Package ‘forecast’

October 13, 2022

Version 8.18

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 (>= 3.5.0),
Imports colorspace, fracdiff, generics (>= 0.1.2), ggplot2 (>= 2.2.1), graphics, lmtest, magrittr, nnet, parallel, Rcpp (>= 0.11.0), stats, timeDate, tseries, urca, zoo
Suggests forecTheta, knitr, methods, rmarkdown, rticles, seasonal, testthat, uroot
LinkingTo Rcpp (>= 0.11.0), RcppArmadillo (>= 0.2.35)
LazyData yes
ByteCompile TRUE

BugReports https://github.com/robjhyndman/forecast/issues
License GPL-3

URL https://pkg.robjhyndman.com/forecast/,
https://github.com/robjhyndman/forecast

VignetteBuilder knitr
Encoding UTF-8
RoxygenNote 7.2.1

NeedsCompilation yes

Author Rob Hyndman [aut, cre, cph] (<https://orcid.org/0000-0002-2140-5352>), George Athanasopoulos [aut] (<https://orcid.org/0000-0002-5389-2802>), Christoph Bergmeir [aut] (<https://orcid.org/0000-0002-3665-9021>), Gabriel Caceres [aut] (<https://orcid.org/0000-0002-2947-2023>), Leanne Chhay [aut], Kirill Kuroptev [aut], Mitchell O'Hara-Wild [aut] (<https://orcid.org/0000-0001-6729-7695>), Fotios Petropoulos [aut] (<https://orcid.org/0000-0003-3039-4955>),
R topics documented:

- forecast-package .................................................. 4
- accuracy.default .................................................. 5
- Acf ................................................................. 7
- arima ............................................................... 9
- Arima ............................................................. 11
- arima.errors ...................................................... 13
- arimaorder ........................................................ 14
- auto.arima ......................................................... 15
- autolayer .......................................................... 18
- autolayer.mts ..................................................... 19
- autoplot.acf ....................................................... 21
- autoplot.decomposed.ts ........................................ 23
- autoplot.mforecast ............................................... 24
- baggedModel ..................................................... 26
- bats ............................................................... 27
- bizdays ............................................................ 29
- bld.mbb.bootstrap ................................................. 30
- BoxCox ............................................................. 31
- BoxCox.lambda ................................................... 32
- checkresiduals ................................................... 33
- croston ............................................................ 34
- CV ................................................................. 36
- CVAR ............................................................. 37
- dm.test ........................................................... 38
- dshw .............................................................. 40
- easter .............................................................. 42
- ets ................................................................. 43
- findfrequency ...................................................... 46
- fitted.ARFIMA .................................................... 47
- forecast.baggedModel .......................................... 48

Maintainer Rob Hyndman <Rob.Hyndman@monash.edu>

Repository CRAN

Date/Publication 2022-10-02 03:10:02 UTC
R topics documented:

- `forecast.bats` .................................................. 50
- `forecast.ets` ................................................. 51
- `forecast.fracdiff` ........................................... 53
- `forecast.HoltWinters` .................................... 56
- `forecast.lm` .................................................. 57
- `forecast.mlm` ................................................ 59
- `forecast.modelAR` .......................................... 61
- `forecast.mts` ................................................ 63
- `forecast.nnetar` ............................................ 65
- `forecast.stl` ................................................ 68
- `forecast.StructTS` ......................................... 71
- `forecast.ts` .................................................. 73
- `fourier` ...................................................... 75
- `getResponse` .................................................. 77
- `gghistogram` ................................................ 78
- `gglagplot` .................................................... 79
- `ggmonthplot` ................................................ 81
- `ggsseasonplot` ............................................... 82
- `ggtsdisplay` ................................................ 84
- `gold` .......................................................... 86
- `is.acf` ........................................................ 86
- `is.constant` .................................................. 87
- `is.forecast` ................................................... 87
- `ma` ............................................................ 88
- `meanf` ........................................................ 89
- `modelAR` ....................................................... 91
- `monthdays` .................................................... 93
- `mstl` ........................................................... 94
- `msts` .......................................................... 95
- `na.interp` ..................................................... 96
- `ndiffs` ........................................................ 97
- `nnetar` ........................................................ 98
- `nsdiffs` ...................................................... 100
- `ocsb.test` ................................................... 102
- `plot.Arima` ................................................. 103
- `plot.bats` .................................................... 105
- `plot.ets` ..................................................... 106
- `plot.forecast` .............................................. 107
- `residuals.forecast` ....................................... 110
- `rwf` .......................................................... 111
- `seasadj` ...................................................... 114
- `seasonal` ...................................................... 115
- `seasonaldummy` ............................................. 116
- `ses` .......................................................... 117
- `simulate.ets` ............................................... 120
- `sindexf` ....................................................... 123
- `splinef` ....................................................... 124
Description

Methods and tools for displaying and analysing univariate time series forecasts including exponential smoothing via state space models and automatic ARIMA modelling.

Author(s)

Maintainer: Rob Hyndman <Rob.Hyndman@monash.edu> (ORCID) [copyright holder]
Authors:

• George Athanasopoulos (ORCID)
• Christoph Bergmeir (ORCID)
• Gabriel Caceres (ORCID)
• Leanne Chhay
• Kirill Kuroptev
• Mitchell O’Hara-Wild (ORCID)
• Fotios Petropoulos (ORCID)
• Slava Razbash
• Earo Wang (ORCID)
• Farah Yasmeen (ORCID)

Other contributors:

• Daniel Reid [contributor]
• David Shaub [contributor]
• Federico Garza [contributor]
• R Core Team [contributor, copyright holder]
accuracy.default

- Ross Ihaka [contributor, copyright holder]
- Xiaoqian Wang [contributor]
- Yuan Tang (ORCID) [contributor]
- Zhenyu Zhou [contributor]

See Also

Useful links:

- https://pkg.robjhyndman.com/forecast/
- https://github.com/robjhyndman/forecast
- Report bugs at https://github.com/robjhyndman/forecast/issues

accuracy.default  Accuracy measures for a forecast model

Description

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

Usage

```r
## Default S3 method:
accuracy(object, x, test = NULL, d = NULL, D = NULL, f = NULL, ...)
```

Arguments

- **object**: 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 training set accuracy measures are returned.
- **x**: An optional numerical vector containing actual values of the same length as \( \text{object} \), or a time series overlapping with the times of \( f \).
- **test**: Indicator of which elements of \( x \) and \( f \) to test. If \( \text{test} \) is \( \text{NULL} \), all elements are used. Otherwise \( \text{test} \) 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.
- **f**: Deprecated. Please use ‘object’ instead.
- **...**: Additional arguments depending on the specific method.
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 training set naive forecasts for non-seasonal time series, training set seasonal naive forecasts for seasonal time series and training set mean forecasts for non-time series data. If \( f \) is a numerical vector rather than a forecast object, the MASE will not be returned as the training data will not be available.

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])
```
**Acf**

**(Partial) Autocorrelation and Cross-Correlation Function Estimation**

**Description**

The function `Acf` computes (and by default plots) an estimate of the autocorrelation function of a (possibly multivariate) time series. Function `Pacf` computes (and by default plots) an estimate of the partial autocorrelation function of a (possibly multivariate) time series. Function `Ccf` computes the cross-correlation or cross-covariance of two univariate series.

**Usage**

```r
Acf(
  x,
  lag.max = NULL,
  type = c("correlation", "covariance", "partial"),
  plot = TRUE,
  na.action = na.contiguous,
  demean = TRUE,
  ...
)
```

```r
Pacf(
  x,
  lag.max = NULL,
  plot = TRUE,
  na.action = na.contiguous,
  demean = TRUE,
  ...
)
```

```r
Ccf(
  x,
  y,
  lag.max = NULL,
  type = c("correlation", "covariance"),
  plot = TRUE,
  na.action = na.contiguous,
  ...
)
```

```r
taperedacf(
  x,
  lag.max = NULL,
  type = c("correlation", "partial"),
  plot = TRUE,
  calc.ci = TRUE,
  ...
)
```
Acf

\begin{verbatim}
level = 95,
nsim = 100,
...
)
taperedpacf(x, ...)
\end{verbatim}

Arguments

- **x**: a univariate or multivariate (not Ccf) numeric time series object or a numeric vector or matrix.
- **lag.max**: maximum lag at which to calculate the acf. Default is $10 \times \log_{10}(N/m)$ 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), "covariance" or "partial".
- **plot**: logical. If TRUE (the default) the resulting acf, pacf or ccf is plotted.
- **na.action**: function to handle missing values. Default is `na.contiguous`. Useful alternatives are `na.pass` and `na.interp`.
- **demean**: Should covariances be about the sample means?
- **y**: a univariate numeric time series object or a numeric vector.
- **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.

Details

The functions improve the `acf`, `pacf` and `ccf` functions. The main differences are that `Acf` does not plot a spike at lag 0 when `type`="correlation" (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 `Acf`, `Pacf` and `Ccf` functions return objects of class "acf" as described in `acf` from the stats package. The `taperedacf` and `taperedpacf` functions return objects of class "mpacf".

Author(s)

Rob J Hyndman
**arfima**

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.

**Usage**

```r
arfima(
  y,
  drange = c(0, 0.5),
  estim = c("mle", "ls"),
  model = NULL,
  lambda = NULL,
  biasadj = FALSE,
  x = y,
  ...
)
```

**References**


**See Also**

acf, pacf, ccf, tsdisplay

**Examples**

```r
Acf(wineind)
Pacf(wineind)
## Not run:
taperedacf(wineind, nsim=50)
taperedpacf(wineind, nsim=50)
## End(Not run)
```
arfima

Arguments

- **y**: a univariate time series (numeric vector).
- **drange**: Allowable values of \(d\) to be considered. Default of \([0,0.5]\) 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.
- **model**: Output from a previous call to arfima. If model is passed, this same model is fitted to \(y\) without re-estimating any parameters.
- **lambda**: Box-Cox transformation parameter. If lambda="auto", then a transformation is automatically selected using BoxCox.lambda. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.
- **biasadj**: Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If biasadj is TRUE, an adjustment will be made to produce mean forecasts and fitted values.
- **x**: Deprecated. Included for backwards compatibility.
- **...**: 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.
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 y.

Usage

Arima(
y,
order = c(0, 0, 0),
seasonal = c(0, 0, 0),
xreg = NULL,
include.mean = TRUE,
include.drift = FALSE,
include.constant,
lambda = model$lambda,
biasadj = FALSE,
method = c("CSS-ML", "ML", "CSS"),
model = NULL,
x = y,
...
)

Arguments

y 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(y)). 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 numerical vector or matrix of external regressors, which must have the same number of rows as y. It should not be a data frame.
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. If lambda="auto", then a transformation is automatically selected using BoxCox.lambda. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.

biasadj Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If biasadj is TRUE, an adjustment will be made to produce mean forecasts and fitted values.

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.

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

x Deprecated. Included for backwards compatibility.

... Additional arguments to be passed to arima.

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).

sigma2 The bias adjusted MLE of the innovations variance.

Author(s)

Rob J Hyndman

See Also

classarima, forecast.Arima.
Examples

library(ggplot2)
WWWusage %>%
  Arima(order=c(3,1,0)) %>%
  forecast(h=20) %>%
  autoplot

# 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,lambda=NULL),
  log(window(AirPassengers,start=1957)))

---

arima.errors Errors from a regression model with ARIMA errors

Description

Returns time series of the regression residuals from a fitted ARIMA model.

Usage

arima.errors(object)

Arguments

object An object containing a time series model of class Arima.

Details

This is a deprecated function which is identical to residuals.Arima(object, type="regression")
Regression residuals are equal to the original data minus the effect of any regression variables. If there are no regression variables, the errors will be identical to the original series (possibly adjusted to have zero mean).
arimaorder

Return the order of an ARIMA or ARFIMA model

Description

Returns the order of a univariate ARIMA or ARFIMA model.

Usage

arimaorder(object)

Arguments

object

An object of class “Arima”, dquote or “fracdiff”. Usually the result of a call to arima, Arima, auto.arima, ar, arfima or fracdiff.

Value

A numerical vector giving the values $p$, $d$ and $q$ of the ARIMA or ARFIMA model. For a seasonal ARIMA model, the returned vector contains the values $p$, $d$, $q$, $P$, $D$, $Q$ and $m$, where $m$ is the period of seasonality.

Author(s)

Rob J Hyndman

See Also

ar, auto.arima, Arima, arima, arfima.

Examples

WWWusage %>% auto.arima %>% arimaorder
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(
        y,
        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,
        nmodels = 94,
        trace = FALSE,
        approximation = (length(x) > 150 | frequency(x) > 12),
        method = NULL,
        truncate = NULL,
        xreg = NULL,
        test = c("kpss", "adf", "pp"),
        test.args = list(),
        seasonal.test = c("seas", "ocsb", "hegy", "ch"),
        seasonal.test.args = list(),
        allowdrift = TRUE,
        allowmean = TRUE,
        lambda = NULL,
        biasadj = FALSE,
        parallel = FALSE,
        num.cores = 2,
        x = y,
        ...
    )
Arguments

y  a univariate time series

D  Order of seasonal-differencing. If missing, will choose a value based on season.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.

nmodels  Maximum number of models considered in the stepwise search.

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.

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. Can be abbreviated.

truncate  An integer value indicating how many observations to use in model selection. The last truncate values of the series are used to select a model when truncate is not NULL and approximation=TRUE. All observations are used if either truncate=NULL or approximation=FALSE.

xreg  Optionally, a numerical vector or matrix of external regressors, which must have the same number of rows as y. (It should not be a data frame.)

test  Type of unit root test to use. See ndiffs for details.

test.args  Additional arguments to be passed to the unit root test.
seasonal.test  This determines which method is used to select the number of seasonal differences. The default method is to use a measure of seasonal strength computed from an STL decomposition. Other possibilities involve seasonal unit root tests.

seasonal.test.args

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. If lambda="auto", then a transformation is automatically selected using BoxCox.lambda. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.

biasadj  Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If biasadj is TRUE, an adjustment will be made to produce mean forecasts and fitted values.

parallel  If TRUE and stepwise = FALSE, then the specification search is done in parallel. This can give a significant speedup on multicore machines.

num.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.

x  Deprecated. Included for backwards compatibility.

...  Additional arguments to be passed to arima.

Details

The default arguments are designed for rapid estimation of models for many time series. If you are analysing just one time series, and can afford to take some more time, it is recommended that you set stepwise=FALSE and approximation=FALSE.

Non-stepwise selection can be slow, especially for seasonal data. The stepwise algorithm outlined in Hyndman & Khandakar (2008) is used except that the default method for selecting seasonal differences is now based on an estimate of seasonal strength (Wang, Smith & Hyndman, 2006) rather than the Canova-Hansen test. There are also some other minor variations to the algorithm described in Hyndman and Khandakar (2008).

Value

Same as for Arima

Author(s)

Rob J Hyndman

References


See Also

Arima

Examples

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

autolayer

Create a ggplot layer appropriate to a particular data type

Description

autolayer() uses ggplot2 to draw a particular layer for an object of a particular class in a single command. This defines the S3 generic that other classes and packages can extend.

Usage

autolayer(object, ...)

Arguments

object an object, whose class will determine the behaviour of autolayer
... other arguments passed to specific methods

Value

a ggplot layer

See Also

autolot(), ggplot() and fortify()
Description

autoplot takes an object of type ts or mts and creates a ggplot object suitable for usage with \texttt{stat_forecast}.

Usage

```r
## S3 method for class 'mts'
autolayer(object, colour = TRUE, series = NULL, ...)

## S3 method for class 'msts'
autolayer(object, series = NULL, ...)

## S3 method for class 'ts'
autolayer(object, colour = TRUE, series = NULL, ...)

## S3 method for class 'ts'
autoplot(
  object,
  series = NULL,
  xlab = "Time",
  ylab = deparse(substitute(object)),
  main = NULL,
  ...
)

## S3 method for class 'mts'
autoplot(
  object,
  colour = TRUE,
  facets = FALSE,
  xlab = "Time",
  ylab = deparse(substitute(object)),
  main = NULL,
  ...
)

## S3 method for class 'msts'
autoplot(object, ...)

## S3 method for class 'ts'
fortify(model, data, ...)
```
Arguments

object Object of class “ts” or “mts”.
colour If TRUE, the time series will be assigned a colour aesthetic
series Identifies the time series with a colour, which integrates well with the functionality of `geom_forecast`.
... Other plotting parameters to affect the plot.
xlab X-axis label.
ylab Y-axis label.
main Main title.
facades If TRUE, multiple time series will be faceted (and unless specified, colour is set to FALSE). If FALSE, each series will be assigned a colour.
model Object of class “ts” to be converted to “data.frame”.
data Not used (required for `fortify` method)

Details

`fortify.ts` takes a ts object and converts it into a data frame (for usage with ggplot2).

Value

None. Function produces a ggplot graph.

Author(s)

Mitchell O’Hara-Wild

See Also

`plot.ts`, `fortify`

Examples

```r
library(ggplot2)
autoplot(USAccDeaths)

lungDeaths <- cbind(mdeaths, fdeaths)
autoplot(lungDeaths)
autoplot(lungDeaths, facets=TRUE)
```
**autoplot.acf**

**ggplot (Partial) Autocorrelation and Cross-Correlation Function Estimation and Plotting**

**Description**

Produces a ggplot object of their equivalent Acf, Pacf, Ccf, taperedacf and taperedpacf functions.

**Usage**

```r
## S3 method for class 'acf'
autoplot(object, ci = 0.95, ...)

ggAcf(
  x,
  lag.max = NULL,
  type = c("correlation", "covariance", "partial"),
  plot = TRUE,
  na.action = na.contiguous,
  demean = TRUE,
  ...
)

ggPacf(
  x,
  lag.max = NULL,
  plot = TRUE,
  na.action = na.contiguous,
  demean = TRUE,
  ...
)

ggCcf(
  x,
  y,
  lag.max = NULL,
  type = c("correlation", "covariance"),
  plot = TRUE,
  na.action = na.contiguous,
  ...
)

## S3 method for class 'mpacf'
autoplot(object, ...)

ggtaperedacf(
  x,
  
```
ggtaperedpacf(x, ...)
autoplot.decomposed.ts

See Also

plot.acf, Acf, acf, taperedacf

Examples

library(ggplot2)
ggAcf(wineind)
wineind %>% Acf(plot=FALSE) %>% autoplot
## Not run:
wineind %>% taperedacf(plot=FALSE) %>% autoplot
ggtaperedacf(wineind)
ggtaperedpacf(wineind)
## End(Not run)
ggCcf(mdeaths, fdeaths)

 autoplot.decomposed.ts

Plot time series decomposition components using ggplot

Description

Produces a ggplot object of seasonally decomposed time series for objects of class “stl” (created with stl), class “seas” (created with seas), or class “decomposed.ts” (created with decompose).

Usage

## S3 method for class 'decomposed.ts'
autoplot(object, labels = NULL, range.bars = NULL, ...)

## S3 method for class 'stl'
autoplot(object, labels = NULL, range.bars = TRUE, ...)

## S3 method for class 'StructTS'
autoplot(object, labels = NULL, range.bars = TRUE, ...)

## S3 method for class 'seas'
autoplot(object, labels = NULL, range.bars = NULL, ...)

## S3 method for class 'mstl'
autoplot(object, ...)

Arguments

object Object of class “seas”, “stl”, or “decomposed.ts”.
labels Labels to replace “seasonal”, “trend”, and “remainder”. 
range.bars Logical indicating if each plot should have a bar at its right side representing relative size. If NULL, automatic selection takes place.
... Other plotting parameters to affect the plot.

Value
Returns an object of class ggplot.

Author(s)
Mitchell O’Hara-Wild

See Also
seas, stl, decompose, StructTS, plot.stl.

Examples

library(ggplot2)
c02 %>%
decompose() %>%
autoplot()
n0ttem %>%
stl(s.window = "periodic") %>%
autoplot()
## Not run:
library(seasonal)
seas(USAccDeaths) %>% autoplot()
## End(Not run)
Arguments

- **object**: Multivariate forecast object of class `mforecast`. Used for ggplot graphics (S3 method consistency).
- **PI**: If `FALSE`, confidence intervals will not be plotted, giving only the forecast line.
- **facets**: If `TRUE`, multiple time series will be faceted. If `FALSE`, each series will be assigned a colour.
- **colour**: If `TRUE`, the time series will be assigned a colour aesthetic... additional arguments to each individual plot.
- **series**: Matches an unidentified forecast layer with a coloured object on the plot.
- **x**: Multivariate forecast object of class `mforecast`.
- **main**: Main title. Default is the forecast method. For autoplot, specify a vector of titles for each plot.
- **xlab**: X-axis label. For autoplot, specify a vector of labels for each plot.

Details

`autoplot` will produce an equivalent plot as a ggplot object.

Author(s)

Mitchell O’Hara-Wild

References


See Also

- `plot.forecast`, `plot.ts`

Examples

```r
library(ggplot2)

lungDeaths <- cbind(mdeaths, fdeaths)
fit <- tslm(lungDeaths ~ trend + season)
fcast <- forecast(fit, h=10)
plot(fcast)
autoplot(fcast)

carPower <- as.matrix(mtcars[,c("qsec","hp")])
carmpg <- mtcars[,"mpg"]
fit <- lm(carPower ~ carmpg)
fcast <- forecast(fit, newdata=data.frame(carmpg=30))
plot(fcast, xlab="Year")
autoplot(fcast, xlab=rep("Year",2))
```
baggedModel  Forecasting using a bagged model

Description

The bagged model forecasting method.

Usage

```r
describeModel(y, bootstrapped_series = bld.mbb.bootstrap(y, 100), fn = ets, ...
describeETS(y, bootstrapped_series = bld.mbb.bootstrap(y, 100), ...)
```

Arguments

- `y`: A numeric vector or time series of class `ts`.
- `bootstrapped_series`: bootstrapped versions of `y`.
- `fn`: the forecast function to use. Default is `ets`.
- `...`: Other arguments passed to the forecast function.

Details

This function implements the bagged model forecasting method described in Bergmeir et al. By default, the `ets` function is applied to all bootstrapped series. Base models other than `ets` can be given by the parameter `fn`. Using the default parameters, the function `bld.mbb.bootstrap` is used to calculate the bootstrapped series with the Box-Cox and Loess-based decomposition (BLD) bootstrap. The function `describe.baggedModel` can then be used to calculate forecasts.

`describeETS` is a wrapper for `describeModel`, setting `fn` to "ets". This function is included for backwards compatibility only, and may be deprecated in the future.

Value

Returns an object of class "baggedModel".

The function print is used to obtain and print a summary of the results.

- `models`: A list containing the fitted ensemble models.
- `method`: The function for producing a forecastable model.
- `y`: The original time series.
- `bootstrapped_series`: The bootstrapped series.
- `modelargs`: The arguments passed through to `fn`.
- `fitted`: Fitted values (one-step forecasts). The mean of the fitted values is calculated over the ensemble.
- `residuals`: Original values minus fitted values.
**Author(s)**

Christoph Bergmeir, Fotios Petropoulos

**References**


**Examples**

```r
fit <- baggedModel(WWWusage)
fcast <- forecast(fit)
plot(fcast)
```

---

**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**

```r
bats(
  y,
  use.box.cox = NULL,
  use.trend = NULL,
  use.damped.trend = NULL,
  seasonal.periods = NULL,
  use arma.errors = TRUE,
  use.parallel = length(y) > 1000,
  num.cores = 2,
  bc.lower = 0,
  bc.upper = 1,
  biasadj = FALSE,
  model = NULL,
  ...
)
```
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.

biasadj Use adjusted back-transformed mean for Box-Cox transformations. If TRUE, point forecasts and fitted values are mean forecast. Otherwise, these points can be considered the median of the forecast densities.

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

... 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
bizdays

References


Examples

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

---

<table>
<thead>
<tr>
<th>bizdays</th>
<th>Number of trading days in each season</th>
</tr>
</thead>
</table>

Description

Returns number of trading days in each month or quarter of the observed time period in a major financial center.

Usage

```r
bizdays(x, FinCenter = c("New York", "London", "NERC", "Toronto", "Zurich"))
```

Arguments

- **x**: Monthly or quarterly time series
- **FinCenter**: Major financial center.

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
bld.mbb.bootstrap

See Also

monthdays

Examples

```r
x <- ts(rnorm(30), start = c(2013, 2), frequency = 12)
bizdays(x, FinCenter = "New York")
```

bld.mbb.bootstrap  

*Box-Cox and Loess-based decomposition bootstrap.*

Description

Generates bootstrapped versions of a time series using the Box-Cox and Loess-based decomposition bootstrap.

Usage

```r
bld.mbb.bootstrap(x, num, block_size = NULL)
```

Arguments

- `x`: Original time series.
- `num`: Number of bootstrapped versions to generate.
- `block_size`: Block size for the moving block bootstrap.

Details

The procedure is described in Bergmeir et al. Box-Cox decomposition is applied, together with STL or Loess (for non-seasonal time series), and the remainder is bootstrapped using a moving block bootstrap.

Value

A list with bootstrapped versions of the series. The first series in the list is the original series.

Author(s)

Christoph Bergmeir, Fotios Petropoulos

References

See Also

baggedETS.

Examples

    bootstrapped_series <- bld.mbb.bootstrap(WWWusage, 100)

BoxCox

Box Cox Transformation

Description

BoxCox() returns a transformation of the input variable using a Box-Cox transformation. InvBoxCox() reverses the transformation.

Usage

    BoxCox(x, lambda)

    InvBoxCox(x, lambda, biasadj = FALSE, fvar = NULL)

Arguments

- **x**: a numeric vector or time series of class ts.
- **lambda**: transformation parameter. If lambda = "auto", then the transformation parameter lambda is chosen using BoxCox.lambda (with a lower bound of -0.9).
- **biasadj**: Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If biasadj is TRUE, an adjustment will be made to produce mean forecasts and fitted values.
- **fvar**: Optional parameter required if biasadj=TRUE. Can either be the forecast variance, or a list containing the interval level, and the corresponding upper and lower intervals.

Details

The Box-Cox transformation (as given by Bickel & Doksum 1981) is given by

\[ f_\lambda(x) = \text{sign}(x)(|x|^\lambda - 1)/\lambda \]

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

\[ f_0(x) = \log(x) \].
Value

A numeric vector of the same length as x.

Author(s)

Rob J Hyndman & Mitchell O'Hara-Wild

References


See Also

`BoxCox.lambda`

Examples

```r
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.

Usage

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

Arguments

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

Details

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.
checkresiduals

Value

A number indicating the Box-Cox transformation parameter.

Author(s)

Leanne Chhay and Rob J Hyndman

References


See Also

BoxCox

Examples

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))

Description

If plot=TRUE, produces a time plot of the residuals, the corresponding ACF, and a histogram. If the degrees of freedom for the model can be determined and test is not FALSE, the output from either a Ljung-Box test or Breusch-Godfrey test is printed.

Usage

checkresiduals(object, lag, df = NULL, test = FALSE, ...)

Arguments

object

Either a time series model, a forecast object, or a time series (assumed to be residuals).

lag

Number of lags to use in the Ljung-Box or Breusch-Godfrey test. If missing, it is set to \text{min}(10, n/5) for non-seasonal data, and \text{min}(2m, n/5) for seasonal data, where \( n \) is the length of the series, and \( m \) is the seasonal period of the data. It is further constrained to be at least \( df+3 \) where \( df \) is the degrees of freedom of the model. This ensures there are at least 3 degrees of freedom used in the chi-squared test.
df  Number of degrees of freedom for fitted model, required for the Ljung-Box or Breusch-Godfrey test. Ignored if the degrees of freedom can be extracted from object.

test  Test to use for serial correlation. By default, if object is of class lm, then test="BG". Otherwise, test="LB". Setting test=FALSE will prevent the test results being printed.

plot  Logical. If TRUE, will produce the plot.

Value
None

Author(s)
Rob J Hyndman

See Also
ggtsdisplay, Box.test, bgtest

Examples

```r
fit <- ets(WWWusage)
checkresiduals(fit)
```
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.

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 y 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

```
y <- rpois(20, lambda = 0.3)
fcast <- croston(y)
plot(fcast)
```
CV 

Cross-validation statistic

Description

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

Usage

CV(obj)

Arguments

obj output from \texttt{lm} or \texttt{tslm}

Value

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

Author(s)

Rob J Hyndman

See Also

\texttt{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)
```
$CVar$ is $k$-fold Cross-Validation applied to an autoregressive model

**Description**

$CVar$ computes the errors obtained by applying an autoregressive modelling function to subsets of the time series $y$ using $k$-fold cross-validation as described in Bergmeir, Hyndman and Koo (2015). It also applies a Ljung-Box test to the residuals. If this test is significant (see returned pvalue), there is serial correlation in the residuals and the model can be considered to be underfitting the data. In this case, the cross-validated errors can underestimate the generalization error and should not be used.

**Usage**

```r
CVar(
  y,
  k = 10,
  FUN = nnetar,
  cvtrace = FALSE,
  blocked = FALSE,
  LBlags = 24,
  ...
)
```

**Arguments**

- **y**: Univariate time series
- **k**: Number of folds to use for cross-validation.
- **FUN**: Function to fit an autoregressive model. Currently, it only works with the `nnetar` function.
- **cvtrace**: Provide progress information.
- **blocked**: choose folds randomly or as blocks?
- **LBlags**: lags for the Ljung-Box test, defaults to 24, for yearly series can be set to 20
- **...**: Other arguments are passed to `FUN`.

**Value**

A list containing information about the model and accuracy for each fold, plus other summary information computed across folds.

**Author(s)**

Gabriel Caceres and Rob J Hyndman
Diebold-Mariano test for predictive accuracy

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

Usage

dm.test(
  e1,
  e2,
  alternative = c("two.sided", "less", "greater"),
  h = 1,
  power = 2,
  varestimator = c("acf", "bartlett")
)

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.
The forecast horizon used in calculating \( e_1 \) and \( e_2 \).

The power used in the loss function. Usually 1 or 2.

A character string specifying the long-run variance estimator. Options are "acf" (default) or "bartlett".

Details

This function implements the modified test proposed by Harvey, Leybourne and Newbold (1997). 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. The long-run variance estimator can either the auto-correlation estimator \( \text{varestimator} = \text{"acf"} \), or the estimator based on Bartlett weights \( \text{varestimator} = \text{"bartlett"} \) which ensures a positive estimate. Both long-run variance estimators are proposed in Diebold and Mariano (1995).

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.
- **varestimator**: a character string describing the long-run variance estimator.
- **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 and Kirill Kuroptev

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])
f1.out <- ets(WWWusage[81:100], model=f1)
f2.out <- Arima(WWWusage[81:100], model=f2)
accuracy(f1.out)
accuracy(f2.out)
dm.test(residuals(f1.out), residuals(f2.out), h=1)

---

dshw  
**Double-Seasonal Holt-Winters Forecasting**

**Description**


**Usage**

```r
dshw(
  y,
  period1 = NULL,
  period2 = NULL,
  h = 2 * max(period1, period2),
  alpha = NULL,
  beta = NULL,
  gamma = NULL,
  omega = NULL,
  phi = NULL,
  lambda = NULL,
  biasadj = FALSE,
  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. If lambda="auto", then a transformation is automatically selected using BoxCox.lambda. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.

biasadj  Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If biasadj is TRUE, an adjustment will be made to produce mean forecasts and fitted values.

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" which is a list that includes 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.
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 dshw.

Author(s)

Rob J Hyndman
References


See Also

`HoltWinters`, `ets`.

Examples

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

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)
## End(Not run)
```

---

**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)
```

Description

Returns ets model applied to `y`.

Usage

```r
esets(
y, 
  model = "ZZZ",
  damped = NULL,
  alpha = NULL,
  beta = NULL,
  gamma = NULL,
  phi = NULL,
  additive.only = FALSE,
  lambda = NULL,
  biasadj = FALSE,
  lower = c(rep(1e-04, 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("aicc", "aic", "bic"),
  restrict = TRUE,
  allow.multiplicative.trend = FALSE,
  use.initial.values = FALSE,
  na.action = c("na.contiguous", "na.interp", "na.fail"),
  ...
)
```
Arguments

- **y**
  - a numeric vector or time series of class ts

- **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. If lambda="auto", then a transformation is automatically selected using BoxCox.lambda. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated. When lambda is specified, additive.only is set to TRUE.

- **biasadj**
  - Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If biasadj is TRUE, an adjustment will be made to produce mean forecasts and fitted values.

- **lower**
  - Lower bounds for the parameters (alpha, beta, gamma, phi). Ignored if bounds=="admissible".

- **upper**
  - Upper bounds for the parameters (alpha, beta, gamma, phi). Ignored if bounds=="admissible".

- **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<=30).

- **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, 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.

na.action
A function which indicates what should happen when the data contains NA values. By default, the largest contiguous portion of the time-series will be used.

... 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`, `rwt`, `Arima`.

Examples
```r
fit <- ets(USAccDeaths)
plot(forecast(fit))
```
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 of class ts

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.ARFIMA

h-step in-sample forecasts for time series models.

Description

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

Usage

```r
## S3 method for class 'ARFIMA'
fitted(object, h = 1, ...)
## S3 method for class 'Arima'
fitted(object, h = 1, ...)
## S3 method for class 'ar'
fitted(object, ...)
## S3 method for class 'bats'
fitted(object, h = 1, ...)
## S3 method for class 'ets'
fitted(object, h = 1, ...)
## S3 method for class 'modelAR'
fitted(object, h = 1, ...)
## S3 method for class 'nnetar'
fitted(object, h = 1, ...)
## S3 method for class 'tbats'
fitted(object, h = 1, ...)
```

Arguments

- `object`: An object of class "ARIMA", "Arima", "bats", "ets" or "nnetar".
- `h`: The number of steps to forecast ahead.
- `...`: Other arguments.

Value

A time series of the h-step forecasts.

Author(s)

Rob J Hyndman & Mitchell O'Hara-Wild
forecast.baggedModel

Forecasting using a bagged model

Description

Returns forecasts and other information for bagged models.

Usage

## S3 method for class 'baggedModel'
forecast(
  object,
  h = ifelse(frequency(object$y) > 1, 2 * frequency(object$y), 10),
  ...,
)

Arguments

- **object**: An object of class "baggedModel" resulting from a call to `baggedModel`.
- **h**: Number of periods for forecasting.
- **...**: Other arguments, passed on to the `forecast` function of the original method.

Details

Intervals are calculated as min and max values over the point forecasts from the models in the ensemble. I.e., the intervals are not prediction intervals, but give an indication of how different the forecasts within the ensemble are.

Examples

```r
fit <- ets(WWWusage)
plot(WWWusage)
lines(fitted(fit), col='red')
lines(fitted(fit, h=2), col='green')
lines(fitted(fit, h=3), col='blue')
legend("topleft", legend=paste("h =",1:3), col=2:4, lty=1)
```
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.

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).
- **xreg**: The external regressors used in fitting (if given).
- **residuals**: Residuals from the fitted model. That is x minus fitted values.
- **fitted**: Fitted values (one-step forecasts)

Author(s)

Christoph Bergmeir, Fotios Petropoulos

References


See Also

- baggedModel

Examples

```r
fit <- baggedModel(WWWusage)
fcast <- forecast(fit)
plot(fcast)

## Not run:
fit2 <- baggedModel(WWWusage, fn="auto.arima")
fcast2 <- forecast(fit2)
plot(fcast2)
accuracy(fcast2)
## End(Not run)
```
Description

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

Usage

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

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(51, 99, by=3)`. This is suitable for fan plots.
- `biasadj` Use adjusted back-transformed mean for Box-Cox transformations. If TRUE, point forecasts and fitted values are mean forecast. Otherwise, these points can be considered the median of the forecast densities.
- `...` 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.values` 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
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**

**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)
```

**Description**
Returns forecasts and other information for univariate ETS models.

**Usage**

```r
## S3 method for class 'ets'
forecast(
  object,
  h = ifelse(object$m > 1, 2 * object$m, 10),
  level = c(80, 95),
  fan = FALSE,
  simulate = FALSE,
```

```r
do iets <- ets(your_data, 
  bootstrap = FALSE, 
  npaths = 5000, 
  PI = TRUE, 
  lambda = object$lambda, 
  biasadj = NULL, 
  ...
)
```

**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(51,99,by=3). This is suitable for fan plots.
- **simulate**: If TRUE, prediction intervals are produced by simulation rather than using analytic formulae. Errors are assumed to be normally distributed.
- **bootstrap**: If TRUE, then prediction intervals are produced by simulation using resampled errors (rather than normally distributed errors).
- **npaths**: Number of sample paths used in computing simulated prediction intervals.
- **PI**: 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.
- **lambda**: Box-Cox transformation parameter. If lambda="auto", then a transformation is automatically selected using BoxCox.lambda. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.
- **biasadj**: Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If biasadj is TRUE, an adjustment will be made to produce mean forecasts and fitted values.
- **...**: Other arguments.

**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.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
**forecast.fracdiff**

Returns forecasts and other information for univariate ARIMA models.

### Description

Returns forecasts and other information for univariate ARIMA models.

### Usage

```r
## S3 method for class 'fracdiff'
forecast(
  object,
  h = 10,
  level = c(80, 95),
  fan = FALSE,
  lambda = object$lambda,
  biasadj = NULL,
  ...
)

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

## S3 method for class 'ar'
forecast(
  object,
  h = 10,
  level = c(80, 95),
  fan = FALSE,
  lambda = NULL,
  bootstrap = FALSE,
npaths = 5000,
biasadj = FALSE,
...
)

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(51,99,by=3). This is suitable for fan plots.
lambda      Box-Cox transformation parameter. If lambda="auto", then a transformation is
            automatically selected using BoxCox.lambda. The transformation is ignored if
            NULL. Otherwise, data transformed before model is estimated.
biasadj     Use adjusted back-transformed mean for Box-Cox transformations. If trans-
            formed data is used to produce forecasts and fitted values, a regular back trans-
            formation will result in median forecasts. If biasadj is TRUE, an adjustment will
            be made to produce mean forecasts and fitted values.
...          Other arguments.
xreg        Future values of an regression variables (for class Arima objects only). A nu-
            merical vector or matrix of external regressors; it should not be a data frame.
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.
```
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**

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,  
  biasadj = 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(51,99,by=3). This is suitable for fan plots.
- `lambda`: Box-Cox transformation parameter. If lambda="auto", then a transformation is automatically selected using `BoxCox.lambda`. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.
- `biasadj`: Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If biasadj is TRUE, an adjustment will be made to produce mean forecasts and fitted values.
- `...`: 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.values 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.
- **fitted**: Fitted values (one-step forecasts)

Author(s)

Rob J Hyndman

See Also


Examples

```r
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,
  biasadj = NULL,
  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.
- **h**: Number of periods for forecasting. Ignored if `newdata` present.
- **level**: Confidence level for prediction intervals.
- **fan**: If `TRUE`, level is set to seq(51,99,by=3). This is suitable for fan plots.
- **lambda**: Box-Cox transformation parameter. If `lambda="auto"`, then a transformation is automatically selected using `BoxCox.lambda`. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.
- **biasadj**: Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If biasadj is TRUE, an adjustment will be made to produce mean forecasts and fitted values.
- **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()`.

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**

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

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

Arguments

- `object`: Object of class "mlm", 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.
- `h`: Number of periods for forecasting. Ignored if `newdata` present.
- `level`: Confidence level for prediction intervals.
- `fan`: If `TRUE`, level is set to `seq(51, 99, by=3)`. This is suitable for fan plots.
- `lambda`: Box-Cox transformation parameter. If `lambda="auto"`, then a transformation is automatically selected using `BoxCox.lambda`. The transformation is ignored if `NULL`. Otherwise, data transformed before model is estimated.
- `biasadj`: Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If `biasadj` is `TRUE`, an adjustment will be made to produce mean forecasts and fitted values.
- `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 `forecast.lm()`.

Details

`forecast.mlm` is largely a wrapper for `forecast.lm()` except that it allows forecasts to be generated on multiple series. Also, the output is reformatted into a `mforecast` object.

Value

An object of class "mforecast".

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 "mforecast" 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 multivariate time series
- `lower`: Lower limits for prediction intervals of each series
- `upper`: Upper limits for prediction intervals of each series
- `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)**

Mitchell O'Hara-Wild

**See Also**

tslm, forecast.lm, lm.

**Examples**

```r
lungDeaths <- cbind(mdeaths, fdeaths)
fit <- tslm(lungDeaths ~ trend + season)
fcast <- forecast(fit, h=10)

carPower <- as.matrix(mtcars[,c("qsec","hp")])
carmpg <- mtcars[,"mpg"]
fit <- lm(carPower ~ carmpg)
fcast <- forecast(fit, newdata=data.frame(carmpg=30))
```

**Description**

Returns forecasts and other information for user-defined models.
Usage

```r
## S3 method for class 'modelAR'
forecast(
  object,
  h = ifelse(object$m > 1, 2 * object$m, 10),
  PI = FALSE,
  level = c(80, 95),
  fan = FALSE,
  xreg = NULL,
  lambda = object$lambda,
  bootstrap = FALSE,
  npaths = 1000,
  innov = NULL,
  ...
)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An object of class &quot;modelAR&quot; resulting from a call to <code>modelAR</code>.</td>
</tr>
<tr>
<td>h</td>
<td>Number of periods for forecasting. If <code>xreg</code> is used, h is ignored and the number of forecast periods is set to the number of rows of <code>xreg</code>.</td>
</tr>
<tr>
<td>PI</td>
<td>If TRUE, prediction intervals are produced, otherwise only point forecasts are calculated. If PI is FALSE, then <code>level</code>, <code>fan</code>, <code>bootstrap</code> and <code>npaths</code> are all ignored.</td>
</tr>
<tr>
<td>level</td>
<td>Confidence level for prediction intervals.</td>
</tr>
<tr>
<td>fan</td>
<td>If TRUE, level is set to <code>seq(51, 99, by=3)</code>. This is suitable for fan plots.</td>
</tr>
<tr>
<td>xreg</td>
<td>Future values of external regressor variables.</td>
</tr>
<tr>
<td>lambda</td>
<td>Box-Cox transformation parameter. If lambda=&quot;auto&quot;, then a transformation is automatically selected using <code>BoxCox.lambda</code>. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.</td>
</tr>
<tr>
<td>bootstrap</td>
<td>If TRUE, then prediction intervals computed using simulations with resampled residuals rather than normally distributed errors. Ignored if <code>innov</code> is not NULL.</td>
</tr>
<tr>
<td>npaths</td>
<td>Number of sample paths used in computing simulated prediction intervals.</td>
</tr>
<tr>
<td>innov</td>
<td>Values to use as innovations for prediction intervals. Must be a matrix with h rows and npaths columns (vectors are coerced into a matrix). If present, bootstrap is ignored.</td>
</tr>
<tr>
<td>...</td>
<td>Additional arguments passed to <code>simulate.nnetar</code></td>
</tr>
</tbody>
</table>

Details

Prediction intervals are calculated through simulations and can be slow. Note that if the model is too complex and overfits the data, the residuals can be arbitrarily small; if used for prediction interval calculations, they could lead to misleadingly small values.
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.nnetar.

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).
- **xreg**: The external regressors used in fitting (if given).
- **residuals**: Residuals from the fitted model. That is x minus fitted values.
- **fitted**: Fitted values (one-step forecasts).
- **...**: Other arguments.

Author(s)

Rob J Hyndman and Gabriel Caceres

See Also

- nnetar.

Description

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

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

Arguments

- **object**: a multivariate time series or multivariate 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(51, 99, by=3)`. 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 `mts`.
- **lambda**: Box-Cox transformation parameter. If `lambda="auto"`, then a transformation is automatically selected using `BoxCox.lambda`. The transformation is ignored if `NULL`. Otherwise, data transformed before model is estimated.
- **biasadj**: Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If `biasadj` is TRUE, an adjustment will be made to produce mean forecasts and fitted values.
- **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.

Details

For example, the function `forecast.mlm` makes multivariate forecasts based on the results produced by `tslm`. 
forecast.nnetar

Value

An object of class "mforecast".

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

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

An object of class "mforecast" 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 & Mitchell O'Hara-Wild

See Also

Other functions which return objects of class "mforecast" are forecast.mlm, forecast.varest.

Description

Returns forecasts and other information for univariate neural network models.

Usage

```r
## S3 method for class 'nnetar'
forecast(
  object,
  h = ifelse(object$m > 1, 2 * object$m, 10),
  PI = FALSE,
  level = c(80, 95),
  fan = FALSE,
```
Arguments

- **object**: An object of class "nnetar" resulting from a call to `nnetar`.
- **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`.
- **PI**: If TRUE, prediction intervals are produced, otherwise only point forecasts are calculated. If PI is FALSE, then level, fan, bootstrap and npaths are all ignored.
- **level**: Confidence level for prediction intervals.
- **fan**: If TRUE, level is set to `seq(51,99,by=3)`. This is suitable for fan plots.
- **xreg**: Future values of external regressor variables.
- **lambda**: Box-Cox transformation parameter. If `lambda="auto"`, then a transformation is automatically selected using `BoxCox.lambda`. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.
- **bootstrap**: If TRUE, then prediction intervals computed using simulations with resampled residuals rather than normally distributed errors. Ignored if `innov` is not NULL.
- **npaths**: Number of sample paths used in computing simulated prediction intervals.
- **innov**: Values to use as innovations for prediction intervals. Must be a matrix with `h` rows and `npaths` columns (vectors are coerced into a matrix). If present, bootstrap is ignored.
- **...**: Additional arguments passed to `simulate.nnetar`.

Details

Prediction intervals are calculated through simulations and can be slow. Note that if the network is too complex and overfits the data, the residuals can be arbitrarily small; if used for prediction interval calculations, they could lead to misleadingly small values. It is possible to use out-of-sample residuals to ameliorate this, see examples.

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.nnetar`.

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).
xreg The external regressors used in fitting (if given).
residuals Residuals from the fitted model. That is x minus fitted values.
fitted Fitted values (one-step forecasts)
... Other arguments

Author(s)
Rob J Hyndman and Gabriel Caceres

See Also
nnetar.

Examples

```r
## Fit & forecast model
fit <- nnetar(USAccDeaths, size=2)
fcast <- forecast(fit, h=20)
plot(fcast)

## Not run:
## Include prediction intervals in forecast
fcast2 <- forecast(fit, h=20, PI=TRUE, npaths=100)
plot(fcast2)

## Set up out-of-sample innovations using cross-validation
fit_cv <- CVar(USAccDeaths, size=2)
res_sd <- sd(fit_cv$residuals, na.rm=TRUE)
myinnovs <- rnorm(20*100, mean=0, sd=res_sd)
## Forecast using new innovations
fcast3 <- forecast(fit, h=20, PI=TRUE, npaths=100, innov=myinnovs)
plot(fcast3)
```

## End(Not run)
**Description**

Forecast 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
## 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,
  biasadj = NULL,
  xreg = NULL,
  newxreg = NULL,
  allow.multiplicative.trend = FALSE,
  ...
)

## S3 method for class 'stlm'
forecast(
  object,
  h = 2 * object$m,
  ...
)
```
level = c(80, 95),
fan = FALSE,
lambda = object$lambda,
biasadj = NULL,
newxreg = NULL,
allow.multiplicative.trend = FALSE,
...
)

stlf(
  y,
  h = frequency(x) * 2,
  s.window = 7 + 4 * seq(6),
  t.window = NULL,
  robust = FALSE,
  lambda = NULL,
  biasadj = FALSE,
  x = y,
  ...
)

Arguments

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.
etsmodel   The ets model specification passed to ets. By default it allows any non-seasonal
            model. If method!="ets", this argument is ignored.
forecastfunction
            An alternative way of specifying the function for forecasting the seasonally ad-
            justed series. If forecastfunction is not NULL, then method is ignored. Other-
            wise method is used to specify the forecasting method to be used.
h         Number of periods for forecasting.
level      Confidence level for prediction intervals.
fan         If TRUE, level is set to seq(51,99,by=3). This is suitable for fan plots.
lambda     Box-Cox transformation parameter. If lambda="auto", then a transformation is
            automatically selected using BoxCox::lambda. The transformation is ignored if
            NULL. Otherwise, data transformed before model is estimated.
biasadj   Use adjusted back-transformed mean for Box-Cox transformations. If trans-
            formed data is used to produce forecasts and fitted values, a regular back trans-
            formation will result in median forecasts. If biasadj is TRUE, an adjustment will
            be made to produce mean forecasts and fitted values.
xreg       Historical regressors to be used in auto.arima() when method=="arima".
newxreg    Future regressors to be used in forecast.Arima().
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 \texttt{forecast.stl}, \texttt{modelfunction} or \texttt{forecastfunction}.

\texttt{y} A univariate numeric time series of class \texttt{ts}.

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

\texttt{robust} If \texttt{TRUE}, robust fitting will used in the loess procedure within \texttt{stl}.

\texttt{modelfunction} An alternative way of specifying the function for modelling the seasonally adjusted series. If \texttt{modelfunction} is not \texttt{NULL}, then \texttt{method} is ignored. Otherwise \texttt{method} is used to specify the time series model to be used.

\texttt{model} Output from a previous call to \texttt{stlm}. If a \texttt{stlm} model is passed, this same model is fitted to \texttt{y} without re-estimating any parameters.

\texttt{x} Deprecated. Included for backwards compatibility.

\texttt{t.window} A number to control the smoothness of the trend. See \texttt{stl} for details.

Details

\texttt{stlm} takes a time series \texttt{y}, applies an STL decomposition, and models the seasonally adjusted data using the model passed as \texttt{modelfunction} or specified using \texttt{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 \texttt{forecast.stlm} for forecasting.

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

\texttt{stlf} combines \texttt{stlm} and \texttt{forecast.stlm}. It takes a \texttt{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 \texttt{forecastfunction}.

\texttt{forecast.stl} is similar to \texttt{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 \texttt{stlm} using either \texttt{method} or \texttt{modelfunction}. The \texttt{method} argument provides a shorthand way of specifying \texttt{modelfunction} for a few special cases. More generally, \texttt{modelfunction} can be any function with first argument a \texttt{ts} object, that returns an object that can be passed to \texttt{forecast}. For example, \texttt{forecastfunction=ar} uses the \texttt{ar} function for modelling the seasonally adjusted series.

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

tsmod <- stlm(USAccDeaths, modelfunction = ar)
plot(forecast(tsmod, h = 36))

decomp <- stl(USAccDeaths, s.window = "periodic")
plot(forecast(decomp))

plot(stlf(AirPassengers, lambda = 0))
```

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,
  biasadj = 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(51,99,by=3). This is suitable for fan plots.
- **lambda**: Box-Cox transformation parameter. If lambda="auto", then a transformation is automatically selected using BoxCox.lambda. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.
- **biasadj**: Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If biasadj is TRUE, an adjustment will be made to produce mean forecasts and fitted values.
- ... 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

`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.

Usage

```r
## 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,
  biasadj = FALSE,
  find.frequency = FALSE,
  allow.multiplicative.trend = FALSE,
  model = NULL,
  ...
)

## Default S3 method:
forecast(object, ...)

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

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(51, 99, by=3). 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. If lambda="auto", then a transformation is automatically selected using BoxCox.lambda. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.

biasadj: Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If biasadj is TRUE, an adjustment will be made to produce mean forecasts and fitted values.

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.

model: An object describing a time series model; e.g., one of of class ets, Arima, bats, tbats, or nnetar.

...: Additional arguments affecting the forecasts produced. If model=NULL, forecast.ts passes these to ets or stlf depending on the frequency of the time series. If model is not NULL, the arguments are passed to the relevant modelling function.

x: a numeric vector or time series of class ts.

Details

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

If model=NULL, 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).

If model is not NULL, forecast.ts will apply the model to the object time series, and then generate forecasts accordingly.

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,

Examples

WWWusage %>% forecast %>% plot
fit <- ets(window(WWWusage, end=60))
fc <- forecast(WWWusage, model=fit)

---

fourier  Fourier terms for modelling seasonality

Description

fourier returns a matrix containing terms from a Fourier series, up to order K, suitable for use in
Arima, auto.arima, or tslm.

Usage

fourier(x, K, h = NULL)

fourierf(x, K, h)

Arguments

x  Seasonal time series: a ts or a msts object

K  Maximum order(s) of Fourier terms

h  Number of periods ahead to forecast (optional)
Details

fourierf is deprecated, instead use the h argument in fourier.

The period of the Fourier terms is determined from the time series characteristics of x. When h is missing, the length of x also determines the number of rows for the matrix returned by fourier. Otherwise, the value of h determines the number of rows for the matrix returned by fourier, typically used for forecasting. The values within x are not used.

Typical use would omit h when generating Fourier terms for training a model and include h when generating Fourier terms for forecasting.

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 2*K 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*sum(K) columns.

Value

Numerical matrix.

Author(s)

Rob J Hyndman

See Also

seasonaldummy

Examples

library(ggplot2)

# Using Fourier series for a "ts" object
# K is chosen to minimize the AICc
deads.model <- auto.arima(USAccDeaths, xreg=fourier(USAccDeaths,K=5), seasonal=FALSE)
deads.fcast <- forecast(deads.model, xreg=fourier(USAccDeaths, K=5, h=36))
autoplot(deads.fcast) + xlab("Year")

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

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, ...)

## Default S3 method:
getResponse(object, ...)

## S3 method for class 'lm'
getResponse(object, ...)

## S3 method for class 'Arima'

## S3 method for class 'ets'

gas
getResponse(object, ...)

## S3 method for class 'fracdiff'
getResponse(object, ...)

## S3 method for class 'ar'
getResponse(object, ...)

## S3 method for class 'tbats'
getResponse(object, ...)

## S3 method for class 'bats'
getResponse(object, ...)

## S3 method for class 'mforecast'
getResponse(object, ...)

## S3 method for class 'baggedModel'
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.

Author(s)

Rob J Hyndman

gghistogram   Histogram with optional normal and kernel density functions

Description

Plots a histogram and density estimates using ggplot.

Usage

gghistogram(
x,
  add.normal = FALSE,
  add.kde = FALSE,
  add.rug = TRUE,
gglagplot

```r
  bins, boundary = 0
)
```

**Arguments**

- `x` a numerical vector.
- `add.normal` Add a normal density function for comparison
- `add.kde` Add a kernel density estimate for comparison
- `add.rug` Add a rug plot on the horizontal axis
- `bins` The number of bins to use for the histogram. Selected by default using the Friedman-Diaconis rule given by `nclass.FD`
- `boundary` A boundary between two bins.

**Value**

None.

**Author(s)**

Rob J Hyndman

**See Also**

- `hist`, `geom_histogram`

**Examples**

```r
gghistogram(lynx, add.kde=TRUE)
```

---

**gglagplot**

*Time series lag ggplots*

**Description**

Plots a lag plot using ggplot.
Usage

gglagplot(
  x,
  lags = ifelse(frequency(x) > 9, 16, 9),
  set.lags = 1:lags,
  diag = TRUE,
  diag.col = "gray",
  do.lines = TRUE,
  colour = TRUE,
  continuous = frequency(x) > 12,
  labels = FALSE,
  seasonal = TRUE,
  ...
)

gglagchull(
  x,
  lags = ifelse(frequency(x) > 1, min(12, frequency(x)), 4),
  set.lags = 1:lags,
  diag = TRUE,
  diag.col = "gray",
  ...
)

Arguments

x a time series object (type ts).
lags number of lag plots desired, see arg set.lags.
set.lags vector of positive integers specifying which lags to use.
diag logical indicating if the x=y diagonal should be drawn.
diag.col color to be used for the diagonal if(diag).
do.lines if TRUE, lines will be drawn, otherwise points will be drawn.
colour logical indicating if lines should be coloured.
continuous Should the colour scheme for years be continuous or discrete?
labels logical indicating if labels should be used.
seasonal Should the line colour be based on seasonal characteristics (TRUE), or sequential (FALSE).
...
Not used (for consistency with lag.plot)

Details

“gglagplot” will plot time series against lagged versions of themselves. Helps visualising 'auto-dependence' even when auto-correlations vanish.

“gglagchull” will layer convex hulls of the lags, layered on a single plot. This helps visualise the change in 'auto-dependence' as lags increase.
ggmonthplot

Value
None.

Author(s)
Mitchell O’Hara-Wild

See Also
lag.plot

Examples

gglagplot(woolyrnq)
gglagplot(woolyrnq,seasonal=FALSE)

lungDeaths <- cbind(mdeaths, fdeaths)
gglagplot(lungDeaths, lags=2)
gglagchull(lungDeaths, lags=6)
gglagchull(woolyrnq)

Description
Plots a subseries plot using ggplot. Each season is plotted as a separate mini time series. The blue lines represent the mean of the observations within each season.

Usage

ggmonthplot(x, labels = NULL, times = time(x), phase = cycle(x), ...)
ggsubseriesplot(x, labels = NULL, times = time(x), phase = cycle(x), ...)

Arguments
x a time series object (type ts).
labels A vector of labels to use for each ’season’
times A vector of times for each observation
phase A vector of seasonal components
... Not used (for consistency with monthplot)
ggseasonplot

Details

The ggmonthplot function is simply a wrapper for ggsubseriesplot as a convenience for users familiar with monthplot.

Value

Returns an object of class ggplot.

Author(s)

Mitchell O’Hara-Wild

See Also

monthplot

Examples

ggsubseriesplot(AirPassengers)
ggsubseriesplot(woolyrnq)

---

ggseasonplot  Seasonal plot

Description

Plots a seasonal plot as described in Hyndman and Athanasopoulos (2014, chapter 2). This is like a time plot except that the data are plotted against the seasons in separate years.

Usage

ggseasonplot(
  x,
  season.labels = NULL,
  year.labels = FALSE,
  year.labels.left = FALSE,
  type = NULL,
  col = NULL,
  continuous = FALSE,
  polar = FALSE,
  labelgap = 0.04,
  ...
)

seasonplot(
  x,
Arguments

x  a numeric vector or time series of class ts.
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). Not yet supported for ggseasonplot.
col  Colour
continuous  Should the colour scheme for years be continuous or discrete?
polar  Plot the graph on seasonal coordinates
labelgap  Distance between year labels and plotted lines
...  additional arguments to plot.
s  seasonal frequency of x
main  Main title.
xlab  X-axis label.
ylab  Y-axis label.

Value

None.

Author(s)

Rob J Hyndman & Mitchell O’Hara-Wild

References

See Also

monthplot

Examples

ggseaseasonplot(AirPassengers, col=rainbow(12), year.labels=TRUE)
ggseaseasonplot(AirPassengers, year.labels=TRUE, continuous=TRUE)

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

---

ggtsdisplay Time series display

Description

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

Usage

ggtsdisplay(
  x,
  plot.type = c("partial", "histogram", "scatter", "spectrum"),
  points = TRUE,
  smooth = FALSE,
  lag.max,
  na.action = na.contiguous,
  theme = NULL,
  ...
)

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

- **x**: a numeric vector or time series of class ts.
- **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.
- **smooth**: logical flag indicating whether to show a smooth loess curve superimposed on the time plot.
- **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`.
- **theme**: adds a ggplot element to each plot, typically a theme.
- **ci.type**: type of confidence limits for ACF that is passed to acf. Should the confidence limits assume a white noise input or for lag k an MA(k – 1) input?
- **main**: Main title.
- **xlab**: X-axis label.
- **ylab**: Y-axis label.
- **pch**: Plotting character.
- **cex**: Character size.

Details

ggtsdisplay will produce the equivalent plot using ggplot graphics.

Value

None.

Author(s)

Rob J Hyndman

References


See Also

`plot.ts, Acf, spec.ar`
Examples

library(ggplot2)
ggtsdisplay(USAccDeaths, plot.type="scatter", theme=theme_bw())

tsdisplay(diff(WWWWusage))
ggtsdisplay(USAccDeaths, plot.type="scatter")

gold                      Daily morning gold prices

Description


Usage
gold

Format

Time series data

Examples
tsdisplay(gold)

is.acf                      Is an object a particular model type?

Description

Returns true if the model object is of a particular type

Usage

is.acf(x)
is.Arima(x)
is.baggedModel(x)
is.bats(x)
is.ets(x)
### is.constant

- **is.modelAR(x)**
- **is.stlm(x)**
- **is.nnetar(x)**
- **is.nnetarmodels(x)**

#### Arguments

- **x**
  - object to be tested

#### Description

Returns true if the object's numerical values do not vary.

#### Usage

- **is.constant(x)**

#### Arguments

- **x**
  - object to be tested

### is.forecast

- **is.mforecast(x)**
- **is.splineforecast(x)**

#### Arguments

- **x**
  - object to be tested
Description

`ma()` computes a simple moving average smoother of a given time series.

Usage

```r
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 for even orders.

Details

The moving average smoother averages the nearest `order` periods of each observation. As neighbouring observations of a time series are likely to be similar in value, averaging eliminates some of the randomness in the data, leaving a smooth trend-cycle component.

\[
\hat{T}_t = \frac{1}{m} \sum_{j=-k}^{k} y_{t+j}
\]

where \( k = \frac{m-1}{2} \)

When an even `order` is specified, the observations averaged will include one more observation from the future than the past (\( k \) is rounded up). If `centre` is TRUE, the value from two moving averages (where \( k \) is rounded up and down respectively) are averaged, centering the moving average.

Value

Numerical time series object containing the simple moving average smoothed values.

Author(s)

Rob J Hyndman

See Also

decompose
Examples

```r
plot(wineind)
sm <- ma(wineind, order=12)
lines(sm, col="red")
```

---

meanf

**Mean Forecast**

Description

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

Usage

```r
meanf(
  y, h = 10, level = c(80, 95),
  fan = FALSE, lambda = NULL, biasadj = FALSE, bootstrap = FALSE,
  npaths = 5000, x = y
)
```

Arguments

- `y` a numeric vector or time series of class `ts`
- `h` Number of periods for forecasting
- `level` Confidence levels for prediction intervals.
- `fan` If TRUE, level is set to seq(51,99,by=3). This is suitable for fan plots.
- `lambda` Box-Cox transformation parameter. If lambda="auto", then a transformation is automatically selected using BoxCox.lambda. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.
- `biasadj` Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If biasadj is TRUE, an adjustment will be made to produce mean forecasts and fitted values.
- `bootstrap` If TRUE, use a bootstrap method to compute prediction intervals. Otherwise, assume a normal distribution.
- `npaths` Number of bootstrapped sample paths to use if bootstrap==TRUE.
- `x` Deprecated. Included for backwards compatibility.
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)
```
**modelAR**

**Time Series Forecasts with a user-defined model**

**Description**

Experimental function to forecast univariate time series with a user-defined model

**Usage**

```r
modelAR(
  y, 
  p, 
  P = 1, 
  FUN, 
  predict.FUN, 
  xreg = NULL, 
  lambda = NULL, 
  model = NULL, 
  subset = NULL, 
  scale.inputs = FALSE, 
  x = y, 
  ...
)
```

**Arguments**

- **y**: A numeric vector or time series of class `ts`.
- **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.
- **FUN**: Function used for model fitting. Must accept argument `x` and `y` for the predictors and response, respectively (*formula* object not currently supported).
- **predict.FUN**: Prediction function used to apply `FUN` to new data. Must accept an object of class `FUN` as its first argument, and a data frame or matrix of new data for its second argument. Additionally, it should return fitted values when new data is omitted.
- **xreg**: Optionally, a vector or matrix of external regressors, which must have the same number of rows as `y`. Must be numeric.
- **lambda**: Box-Cox transformation parameter. If `lambda="auto"`, then a transformation is automatically selected using `BoxCox.lambda`. The transformation is ignored if `NULL`. Otherwise, data transformed before model is estimated.
modelAR

model  Output from a previous call to nnetar. If model is passed, this same model is fitted to y without re-estimating any parameters.
subset  Optional vector specifying a subset of observations to be used in the fit. Can be an integer index vector or a logical vector the same length as y. All observations are used by default.
scale.inputs  If TRUE, inputs are scaled by subtracting the column means and dividing by their respective standard deviations. If lambda is not NULL, scaling is applied after Box-Cox transformation.
x  Deprecated. Included for backwards compatibility.
...  Other arguments passed to FUN for modelAR.

Details

This is an experimental function and only recommended for advanced users. The selected model is fitted with lagged values of y as inputs. The inputs are for lags 1 to p, and lags m to mP where m=frequency(y). If xreg is provided, its columns are also used as inputs. If there are missing values in y or xreg, the corresponding rows (and any others which depend on them as lags) are omitted from the fit. The model is trained for one-step forecasting. Multi-step forecasts are computed recursively.

Value

Returns an object of class "modelAR".
The function summary is used to obtain and print a summary of the results.
The generic accessor functions fitted.values and residuals extract useful features of the value returned by nnetar.

model  A list containing information about the fitted model
method  The name of the forecasting method as a character string
x  The original time series.
xreg  The external regressors used in fitting (if given).
residuals  Residuals from the fitted model. That is x minus fitted values.
fitted  Fitted values (one-step forecasts)
...  Other arguments

Author(s)

Rob J Hyndman and Gabriel Caceres
Description

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

Usage

monthdays(x)

Arguments

x  time series

Details

Useful for month length adjustments

Value

Time series

Author(s)

Rob J Hyndman

See Also

bizdays

Examples

par(mfrow=c(2,1))
plot(ldeaths,xlab="Year",ylab="pounds",
     main="Monthly deaths from lung disease (UK)")
ldeaths.adj <- ldeaths/monthdays(ldeaths)*365.25/12
plot(ldeaths.adj,xlab="Year",ylab="pounds",
     main="Adjusted monthly deaths from lung disease (UK)")
mstl

Multiple seasonal decomposition

Description

Decompose a time series into seasonal, trend and remainder components. Seasonal components are estimated iteratively using STL. Multiple seasonal periods are allowed. The trend component is computed for the last iteration of STL. Non-seasonal time series are decomposed into trend and remainder only. In this case, supsmu is used to estimate the trend. Optionally, the time series may be Box-Cox transformed before decomposition. Unlike stl, mstl is completely automated.

Usage

mstl(x, lambda = NULL, iterate = 2, s.window = 7 + 4 * seq(6), ...)

Arguments

x  Univariate time series of class msts or ts.
lambda  Box-Cox transformation parameter. If lambda="auto", then a transformation is automatically selected using BoxCox.lambda. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.
iterate  Number of iterations to use to refine the seasonal component.
s.window  Seasonal windows to be used in the decompositions. If scalar, the same value is used for all seasonal components. Otherwise, it should be a vector of the same length as the number of seasonal components (or longer).
...
Other arguments are passed to stl.

See Also

stl, supsmu

Examples

library(ggplot2)
mstl(taylor) %>% autoplot()
mstl(AirPassengers, lambda = "auto") %>% autoplot()
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 period 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"). If there is only one seasonal period (i.e., length(seasonal.periods)==1), then the object is of class "ts".

Author(s)

Slava Razbash and Rob J Hyndman

Examples

x <- msts(taylor, seasonal.periods=c(2*24,2*24*7,2*24*365), start=2000+22/52)
y <- msts(USAccDeaths, seasonal.periods=12, start=1949)
Interpolate missing values in a time series

Description

By default, uses linear interpolation for non-seasonal series. For seasonal series, a robust STL decomposition is first computed. Then a linear interpolation is applied to the seasonally adjusted data, and the seasonal component is added back.

Usage

na.interp(
  x,
  lambda = NULL,
  linear = (frequency(x) <= 1 | sum(!is.na(x)) <= 2 * frequency(x))
)

Arguments

x          time series
lambda     Box-Cox transformation parameter. If lambda="auto", then a transformation is automatically selected using BoxCox.lambda. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.
linear     Should a linear interpolation be used.

Details

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

Value

time series

Author(s)

Rob J Hyndman

See Also

tssoutliers

Examples

data(gold)
plot(na.interp(gold))
**Description**

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

**Usage**

```r
ndiffs(
  x, 
  alpha = 0.05, 
  test = c("kpss", "adf", "pp"),
  type = c("level", "trend"),
  max.d = 2,
  ...
)
```

**Arguments**

- `x`: A univariate time series
- `alpha`: Level of the test, possible values range from 0.01 to 0.1.
- `test`: Type of unit root test to use
- `type`: Specification of the deterministic component in the regression
- `max.d`: Maximum number of non-seasonal differences allowed
- `...`: Additional arguments to be passed on to the unit root test

**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`.

**Value**

An integer indicating the number of differences required for stationarity.

**Author(s)**

Rob J Hyndman, Slava Razbash & Mitchell O’Hara-Wild
References


See Also

*auto.arima* and *ndiffs*

Examples

```r
ndiffs(WWWusage)
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**

```r
nnetar(
  y,
  p,
  P = 1,
  size,
  repeats = 20,
  xreg = NULL,
  lambda = NULL,
  model = NULL,
  subset = NULL,
  scale.inputs = TRUE,
  x = y,
  ...
)
```
Arguments

- **y**
  A numeric vector or time series of class `ts`.

- **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 (including external regressors, if given) plus 1.

- **repeats**
  Number of networks to fit with different random starting weights. These are then averaged when producing forecasts.

- **xreg**
  Optionally, a vector or matrix of external regressors, which must have the same number of rows as `y`. Must be numeric.

- **lambda**
  Box-Cox transformation parameter. If `lambda="auto"`, then a transformation is automatically selected using `BoxCox.lambda`. The transformation is ignored if `NULL`. Otherwise, data transformed before model is estimated.

- **model**
  Output from a previous call to `nnetar`. If model is passed, this same model is fitted to `y` without re-estimating any parameters.

- **subset**
  Optional vector specifying a subset of observations to be used in the fit. Can be an integer index vector or a logical vector the same length as `y`. All observations are used by default.

- **scale.inputs**
  If TRUE, inputs are scaled by subtracting the column means and dividing by their respective standard deviations. If `lambda` is not `NULL`, scaling is applied after Box-Cox transformation.

- **x**
  Deprecated. Included for backwards compatibility.

- **...**
  Other arguments passed to `nnet` for `nnetar`.

Details

A feed-forward neural network is fitted with lagged values of `y` as inputs and a single hidden layer with `size` nodes. The inputs are for lags 1 to `p`, and lags `m` to `mP` where `m=frequency(y)`. If `xreg` is provided, its columns are also used as inputs. If there are missing values in `y` or `xreg`, the corresponding rows (and any others which depend on them as lags) are omitted from the fit. 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.

For non-seasonal data, the fitted model is denoted as an NNAR(p,k) model, where k is the number of hidden nodes. This is analogous to an AR(p) model but with nonlinear functions. For seasonal data, the fitted model is called an NNAR(p,P,k)[m] model, which is analogous to an ARIMA(p,0,0)(P,0,0)[m] model but with nonlinear functions.
Value

Returns an object of class "nnetar".

The function summary is used to obtain and print a summary of the results.

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

model A list containing information about the fitted model
method The name of the forecasting method as a character string
x The original time series.
xreg The external regressors used in fitting (if given).
residuals Residuals from the fitted model. That is x minus fitted values.
fitted Fitted values (one-step forecasts)
... Other arguments

Author(s)

Rob J Hyndman and Gabriel Caceres

Examples

fit <- nnetar(lynx)
fcast <- forecast(fit)
plot(fcast)

## Arguments can be passed to nnet()
fit <- nnetar(lynx, decay=0.5, maxit=150)
plot(forecast(fit))
lines(lynx)

## Fit model to first 100 years of lynx data
fit <- nnetar(window(lynx,end=1920), decay=0.5, maxit=150)
plot(forecast(fit,h=14))
lines(lynx)

## Apply fitted model to later data, including all optional arguments
fit2 <- nnetar(window(lynx,start=1921), model=fit)

nsdiffs

Number of differences required for a seasonally stationary series

Description

Functions to estimate the number of differences required to make a given time series stationary. nsdiffs estimates the number of seasonal differences necessary.
Usage

```r
nsdiffs(
  x,
  alpha = 0.05,
  m = frequency(x),
  test = c("seas", "ocsb", "hegy", "ch"),
  max.D = 1,
  ...
)
```

Arguments

- `x` A univariate time series
- `alpha` Level of the test, possible values range from 0.01 to 0.1.
- `m` Deprecated. Length of seasonal period
- `test` Type of unit root test to use
- `max.D` Maximum number of seasonal differences allowed
- `...` Additional arguments to be passed on to the unit root test

Details

`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).

Several different tests are available:

- If `test="seas"` (default), a measure of seasonal strength is used, where differencing is selected if the seasonal strength (Wang, Smith & Hyndman, 2006) exceeds 0.64 (based on minimizing MASE when forecasting using auto.arima on M3 and M4 data).
- If `test="ch"`, the Canova-Hansen (1995) test is used (with null hypothesis of deterministic seasonality)
- If `test="hegy"`, the Hylleberg, Engle, Granger & Yoo (1990) test is used.
- If `test="ocsb"`, the Osborn-Chui-Smith-Birchenhall (1988) test is used (with null hypothesis that a seasonal unit root exists).

Value

An integer indicating the number of differences required for stationarity.

Author(s)

Rob J Hyndman, Slava Razbash and Mitchell O’Hara-Wild
References

Wang, X, Smith, KA, Hyndman, RJ (2006) "Characteristic-based clustering for time series data", 
*Data Mining and Knowledge Discovery*, 13(3), 335-364.

Osborn DR, Chui APL, Smith J, and Birchenhall CR (1988) "Seasonality and the order of integration 

Canova F and Hansen BE (1995) "Are Seasonal Patterns Constant over Time? A Test for Seasonal 


See Also

auto.arima, ndiffs, ocsb.test, hegy.test, and ch.test

Examples

nsdiffs(AirPassengers)

---

**ocsb.test**

*Osborn, Chui, Smith, and Birchenhall Test for Seasonal Unit Roots*

**Description**

An implementation of the Osborn, Chui, Smith, and Birchenhall (OCSB) test.

**Usage**

```r
ocsb.test(x, lag.method = c("fixed", "AIC", "BIC", "AICc"), maxlag = 0)
```

**Arguments**

- `x` a univariate seasonal time series.
- `lag.method` a character specifying the lag order selection method.
- `maxlag` the maximum lag order to be considered by `lag.method`.

**Details**

The regression equation may include lags of the dependent variable. When `lag.method = "fixed"`, 
the lag order is fixed to `maxlag`; otherwise, `maxlag` is the maximum number of lags considered 
in a lag selection procedure that minimises the `lag.method` criterion, which can be AIC or BIC or 
corrected AIC, AICc, obtained as AIC + (2k(k+1))/(n-k-1), where k is the number of parameters 
and n is the number of available observations in the model.

Critical values for the test are based on simulations, which has been smoothed over to produce 
critical values for all seasonal periods.
Value

ocsb.test returns a list of class "OCSBtest" with the following components: * statistics the value of the test statistics. * pvalues the p-values for each test statistics. * method a character string describing the type of test. * data.name a character string giving the name of the data. * fitted.model the fitted regression model.

References


See Also

nsdiffs

Examples

ocsb.test(AirPassengers)

plot.Arima

*Plot characteristic roots from ARIMA model*

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", ...)

## S3 method for class 'Arima'
autoplot(object, type = c("both", "ar", "ma"), ...)

## S3 method for class 'ar'
autoplot(object, ...)
```
**plot.Arima**

**Arguments**

- `x` Object of class “Arima” or “ar”.
- `type` Determines if both AR and MA roots are plotted, or 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`.
- `object` Object of class “Arima” or “ar”. Used for ggplot graphics (S3 method consistency).

**Details**

`autoplot` will produce an equivalent plot as a ggplot object.

**Value**

None. Function produces a plot

**Author(s)**

Rob J Hyndman & Mitchell O’Hara-Wild

**See Also**

`Arima`, `ar`

**Examples**

```r
library(ggplot2)

fit <- Arima(WWWusage, order = c(3, 1, 0))
plot(fit)
autoplot(fit)

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

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

Plot components from BATS model

Description

Produces a plot of the level, slope and seasonal components from a BATS or TBATS model. The plotted components are Box-Cox transformed using the estimated transformation parameter.

Usage

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

## S3 method for class 'tbats'
autoplot(object, range.bars = FALSE, ...)

## S3 method for class 'bats'
autoplot(object, range.bars = FALSE, ...)

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

Arguments

- `x` Object of class “bats.tbats”.
- `main` Main title for plot.
- `...` Other plotting parameters passed to `par`.
- `object` Object of class “bats.tbats”.
- `range.bars` Logical indicating if each plot should have a bar at its right side representing relative size. If NULL, automatic selection takes place.

Value

None. Function produces a plot

Author(s)

Rob J Hyndman

See Also

- `bats`, `tbats`
Examples

```r
## Not run:
fit <- tbats(USAccDeaths)
plot(fit)
autoplot(fit, range.bars = TRUE)
## 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, ...)
## S3 method for class 'ets'
autoplot(object, range.bars = NULL, ...)
```

Arguments

- **x**: Object of class “ets”.
- **...**: Other plotting parameters to affect the plot.
- **object**: Object of class “ets”. Used for ggplot graphics (S3 method consistency).
- **range.bars**: Logical indicating if each plot should have a bar at its right side representing relative size. If NULL, automatic selection takes place.

Details

autoplot will produce an equivalent plot as a ggplot object.

Value

None. Function produces a plot

Author(s)

Rob J Hyndman & Mitchell O’Hara-Wild

See Also

ets
plot.forecast

Examples

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

library(ggplot2)
autoplot(fit)

plot.forecast  
Forecast plot

Description

Plots historical data with forecasts and prediction intervals.

Usage

## S3 method for class 'forecast'
plot(
x, 
include, 
PI = TRUE, 
showgap = TRUE, 
shaded = TRUE, 
shadebars = (length(x$mean) < 5), 
shadecols = NULL, 
col = 1, 
fcol = 4, 
pi.col = 1, 
pi.lty = 2, 
ylim = NULL, 
main = NULL, 
ylab = "", 
lab = "", 
type = "l", 
flty = 1, 
flwd = 2,
...
)

## S3 method for class 'forecast'
autoplot(
object, 
include, 
PI = TRUE,
shadecols = c("#596DD5", "#D5DBFF"),
fcol = "#0000AA",
flwd = 0.5,
...

## S3 method for class 'splineforecast'
autoplot(object, PI = TRUE, ...)

## S3 method for class 'forecast'
autolayer(object, series = NULL, PI = TRUE, showgap = TRUE, ...)

## 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. Default is all values.
- **PI** Logical flag indicating whether to plot prediction intervals.
- **showgap** If showgap=FALSE, the gap between the historical observations and the forecasts is removed.
- **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.
- **pi.col** If shaded=FALSE and PI=TRUE, the prediction intervals are plotted in this colour.
- **pi.lty** If shaded=FALSE and PI=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.
- **type** 1-character string giving the type of plot desired. As for `plot.default`.
- **flty** Line type for the forecast line.
- **flwd** Line width for the forecast line.
- **...** Other plotting parameters to affect the plot.
object  Forecast object produced by `forecast`. Used for ggplot graphics (S3 method consistency).
series  Matches an unidentified forecast layer with a coloured object on the plot.
fitcol  Line colour for fitted values.
pch  Plotting character (if type="p" or type="o").

Details

`autoplot` will produce a ggplot object.

Value

None.

Author(s)

Rob J Hyndman & Mitchell O’Hara-Wild

References


See Also

`plot.ts`

Examples

```r
library(ggplot2)

wine.fit <- hw(wineind,h=48)
plot(wine.fit)
autoplot(wine.fit)

fit <- tslm(wineind ~ fourier(wineind,4))
fcast <- forecast(fit, newdata=data.frame(fourier(wineind,4,20)))
autoplot(fcast)

fcast <- splinef(airmiles,h=5)
plot(fcast)
autoplot(fcast)
```
residuals.forecast  Residuals for various time series models

Description

Returns time series of residuals from a fitted model.

Usage

```r
## S3 method for class 'forecast'
residuals(object, type = c("innovation", "response"), ...)

## S3 method for class 'ar'
residuals(object, type = c("innovation", "response"), ...)

## S3 method for class 'Arima'
residuals(object, type = c("innovation", "response", "regression"), h = 1, ...)

## S3 method for class 'bats'
residuals(object, type = c("innovation", "response"), h = 1, ...)

## S3 method for class 'tbats'
residuals(object, type = c("innovation", "response"), h = 1, ...)

## S3 method for class 'ets'
residuals(object, type = c("innovation", "response"), h = 1, ...)

## S3 method for class 'ARFIMA'
residuals(object, type = c("innovation", "response"), ...)

## S3 method for class 'nnetar'
residuals(object, type = c("innovation", "response"), h = 1, ...)

## S3 method for class 'stlm'
residuals(object, type = c("innovation", "response"), ...)

## S3 method for class 'tslm'
residuals(object, type = c("innovation", "response", "deviance"), ...)
```

Arguments

- `object` An object containing a time series model of class ar, Arima, bats, ets, arfima, nnetar or stlm. If object is of class forecast, then the function will return `object$residuals` if it exists, otherwise it returns the differences between the observations and their fitted values.
- `type` Type of residual.
Other arguments not used.

If type='response', then the fitted values are computed for h-step forecasts.

Details

Innovation residuals correspond to the white noise process that drives the evolution of the time series model. Response residuals are the difference between the observations and the fitted values (equivalent to h-step forecasts). For functions with no h argument, h=1. For homoscedastic models, the innovation residuals and the response residuals for h=1 are identical. Regression residuals are available for regression models with ARIMA errors, and are equal to the original data minus the effect of the regression variables. If there are no regression variables, the errors will be identical to the original series (possibly adjusted to have zero mean). arima.errors is a deprecated function which is identical to residuals.Arima(object, type="regression"). For nnetar objects, when type="innovations" and lambda is used, a matrix of time-series consisting of the residuals from each of the fitted neural networks is returned.

Value

A ts object.

Author(s)

Rob J Hyndman

See Also

fitted.Arima, checkresiduals.

Examples

```r
fit <- Arima(lynx, order=c(4,0,0), lambda=0.5)
plot(residuals(fit))
plot(residuals(fit, type='response'))
```

Description

`rwf()` returns forecasts and prediction intervals for a random walk with drift model applied to y. This is equivalent to an ARIMA(0,1,0) model with an optional drift coefficient. naive() is simply a wrapper to rwf() for simplicity. snaive() returns forecasts and prediction intervals from an ARIMA(0,0,0)(0,1,0)m model where m is the seasonal period.
Usage

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

```r
naive(
  y,
  h = 10,
  level = c(80, 95),
  fan = FALSE,
  lambda = NULL,
  biasadj = FALSE,
  ..., 
  x = y
)
```

```r
snaive(
  y,
  h = 2 * frequency(x),
  level = c(80, 95),
  fan = FALSE,
  lambda = NULL,
  biasadj = FALSE,
  ..., 
  x = y
)
```

Arguments

- **y**: a numeric vector or time series of class `ts`
- **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(51,99,by=3)`. This is suitable for fan plots.
- **lambda**: Box-Cox transformation parameter. If `lambda="auto"`, then a transformation is automatically selected using `BoxCox.lambda`. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.
biasadj  Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If biasadj is TRUE, an adjustment will be made to produce mean forecasts and fitted values.

Additional arguments affecting the forecasts produced. If model=NULL, forecast.ts passes these to ets or stlf depending on the frequency of the time series. If model is not NULL, the arguments are passed to the relevant modelling function.

x  Deprecated. Included for backwards compatibility.

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 (as in naive), the drift parameter $c=0$. Forecast standard errors allow for uncertainty in estimating the drift parameter (unlike the corresponding forecasts obtained by fitting an ARIMA model directly).

The seasonal naive model is

$$Y_t = Y_{t-m} + Z_t$$

where $Z_t$ is a normal iid error.

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 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)
seasadj

Author(s)

Rob J Hyndman

See Also

Arima

Examples

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

plot(naive(gold,h=50),include=200)

plot(snaive(wineind))

seasadj Seasonal adjustment

Description

Returns seasonally adjusted data constructed by removing the seasonal component.

Usage

seasadj(object, ...)

## S3 method for class 'stl'
seasadj(object, ...)

## S3 method for class 'mstl'
seasadj(object, ...)

## S3 method for class 'decomposed.ts'
seasadj(object, ...)

## S3 method for class 'tbats'
seasadj(object, ...)

## S3 method for class 'seas'
seasadj(object, ...)
seasonal

Arguments

object Object created by decompose, stl or tbats.

... Other arguments not currently used.

Value

Univariate time series.

Author(s)

Rob J Hyndman

See Also

stl, decompose, tbats.

Examples

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

seasonal Extract components from a time series decomposition

Description

Returns a univariate time series equal to either a seasonal component, trend-cycle component or
remainder component from a time series decomposition.

Usage

seasonal(object)

trendcycle(object)

remainder(object)

Arguments

object Object created by decompose, stl or tbats.

Value

Univariate time series.
seasonaldummy

Author(s)

Rob J Hyndman

See Also

stl, decompose, tbats, seasadj.

Examples

plot(USAccDeaths)
fit <- stl(USAccDeaths, s.window="periodic")
lines(trendcycle(fit),col="red")

library(ggplot2)
autoplot(cbind(
  Data=USAccDeaths,
  Seasonal=seasonal(fit),
  Trend=trendcycle(fit),
  Remainder=remainder(fit)),
  facets=TRUE) +
ylab("") + xlab("Year")

seasonaldummy

Seasonal dummy variables

Description

seasonaldummy returns a matrix of dummy variables suitable for use in Arima, auto.arima or tslm. The last season is omitted and used as the control.

Usage

seasonaldummy(x, h = NULL)
seasonaldummyf(x, h)

Arguments

x  Seasonal time series: a ts or a msts object
h  Number of periods ahead to forecast (optional)

Details

seasonaldummyf is deprecated, instead use the h argument in seasonaldummy.

The number of dummy variables is determined from the time series characteristics of x. When h is missing, the length of x also determines the number of rows for the matrix returned by seasonaldummy. The value of h determines the number of rows for the matrix returned by seasonaldummy, typically used for forecasting. The values within x are not used.
ses is a function in the **forecast** package. It performs exponential smoothing forecasts for a given time series. Here’s a breakdown of the documentation:

### Description

Returns forecasts and other information for exponential smoothing forecasts applied to `y`.

### Usage

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

### Examples

```r
plot(ldeaths)

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

# A simpler approach to seasonal dummy variables
deaths.lm <- tslm(ldeaths ~ season)
deaths.fcast <- forecast(deaths.lm, h=36)
plot(deaths.fcast)
```
\begin{verbatim}
x = y,
...

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

hw(
  y,
  h = 2 * frequency(x),
  seasonal = c("additive", "multiplicative"),
  damped = FALSE,
  level = c(80, 95),
  fan = FALSE,
  initial = c("optimal", "simple"),
  exponential = FALSE,
  alpha = NULL,
  beta = NULL,
  gamma = NULL,
  phi = NULL,
  lambda = NULL,
  biasadj = FALSE,
  x = y,
  ...
)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{y} \hspace{1cm} a numeric vector or time series of class \texttt{ts}
  \item \texttt{h} \hspace{1cm} Number of periods for forecasting.
  \item \texttt{level} \hspace{1cm} Confidence level for prediction intervals.
  \item \texttt{fan} \hspace{1cm} If \texttt{TRUE}, level is set to \texttt{seq(51,99,by=3)}. This is suitable for fan plots.
  \item \texttt{initial} \hspace{1cm} Method used for selecting initial state values. If \texttt{optimal}, the initial values are optimized along with the smoothing parameters using \texttt{ets}. If \texttt{simple}, the
\end{itemize}
initial values are set to values obtained using simple calculations on the first few observations. See Hyndman & Athanasopoulos (2014) for details.

alpha
Value of smoothing parameter for the level. If NULL, it will be estimated.

lambda
Box-Cox transformation parameter. If lambda="auto", then a transformation is automatically selected using BoxCox. lambda. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.

biasadj
Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If biasadj is TRUE, an adjustment will be made to produce mean forecasts and fitted values.

x
Deprecated. Included for backwards compatibility.

... Other arguments passed to forecast.ets.

damped
If TRUE, use a damped trend.

exponential
If TRUE, an exponential trend is fitted. Otherwise, the trend is (locally) linear.

beta
Value of smoothing parameter for the trend. If NULL, it will be estimated.

phi
Value of damping parameter if damped=TRUE. If NULL, it will be estimated.

seasonal
Type of seasonality in hw model. "additive" or "multiplicativc".

gamma
Value of smoothing parameter for the seasonal component. If NULL, it will be estimated.

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.
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,
  ...
)
```

```r
## 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 'ar'
simulate(
  object,
  nsim = object$n.used,
  seed = NULL,
  future = TRUE,
  bootstrap = FALSE,
  innov = NULL,
  ...
)

## S3 method for class 'lagwalk'
simulate(
  object,
  nsim = length(object$x),
  seed = NULL,
  future = TRUE,
  bootstrap = FALSE,
  innov = NULL,
  lambda = object$lambda,
  ...
)

## S3 method for class 'fracdiff'
simulate(
  object,
  nsim = object$n,
  seed = NULL,
  future = TRUE,
  bootstrap = FALSE,
  innov = NULL,
  ...
)

## S3 method for class 'nnetar'
simulate(
  object,
  nsim = length(object$x),
simulate.ets

```r
seed = NULL,
xreg = NULL,
future = TRUE,
bootstrap = FALSE,
innov = NULL,
lambda = object$lambda,
...
)
```

## S3 method for class 'modelAR'
simulate(
  object,
  nsim = length(object$x),
  seed = NULL,
  xreg = NULL,
  future = TRUE,
  bootstrap = FALSE,
  innov = NULL,
  lambda = object$lambda,
  ...
)

## S3 method for class 'tbats'
simulate(
  object,
  nsim = length(object$y),
  seed = NULL,
  xreg = NULL,
  future = TRUE,
  bootstrap = FALSE,
  innov = NULL,
  ...
)
```

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An object of class &quot;ets&quot;, &quot;Arima&quot;, &quot;ar&quot; or &quot;nnetar&quot;.</td>
</tr>
<tr>
<td>nsim</td>
<td>Number of periods for the simulated series. Ignored if either xreg or innov are not NULL. Otherwise the default is the length of series used to train model (or 100 if no data found).</td>
</tr>
<tr>
<td>seed</td>
<td>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.</td>
</tr>
<tr>
<td>future</td>
<td>Produce sample paths that are future to and conditional on the data in object. Otherwise simulate unconditionally.</td>
</tr>
<tr>
<td>bootstrap</td>
<td>Do simulation using resampled errors rather than normally distributed errors or errors provided as innov.</td>
</tr>
<tr>
<td>innov</td>
<td>A vector of innovations to use as the error series. Ignored if bootstrap==TRUE. If not NULL, the value of nsim is set to length of innov.</td>
</tr>
</tbody>
</table>
... Other arguments, not currently used.

xreg New values of xreg to be used for forecasting. The value of nsim is set to the number of rows of xreg if it is not NULL.

lambda Box-Cox transformation parameter. If lambda="auto", then a transformation is automatically selected using BoxCox.lambda. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.

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

et, Arima, auto.arima, ar, arfima, nnetar.

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

Arguments

- **y**: a numeric vector or time series of class ts
- **h**: Number of periods for forecasting
- **level**: Confidence level for prediction intervals.
- **fan**: If TRUE, level is set to seq(51,99,by=3). This is suitable for fan plots.
- **lambda**: Box-Cox transformation parameter. If lambda="auto", then a transformation is automatically selected using BoxCox.lambda. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.
- **biasadj**: Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If biasadj is TRUE, an adjustment will be made to produce mean forecasts and fitted values.
- **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.
- **x**: Deprecated. Included for backwards compatibility.

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.
StatForecast

Author(s)

Rob J Hyndman

References


See Also

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

Examples

```r
fcast <- splinef(uspop, h=5)
plot(fcast)
summary(fcast)
```

---

StatForecast  Forecast plot

Description

Generates forecasts from `forecast.ts` and adds them to the plot. Forecasts can be modified via sending forecast specific arguments above.

Usage

StatForecast

GeomForecast

gem_forecast(
  mapping = NULL,
  data = NULL,
  stat = "forecast",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  PI = TRUE,
  showgap = TRUE,
  series = NULL,
  ...
)
Arguments

- **mapping**: Set of aesthetic mappings created by `aes` or `aes_*`. If specified and `inherit.aes` = `TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

- **data**: The data to be displayed in this layer. There are three options:
  - If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot`.
  - A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify` for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data.

- **stat**: The stat object to use calculate the data.

- **position**: Position adjustment, either as a string, or the result of a call to a position adjustment function.

- **na.rm**: If `FALSE` (the default), removes missing values with a warning. If `TRUE` silently removes missing values.

- **show.legend**: logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes.

- **inherit.aes**: If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders`.

- **PI**: If `FALSE`, confidence intervals will not be plotted, giving only the forecast line.

- **showgap**: If `showgap=FALSE`, the gap between the historical observations and the forecasts is removed.

- **series**: Matches an unidentified forecast layer with a coloured object on the plot.

- **...**: Additional arguments for `forecast.ts`, other arguments are passed on to `layer`. These are often aesthetics, used to set an aesthetic to a fixed value, like `color = "red"` or `alpha = .5`. They may also be parameters to the paired geom/stat.

Format

An object of class `StatForecast` (inherits from `Stat`, `ggproto`, `gg`) of length 3.

An object of class `GeomForecast` (inherits from `Geom`, `ggproto`, `gg`) of length 7.

Details

Multivariate forecasting is supported by having each time series on a different group.

You can also pass `geom_forecast` a forecast object to add it to the plot.

The aesthetics required for the forecasting to work includes forecast observations on the y axis, and the time of the observations on the x axis. Refer to the examples below. To automatically set up aesthetics, use `autoplot`.

Value

A layer for a ggplot graph.

Author(s)

Mitchell O’Hara-Wild

See Also

forecast, ggproto

Examples

```r
## Not run:
library(ggplot2)
autoplot(USAccDeaths) + geom_forecast()

lungDeaths <- cbind(mdeaths, fdeaths)
autoplot(lungDeaths) + geom_forecast()

# Using fortify.ts
p <- ggplot(aes(x=x, y=y), data=USAccDeaths)
p <- p + geom_line()
p + geom_forecast()

# Without fortify.ts
data <- data.frame(USAccDeaths=as.numeric(USAccDeaths), time=as.numeric(time(USAccDeaths)))
p <- ggplot(aes(x=time, y=USAccDeaths), data=data)
p <- p + geom_line()
p + geom_forecast()
p + geom_forecast(h=60)
p <- ggplot(aes(x=time, y=USAccDeaths), data=data)
p + geom_forecast(level=c(70,98))
p + geom_forecast(level=c(70,98), colour="lightblue")

#Add forecasts to multivariate series with colour groups
lungDeaths <- cbind(mdeaths, fdeaths)
autoplot(lungDeaths) + geom_forecast(forecast(mdeaths), series="mdeaths")

## End(Not run)
```
subset.ts

Description

Various types of subsetting of a time series. Allows subsetting by index values (unlike window). Also allows extraction of the values of a specific season or subset of seasons 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,
  start = NULL,
  end = NULL,
  ...
)
```

```r
## S3 method for class 'msts'
subset(x, subset = NULL, start = NULL, end = NULL, ...)
```

Arguments

- `x` a univariate time series to be subsetted
- `subset` optional logical expression indicating elements to keep; missing values are taken as false. `subset` must be the same length as `x`.
- `month` Numeric or character vector of months to retain. Partial matching on month names used.
- `quarter` Numeric or character vector of quarters to retain.
- `season` Numeric vector of seasons to retain.
- `start` Index of start of contiguous subset.
- `end` Index of end of contiguous subset.
- `...` Other arguments, unused.

Details

If character values for months are used, either upper or lower case may be used, and partial unambiguous names are acceptable. Possible character values for quarters are "Q1", "Q2", "Q3", and "Q4".

Value

If `subset` is used, a numeric vector is returned with no ts attributes. If `start` and/or `end` are used, a ts object is returned consisting of `x[start:end]`, with the appropriate time series attributes retained. Otherwise, a ts object is returned with frequency equal to the length of `month`, `quarter` or `season`. 
**Author(s)**

Rob J Hyndman

**See Also**

`subset`, `window`

**Examples**

```r
plot(subset(gas, month="November"))
subset(woolyrnq, quarter=3)
subset(USAccDeaths, start=49)
```

---

**taylor**

**Half-hourly electricity demand**

**Description**

Half-hourly electricity demand in England and Wales from Monday 5 June 2000 to Sunday 27 August 2000. Discussed in Taylor (2003), and kindly provided by James W Taylor. Units: Megawatts

**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

```r
tbats(
  y,  
  use.box.cox = NULL,  
  use.trend = NULL,  
  use.damped.trend = NULL,  
  seasonal.periods = NULL,  
  use.arma.errors = TRUE,  
  use.parallel = length(y) > 1000,  
  num.cores = 2,  
  bc.lower = 0,  
  bc.upper = 1,  
  biasadj = FALSE,  
  model = NULL,  
  ...  
)
```

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 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.
biasadj Use adjusted back-transformed mean for Box-Cox transformations. If TRUE, point forecasts and fitted values are mean forecast. Otherwise, these points can be considered the median of the forecast densities.
model Output from a previous call to tbats. If model is passed, this same model is fitted to y without re-estimating any parameters.
... 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 with class c("tbats", "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 TBATS(omega, p,q, phi, <m1,k1>,...,<mJ,kJ>) 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 and k1,...,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)
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. The extracted components are Box-Cox transformed using the estimated transformation parameter.

Usage

tbats.components(x)

Arguments

x

A tbats object created by tbats.

Value

A multiple time series (mts) object. The first series is the observed time series. The second series is the trend component of the fitted model. Series three onwards are the seasonal components of the fitted model with one time series for each of the seasonal components. All components are transformed using estimated Box-Cox parameter.

Author(s)

Slava Razbash and Rob J Hyndman

References


See Also

tbats.

Examples

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

Returns forecasts and prediction intervals for a theta method forecast.

Usage

\[
\text{thetaf}(y, h = \text{ifelse}(\text{frequency}(y) > 1, 2 \times \text{frequency}(y), 10), \text{level} = c(80, 95), \text{fan} = \text{FALSE}, x = y)
\]

Arguments

- **y**: a numeric vector or time series of class ts
- **h**: Number of periods for forecasting
- **level**: Confidence levels for prediction intervals.
- **fan**: If TRUE, level is set to seq(51,99,by=3). This is suitable for fan plots.
- **x**: Deprecated. Included for backwards compatibility.

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).

The series is tested for seasonality using the test outlined in A&N. If deemed seasonal, the series is seasonally adjusted using a classical multiplicative decomposition before applying the theta method. The resulting forecasts are then reseasonalized.

Prediction intervals are computed using the underlying state space model.

More general theta methods are available in the forecTheta package.

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 `rwf`.

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

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

**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 supsmu for non-seasonal series and a robust STL decomposition for seasonal series. To estimate missing values and outlier replacements, linear interpolation is used on the (possibly seasonally adjusted) series.

**Usage**

`tsclean(x, replace.missing = TRUE, iterate = 2, lambda = NULL)`
Arguments

- **x**: time series
- **replace.missing**: If TRUE, it not only replaces outliers, but also interpolates missing values
- **iterate**: the number of iterations required
- **lambda**: Box-Cox transformation parameter. If lambda="auto", then a transformation is automatically selected using BoxCox.lambda. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.

Value

Time series

Author(s)

Rob J Hyndman

References

Hyndman (2021) "Detecting time series outliers" [https://robjhyndman.com/hyndsight/tsoutliers/](https://robjhyndman.com/hyndsight/tsoutliers/).

See Also

- `na.interp`, `tsoutliers`, `supsmu`

Examples

```r
cleangold <- tsclean(gold)
```

---

**tsCV**

*Time series cross-validation*

Description

**tsCV** computes the forecast errors obtained by applying **forecastfunction** to subsets of the time series **y** using a rolling forecast origin.

Usage

```r
tscv(y, forecastfunction, h = 1, window = NULL, xreg = NULL, initial = 0, ...)
```
Arguments

- **y**: Univariate time series
- **forecastfunction**: Function to return an object of class forecast. Its first argument must be a univariate time series, and it must have an argument h for the forecast horizon. If exogenous predictors are used, then it must also have xreg and newxreg arguments corresponding to the training and test periods.
- **h**: Forecast horizon
- **window**: Length of the rolling window, if NULL, a rolling window will not be used.
- **xreg**: Exogeneous predictor variables passed to the forecast function if required.
- **initial**: Initial period of the time series where no cross-validation is performed.
- **...**: Other arguments are passed to forecastfunction.

Details

Let y contain the time series \( y_1, \ldots, y_T \). Then forecastfunction is applied successively to the time series \( y_1, \ldots, y_t \), for \( t = 1, \ldots, T - h \), making predictions \( \hat{y}_{t+h|t} \). The errors are given by \( e_{t+h} = y_{t+h} - \hat{y}_{t+h|t} \). If \( h=1 \), these are returned as a vector, \( e_1, \ldots, e_T \). For \( h>1 \), they are returned as a matrix with the hth column containing errors for forecast horizon h. The first few errors may be missing as it may not be possible to apply forecastfunction to very short time series.

Value

Numerical time series object containing the forecast errors as a vector (if \( h=1 \)) and a matrix otherwise. The time index corresponds to the last period of the training data. The columns correspond to the forecast horizons.

Author(s)

Rob J Hyndman

See Also


Examples

```r
# Fit an AR(2) model to each rolling origin subset
far2 <- function(x, h){forecast(Arima(x, order=c(2,0,0)), h=h)}
e <- tsCV(lynx, far2, h=1)

# Fit the same model with a rolling window of length 30
e <- tsCV(lynx, far2, h=1, window=30)

# Example with exogenous predictors
far2_xreg <- function(x, h, xreg, newxreg) {
  forecast(Arima(x, order=c(2,0,0), xreg=xreg), newxreg=xreg)
```
y <- ts(rnorm(50))
xreg <- matrix(rnorm(100), ncol=2)
e <- tsCV(y, far2_xreg, h=3, xreg=xreg)

**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**

```r
tslm(formula, data, subset, lambda = NULL, biasadj = FALSE, ...)
```

**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.
- **subset**: an optional subset containing rows of data to keep. For best results, pass a logical vector of rows to keep. Also supports `subset()` functions.
- **lambda**: Box-Cox transformation parameter. If `lambda="auto"`, then a transformation is automatically selected using `BoxCox.lambda`. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.
- **biasadj**: Use adjusted back-transformed mean for Box-Cox transformations. If transformed data is used to produce forecasts and fitted values, a regular back transformation will result in median forecasts. If biasadj is TRUE, an adjustment will be made to produce mean forecasts and fitted values.
- **...**: Other arguments passed to `lm()`

**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".
tsoutliers

Author(s)
Mitchell O’Hara-Wild and 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 supsmu for non-seasonal series and a periodic stl decomposition with seasonal series to identify outliers and estimate their replacements.

**Usage**

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

**Arguments**

- `x` time series
- `iterate` the number of iterations required
- `lambda` Box-Cox transformation parameter. If `lambda="auto"`, then a transformation is automatically selected using BoxCox.lambda. The transformation is ignored if NULL. Otherwise, data transformed before model is estimated.

**Value**

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

**Author(s)**

Rob J Hyndman

**References**

Hyndman (2021) "Detecting time series outliers" [https://robjhyndman.com/hyndsight/tsoutliers/](https://robjhyndman.com/hyndsight/tsoutliers/).
See Also

na.interp, tsclean

Examples

data(gold)
tsoutliers(gold)

wineind  Australian total wine sales

Description


Usage

wineind

Format

Time series data

Source


Examples

tsdisplay(wineind)

woolyrnq  Quarterly production of woollen yarn in Australia

Description


Usage

woolyrnq
woolyrnq

Format

Time series data

Source


Examples

tsdisplay(woolyrnq)
Index

* datasets
  gas, 77
  gold, 86
  StatForecast, 126
  taylor, 130
  wineind, 140
  woolyrnq, 140
* hplot
  plot.Arima, 103
  plot.bats, 105
  plot.ets, 106
* htest
  dm.test, 38
* models
  CV, 36
* package
  forecast-package, 4
* stats
  forecast.lm, 57
  tsllm, 138
* ts
  accuracy.default, 5
  Acf, 7
  arima.errors, 13
  arimaorder, 14
  auto.arima, 15
  autoplot.mforecast, 24
  baggedModel, 26
  bats, 27
  bizdays, 29
  bld.mbb.bootstrap, 30
  BoxCox, 31
  BoxCox.lambda, 32
  croston, 34
  CVar, 37
  dm.test, 38
  dshw, 40
  easter, 42
  ets, 43
  findfrequency, 46
  fitted.ARFIMA, 47
  forecast.baggedModel, 48
  forecast.bats, 50
  forecast.ets, 51
  forecast.fracdiff, 53
  forecast.HoltWinters, 56
  forecast.modelAR, 61
  forecast.nnetar, 65
  forecast.stl, 68
  forecast.StructTS, 71
  forecast.ts, 73
  fourier, 75
  getResponse, 77
  ggseasonplot, 82
  ggtsdisplay, 84
  ma, 88
  meanf, 89
  modelAR, 91
  monthdays, 93
  msts, 95
  na.interp, 96
  ndiffs, 97
  nnetar, 98
  plot.forecast, 107
  residuals.forecast, 110
  rwf, 111
  seasadj, 114
  seasonal, 115
  seasonaldummy, 116
  ses, 117
  simulate.ets, 120
  sindexf, 123
  splinef, 124
  subset.ts, 128
  tbats, 131
  tbats.components, 133
thetaf, 134
tsclean, 135
tsCV, 136
tsoutliers, 139
_PACKAGE (forecast-package), 4
'[.msts' (msts), 95
accuracy.default, 5
Acf, 7, 23, 85
acf, 8, 9, 23, 85
aes, 127
aes_, 127
AIC, 36
ar, 14, 54, 55, 70, 104, 123
arima, 9, 14, 54, 55, 123
Arima, 11, 14, 17, 18, 45, 55, 75, 104, 114, 116, 123
arima, 10–12, 14, 17, 54, 55, 74, 120, 123, 126, 135
arima.errors, 13
arimaorder, 14
as.character.Arima(Arima), 11
as.character.bats(bats), 27
as.character.ets(ets), 43
as.character.tbats(tbats), 131
as.data.frame.forecast (forecast.ts), 73
as.data.frame.mforecast (forecast.mts), 63
as.ts.forecast (forecast.ts), 73
auto.arima, 10, 12, 14, 15, 54, 55, 69, 75, 98, 102, 116, 123
autolayer, 18
autolayer.forecast (plot.forecast), 107
autolayer.mforecast (autoplot.mforecast), 24
autolayer.msts (autolayer.mts), 19
autolayer.mts, 19
autolayer.ts (autolayer.mts), 19
autoplot(), 18
autoplot.acf, 21
autoplot.ar (plot.Arima), 103
autoplot.Arima (plot.Arima), 103
autoplot.bats (plot.bats), 105
autoplot.decomposed.ts, 23
autoplot.ets (plot.ets), 106
autoplot.forecast (plot.forecast), 107
autoplot.mforecast, 24
autoplot.mpacf (autoplot.acf), 21
autoplot.mstl (autoplot.decomposed.ts), 23
autoplot.msts (autolayer.mts), 19
autoplot.mts (autolayer.mts), 19
autoplot.seas (autoplot.decomposed.ts), 23
autoplot.splineforecast (plot.forecast), 107
autoplot.stl (autoplot.decomposed.ts), 23
autoplot.StructTS (autoplot.decomposed.ts), 23
autoplot.tbats (plot.bats), 105
autoplot.ts (autolayer.mts), 19
baggedETS, 31
baggedETS (baggedModel), 26
baggedModel, 26, 48, 49
bats, 27, 50, 51, 105
bgtest, 34
bizdays, 29, 93
bld.mbb.bootstrap, 26, 30
borders, 127
Box.test, 34
BoxCox, 31, 33
BoxCox.lambda, 32, 32
Ccf (Acf), 7
ccf, 8, 9
ch.test, 102
checkresiduals, 33, 111
coeff.ets (ets), 43
croston, 34, 75
CV, 36, 38, 137
CVar, 37, 137
decompose, 23, 24, 88, 115, 116, 124
dm.test, 38
dshw, 40
easter, 42
ets, 26, 41, 42, 43, 52, 53, 56, 69, 74, 106, 113, 118, 120, 123
findfrequency, 46
fitted.ar (fitted.ARFIMA), 47
fitted.ARFIMA, 47
fitted.Arima, 111
fitted.Arima (fitted.ARFIMA), 47
fitted.bats (fitted.ARFIMA), 47
fitted.ets (fitted.ARFIMA), 47
fitted.forecast_ARIMA (fitted.ARFIMA), 47
fitted.modelAR (fitted.ARFIMA), 47
fitted.nnetar (fitted.ARFIMA), 47
fitted.tbats (fitted.ARFIMA), 47
forecast, 48, 70, 71, 108, 109, 128
forecast-package, 4
forecast.ar (forecast.fracdiff), 53
forecast.Arima, 12, 48, 69, 71, 74, 75
forecast.Arima (forecast.fracdiff), 53
forecast.baggedModel, 26, 48
forecast.bats, 48, 50
forecast.default (forecast.ts), 73
forecast.ets, 48, 51, 51, 71, 75
forecast.forecast_ARIMA (forecast.fracdiff), 53
forecast.fracdiff, 10, 53, 55
forecast.HoltWinters, 56, 75
forecast.lm, 57, 60, 61, 139
forecast.mlm, 59, 64, 65
forecast.modelAR, 61
forecast.mts, 63
forecast.nnetar, 48, 65
forecast.stl, 68
forecast.stlm (forecast.stl), 68
forecast.StructTS, 71, 75
forecast.tbats, 48
forecast.tbats (forecast.bats), 50
forecast.ts, 73, 74, 127
forecTheta, 134
fortify, 20, 127
fortify(), 18
fortify.ts (autolayer.mts), 19
fourier, 75, 117
fourierf (fourier), 75
fracdiff, 10, 14, 54
gas, 77
gem_forecast, 20
gem_forecast (StatForecast), 126
gem_histogram, 79
GeomForecast (StatForecast), 126
getResponse, 77
ggAcf (autoplot.acf), 21
ggCcf (autoplot.acf), 21
gghistogram, 78
gglagchull (gglagplot), 79
ggplot, 127
ggmplot(), 18
ggproto, 128
ggsseasonplot, 82
ggsubseriesplot (ggmonthplot), 81
ggtaperedacf (autoplot.acf), 21
ggtaperedpacf (autoplot.acf), 21
ggtsdisplay, 34, 84
gold, 86
hegy.test, 102
hist, 79
holt, 53, 75, 126
holt (ses), 117
HoltWinters, 42, 45, 56, 57, 120
hw, 53, 75
hw (ses), 117
InvBoxCox (BoxCox), 31
is.acf, 86
is.Arima (is.acf), 86
is.baggedModel (is.acf), 86
is.bats (is.acf), 86
is.constant, 87
is.ets (is.acf), 86
is.forecast, 87
is.forecast, 87
is.mforecast (is.forecast), 87
is.modelAR (is.acf), 86
is.nnetar (is.acf), 86
is.nnetarmodels (is.acf), 86
is.splineforecast (is.forecast), 87
is.stlm (is.acf), 86
isBizday, 29
lag.plot, 81
layer, 127
lm, 36, 58–61, 138, 139
ma, 88
meanf, 75, 89, 135
mforecast (forecast.mts), 63
modelAR, 62, 91
monthdays, 30, 93
monthplot, 82, 84
mstl, 94
msts, 40, 95
INDEX

na.contiguous, 8, 22, 85
na.interp, 8, 22, 85, 96, 136, 140
na.pass, 8, 22, 85
naive (rwf), 111
nclass. FD, 79
diffs, 16, 97, 98, 102
nnet, 99
nnetar, 37, 63, 66, 67, 98, 123
nsdiffs, 17, 100, 103
ocsb.test, 102, 102

Pacf (Acf), 7
pacf, 8, 9
par, 104, 105
plot, 83
plot.acf, 23
plot.ar (plot.Arima), 103
plot.Arima, 103
plot.bats, 105
plot.default, 108
plot.ets, 106
plot.forecast, 25, 107
plot.mforecast (autoplot.mforecast), 24
plot.splineforecast (plot.forecast), 107
plot.stl, 24
plot.tbats (plot.bats), 105
plot.ts, 20, 25, 85, 109
predict.ar, 55
predict.Arima, 55
predict.HoltWinters, 56, 57
predict.lm, 58
print.ARIMA (Arima), 11
print.baggedModel (baggedModel), 26
print.bats (bats), 27
print.CVar (CVar), 37
print.ets (ets), 43
print.forecast (forecast.ts), 73
print.mforecast (forecast.mts), 63
print.modelAR (modelAR), 91
print.msts (msts), 95
print.naive (rwf), 111
print.nnetar (nnetar), 98
print.nnetarmodels (nnetar), 98
print.OCSBtest (ocsb.test), 102
print.tbats (tbats), 131

remainder (seasonal), 115
residuals.ar (residuals.forecast), 110
residuals.ARFIMA (residuals.forecast), 110
residuals.Arima, 13, 14, 48, 137
residuals.Arima (residuals.forecast), 110
residuals.bats, 48
residuals.bats (residuals.forecast), 110
residuals.ets, 48
residuals.ets (residuals.forecast), 110
residuals.forecast, 110
residuals.forecast_ARIMA
(residuals.forecast), 110
residuals.nnetar, 48
residuals.nnetar (residuals.forecast), 110
residuals.stlm (residuals.forecast), 110
residuals.tbats, 48
residuals.tbats (residuals.forecast), 110
residuals.tslm (residuals.forecast), 110
rnorm, 123
rwf, 45, 75, 90, 111, 120, 135

seas, 23, 24
seasadj, 114, 116
seasonal, 115
seasonaldummy, 76, 116
seasonaldummyf (seasonaldummy), 116
seasonplot (ggseasonplot), 82
ses, 35, 53, 75, 117, 135
set.seed, 122
simulate.ar (simulate.ets), 120
simulate.Arima (simulate.ets), 120
simulate.ets, 120
simulate.fracdiff (simulate.ets), 120
simulate.lagwalk (simulate.ets), 120
simulate.modelAR (simulate.ets), 120
simulate.nnetar, 62, 66
simulate.nnetar (simulate.ets), 120
simulate.tbats (simulate.ets), 120
sindexf, 123
smooth.spline, 125, 126
snaive (rwf), 111
spec.ar, 85
splinef, 75, 124
StatForecast, 126
stlf, 74, 113
stlf (forecast.stl), 68
stlm(forecast.stl), 68
StructTS, 24, 72, 73
subset, 130, 138
subset.msts(subset.ts), 128
subset.ts, 128
summary.Arima(Arima), 11
summary.ets(ets), 43
summary.forecast(forecast.ts), 73
summary.mforecast(forecast.mts), 63
supsmu, 94, 136
taperedacf, 23
taperedacf(Acf), 7
taperedpacf(Acf), 7
taylor, 130
tbats, 51, 105, 115, 116, 131, 133
tbats.components, 132, 133
thetaf, 70, 75, 134
trendcycle(seasonal), 115
tsclean, 135, 140
tscv, 38, 136
tsdia.ets(ets), 43
tsddisplay, 9
tsddisplay(ggtsdisplay), 84
tslm, 36, 58–61, 64, 75, 116, 138
tsinkliers, 96, 136, 139
window, 129, 130
window.msts(msts), 95
wineind, 140
woolyrnq, 140