables, but leaving in the serial correlation that is modelled with the AR and MA terms. If you want the "residuals", then use residuals(z).

Usage

```r
arima.errors(z)
```

Arguments

- `z`: Fitted ARIMA model from `arima`

Value

A time series containing the "errors".

Author(s)

Rob J Hyndman
arimaorder

See Also

arima, residuals

Examples

www.fit <- auto.arima(WWWusage)
www.errors <- arima.errors(www.fit)
par(mfrow=c(2,1))
plot(WWWusage)
plot(www.errors)
auto.arima

Fit best ARIMA model to univariate time series

Description

Returns best ARIMA model according to either AIC, AICc or BIC value. The function conducts a search over possible model within the order constraints provided.

Usage

auto.arima(x, d=NA, D=NA, max.p=5, max.q=5, max.P=2, max.Q=2, max.order=5, max.d=2, max.D=1, start.p=2, start.q=2, start.P=1, start.Q=1, stationary=FALSE, seasonal=TRUE, ic=c("aicc", "aic", "bic"), stepwise=TRUE, trace=FALSE, approximation=(length(x)>100 | frequency(x)>12), xreg=NULL, test=c("kpss", "adf", "pp"), seasonal.test=c("ocsb", "ch"), allowdrift=TRUE, allowmean=TRUE, lambda=NULL, parallel=FALSE, num.cores=2)

Arguments

- **x**: a univariate time series
- **d**: Order of first-differencing. If missing, will choose a value based on KPSS test.
- **D**: Order of seasonal-differencing. If missing, will choose a value based on OCSB test.
- **max.p**: Maximum value of p
- **max.q**: Maximum value of q
- **max.P**: Maximum value of P
- **max.Q**: Maximum value of Q
- **max.order**: Maximum value of p+q+P+Q if model selection is not stepwise.
- **max.d**: Maximum number of non-seasonal differences
- **max.D**: Maximum number of seasonal differences
- **start.p**: Starting value of p in stepwise procedure.
- **start.q**: Starting value of q in stepwise procedure.
- **start.P**: Starting value of P in stepwise procedure.
- **start.Q**: Starting value of Q in stepwise procedure.
- **stationary**: If TRUE, restricts search to stationary models.
- **seasonal**: If FALSE, restricts search to non-seasonal models.
- **ic**: Information criterion to be used in model selection.
- **stepwise**: If TRUE, will do stepwise selection (faster). Otherwise, it searches over all models. Non-stepwise selection can be very slow, especially for seasonal models.
trace If TRUE, the list of ARIMA models considered will be reported.
approximation If TRUE, estimation is via conditional sums of squares and the information criteria used for model selection are approximated. The final model is still computed using maximum likelihood estimation. Approximation should be used for long time series or a high seasonal period to avoid excessive computation times.
xreg Optionally, a vector or matrix of external regressors, which must have the same number of rows as x.
test Type of unit root test to use. See ndiffs for details.
seasonal.test This determines which seasonal unit root test is used. See nsdiffs for details.
allowdrift If TRUE, models with drift terms are considered.
allowmean If TRUE, models with a non-zero mean are considered.
lambda Box-Cox transformation parameter. Ignored if NULL. Otherwise, data transformed before model is estimated.
parallel If TRUE and stepwise = FALSE, then the specification search is done in parallel. This can give a significant speedup on multicore machines.
um.cores Allows the user to specify the amount of parallel processes to be used if parallel = TRUE and stepwise = FALSE. If NULL, then the number of logical cores is automatically detected and all available cores are used.

Details
Non-stepwise selection can be slow, especially for seasonal data. Stepwise algorithm outlined in Hyndman and Khandakar (2008) except that the default method for selecting seasonal differences is now the OCSB test rather than the Canova-Hansen test.

Value
Same as for arima

Author(s)
Rob J Hyndman

References

See Also
Arima

Examples
fit <- auto.arima(WWWusage)
plot(forecast(fit, h=20))
**bats**

*BATS model (Exponential smoothing state space model with Box-Cox transformation, ARMA errors, Trend and Seasonal components)*

**Description**

Fits a BATS model applied to \( y \), as described in De Livera, Hyndman & Snyder (2011). Parallel processing is used by default to speed up the computations.

**Usage**

\[
bats(y, \text{use.box.cox}=\text{NULL}, \text{use.trend}=\text{NULL}, \text{use.damped.trend}=\text{NULL}, \text{seasonal.periods}=\text{NULL}, \text{use.arma.errors}=\text{TRUE}, \text{use.parallel}=\text{TRUE}, \text{num.cores}=2, \text{bc.lower}=0, \text{bc.upper}=1, \ldots)
\]

**Arguments**

- **y**
  The time series to be forecast. Can be numeric, msts or ts. Only univariate time series are supported.

- **use.box.cox**
  TRUE/FALSE indicates whether to use the Box-Cox transformation or not. If NULL then both are tried and the best fit is selected by AIC.

- **use.trend**
  TRUE/FALSE indicates whether to include a trend or not. If NULL then both are tried and the best fit is selected by AIC.

- **use.damped.trend**
  TRUE/FALSE indicates whether to include a damping parameter in the trend or not. If NULL then both are tried and the best fit is selected by AIC.

- **seasonal.periods**
  If \( y \) is a numeric then seasonal periods can be specified with this parameter.

- **use.arma.errors**
  TRUE/FALSE indicates whether to include ARMA errors or not. If TRUE the best fit is selected by AIC. If FALSE then the selection algorithm does not consider ARMA errors.

- **use.parallel**
  TRUE/FALSE indicates whether or not to use parallel processing.

- **num.cores**
  The number of parallel processes to be used if using parallel processing. If NULL then the number of logical cores is detected and all available cores are used.

- **bc.lower**
  The lower limit (inclusive) for the Box-Cox transformation.

- **bc.upper**
  The upper limit (inclusive) for the Box-Cox transformation.

- **...**
  Additional arguments to be passed to auto.arima when choose an ARMA(p, q) model for the errors. (Note that xreg will be ignored, as will any arguments concerning seasonality and differencing, but arguments controlling the values of p and q will be used.)
Value

An object of class "bats". The generic accessor functions fitted.values and residuals extract useful features of the value returned by bats and associated functions. The fitted model is designated BATS(omega, p,q, phi, m1,...,mJ) where omega is the Box-Cox parameter and phi is the damping parameter; the error is modelled as an ARMA(p,q) process and m1,...,mJ list the seasonal periods used in the model.

Author(s)

Slava Razbash and Rob J Hyndman

References


Examples

```r
## Not run:
fit <- bats(USAccDeaths, use.parallel=FALSE)
plot(forecast(fit))

taylor.fit <- bats(taylor)
plot(forecast(taylor.fit))
## End(Not run)
```

### bizdays

**Number of trading days in each season**

Description

Returns number of trading days in each month or quarter of the observed time period.

Usage

```r
bizdays(x, FinCenter)
```

Arguments

- **x**: Monthly or quarterly time series
- **FinCenter**: A character with the the location of the financial center named as "continent/city". This concept allows to handle data records collected in different time zones and mix them up to have always the proper time stamps with respect to your personal financial center, or alternatively to the GMT reference time. More details on `finCenter`. 
Details

Useful for trading days length adjustments. More on how to define "business days", please refer to isBizday.

Value

Time series

Author(s)

Earo Wang

See Also

monthdays

Examples

bizdays(wineind, FinCenter = "Sydney")

\[ f_\lambda(x) = \frac{x^\lambda - 1}{\lambda} \]

if \( \lambda \neq 0 \). For \( \lambda = 0 \),

\[ f_0(x) = \log(x) \]
BoxCox.lambda

Value

a numeric vector of the same length as x.

Author(s)

Rob J Hyndman

References


See Also

BoxCox.lambda

Examples

lambda <- BoxCox.lambda(lynx)
lynx.fit <- ar(BoxCox(lynx, lambda))
plot(forecast(lynx.fit, h=20, lambda=lambda))

Description

If method=="guerrero", Guerrero's (1993) method is used, where lambda minimizes the coefficient of variation for subseries of x.

If method=="loglik", the value of lambda is chosen to maximize the profile log likelihood of a linear model fitted to x. For non-seasonal data, a linear time trend is fitted while for seasonal data, a linear time trend with seasonal dummy variables is used.

Usage

BoxCox.lambda(x, method=c("guerrero","loglik"), lower=-1, upper=2)

Arguments

x a numeric vector or time series
method Choose method to be used in calculating lambda.
lower Lower limit for possible lambda values.
upper Upper limit for possible lambda values.

Value

a number indicating the Box-Cox transformation parameter.
**Author(s)**

Leanne Chhay and Rob J Hyndman

**References**


**See Also**

BoxCox

**Examples**

```r
lambda <- BoxCox.lambda(AirPassengers,lower=0)
air.fit <- Arima(AirPassengers, order=c(0,1,1),
seasonal=list(order=c(0,1,1),period=12), lambda=lambda)
plot(forecast(air.fit))
```

**croston**  
*Forecasts for intermittent demand using Croston’s method*

**Description**

Returns forecasts and other information for Croston’s forecasts applied to x.

**Usage**

`croston(x, h=10, alpha=0.1)`

**Arguments**

- `x`  
a numeric vector or time series
- `h`  
Number of periods for forecasting.
- `alpha`  
Value of alpha. Default value is 0.1.

**Details**

Based on Croston’s (1972) method for intermittent demand forecasting, also described in Shenstone and Hyndman (2005). Croston’s method involves using simple exponential smoothing (SES) on the non-zero elements of the time series and a separate application of SES to the times between non-zero elements of the time series. The smoothing parameters of the two applications of SES are assumed to be equal and are denoted by alpha.

Note that prediction intervals are not computed as Croston’s method has no underlying stochastic model. The separate forecasts for the non-zero demands, and for the times between non-zero demands do have prediction intervals based on ETS(A,N,N) models.
Value

An object of class "forecast" is a list containing at least the following elements:

- **model**: A list containing information about the fitted model. The first element gives the model used for non-zero demands. The second element gives the model used for times between non-zero demands. Both elements are of class forecast.
- **method**: The name of the forecasting method as a character string.
- **mean**: Point forecasts as a time series.
- **x**: The original time series (either object itself or the time series used to create the model stored as object).
- **residuals**: Residuals from the fitted model. That is x minus fitted values.
- **fitted**: Fitted values (one-step forecasts).

The function `summary` is used to obtain and print a summary of the results, while the function `plot` produces a plot of the forecasts.

The generic accessor functions `fitted`, `values` and `residuals` extract useful features of the value returned by `croston` and associated functions.

Author(s)

Rob J Hyndman

References


See Also

- `ses`

Examples

```r
x <- rpois(20, lambda = .3)
fcast <- croston(x)
plot(fcast)
```
**Cross-validation statistic**

**Description**
Computes the leave-one-out cross-validation statistic (also known as PRESS – prediction residual sum of squares), AIC, corrected AIC, BIC and adjusted R^2 values for a linear model.

**Usage**

```r
CV(obj)
```

**Arguments**

- `obj`: output from `lm` or `tslm`

**Value**
Numerical vector containing CV, AIC, AICc, BIC and AdjR2 values.

**Author(s)**
Rob J Hyndman

**See Also**
AIC

**Examples**

```r
y <- ts(rnorm(120,0,3) + 20*sin(2*pi*(1:120)/12), frequency=12)
fit1 <- tslm(y ~ trend + season)
fit2 <- tslm(y ~ season)
CV(fit1)
CV(fit2)
```

---

**Diebold-Mariano test for predictive accuracy**

**Description**
The Diebold-Mariano test compares the forecast accuracy of two forecast methods.

**Usage**

```r
dm.test(e1, e2, alternative=c("two.sided","less","greater"),
        h=1, power=2)
```
Arguments

- **e1**: Forecast errors from method 1.
- **e2**: Forecast errors from method 2.
- **alternative**: A character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.
- **h**: The forecast horizon used in calculating e1 and e2.
- **power**: The power used in the loss function. Usually 1 or 2.

Details

The null hypothesis is that the two methods have the same forecast accuracy. For alternative="less", the alternative hypothesis is that method 2 is less accurate than method 1. For alternative="greater", the alternative hypothesis is that method 2 is more accurate than method 1. For alternative="two.sided", the alternative hypothesis is that method 1 and method 2 have different levels of accuracy.

Value

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

- **statistic**: the value of the DM-statistic.
- **parameter**: the forecast horizon and loss function power used in the test.
- **alternative**: a character string describing the alternative hypothesis.
- **p.value**: the p-value for the test.
- **method**: a character string with the value "Diebold-Mariano Test".
- **data.name**: a character vector giving the names of the two error series.

Author(s)

George Athanasopoulos, Yousaf Khan and Rob Hyndman

References


Examples

```r
# Test on in-sample one-step forecasts
f1 <- ets(WWWusage)
f2 <- auto.arima(WWWusage)
accuracy(f1)
accuracy(f2)
dm.test(residuals(f1), residuals(f2), h=1)

# Test on out-of-sample one-step forecasts
f1 <- ets(WWWusage[1:80])
f2 <- auto.arima(WWWusage[1:80])
```
Description


Usage

dshw(y, period1, period2, h=2*max(period1,period2),
   alpha=NULL, beta=NULL, gamma=NULL, omega=NULL, phi=NULL,
   lambda=NULL, armethod=TRUE, model = NULL)

Arguments

y Either an msts object with two seasonal periods or a numeric vector.
period1 Period of the shorter seasonal period. Only used if y is not an msts object.
period2 Period of the longer seasonal period. Only used if y is not an msts object.
h Number of periods for forecasting.
alpha Smoothing parameter for the level. If NULL, the parameter is estimated using least squares.
beta Smoothing parameter for the slope. If NULL, the parameter is estimated using least squares.
gamma Smoothing parameter for the first seasonal period. If NULL, the parameter is estimated using least squares.
omega Smoothing parameter for the second seasonal period. If NULL, the parameter is estimated using least squares.
phi Autoregressive parameter. If NULL, the parameter is estimated using least squares.
lambda Box-Cox transformation parameter. Ignored if NULL. Otherwise, data transformed before model is estimated.
armethod If TRUE, the forecasts are adjusted using an AR(1) model for the errors.
model If it’s specified, an existing model is applied to a new data set.

Details

Taylor’s (2003) double-seasonal Holt-Winters method uses additive trend and multiplicative seasonality, where there are two seasonal components which are multiplied together. For example, with a series of half-hourly data, one would set period1=48 for the daily period and period2=336 for the weekly period. The smoothing parameter notation used here is different from that in Taylor (2003); instead it matches that used in Hyndman et al (2008) and that used for the ets function.
Value

An object of class "forecast".

The function summary is used to obtain and print a summary of the results, while the function plot produces a plot of the forecasts.

The generic accessor functions fitted.values and residuals extract useful features of the value returned by meanf.

An object of class "forecast" is a list containing at least the following elements:

- **model**: A list containing information about the fitted model
- **method**: The name of the forecasting method as a character string
- **mean**: Point forecasts as a time series
- **x**: The original time series (either object itself or the time series used to create the model stored as object).
- **residuals**: Residuals from the fitted model. That is x minus fitted values.
- **fitted**: Fitted values (one-step forecasts)

Author(s)

Rob J Hyndman

References


See Also

HoltWinters, ets.

Examples

```r
## Not run:
fcast <- dshw(taylor)
plot(fcast)
## End(Not run)

t <- seq(0, 5, by=1/20)
x <- exp(sin(2*pi*t) + cos(2*pi*t*4) + rnorm(length(t), 0, 1))
fit <- dshw(x, 20, 5)
plot(fit)
```
easter  

*Easter holidays in each season*

**Description**

Returns a vector of 0’s and 1’s or fractional results if Easter spans March and April in the observed time period. Easter is defined as the days from Good Friday to Easter Sunday inclusively, plus optionally Easter Monday if `easter.mon`=TRUE.

**Usage**

```r
easter(x, easter.mon = FALSE)
```

**Arguments**

- `x` Monthly or quarterly time series
- `easter.mon` If TRUE, the length of Easter holidays includes Easter Monday.

**Details**

Useful for adjusting calendar effects.

**Value**

Time series

**Author(s)**

Earo Wang

**Examples**

```r
easter(wineind, easter.mon = TRUE)
```

---

ets  

*Exponential smoothing state space model*

**Description**

Returns ets model applied to y.
Usage

ets(y, model="ZZZ", damped=NULL, alpha=NULL, beta=NULL, gamma=NULL,
phi=NULL, additive.only=FALSE, lambda=NULL,
lower=c(rep(0.0001,3), 0.8), upper=c(rep(0.9999,3), 0.98),
opt.crit=c("lik","amse","mse","sigma","mae"), nmse=3,
bounds=c("both","usual","admissible"), ic=c("aic","aicc","bic"),
restrict=TRUE, allow.multiplicative.trend=FALSE, use.initial.values=FALSE,...)

Arguments

y a numeric vector or time series

model Usually a three-character string identifying method using the framework terminology of Hyndman et al. (2002) and Hyndman et al. (2008). The first letter denotes the error type ("A", "M" or "Z"); the second letter denotes the trend type ("N","A","M" or "Z"); and the third letter denotes the season type ("N","A","M" or "Z"). In all cases, "N"=none, "A"=additive, "M"=multiplicative and "Z"=automatically selected. So, for example, "ANN" is simple exponential smoothing with additive errors, "MAM" is multiplicative Holt-Winters' method with multiplicative errors, and so on.

It is also possible for the model to be of class "ets", and equal to the output from a previous call to ets. In this case, the same model is fitted to y without re-estimating any smoothing parameters. See also the use.initial.values argument.

damped If TRUE, use a damped trend (either additive or multiplicative). If NULL, both damped and non-damped trends will be tried and the best model (according to the information criterion ic) returned.

alpha Value of alpha. If NULL, it is estimated.

beta Value of beta. If NULL, it is estimated.

gamma Value of gamma. If NULL, it is estimated.

phi Value of phi. If NULL, it is estimated.

additive.only If TRUE, will only consider additive models. Default is FALSE.

lambda Box-Cox transformation parameter. Ignored if NULL. Otherwise, data transformed before model is estimated. When lambda=TRUE, additive.only is set to FALSE.

lower Lower bounds for the parameters (alpha, beta, gamma, phi)

upper Upper bounds for the parameters (alpha, beta, gamma, phi)

opt.crit Optimization criterion. One of "mse" (Mean Square Error), "amse" (Average MSE over first nmse forecast horizons), "sigma" (Standard deviation of residuals), "mae" (Mean of absolute residuals), or "lik" (Log-likelihood, the default).

nmse Number of steps for average multistep MSE (1<=nmse<=10).

bounds Type of parameter space to impose: "usual" indicates all parameters must lie between specified lower and upper bounds; "admissible" indicates parameters must lie in the admissible space; "both" (default) takes the intersection of these regions.
ic Information criterion to be used in model selection.
restrict If TRUE (default), the models with infinite variance will not be allowed.
allow.multiplicative.trend If TRUE (default), models with multiplicative trend are allowed when searching for a model. Otherwise, the model space excludes them. This argument is ignored if a multiplicative trend model is explicitly requested (e.g., using model="MMN").
use.initial.values If TRUE and model is of class "ets", then the initial values in the model are also not re-estimated.
... Other undocumented arguments.

Details

Based on the classification of methods as described in Hyndman et al (2008).
The methodology is fully automatic. The only required argument for ets is the time series. The model is chosen automatically if not specified. This methodology performed extremely well on the M3-competition data. (See Hyndman, et al, 2002, below.)

Value

An object of class "ets".
The generic accessor functions fitted.values and residuals extract useful features of the value returned by ets and associated functions.

Author(s)

Rob J Hyndman

References


See Also

*HoltWinters*, *rwf*, *Arima*.

Examples

```r
fit <- ets(USAccDeaths)
plot(forecast(fit))
```
findfrequency

**findfrequency**

Find dominant frequency of a time series

**Description**

findfrequency returns the period of the dominant frequency of a time series. For seasonal data, it will return the seasonal period. For cyclic data, it will return the average cycle length.

**Usage**

findfrequency(x)

**Arguments**

x a numeric vector or time series

**Details**

The dominant frequency is determined from a spectral analysis of the time series. First, a linear trend is removed, then the spectral density function is estimated from the best fitting autoregressive model (based on the AIC). If there is a large (possibly local) maximum in the spectral density function at frequency $f$, then the function will return the period $1/f$ (rounded to the nearest integer). If no such dominant frequency can be found, the function will return 1.

**Value**

an integer value

**Author(s)**

Rob J Hyndman

**Examples**

findfrequency(USAccDeaths) # Monthly data
findfrequency(taylor) # Half-hourly data
findfrequency(lynx) # Annual data
fitted.Arima

One-step in-sample forecasts using ARIMA models

Description

Returns one-step forecasts for the data used in fitting the ARIMA model.

Usage

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

Arguments

- `object`: An object of class "Arima". Usually the result of a call to `arima`.
- `...`: Other arguments.

Value

An time series of the one-step forecasts.

Author(s)

Rob J Hyndman

See Also

`forecast.Arima`.

Examples

```r
fit <- Arima(WWWusage, c(3,1,0))
plot(WWWusage)
lines(fitted(fit), col=2)
```

forecast

Forecasting time series

Description

`forecast` is a generic function for forecasting from time series or time series models. The function invokes particular `methods` which depend on the class of the first argument.

For example, the function `forecast.Arima` makes forecasts based on the results produced by `arima`.

The function `forecast.ts` makes forecasts using `ets` models (if the data are non-seasonal or the seasonal period is 12 or less) or `stlf` (if the seasonal period is 13 or more).
Usage

```r
forecast(object,...)
# S3 method for class 'ts'
forecast(object, h = ifelse(frequency(object) > 1, 2 * frequency(object), 10),
         level=c(80,95), fan=FALSE, robust=FALSE, lambda=NULL, find.frequency=FALSE,
         allow.multiplicative.trend=FALSE, ...)
```

Arguments

- `object` a time series or time series model for which forecasts are required
- `h` Number of periods for forecasting
- `level` Confidence level for prediction intervals.
- `fan` If TRUE, level is set to `seq(50,99,by=1)`. This is suitable for fan plots.
- `robust` If TRUE, the function is robust to missing values and outliers in object. This argument is only valid when `object` is of class `ts`.
- `lambda` Box-Cox transformation parameter.
- `find.frequency` If TRUE, the function determines the appropriate period, if the data is of unknown period.
- `allow.multiplicative.trend` If TRUE, then ETS models with multiplicative trends are allowed. Otherwise, only additive or no trend ETS models are permitted.
- `...` Additional arguments affecting the forecasts produced. `forecast.ts` passes these to `forecast.ets` or `stlf` depending on the frequency of the time series.

Value

An object of class "forecast".

The function `summary` is used to obtain and print a summary of the results, while the function `plot` produces a plot of the forecasts and prediction intervals.

The generic accessor functions `fitted.values` and `residuals` extract various useful features of the value returned by `forecast$model`.

An object of class "forecast" is a list usually containing at least the following elements:

- `model` A list containing information about the fitted model
- `method` The name of the forecasting method as a character string
- `mean` Point forecasts as a time series
- `lower` Lower limits for prediction intervals
- `upper` Upper limits for prediction intervals
- `level` The confidence values associated with the prediction intervals
- `x` The original time series (either `object` itself or the time series used to create the model stored as `object`).
- `residuals` Residuals from the fitted model. For models with additive errors, the residuals will be `x` minus the fitted values.
- `fitted` Fitted values (one-step forecasts)
Author(s)
Rob J Hyndman

See Also
Other functions which return objects of class "forecast" are forecast.ets, forecast.Arima, forecast.HoltWinters, forecast.StructTS, meanf, rwf, splinef, thetaf, croston, ses, holt, hw.

forecast.Arima  Forecasting using ARIMA or ARFIMA models

Description
Returns forecasts and other information for univariate ARIMA models.

Usage

## S3 method for class 'Arima'
forecast(object, h=ifelse(object$arma[5]>1,2+object$arma[5],10),
  level=c(80,95), fan=FALSE, xreg=NULL, lambda=object$lambda,
  bootstrap=FALSE, npaths=5000, ...)
## S3 method for class 'ar'
forecast(object, h=10, level=c(80,95), fan=FALSE, lambda=NULL,
  bootstrap=FALSE, npaths=5000, ...)
## S3 method for class 'fracdiff'
forecast(object, h=10, level=c(80,95), fan=FALSE, lambda=object$lambda, ...)

Arguments

object  An object of class "Arima", "ar" or "fracdiff". Usually the result of a call to arima, auto.arima, ar, arfima or fracdiff.
h  Number of periods for forecasting. If xreg is used, h is ignored and the number of forecast periods is set to the number of rows of xreg.
level  Confidence level for prediction intervals.
fan  If TRUE, level is set to seq(50,99,by=1). This is suitable for fan plots.
xreg  Future values of an regression variables (for class Arima objects only).
lambda  Box-Cox transformation parameter. Ignored if NULL. Otherwise, forecasts back-transformed via an inverse Box-Cox transformation.
bootstrap  If TRUE, then prediction intervals computed using simulation with resampled errors.
npaths  Number of sample paths used in computing simulated prediction intervals when bootstrap=TRUE.
...  Other arguments.
Details

For Arima or ar objects, the function calls `predict.Arima` or `predict.ar` and constructs an object of class "forecast" from the results. For fracdiff objects, the calculations are all done within `forecast.fracdiff` using the equations given by Peiris and Perera (1988).

Value

An object of class "forecast".

The function `summary` is used to obtain and print a summary of the results, while the function `plot` produces a plot of the forecasts and prediction intervals.

The generic accessor functions `fitted.values` and `residuals` extract useful features of the value returned by `forecast.Arima`.

An object of class "forecast" is a list containing at least the following elements:

- `model`: A list containing information about the fitted model
- `method`: The name of the forecasting method as a character string
- `mean`: Point forecasts as a time series
- `lower`: Lower limits for prediction intervals
- `upper`: Upper limits for prediction intervals
- `level`: The confidence values associated with the prediction intervals
- `x`: The original time series (either object itself or the time series used to create the model stored as object).
- `residuals`: Residuals from the fitted model. That is x minus fitted values.
- `fitted`: Fitted values (one-step forecasts)

Author(s)

Rob J Hyndman

References


See Also

`predict.Arima`, `predict.ar`, `auto.arima`, `Arima`, `arima`, `ar`, `arfima`.

Examples

```r
fit <- Arima(WWWusage,c(3,1,0))
plot(forecast(fit))

library(fracdiff)
x <- fracdiff.sim( 100, ma=-.4, d=.3)$series
fit <- arfima(x)
plot(forecast(fit,h=30))
```
Description

Forecasts \( h \) steps ahead with a BATS model. Prediction intervals are also produced.

Usage

```
## S3 method for class 'bats'
forecast(object, h, level=c(80,95), fan=FALSE, ...)
## S3 method for class 'tbats'
forecast(object, h, level=c(80,95), fan=FALSE, ...)
```

Arguments

- **object**: An object of class "bats". Usually the result of a call to `bats`.
- **h**: Number of periods for forecasting. Default value is twice the largest seasonal period (for seasonal data) or ten (for non-seasonal data).
- **level**: Confidence level for prediction intervals.
- **fan**: If TRUE, level is set to `seq(50, 99, by=1)`. This is suitable for fan plots.
- **...**: Other arguments, currently ignored.

Value

An object of class "forecast".

The function `summary` is used to obtain and print a summary of the results, while the function `plot` produces a plot of the forecasts and prediction intervals.

The generic accessor functions `fitted` and `residuals` extract useful features of the value returned by `forecast.bats`.

An object of class "forecast" is a list containing at least the following elements:

- **model**: A copy of the `bats` object
- **method**: The name of the forecasting method as a character string
- **mean**: Point forecasts as a time series
- **lower**: Lower limits for prediction intervals
- **upper**: Upper limits for prediction intervals
- **level**: The confidence values associated with the prediction intervals
- **x**: The original time series (either `object` itself or the time series used to create the model stored as `object`).
- **residuals**: Residuals from the fitted model.
- **fitted**: Fitted values (one-step forecasts)
Author(s)
Slava Razbash and Rob J Hyndman

References
seasonal patterns using exponential smoothing, *Journal of the American Statistical Association*,
106(496), 1513-1527.

See Also
bats, tbats, forecast.ets.

Examples
```r
## Not run:
fit <- bats(USAccDeaths)
plot(forecast(fit))

taylor.fit <- bats(taylor)
plot(forecast(taylor.fit))

## End(Not run)
```

## S3 method for class 'ets'
forecast(object, h=ifelse(object$m>1, 2+object$m, 10),
level=c(80,95), fan=FALSE, simulate=FALSE, bootstrap=FALSE,
npaths=5000, PI=TRUE, lambda=object$lambda, ...)

Arguments
- **object**: An object of class "ets". Usually the result of a call to `ets`.
- **h**: Number of periods for forecasting.
- **level**: Confidence level for prediction intervals.
- **fan**: If TRUE, level is set to seq(50,99,by=1). This is suitable for fan plots.
- **simulate**: If TRUE, prediction intervals produced by simulation rather than using analytic
  formulae.
If TRUE, and if simulate=TRUE, then simulation uses resampled errors rather than normally distributed errors.

Number of sample paths used in computing simulated prediction intervals.

If TRUE, prediction intervals are produced, otherwise only point forecasts are calculated. If PI is FALSE, then level, fan, simulate, bootstrap and npaths are all ignored.

Box-Cox transformation parameter. Ignored if NULL. Otherwise, forecasts back-transformed via an inverse Box-Cox transformation.

Other arguments.

An object of class "forecast".

The function summary is used to obtain and print a summary of the results, while the function plot produces a plot of the forecasts and prediction intervals.

The generic accessor functions fitted.values and residuals extract useful features of the value returned by forecast.ets.

An object of class "forecast" is a list containing at least the following elements:

- model: A list containing information about the fitted model
- method: The name of the forecasting method as a character string
- mean: Point forecasts as a time series
- lower: Lower limits for prediction intervals
- upper: Upper limits for prediction intervals
- level: The confidence values associated with the prediction intervals
- x: The original time series (either object itself or the time series used to create the model stored as object).
- residuals: Residuals from the fitted model. For models with additive errors, the residuals are x - fitted values. For models with multiplicative errors, the residuals are equal to x/(fitted values) - 1.
- fitted: Fitted values (one-step forecasts)

Author(s)

Rob J Hyndman

See Also

et, ses, holt, hw.

Examples

fit <- ets(USAccDeaths)
plot(forecast(fit, h=48))
**Description**

Returns forecasts and other information for univariate Holt-Winters time series models.

**Usage**

```r
## S3 method for class 'HoltWinters'
forecast(object, h=ifelse(frequency(object$x)>1,2*frequency(object$x),10),
          level=c(80,95),fan=FALSE,lambda=NULL,...)
```

**Arguments**

- `object`: An object of class "HoltWinters". Usually the result of a call to `HoltWinters`.
- `h`: Number of periods for forecasting.
- `level`: Confidence level for prediction intervals.
- `fan`: If TRUE, level is set to seq(50,99,by=1). This is suitable for fan plots.
- `lambda`: Box-Cox transformation parameter. Ignored if NULL. Otherwise, forecasts back-transformed via an inverse Box-Cox transformation.
- `...`: Other arguments.

**Details**

This function calls `predict.HoltWinters` and constructs an object of class "forecast" from the results.

It is included for completeness, but the `ets` is recommended for use instead of `HoltWinters`.

**Value**

An object of class "forecast".

The function `summary` is used to obtain and print a summary of the results, while the function `plot` produces a plot of the forecasts and prediction intervals.

The generic accessor functions `fitted` and `residuals` extract useful features of the value returned by `forecast.HoltWinters`.

An object of class "forecast" is a list containing at least the following elements:

- `model`: A list containing information about the fitted model.
- `method`: The name of the forecasting method as a character string.
- `mean`: Point forecasts as a time series.
- `lower`: Lower limits for prediction intervals.
- `upper`: Upper limits for prediction intervals.
level
- The confidence values associated with the prediction intervals.

x
- The original time series (either object itself or the time series used to create the model stored as object).

residuals
- Residuals from the fitted model. That is x minus fitted values.

fitted
- Fitted values (one-step forecasts).

Author(s)
- Rob J Hyndman

See Also

Examples
- `fit <- HoltWinters(WWWusage,gamma=FALSE)`
- `plot(forecast(fit))`

**forecast.lm**

*Forecast a linear model with possible time series components*

**Description**

`forecast.lm` is used to predict linear models, especially those involving trend and seasonality components.

**Usage**

```r
## S3 method for class 'lm'
forecast(object, newdata, h=10, level=c(80,95), fan=FALSE,
         lambda=object$lambda, ts=TRUE, ...)
```

**Arguments**

- `object` Object of class "lm", usually the result of a call to `lm` or `tslm`.
- `newdata` An optional data frame in which to look for variables with which to predict. If omitted, it is assumed that the only variables are trend and season, and h forecasts are produced.
- `level` Confidence level for prediction intervals.
- `fan` If TRUE, level is set to seq(50,99,by=1). This is suitable for fan plots.
- `h` Number of periods for forecasting. Ignored if newdata present.
- `lambda` Box-Cox transformation parameter. Ignored if NULL. Otherwise, forecasts back-transformed via an inverse Box-Cox transformation.
- `ts` If TRUE, the forecasts will be treated as time series provided the original data is a time series; the newdata will be interpreted as related to the subsequent time periods. If FALSE, any time series attributes of the original data will be ignored.
- `...` Other arguments passed to `predict.lm`.

**Note**

The confidence values associated with the prediction intervals are used to provide a measure of uncertainty around the forecast. The original time series (either object itself or the time series used to create the model stored as object) is then used to calculate predictions. Residuals from the fitted model are also provided, along with fitted values (one-step forecasts).
forecast.lm

Details

forecast.lm is largely a wrapper for predict.lm() except that it allows variables "trend" and "season" which are created on the fly from the time series characteristics of the data. Also, the output is reformatted into a forecast object.

Value

An object of class "forecast".

The function summary is used to obtain and print a summary of the results, while the function plot produces a plot of the forecasts and prediction intervals.

The generic accessor functions fitted.values and residuals extract useful features of the value returned by forecast.lm.

An object of class "forecast" is a list containing at least the following elements:

- **model**: A list containing information about the fitted model
- **method**: The name of the forecasting method as a character string
- **mean**: Point forecasts as a time series
- **lower**: Lower limits for prediction intervals
- **upper**: Upper limits for prediction intervals
- **level**: The confidence values associated with the prediction intervals
- **x**: The historical data for the response variable.
- **residuals**: Residuals from the fitted model. That is x minus fitted values.
- **fitted**: Fitted values

Author(s)

Rob J Hyndman

See Also

tslm, lm.

Examples

```r
y <- ts(rnorm(120,0,3) + 1:120 + 20*sin(2*pi*(1:120)/12), frequency=12)
fit <- tslm(y ~ trend + season)
plot(forecast(fit, h=20))
```
Description
Forecasts of STL objects are obtained by applying a non-seasonal forecasting method to the seasonally adjusted data and re-seasonalizing using the last year of the seasonal component.

Usage

```r
stlm(x, s.window=7, robust=FALSE, method=c("ets","arima"),
    modelfunction=NULL, etsmodel="ZZN", lambda=NULL, xreg=NULL,
    allow.multiplicative.trend=FALSE, ...)
```

```r
stlf(x, h=frequency(x)*2, s.window=7, t.window=NULL, robust=FALSE, lambda=NULL, ...)
```

```r
## S3 method for class 'stlm'
forecast(object, h = 2*object$m,
    level = c(80, 95), fan = FALSE, lambda=object$lambda, newxreg=NULL,
    allow.multiplicative.trend=FALSE, ...)
## S3 method for class 'stl'
forecast(object, method=c("ets","arima","naive","rwdrift"),
    etsmodel="ZZN", forecastfunction=NULL,
    h=frequency(object$time.series)*2, level=c(80,95),
    fan=FALSE, lambda=NULL, xreg=NULL, newxreg=NULL,
    allow.multiplicative.trend=FALSE, ...)
```

Arguments

- **x**: A univariate numeric time series of class `ts`
- **object**: An object of class `stl` or `stlm`. Usually the result of a call to `stl` or `stlm`.
- **method**: Method to use for forecasting the seasonally adjusted series.
- **modelfunction**: An alternative way of specifying the function for modelling the seasonally adjusted series. If `modelfunction` is not `NULL`, then `method` is ignored. Otherwise `method` is used to specify the time series model to be used.
- **forecastfunction**: An alternative way of specifying the function for forecasting the seasonally adjusted series. If `forecastfunction` is not `NULL`, then `method` is ignored. Otherwise `method` is used to specify the forecasting method to be used.
- **etsmodel**: The ets model specification passed to `ets`. By default it allows any non-seasonal model. If `method!="ets"`, this argument is ignored.
- **xreg**: Historical regressors to be used in `auto.arima()` when `method="arima"`.
- **newxreg**: Future regressors to be used in `forecast.Arima()`.
- **h**: Number of periods for forecasting.
- **level**: Confidence level for prediction intervals.
- **fan**: If `TRUE`, level is set to `seq(50,99,by=1)`. This is suitable for fan plots.
### lambda
Box-Cox transformation parameter. Ignored if NULL. Otherwise, data transformed before decomposition and back-transformed after forecasts are computed.

### s.window
Either the character string “periodic” or the span (in lags) of the loess window for seasonal extraction.

### t.window
A number to control the smoothness of the trend. See stl for details.

### robust
If TRUE, robust fitting will used in the loess procedure within stl.

### allow.multiplicative.trend
If TRUE, then ETS models with multiplicative trends are allowed. Otherwise, only additive or no trend ETS models are permitted.

... Other arguments passed to forecast.stl, modelfunction or forecastfunction.

### Details

**stlm** takes a time series \( x \), applies an STL decomposition, and models the seasonally adjusted data using the model passed as \( \text{modelfunction} \) or specified using \( \text{method} \). It returns an object that includes the original STL decomposition and a time series model fitted to the seasonally adjusted data. This object can be passed to the forecast.stlm for forecasting.

**forecast.stlm** forecasts the seasonally adjusted data, then re-seasonalizes the results by adding back the last year of the estimated seasonal component.

**stlf** combines stlm and forecast.stlm. It takes a ts argument, applies an STL decomposition, models the seasonally adjusted data, reseasonalizes, and returns the forecasts. However, it allows more general forecasting methods to be specified via forecastfunction.

**forecast.stl** is similar to stlf except that it takes the STL decomposition as the first argument, instead of the time series.

Note that the prediction intervals ignore the uncertainty associated with the seasonal component. They are computed using the prediction intervals from the seasonally adjusted series, which are then reseasonalized using the last year of the seasonal component. The uncertainty in the seasonal component is ignored.

The time series model for the seasonally adjusted data can be specified in stlm using either method or modelfunction. The method argument provides a shorthand way of specifying modelfunction for a few special cases. More generally, modelfunction can be any function with first argument a ts object, that returns an object that can be passed to forecast. For example, forecastfunction=ar uses the ar function for modelling the seasonally adjusted series.

The forecasting method for the seasonally adjusted data can be specified in stlf and forecast.stl using either method or forecastfunction. The method argument provides a shorthand way of specifying forecastfunction for a few special cases. More generally, forecastfunction can be any function with first argument a ts object, and other h and level, which returns an object of class forecast. For example, forecastfunction=thetaf uses the thetaf function for forecasting the seasonally adjusted series.

### Value

**stlm** returns an object of class stlm. The other functions return objects of class forecast.
There are many methods for working with `forecast` objects including `summary` to obtain and print a summary of the results, while `plot` produces a plot of the forecasts and prediction intervals. The generic accessor functions `fitted.values` and `residuals` extract useful features.

**Author(s)**

Rob J Hyndman

**See Also**

`stl`, `forecast.ets`, `forecast.Arima`.

**Examples**

```r
# Fit a seasonal model
mod <- stlm(USAccDeaths, modelfunction=ar)
plot(forecast(mod, h=36))

plot(stlf(AirPassengers, lambda=0))

decom <- stlm(USAccDeaths, s.window="periodic")
plot(forecast(decom))
```

### `forecast.StructTS`

**Forecasting using Structural Time Series models**

**Description**

Returns forecasts and other information for univariate structural time series models.

**Usage**

```r
# S3 method for class 'StructTS'
forecast(object, 
  h=ifelse(object$coef["epsilon"] > 1e-10, 2*object$xtsp[3],10), 
  level=c(80,95), fan=FALSE, lambda=NULL, ...)
```

**Arguments**

- `object`: An object of class "StructTS". Usually the result of a call to `StructTS`.
- `h`: Number of periods for forecasting.
- `level`: Confidence level for prediction intervals.
- `fan`: If TRUE, level is set to seq(50,99,by=1). This is suitable for fan plots.
- `lambda`: Box-Cox transformation parameter. Ignored if NULL. Otherwise, forecasts back-transformed via an inverse Box-Cox transformation.
- `...`: Other arguments.
Details

This function calls predict.StructTS and constructs an object of class "forecast" from the results.

Value

An object of class "forecast".

The function summary is used to obtain and print a summary of the results, while the function plot produces a plot of the forecasts and prediction intervals.

The generic accessor functions fitted.values and residuals extract useful features of the value returned by forecast.StructTS.

An object of class "forecast" is a list containing at least the following elements:

- **model**: A list containing information about the fitted model
- **method**: The name of the forecasting method as a character string
- **mean**: Point forecasts as a time series
- **lower**: Lower limits for prediction intervals
- **upper**: Upper limits for prediction intervals
- **level**: The confidence values associated with the prediction intervals
- **x**: The original time series (either object itself or the time series used to create the model stored as object).
- **residuals**: Residuals from the fitted model. That is x minus fitted values.
- **fitted**: Fitted values (one-step forecasts)

Author(s)

Rob J Hyndman

See Also

- StructTS.

Examples

```r
fit <- StructTS(WWWusage,"level")
plot(forecast(fit))
```
**Description**


**Usage**

gas

**Format**

Time series data

**Source**

Australian Bureau of Statistics.

**Examples**

plot(gas)
seasonplot(gas)
tsdisplay(gas)

---

**getResponse**

Get response variable from time series model.

**Description**

getResponse is a generic function for extracting the historical data from a time series model (including Arima, ets, ar, fracdiff), a linear model of class lm, or a forecast object. The function invokes particular methods which depend on the class of the first argument.

**Usage**

getResponse(object,...)

**Arguments**

object a time series model or forecast object.

... Additional arguments that are ignored.

**Value**

A numerical vector or a time series object of class ts.
gold

Author(s)
Rob J Hyndman

gold Daily morning gold prices

Description

Usage
data(gold)

Format
Time series data

Source
Time Series Data Library. http://data.is/TSDLdemo

Examples
tsdisplay(gold)

logLik.ets Log-Likelihood of an ets object

Description
Returns the log-likelihood of the ets model represented by object evaluated at the estimated parameters.

Usage
## S3 method for class 'ets'
logLik(object, ...)

Arguments
object an object of class ets, representing an exponential smoothing state space model.
... some methods for this generic require additional arguments. None are used in this method.
Value

the log-likelihood of the model represented by object evaluated at the estimated parameters.

Author(s)

Rob J Hyndman

References


See Also

ets

Examples

```r
fit <- ets(USAccDeaths)
logLik(fit)
```

---

**ma**

*Moving-average smoothing*

Description

Computes a simple moving average smoother.

Usage

`ma(x, order, centre=TRUE)`

Arguments

- `x` : Univariate time series
- `order` : Order of moving average smoother
- `centre` : If TRUE, then the moving average is centred.

Value

Numerical time series object containing the smoothed values.

Author(s)

Rob J Hyndman
meanf

Description

Returns forecasts and prediction intervals for an iid model applied to x.

Usage

meanf(x, h=10, level=c(80,95), fan=FALSE, lambda=NULL)

Arguments

x a numeric vector or time series
h Number of periods for forecasting
level Confidence levels for prediction intervals.
fan If TRUE, level is set to seq(50,99,by=1). This is suitable for fan plots.
lambda Box-Cox transformation parameter. Ignored if NULL. Otherwise, forecasts
back-transformed via an inverse Box-Cox transformation.

Details

The iid model is

\[ Y_t = \mu + Z_t \]

where \( Z_t \) is a normal iid error. Forecasts are given by

\[ Y_n(h) = \mu \]

where \( \mu \) is estimated by the sample mean.
Value

An object of class "forecast".

The function summary is used to obtain and print a summary of the results, while the function plot produces a plot of the forecasts and prediction intervals.

The generic accessor functions fitted.values and residuals extract useful features of the value returned by meanf.

An object of class "forecast" is a list containing at least the following elements:

- **model**: A list containing information about the fitted model
- **method**: The name of the forecasting method as a character string
- **mean**: Point forecasts as a time series
- **lower**: Lower limits for prediction intervals
- **upper**: Upper limits for prediction intervals
- **level**: The confidence values associated with the prediction intervals
- **x**: The original time series (either object itself or the time series used to create the model stored as object).
- **residuals**: Residuals from the fitted model. That is x minus fitted values.
- **fitted**: Fitted values (one-step forecasts)

Author(s)

Rob J Hyndman

See Also

rwf

Examples

```r
nile.fcast <- meanf(Nile, h=10)
plot(nile.fcast)
```

Description

Returns number of days in each month or quarter of the observed time period.

Usage

```r
monthdays(x)
```
msts

Multi-Seasonal Time Series

Description

msts is an S3 class for multi seasonal time series objects, intended to be used for models that support multiple seasonal periods. The msts class inherits from the ts class and has an additional "msts" attribute which contains the vector of seasonal periods. All methods that work on a ts class, should also work on a msts class.

Usage

msts(data, seasonal.periods, ts.frequency=floor(max(seasonal.periods)),
     ... )
Arguments

data A numeric vector, ts object, matrix or data frame. It is intended that the time series data is univariate, otherwise treated the same as ts().

seasonal.periods A vector of the seasonal periods of the msts.

ts.frequency The seasonal periods that should be used as frequency of the underlying ts object. The default value is max(seasonal.periods).

... Arguments to be passed to the underlying call to ts(). For example start=c(1987,5).

Value

An object of class c("msts", "ts").

Author(s)

Slava Razbash and Rob J Hyndman

Examples

x <- msts(taylor, seasonal.periods=c(48,336), ts.frequency=48, start=2000+22/52)
y <- msts(USAccDeaths, seasonal.periods=12, ts.frequency=12, start=1949)

Description

Uses linear interpolation for non-seasonal series and a periodic stl decomposition with seasonal series to replace missing values.

Usage

na.interp(x, lambda = NULL)

Arguments

x time series

lambda a numeric value suggesting Box-cox transformation

Details

A more general and flexible approach is available using na.approx in the zoo package.

Value

Time series
\textit{naive} \quad 49

\textbf{Author(s)}

Rob J Hyndman

\textbf{See Also}

\texttt{na.interp}, \texttt{tsoutliers}

\textbf{Examples}

\begin{verbatim}
data(gold)
plot(na.interp(gold))
\end{verbatim}

\begin{tabular}{ll}
\textbf{naive} & \textit{Naive forecasts} \\
\end{tabular}

\textbf{Description}

\texttt{naive()} returns forecasts and prediction intervals for an ARIMA(0,1,0) random walk model applied to \texttt{x}. \texttt{snaive()} returns forecasts and prediction intervals from an ARIMA(0,0,0)(0,1,0) model where \textit{m} is the seasonal period.

\textbf{Usage}

\begin{verbatim}
naive(x, h=10, level=c(80,95), fan=FALSE, lambda=NULL)
snaive(x, h=2*frequency(x), level=c(80,95), fan=FALSE, lambda=NULL)
\end{verbatim}

\textbf{Arguments}

\begin{verbatim}
x \quad \text{a numeric vector or time series}
h \quad \text{Number of periods for forecasting}
level \quad \text{Confidence levels for prediction intervals.}
fan \quad \text{If TRUE, level is set to seq(50,99,by=1). This is suitable for fan plots.}
lambda \quad \text{Box-Cox transformation parameter. Ignored if NULL. Otherwise, forecasts back-transformed via an inverse Box-Cox transformation.}
\end{verbatim}

\textbf{Details}

These functions are simply convenient wrappers to \texttt{Arima} with the appropriate arguments to return naive and seasonal naive forecasts.
Value

An object of class "forecast".

The function `summary` is used to obtain and print a summary of the results, while the function `plot` produces a plot of the forecasts and prediction intervals.

The generic accessor functions `fitted` and `residuals` extract useful features of the value returned by `naive` or `snaive`.

An object of class "forecast" is a list containing at least the following elements:

- `model`: A list containing information about the fitted model
- `method`: The name of the forecasting method as a character string
- `mean`: Point forecasts as a time series
- `lower`: Lower limits for prediction intervals
- `upper`: Upper limits for prediction intervals
- `level`: The confidence values associated with the prediction intervals
- `x`: The original time series (either object itself or the time series used to create the model stored as object).
- `residuals`: Residuals from the fitted model. That is x minus fitted values.
- `fitted`: Fitted values (one-step forecasts)

Author(s)

Rob J Hyndman

See Also

`Arima`, `rwf`

Examples

```r
plot(naive(gold,h=50),include=200)
plot(snaive(wineind))
```

---

### ndiffs

**Number of differences required for a stationary series**

Description

Functions to estimate the number of differences required to make a given time series stationary. `ndiffs` estimates the number of first differences and `nsdiffs` estimates the number of seasonal differences.

Usage

```r
ndiffs(x, alpha=0.05, test=c("kpss","adf","pp"), max.d=2)
nsdiffs(x, m=frequency(x), test=c("ocsb","ch"), max.D=1)
```
Arguments

- **x**: A univariate time series
- **alpha**: Level of the test
- **m**: Length of seasonal period
- **test**: Type of unit root test to use
- **max.d**: Maximum number of non-seasonal differences allowed
- **max.D**: Maximum number of seasonal differences allowed

Details

`ndiffs` uses a unit root test to determine the number of differences required for time series `x` to be made stationary. If `test="kpss"`, the KPSS test is used with the null hypothesis that `x` has a stationary root against a unit-root alternative. Then the test returns the least number of differences required to pass the test at the level `alpha`. If `test="adf"`, the Augmented Dickey-Fuller test is used and if `test="pp"` the Phillips-Perron test is used. In both of these cases, the null hypothesis is that `x` has a unit root against a stationary root alternative. Then the test returns the least number of differences required to fail the test at the level `alpha`.

`nsdiffs` uses seasonal unit root tests to determine the number of seasonal differences required for time series `x` to be made stationary (possibly with some lag-one differencing as well). If `test="ch"`, the Canova-Hansen (1995) test is used (with null hypothesis of deterministic seasonality) and if `test="ocsb"`, the Osborn-Chui-Smith-Birchenhall (1988) test is used (with null hypothesis that a seasonal unit root exists).

Value

An integer.

Author(s)

Rob J Hyndman and Slava Razbash

References


See Also

auto.arima

Examples

ndiffs WWWusage
nsdiffs(log(AirPassengers))
ndiffs(diff(log(AirPassengers),12))

nnetar Neural Network Time Series Forecasts

Description

Feed-forward neural networks with a single hidden layer and lagged inputs for forecasting univariate

time series.

Usage

nnetar(x, p, P=1, size, repeats=20, lambda=NULL)

## S3 method for class 'nnetar'
forecast(object, h=ifelse(object$m > 1, 2 * object$m, 10),
         lambda=object$lambda, ...)

Arguments

x a numeric vector or time series
p Embedding dimension for non-seasonal time series. Number of non-seasonal
   lags used as inputs. For non-seasonal time series, the default is the optimal
   number of lags (according to the AIC) for a linear AR(p) model. For seasonal
   time series, the same method is used but applied to seasonally adjusted data
   (from an stl decomposition).
P Number of seasonal lags used as inputs.
size Number of nodes in the hidden layer. Default is half of the number of input
      nodes plus 1.
repeats Number of networks to fit with different random starting weights. These are
         then averaged when producing forecasts.
lambda Box-Cox transformation parameter.
object An object of class nnetar generated by nnetar.
h Number of periods for forecasting.
... Other arguments.
A feed-forward neural network is fitted with lagged values of \( x \) as inputs and a single hidden layer with size \( \text{nodes} \). The inputs are for lags 1 to \( p \), and lags \( m \) to \( mp \) where \( m = \text{frequency}(x) \). A total of repeats networks are fitted, each with random starting weights. These are then averaged when computing forecasts. The network is trained for one-step forecasting. Multi-step forecasts are computed recursively. The fitted model is called an NNAR(p,P) model and is analogous to an ARIMA(p,0,0)(P,0,0) model but with nonlinear functions.

Value

\text{nnetar} returns an object of class "nnetar". \text{forecast.nnetar} returns an object of class "forecast". The function \text{summary} is used to obtain and print a summary of the results, while the function \text{plot} produces a plot of the forecasts.

The generic accessor functions \text{fitted} and \text{residuals} extract useful features of the value returned by \text{nnetar}.

An object of class "forecast" is a list containing at least the following elements:

- \text{model} A list containing information about the fitted model
- \text{method} The name of the forecasting method as a character string
- \text{mean} Point forecasts as a time series
- \text{x} The original time series (either \text{object} itself or the time series used to create the model stored as \text{object}).
- \text{residuals} Residuals from the fitted model. That is \text{x} minus fitted values.
- \text{fitted} Fitted values (one-step forecasts)
- ... Other arguments

Author(s)

Rob J Hyndman

Examples

\begin{verbatim}
fit <- nnetar(lynx)
fcast <- forecast(fit)
plot(fcast)
\end{verbatim}

Description

Produces a plot of the inverse AR and MA roots of an ARIMA model. Inverse roots outside the unit circle are shown in red.
Usage

```r
## S3 method for class 'Arima'
plot(x, type=c("both","ar","ma"),
     main, xlab="Real", ylab="Imaginary", ...)
## S3 method for class 'ar'
plot(x, main, xlab="Real", ylab="Imaginary", ...)
```

Arguments

- `x` Object of class “Arima” or “ar”.
- `type` Determines if both AR and MA roots are plotted, of if just one set is plotted.
- `main` Main title. Default is “Inverse AR roots” or “Inverse MA roots”.
- `xlab` X-axis label.
- `ylab` Y-axis label.
- `...` Other plotting parameters passed to `par`.

Value

None. Function produces a plot.

Author(s)

Rob J Hyndman

See Also

`Arima`, `ar`

Examples

```r
fit <- Arima(WWusage, order=c(3,1,0))
plot(fit)

fit <- Arima(woolyrnq,order=c(2,0,0),seasonal=c(2,1,1))
plot(fit)

plot(ar.ols(gold[1:61]))
```
plot.bats

### Description

Produces a plot of the level, slope and seasonal components from a BATS or TBATS model.

### Usage

```r
## S3 method for class 'bats'
plot(x, main="Decomposition by BATS model", ...) 
## S3 method for class 'tbats'
plot(x, main="Decomposition by TBATS model", ...)
```

### Arguments

- `x`: Object of class “ets”.
- `main`: Main title for plot.
- `...`: Other plotting parameters passed to `par`.

### Value

None. Function produces a plot

### Author(s)

Rob J Hyndman

### See Also

- `bats`
- `tbats`

### Examples

```r
## Not run:
fit <- tbats(USAccDeaths)
plot(fit)
## End(Not run)
```
### plot.ets

**Plot components from ETS model**

**Description**

Produces a plot of the level, slope and seasonal components from an ETS model.

**Usage**

```r
## S3 method for class 'ets'
plot(x, ...)
```

**Arguments**

- `x` Object of class “ets”.
- `...` Other plotting parameters passed to `par`.

**Value**

None. Function produces a plot

**Author(s)**

Rob J Hyndman

**See Also**

`ets`

**Examples**

```r
fit <- ets(USAccDeaths)
plot(fit)
plot(fit, plot.type="single", ylab="", col=1:3)
```

---

### plot.forecast

**Forecast plot**

**Description**

Plots historical data with forecasts and prediction intervals.
Usage

```r
## S3 method for class 'forecast'
plot(x, include, plot.conf=TRUE, shaded=TRUE,
     shadebars=(length(x$mean)<5), shadecols=NULL, col=1, fcol=4,
     pi.col=1, pi.lty=2, ylim=NULL, main=NULL, xlab='', ylab='', type='l',
     f1ty=1, flwd=2, ...)

## S3 method for class 'splineforecast'
plot(x, fitcol=2, type="o", pch=19, ...)
```

Arguments

- **x**  
  Forecast object produced by `forecast`.
- **include**  
  Number of values from time series to include in plot.
- **plot.conf**  
  Logical flag indicating whether to plot prediction intervals.
- **shaded**  
  Logical flag indicating whether prediction intervals should be shaded (TRUE) or lines (FALSE).
- **shadebars**  
  Logical flag indicating if prediction intervals should be plotted as shaded bars (if TRUE) or a shaded polygon (if FALSE). Ignored if `shaded=FALSE`. Bars are plotted by default if there are fewer than five forecast horizons.
- **shadecols**  
  Colors for shaded prediction intervals. To get default colors used prior to v3.26, set `shadecols="oldstyle"`.
- **col**  
  Colour for the data line.
- **fcol**  
  Colour for the forecast line.
- **flty**  
  Line type for the forecast line.
- **flwd**  
  Line width for the forecast line.
- **pi.col**  
  If `shade=FALSE` and `plot.conf=TRUE`, the prediction intervals are plotted in this colour.
- **pi.lty**  
  If `shade=FALSE` and `plot.conf=TRUE`, the prediction intervals are plotted using this line type.
- **ylim**  
  Limits on y-axis.
- **main**  
  Main title.
- **xlab**  
  X-axis label.
- **ylab**  
  Y-axis label.
- **fitcol**  
  Line colour for fitted values.
- **type**  
  1-character string giving the type of plot desired. As for `plot.default`.
- **pch**  
  Plotting character (if type="p" or type="o").
  ...
  Additional arguments to `plot`.

Value

None.
Author(s)
Rob J Hyndman

References

See Also
plot.ts

Examples

deads.fit <- hw(USAccDeaths,h=48)
plot(deads.fit)

---

**rwf**  
*Random Walk Forecast*

**Description**

Returns forecasts and prediction intervals for a random walk with drift model applied to x.

**Usage**

`rwf(x, h=10, drift=FALSE, level=c(80,95), fan=FALSE, lambda=NULL)`

**Arguments**

- `x`: a numeric vector or time series
- `h`: Number of periods for forecasting
- `drift`: Logical flag. If TRUE, fits a random walk with drift model.
- `level`: Confidence levels for prediction intervals.
- `fan`: If TRUE, level is set to seq(50,99,by=1). This is suitable for fan plots.
- `lambda`: Box-Cox transformation parameter. Ignored if NULL. Otherwise, forecasts back-transformed via an inverse Box-Cox transformation.

**Details**

The random walk with drift model is

\[ Y_t = c + Y_{t-1} + Z_t \]

where \( Z_t \) is a normal iid error. Forecasts are given by

\[ Y_n(h) = ch + Y_n \]

If there is no drift, the drift parameter \( c=0 \). Forecast standard errors allow for uncertainty in estimating the drift parameter.
Value

An object of class "forecast".

The function `summary` is used to obtain and print a summary of the results, while the function `plot` produces a plot of the forecasts and prediction intervals.

The generic accessor functions `fitted` and `residuals` extract useful features of the value returned by `rwf`.

An object of class "forecast" is a list containing at least the following elements:

- **model**: A list containing information about the fitted model
- **method**: The name of the forecasting method as a character string
- **mean**: Point forecasts as a time series
- **lower**: Lower limits for prediction intervals
- **upper**: Upper limits for prediction intervals
- **level**: The confidence values associated with the prediction intervals
- **x**: The original time series (either `object` itself or the time series used to create the model stored as `object`).
- **residuals**: Residuals from the fitted model. That is `x` minus fitted values.
- **fitted**: Fitted values (one-step forecasts)

Author(s)

Rob J Hyndman

See Also

- `arima`, `meanf`

Examples

```r
gold.fcast <- rwf(gold[1:60], h=50)
plot(gold.fcast)
```

---

seasadj  

*Seasonal adjustment*

Description

Returns seasonally adjusted data constructed by removing the seasonal component.

Usage

```r
seasadj(object)
```
Arguments

object Object created by `decompose`, `stl` or `tbats`.

Value

Univariate time series.

Author(s)

Rob J Hyndman

See Also

`stl`, `decompose`, `tbats`.

Examples

```r
plot(AirPassengers)
lines(seasadj(decompose(AirPassengers,"multiplicative")),col=4)
```

---

**seasonaldummy**  
*Seasonal dummy variables*

Description

`seasonaldummy` and `seasonaldummyf` return matrices of dummy variables suitable for use in `arima`, `lm` or `tslm`. The last season is omitted and used as the control.

`fourier` and `fourierf` return matrices containing terms from a Fourier series, up to order K, suitable for use in `arima`, `lm` or `tslm`.

Usage

```r
seasonaldummy(x)
seasonaldummyf(x,h)
fourier(x,K)
fourierf(x,K,h)
```

Arguments

- **x**: Seasonal time series: a `ts` or a `msts` object
- **h**: Number of periods ahead to forecast
- **K**: Maximum order(s) of Fourier terms
Details

The number of dummy variables, or the period of the Fourier terms, is determined from the time series characteristics of \( x \). The length of \( x \) also determines the number of rows for the matrices returned by `seasonaldummy` and `fourier`. The value of \( h \) determines the number of rows for the matrices returned by `seasonaldummyf` and `fourierf`. The values within \( x \) are not used in any function.

When \( x \) is a `ts` object, the value of \( K \) should be an integer and specifies the number of sine and cosine terms to return. Thus, the matrix returned has \( 2K \) columns.

When \( x \) is a `msts` object, then \( K \) should be a vector of integers specifying the number of sine and cosine terms for each of the seasonal periods. Then the matrix returned will have \( 2*\text{sum}(K) \) columns.

Value

Numerical matrix.

Author(s)

Rob J Hyndman

Examples

```r
plot(Ideaths)

# Using seasonal dummy variables
month <- seasonaldummy(Ideaths)
deaths.lm <- tslm(Ideaths ~ month)
tsdisplay(residuals(deaths.lm))
Idantas.fcast <- forecast(deaths.lm, 
    data.frame(month=I(seasonaldummyf(Ideaths,36))))
plot(Idantas.fcast)

# A simpler approach to seasonal dummy variables
deaths.lm <- tslm(Ideaths ~ season)
Idantas.fcast <- forecast(deaths.lm, h=36)
plot(Idantas.fcast)

# Using Fourier series
X <- fourier(Ideaths,3)
deaths.lm <- tslm(Ideaths ~ X)
Idantas.fcast <- forecast(deaths.lm, 
    data.frame(X=I(fourierf(Ideaths,3,36))))
plot(Idantas.fcast)

# Using Fourier series for a "msts" object
Z <- fourier(taylor, K = c(3, 3))
taylor.lm <- tslm(taylor ~ Z)
taylor.fcast <- forecast(taylor.lm, 
    data.frame(Z = I(fourierf(taylor, K = c(3, 3), h = 270))))
plot(taylor.fcast)
```
seasonplot  

**Seasonal plot**

**Description**

Plots a seasonal plot as described in Hyndman and Athanasopoulos (2014, chapter 2).

**Usage**

```r
seasonplot(x, s, season.labels=NULL, year.labels=FALSE,
year.labels.left=FALSE, type="o", main, xlab=NULL, ylab="",
col=1, labelgap=0.1, ...)  
```

**Arguments**

- `x`  
  a numeric vector or time series.
- `s`  
  seasonal frequency of x
- `season.labels`  
  Labels for each season in the "year"
- `year.labels`  
  Logical flag indicating whether labels for each year of data should be plotted on the right.
- `year.labels.left`  
  Logical flag indicating whether labels for each year of data should be plotted on the left.
- `type`  
  plot type (as for `plot`)
- `main`  
  Main title.
- `xlab`  
  X-axis label.
- `ylab`  
  Y-axis label.
- `col`  
  Colour
- `labelgap`  
  Distance between year labels and plotted lines
- `...` additional arguments to `plot`.

**Value**

None.

**Author(s)**

Rob J Hyndman

**References**

**ses**

*Exponential smoothing forecasts*

**Description**

Returns forecasts and other information for exponential smoothing forecasts applied to x.

**Usage**

```r
ses(x, h=10, level=c(80,95), fan=FALSE,
    initial=c("optimal","simple"), alpha=NULL, ...)
```

```r
holt(x, h=10, damped=FALSE, level=c(80,95), fan=FALSE,
    initial=c("optimal","simple"), exponential=FALSE,
    alpha=NULL, beta=NULL, ...)
```

```r
hw(x, h=2*frequency(x), seasonal="additive", damped=FALSE,
    level=c(80,95), fan=FALSE, initial=c("optimal","simple"),
    exponential=FALSE, alpha=NULL, beta=NULL, gamma=NULL, ...)
```

**Arguments**

- **x**: a numeric vector or time series
- **h**: Number of periods for forecasting.
- **damped**: If TRUE, use a damped trend.
- **seasonal**: Type of seasonality in hw model. "additive" or "multiplicativie"
- **level**: Confidence level for prediction intervals.
- **fan**: If TRUE, level is set to seq(50,99,by=1). This is suitable for fan plots.
- **initial**: Method used for selecting initial state values. If optimal, the initial values are optimized along with the smoothing parameters using ets. If simple, the initial values are set to values obtained using simple calculations on the first few observations. See Hyndman & Athanasopoulos (2014) for details.
- **exponential**: If TRUE, an exponential trend is fitted. Otherwise, the trend is (locally) linear.
- **alpha**: Value of smoothing parameter for the level. If NULL, it will be estimated.
- **beta**: Value of smoothing parameter for the trend. If NULL, it will be estimated.
- **gamma**: Value of smoothing parameter for the seasonal component. If NULL, it will be estimated.
- **...**: Other arguments passed to forecast.ets.

**Examples**

```r
seasonplot(AirPassengers,col=rainbow(12),year.labels=TRUE)
```
Details

ses, holt and hw are simply convenient wrapper functions for forecast(ets(...)).

Value

An object of class "forecast".

The function summary is used to obtain and print a summary of the results, while the function plot produces a plot of the forecasts and prediction intervals.

The generic accessor functions fitted, values and residuals extract useful features of the value returned by ets and associated functions.

An object of class "forecast" is a list containing at least the following elements:

- **model**: A list containing information about the fitted model
- **method**: The name of the forecasting method as a character string
- **mean**: Point forecasts as a time series
- **lower**: Lower limits for prediction intervals
- **upper**: Upper limits for prediction intervals
- **level**: The confidence values associated with the prediction intervals
- **x**: The original time series (either object itself or the time series used to create the model stored as object).
- **residuals**: Residuals from the fitted model. That is x minus fitted values.
- **fitted**: Fitted values (one-step forecasts)

Author(s)

Rob J Hyndman

References


See Also

ets, HoltWinters, rwf, arima.

Examples

```r
fcast <- holt(airmiles)
plot(fcast)
deaths.fcast <- hw(USAccDeaths,h=48)
plot(deaths.fcast)
```
**simulate.ets**

*Simulation from a time series model*

**Description**

Returns a time series based on the model object object.

**Usage**

```r
## S3 method for class 'ets'
simulate(object, nsim=length(object$x), seed=NULL, future=TRUE,
     bootstrap=FALSE, innov=NULL, ...)
## S3 method for class 'ar'
simulate(object, nsim=object$n.used, seed=NULL, future=TRUE,
     bootstrap=FALSE, innov=NULL, ...)
## S3 method for class 'Arima'
simulate(object, nsim=length(object$x), seed=NULL, xreg=NULL, future=TRUE,
     bootstrap=FALSE, innov=NULL, lambda=object$lambda, ...)
## S3 method for class 'fracdiff'
simulate(object, nsim=object$n, seed=NULL, future=TRUE,
     bootstrap=FALSE, innov=NULL, ...)
```

**Arguments**

- **object**
  An object of class "ets", "Arima" or "ar".
- **nsim**
  Number of periods for the simulated series
- **seed**
  Either NULL or an integer that will be used in a call to `set.seed` before simulating the time series. The default, NULL will not change the random generator state.
- **future**
  Produce sample paths that are future to and conditional on the data in object.
- **bootstrap**
  If TRUE, simulation uses resampled errors rather than normally distributed errors.
- **innov**
  A vector of innovations to use as the error series. If present, bootstrap and seed are ignored.
- **xreg**
  New values of xreg to be used for forecasting. Must have nsim rows.
- **lambda**
  Box-Cox parameter. If not NULL, the simulated series is transformed using an inverse Box-Cox transformation with parameter `lambda`.
- **...**
  Other arguments.

**Details**

With `simulate.Arima()`, the object should be produced by `Arima` or `auto.arima`, rather than `arima`. By default, the error series is assumed normally distributed and generated using `rnorm`. If `innov` is present, it is used instead. If `bootstrap=TRUE` and `innov=NULL`, the residuals are resampled instead.
When `future=TRUE`, the sample paths are conditional on the data. When `future=FALSE` and the model is stationary, the sample paths do not depend on the data at all. When `future=FALSE` and the model is non-stationary, the location of the sample paths is arbitrary, so they all start at the value of the first observation.

**Value**

An object of class "ts".

**Author(s)**

Rob J Hyndman

**See Also**

`ets`, `Arima`, `auto.arima`, `ar`, `arfima`.

**Examples**

```r
fit <- ets(USAccDeaths)
plot(USAccDeaths, xlim=c(1973,1982))
lines(simulate(fit, 36), col="red")
```

---

### sindexf

**Forecast seasonal index**

**Description**

Returns vector containing the seasonal index for h future periods. If the seasonal index is non-periodic, it uses the last values of the index.

**Usage**

`sindexf(object, h)`

**Arguments**

- `object`: Output from `decompose` or `stl`.
- `h`: Number of periods ahead to forecast

**Value**

Time series

**Author(s)**

Rob J Hyndman
Examples

```r
uk.stl <- stl(UKDriverDeaths,"periodic")
uk.sa <- seasadj(uk.stl)
uk.fcast <- holt(uk.sa,36)
seasf <- sindexf(uk.stl,36)
uk.fcast$mean <- uk.fcast$mean + seasf
uk.fcast$lower <- uk.fcast$lower + cbind(seasf,seasf)
uk.fcast$upper <- uk.fcast$upper + cbind(seasf,seasf)
uk.fcast$x <- UKDriverDeaths
plot(uk.fcast,main="Forecasts from Holt’s method with seasonal adjustment")
```

---

**splinef**

**Cubic Spline Forecast**

### Description

Returns local linear forecasts and prediction intervals using cubic smoothing splines.

### Usage

```r
splinef(x, h=10, level=c(80,95), fan=FALSE, lambda=NULL, method=c("gcv","mle"))
```

### Arguments

- **x**: a numeric vector or time series
- **h**: Number of periods for forecasting
- **level**: Confidence level for prediction intervals.
- **fan**: If TRUE, level is set to seq(50,99,by=1). This is suitable for fan plots.
- **lambda**: Box-Cox transformation parameter. Ignored if NULL. Otherwise, forecasts back-transformed via an inverse Box-Cox transformation.
- **method**: Method for selecting the smoothing parameter. If method="gcv", the generalized cross-validation method from `smooth.spline` is used. If method="mle", the maximum likelihood method from Hyndman et al (2002) is used.

### Details

The cubic smoothing spline model is equivalent to an ARIMA(0,2,2) model but with a restricted parameter space. The advantage of the spline model over the full ARIMA model is that it provides a smooth historical trend as well as a linear forecast function. Hyndman, King, Pitrun, and Billah (2002) show that the forecast performance of the method is hardly affected by the restricted parameter space.
**Value**

An object of class "forecast".

The function `summary` is used to obtain and print a summary of the results, while the function `plot` produces a plot of the forecasts and prediction intervals.

The generic accessor functions `fitted.values` and `residuals` extract useful features of the value returned by `splinef`.

An object of class "forecast" containing the following elements:

- `model`: A list containing information about the fitted model
- `method`: The name of the forecasting method as a character string
- `mean`: Point forecasts as a time series
- `lower`: Lower limits for prediction intervals
- `upper`: Upper limits for prediction intervals
- `level`: The confidence values associated with the prediction intervals
- `x`: The original time series (either object itself or the time series used to create the model stored as `object`).
- `onestepf`: One-step forecasts from the fitted model.
- `fitted`: Smooth estimates of the fitted trend using all data.
- `residuals`: Residuals from the fitted model. That is `x` minus one-step forecasts.

**Author(s)**

Rob J Hyndman

**References**


**See Also**

`smooth.spline`, `arima`, `holt`.

**Examples**

```r
fcast <- splinef(uspop,h=5)
plot(fcast)
s summarize(fcast)
```
Description

The main purpose of this function is to extract the values of a specific season in each year. For example, to extract all values for the month of May from a time series.

Usage

```r
## S3 method for class 'ts'
subset(x, subset=NULL, month=NULL, quarter=NULL, season=NULL, ...)
```

Arguments

- `x`: a univariate time series to be subsetted
- `subset`: optional logical expression indicating elements to keep; missing values are taken as false.
- `month`: Character list of months to retain. Partial matching on month names used.
- `quarter`: Numeric list of quarters to retain.
- `season`: Numeric list of seasons to retain.
- `...`: Other arguments, unused.

Value

If one season per year is extracted, then a ts object is returned with frequency 1. Otherwise, a numeric vector is returned with no ts attributes.

Author(s)

Rob J Hyndman

See Also

`subset`

Examples

```r
plot(subset(gas,month="November"))
subset(woolyrnq,quarter=3)
```
**taylor**  
*Half-hourly electricity demand*

**Description**


**Usage**

taylor

**Format**

Time series data

**Source**

James W Taylor

**References**


**Examples**

```r
plot(taylor)
```

---

**tbats**  
*TBATS model (Exponential smoothing state space model with Box-Cox transformation, ARMA errors, Trend and Seasonal components)*

**Description**

Fits a TBATS model applied to \( y \), as described in De Livera, Hyndman & Snyder (2011). Parallel processing is used by default to speed up the computations.

**Usage**

tbats(y, use.box.cox=NULL, use.trend=NULL, use.damped.trend=NULL, seasonal.periods=NULL, use.arma.errors=TRUE, use.parallel=TRUE, num.cores=2, bc.lower=0, bc.upper=1, ...)

Arguments

\texttt{y} \hspace{1cm} \text{The time series to be forecast. Can be numeric, msts or ts. Only univariate time series are supported.}

\texttt{use.box.cox} \hspace{1cm} \text{TRUE/FALSE indicates whether to use the Box-Cox transformation or not. If NULL then both are tried and the best fit is selected by AIC.}

\texttt{use.trend} \hspace{1cm} \text{TRUE/FALSE indicates whether to include a trend or not. If NULL then both are tried and the best fit is selected by AIC.}

\texttt{use.damped.trend} \hspace{1cm} \text{TRUE/FALSE indicates whether to include a damping parameter in the trend or not. If NULL then both are tried and the best fit is selected by AIC.}

\texttt{seasonal.periods} \hspace{1cm} \text{If y is numeric then seasonal periods can be specified with this parameter.}

\texttt{use.arma.errors} \hspace{1cm} \text{TRUE/FALSE indicates whether to include ARMA errors or not. If TRUE the best fit is selected by AIC. If FALSE then the selection algorithm does not consider ARMA errors.}

\texttt{use.parallel} \hspace{1cm} \text{TRUE/FALSE indicates whether or not to use parallel processing.}

\texttt{num.cores} \hspace{1cm} \text{The number of parallel processes to be used if using parallel processing. If NULL then the number of logical cores is detected and all available cores are used.}

\texttt{bc.lower} \hspace{1cm} \text{The lower limit (inclusive) for the Box-Cox transformation.}

\texttt{bc.upper} \hspace{1cm} \text{The upper limit (inclusive) for the Box-Cox transformation.}

... \hspace{1cm} \text{Additional arguments to be passed to \texttt{auto.arima} when choose an ARMA(p, q) model for the errors. (Note that \texttt{xreg} will be ignored, as will any arguments concerning seasonality and differencing, but arguments controlling the values of p and q will be used.)}

Value

An object with class \texttt{c("tbats", "bats"). The generic accessor functions \texttt{fitted.values} and \texttt{residuals} extract useful features of the value returned by \texttt{bats} and associated functions. The fitted model is designated TBATS(omega, p,q, phi, \langle m1,k1\rangle,\ldots,\langle mJ,kJ\rangle) where omega is the Box-Cox parameter and phi is the damping parameter; the error is modelled as an ARMA(p,q) process and m1,\ldots,mJ list the seasonal periods used in the model and k1,\ldots,kJ are the corresponding number of Fourier terms used for each seasonality.

Author(s)

Slava Razbash and Rob J Hyndman

References

See Also

tbats.components.

Examples

```r
## Not run:
fit <- tbats(USAccDeaths, use.parallel=FALSE)
plot(forecast(fit))

taylor.fit <- tbats(taylor)
plot(forecast(taylor.fit))
## End(Not run)
```

tbats.components  Extract components of a TBATS model

Description

Extract the level, slope and seasonal components of a TBATS model.

Usage

tbats.components(x)

Arguments

x A tbats object created by `tbats`.

Value

A multiple time series (mts) object.

Author(s)

Slava Razbash and Rob J Hyndman

References


See Also

tbats.
The `thetaf` function returns forecasts and prediction intervals for a theta method forecast. The theta method of Assimakopoulos and Nikolopoulos (2000) is equivalent to simple exponential smoothing with drift. This is demonstrated in Hyndman and Billah (2003). Prediction intervals are computed using the underlying state space model.

### Examples

```r
## Not run:
fit <- tbats(USAccDeaths, use.parallel=FALSE)
components <- tbats.components(fit)
plot(components)
## End(Not run)
```

### Usage

```r
thetaf(x, h=10, level=c(80,95), fan=FALSE)
```

### Arguments

- `x` : a numeric vector or time series
- `h` : Number of periods for forecasting
- `level` : Confidence levels for prediction intervals.
- `fan` : If TRUE, level is set to seq(50,99,by=1). This is suitable for fan plots.

### Details

The theta method of Assimakopoulos and Nikolopoulos (2000) is equivalent to simple exponential smoothing with drift. This is demonstrated in Hyndman and Billah (2003). Prediction intervals are computed using the underlying state space model.

### Value

An object of class "forecast".

The function `summary` is used to obtain and print a summary of the results, while the function `plot` produces a plot of the forecasts and prediction intervals.

The generic accessor functions `fitted.values` and `residuals` extract useful features of the value returned by `thetaf`.

An object of class "forecast" is a list containing at least the following elements:

- `model` : A list containing information about the fitted model
- `method` : The name of the forecasting method as a character string
- `mean` : Point forecasts as a time series
- `lower` : Lower limits for prediction intervals
- `upper` : Upper limits for prediction intervals
level
The confidence values associated with the prediction intervals

x
The original time series (either object itself or the time series used to create the model stored as object).

residuals
Residuals from the fitted model. That is x minus fitted values.

fitted
Fitted values (one-step forecasts)

Author(s)
Rob J Hyndman

References


See Also
arima, meanf, rwf, ses

Examples

```r
nile.fcast <- thetaf(Nile)
plot(nile.fcast)
```

---

tsclean

*Identify and replace outliers and missing values in a time series*

Description
Uses loess for non-seasonal series and a periodic stl decomposition with seasonal series to identify and replace outliers. To estimate missing values, linear interpolation is used for non-seasonal series, and a periodic stl decomposition is used with seasonal series.

Usage

```r
tsclean(x, replace.missing = TRUE, lambda = NULL)
```

Arguments

- **x**: time series
- **replace.missing**: If TRUE, it not only replaces outliers, but also interpolates missing values
- **lambda**: a numeric value giving the Box-Cox transformation parameter
tsdisplay

Value
Time series

Author(s)
Rob J Hyndman

See Also
na.interp, tsoutliers

Examples

data(gold)
tsdisplay(gold)

tsd

Description
Plots a time series along with its acf and either its pacf, lagged scatterplot or spectrum.

Usage
tsdisplay(x, plot.type=c("partial","scatter","spectrum"), points=TRUE, ci.type="white",
  lag.max, na.action=na.contiguous,
  main=NULL, xlab='', ylab='', pch=1, cex=0.5, ...)

Arguments

  x
    a numeric vector or time series.

  plot.type
    type of plot to include in lower right corner.

  points
    logical flag indicating whether to show the individual points or not in the time plot.

  ci.type
    type of confidence limits for ACF. Possible values are as for acf.

  lag.max
    the maximum lag to plot for the acf and pacf. A suitable value is selected by default if the argument is missing.

  na.action
    function to handle missing values in acf, pacf and spectrum calculations. The default is na.contiguous. Useful alternatives are na.pass and na.interp.

  main
    Main title.

  xlab
    X-axis label.

  ylab
    Y-axis label.

  pch
    Plotting character.

  cex
    Character size.

  ...
    additional arguments to acf.
tslm

Author(s)

Rob J Hyndman

References


See Also

plot.ts, Acf, spec.ar

Examples

tsddisplay(diff(WWWusage))

tslm

Fit a linear model with time series components

Description

tslm is used to fit linear models to time series including trend and seasonality components.

Usage

tslm(formula, data, lambda=NULL, ...)

Arguments

formula an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted.
data an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which lm is called.
lambda Box-Cox transformation parameter. Ignored if NULL. Otherwise, data are transformed via a Box-Cox transformation.
... Other arguments passed to lm().
tsoutliers

Details

`tslm` is largely a wrapper for `lm()` except that it allows variables "trend" and "season" which are created on the fly from the time series characteristics of the data. The variable "trend" is a simple time trend and "season" is a factor indicating the season (e.g., the month or the quarter depending on the frequency of the data).

Value

Returns an object of class "lm".

Author(s)

Rob J Hyndman

See Also

`forecast.lm`, `lm`.

Examples

```r
y <- ts(rnorm(120,0,3) + 1:120 + 20*sin(2*pi*(1:120)/12), frequency=12)
fit <- tslm(y ~ trend + season)
plot(forecast(fit, h=20))
```

---

**tsoutliers**

*Identify and replace outliers in a time series*

Description

Uses loess for non-seasonal series and a periodic stl decomposition with seasonal series to identify and replace outliers.

Usage

```r
tsoutliers(x, iterate = 2, lambda = NULL)
```

Arguments

- `x` time series
- `iterate` the number of iteration only for non-seasonal series
- `lambda` Allowing Box-cox transformation

Value

- `index` Indicating the index of outlier(s)
- `replacement` Suggested numeric values to replace identified outliers
Author(s)

Rob J Hyndman

See Also

na.interp, tsclean

Examples

data(gold)
tsoutliers(gold)

tsdemo(wineind)

wineind  
Australian total wine sales

Description


Usage

wineind

Format

Time series data

Source

Time Series Data Library. http://data.is/TSDLdemo

Examples

tsdemo(wineind)
woolyrnq

<table>
<thead>
<tr>
<th>woolyrnq</th>
<th>Quarterly production of woollen yarn in Australia</th>
</tr>
</thead>
</table>

**Description**

**Usage**
woolyrnq

**Format**
Time series data

**Source**
Time Series Data Library. [http://data.is/TSDLdemo](http://data.is/TSDLdemo)

**Examples**
`ttsdisplay(woolyrnq)`
Index

*Topic **datasets**
  - gas, 42
  - gold, 43
  - taylor, 70
  - wineind, 78
  - woolyrnq, 79

*Topic **hplot**
  - plot.Arima, 53
  - plot.bats, 55
  - plot.ets, 56

*Topic **htest**
  - dm.test, 20

*Topic **models**
  - CV, 20

*Topic **stats**
  - forecast.lm, 36
  - tslm, 76

*Topic **ts**
  - accuracy, 3
  - Acf, 5
  - arfima, 6
  - Arima, 8
  - arima.errors, 10
  - arimaorder, 11
  - auto.arima, 12
  - bats, 14
  - bizdays, 15
  - BoxCox, 16
  - BoxCox.lambda, 17
  - croston, 18
  - dm.test, 20
  - dshw, 22
  - easter, 24
  - ets, 24
  - findfrequency, 27
  - fitted.Arima, 28
  - forecast, 28
  - forecast.Arima, 30
  - forecast.bats, 32
  - forecast.ets, 33
  - forecast.HoltWinters, 35
  - forecast.stl, 38
  - forecast.StructTS, 40
  - getResponse, 42
  - logLik.ets, 43
  - ma, 44
  - meanf, 45
  - monthdays, 46
  - msts, 47
  - na.interp, 48
  - naive, 49
  - ndiffs, 50
  - nnets, 52
  - plot.forecast, 56
  - rwf, 58
  - seasadj, 59
  - seasonaldummy, 60
  - seasonplot, 62
  - ses, 63
  - simulate.ets, 65
  - sindexf, 66
  - splinef, 67
  - subset.ts, 69
  - tbats, 70
  - tbats.components, 72
  - thetadf, 73
  - tsclean, 74
  - tsdisplay, 75
  - tsoutliers, 77

  accuracy, 3
  Acf, 5, 76
  acf, 5, 6, 75
  AIC, 20
  ar, 11, 30, 31, 39, 54, 66
  arfima, 6, 11, 30, 31, 66
  Arima, 8, 11, 13, 26, 31, 49, 50, 54, 59, 65, 66
  arima, 7–11, 13, 28, 30, 31, 60, 64, 65, 68, 74
  arima.errors, 10
INDEX

arimaorder, 11
auto.arima, 7, 11, 12, 30, 31, 38, 52, 65, 66
bats, 14, 32, 33, 55
best.arima (auto.arima), 12
bizdays, 15, 47
BoxCox, 16, 18
BoxCox.lambda, 17, 17
croston, 18, 30
cV, 20
decompose, 45, 60, 66
dm.test, 20
dshw, 22
easter, 24
est, 22, 23, 24, 28, 33–35, 38, 44, 56, 63, 64, 66
finCenter, 15
findfrequency, 27
fitted.Arima, 28
forecast, 28, 39, 40, 57
forecast.ar (forecast.Arima), 30
forecast.Arima, 9, 28, 30, 38, 40
forecast.bats, 32
forecast.ets, 29, 30, 33, 33, 40
forecast.fracdiff, 7, 31
forecast.fracdiff (forecast.Arima), 30
forecast.HoltWinters, 30, 35
forecast.lm, 36, 77
forecast.nnetar (nnetar), 52
forecast.stl, 38
forecast.stlm (forecast.stl), 38
forecast.StructTS, 30, 40
forecast.tbats (forecast.bats), 32
forecast.ts, 28
fourier (seasonaldummy), 60
fourierf (seasonaldummy), 60
fracdiff, 7, 11, 30

gas, 42
getResponse, 42
gold, 43

holt, 30, 34, 68
holt (ses), 63
HoltWinters, 23, 26, 35, 36, 64
hw, 30, 34

hw (ses), 63
InvBoxCox (BoxCox), 16
isBizday, 16
ksmooth, 45
lm, 20, 36, 37, 60, 76, 77
logLik.ets, 43
ma, 44
meanf, 30, 45, 59, 74
monthdays, 16, 46
monthplot, 63
msts, 22, 47

na.contiguous, 5, 75
na.interp, 5, 48, 49, 75, 78
na.pass, 5, 75
naive, 49
ndiffs, 13, 50
nnetar, 52, 52
nsdiffs, 13
nsdiffs (ndiffs), 50

Pacf (Acf), 5
pacf, 6
par, 54–56
plot, 57, 62
plot.ar (plot.Arima), 53
plot.Arima, 53
plot.bats, 55
plot.default, 57
plot.ets, 56
plot.forecast, 56
plot.splineforecast (plot.forecast), 56
plot.tbats (plot.bats), 55
plot.ts, 58, 76
predict.ar, 31
predict.Arima, 31
predict.HoltWinters, 35, 36
predict.lm, 36, 37
print.forecast (forecast), 28

residuals, 11
rnorm, 65
rwf, 26, 30, 46, 50, 58, 64, 74

seasadj, 59
seasonaldummy, 60
seasonaldummy (seasonaldummy), 60
seasonplot, 62
ses, 19, 30, 34, 63, 74
set.seed, 65
simulate.ar (simulate.ets), 65
simulate.Arima (simulate.ets), 65
simulate.ets, 65
simulate.fracdiff (simulate.ets), 65
sindexf, 66
smooth.spline, 67, 68
snaive (naive), 49
spec.ar, 76
splinef, 30, 67
stl, 38–40, 60, 66
stl (forecast.stl), 38
stlf, 28, 29
stlf (forecast.stl), 38
stlm (forecast.stl), 38
StructTS, 40, 41
subset, 69
subset.ts, 69
summary.forecast (forecast), 28
taperedacf (Acf), 5
taperedpacf (Acf), 5
taylor, 70
tbats, 33, 55, 60, 70, 72
tbats.components, 72, 72
thetaf, 30, 39, 73
tsclean, 74, 78
tsdisplay, 6, 75
tslm, 20, 36, 37, 60, 76
toutliers, 49, 75, 77
wineind, 78
woolyrnq, 79