Package ‘forecast’

January 11, 2026

Version 9.0.0

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 (>= 4.1.0)

Imports colorspace, fracdiff, generics (>= 0.1.2), ggplot2 (>= 3.4.0), graphics, lmtest, magrittr, nnet, parallel, Rcpp (>= 0.11.0), stats, timeDate, tseries, urca, withr, zoo

Suggests forecTheta, knitr, methods, rmarkdown, rticles, scales, seasonal, testthat (>= 3.3.0), 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

RoxygenNote 7.3.3

Encoding UTF-8

Config/testthat/edition 3

NeedsCompilation yes

Author Rob Hyndman [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-2140-5352>),
George Athanasopoulos [aut] (ORCID: <https://orcid.org/0000-0002-5389-2802>),
Christoph Bergmeir [aut] (ORCID: <https://orcid.org/0000-0002-3665-9021>),
Gabriel Caceres [aut] (ORCID: <https://orcid.org/0000-0002-2947-2023>),

Leanne Chhay [aut],
Kirill Kuroptev [aut],
Maximilian Mücke [aut] (ORCID: <https://orcid.org/0009-0000-9432-9795>),
Mitchell O'Hara-Wild [aut] (ORCID:
<https://orcid.org/0000-0001-6729-7695>),
Fotios Petropoulos [aut] (ORCID:
<https://orcid.org/0000-0003-3039-4955>),
Slava Razbash [aut],
Earo Wang [aut] (ORCID: <https://orcid.org/0000-0001-6448-5260>),
Farah Yasmeen [aut] (ORCID: <https://orcid.org/0000-0002-1479-5401>),
Federico Garza [ctb],
Daniele Girolimetto [ctb],
Ross Ihaka [ctb, cph],
R Core Team [ctb, cph],
Daniel Reid [ctb],
David Shaub [ctb],
Yuan Tang [ctb] (ORCID: <https://orcid.org/0000-0001-5243-233X>),
Xiaoqian Wang [ctb],
Zhenyu Zhou [ctb]

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

Repository CRAN

Date/Publication 2026-01-11 08:40:02 UTC

Contents

`accuracy.forecast` *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"] - fitted(f)`. All measures are defined and discussed in Hyndman and Koehler (2006).

Usage

```
## S3 method for class 'forecast'
accuracy(object, x, test = NULL, d = NULL, D = NULL, ...)

## S3 method for class 'mforecast'
accuracy(object, x, test = NULL, d = NULL, D = NULL, ...)

## S3 method for class 'fc_model'
```

```
accuracy(object, x, test = NULL, d = NULL, D = NULL, ...)

## S3 method for class 'Arima'
accuracy(object, x, test = NULL, d = NULL, D = NULL, ...)

## S3 method for class 'lm'
accuracy(object, x, test = NULL, d = NULL, D = NULL, ...)

## S3 method for class 'ts'
accuracy(object, x, test = NULL, d = NULL, D = NULL, ...)

## S3 method for class 'numeric'
accuracy(object, x, test = NULL, d = NULL, D = 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 object, or a time series overlapping with the times of `f`.
test	Indicator of which elements of `x` and `f` to test. If `test` is `NULL`, all elements are used. Otherwise test is a numeric vector containing the indices of the elements to use in the test.
d	An integer indicating the number of lag-1 differences to be used for the denominator in MASE calculation. Default value is 1 for non-seasonal series and 0 for seasonal series.
D	An integer indicating the number of seasonal differences to be used for the denominator in MASE calculation. Default value is 0 for non-seasonal series and 1 for seasonal series.
...	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

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

Hyndman, R.J. and Athanasopoulos, G. (2018) "Forecasting: principles and practice", 2nd ed., OTexts, Melbourne, Australia. Section 3.4 "Evaluating forecast accuracy". https://otexts.com/fpp2/accuracy.html.

Examples

```
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

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

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

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

taperedacf(
  x,
  lag.max = NULL,
  type = c("correlation", "partial"),
  plot = TRUE,
  calc.ci = TRUE,
  level = 95,
  nsim = 100,
  ...
)

taperedpacf(x, ...)
```

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 $10*log10(N/m)$ where

	N is the number of observations and m the number of series. Will be automatically limited to one less than the number of observations in the series.
type	Character string giving the type of acf to be computed. Allowed values are "correlation" (the default), "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 stats::na.contiguous(). Useful alternatives are stats::na.pass() and na.interp().
demean	Should covariances be about the sample means?
...	Additional arguments passed to the plotting function.
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 stats::acf(), stats::pacf() and stats::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 stats::acf() from the stats package. The taperedacf and taperedpacf functions return objects of class "mpacf".

Author(s)

Rob J Hyndman

References

Hyndman, R.J. (2015). Discussion of “High-dimensional autocovariance matrices and optimal linear prediction”. *Electronic Journal of Statistics*, 9, 792-796.

McMurry, T. L., & Politis, D. N. (2010). Banded and tapered estimates for autocovariance matrices and the linear process bootstrap. *Journal of Time Series Analysis*, 31(6), 471-482.

See Also

stats::acf(), stats::pacf(), stats::ccf(), tsdisplay()

Examples

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

## End(Not run)
```

---

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

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

Arguments

y	a numeric vector or univariate time series of class `ts`
drange	Allowable values of d to be considered. Default of `c(0, 0.5)` ensures a stationary model is returned.
estim	If `estim = "ls"`, then the ARMA parameters are calculated using the Haslett-Raftery algorithm. If `estim = "mle"`, then the ARMA parameters are calculated using full MLE via the `stats::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.

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.

x Deprecated. Included for backwards compatibility.

... Other arguments passed to `auto.arima()` when selecting p and q.

Details

This function combines `fracdiff::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::fracdiff()`. If `estim = "mle"`, the ARMA coefficients are refined using `stats::arima()`.

Value

A list object of S3 class `fracdiff`, which is described in the `fracdiff::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

J. Haslett and A. E. Raftery (1989) Space-time Modelling with Long-memory Dependence: Assessing Ireland's Wind Power Resource (with discussion); *Applied Statistics* **38**, 1-50.

Hyndman, R.J. and Khandakar, Y. (2008) "Automatic time series forecasting: The forecast package for R", *Journal of Statistical Software*, **26**(3).

See Also

`fracdiff::fracdiff()`, `auto.arima()`, `forecast.fracdiff()`.

Examples

```
library(fracdiff)
x <- fracdiff.sim(100, ma = -0.4, d = 0.3)$series
fit <- arfima(x)
tsdisplay(residuals(fit))
```

Arima *Fit ARIMA model to univariate time series*

Description

Largely a wrapper for the `stats::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 = NULL,
  lambda = model$lambda,
  biasadj = attr(lambda, "biasadj"),
  method = c("CSS-ML", "ML", "CSS"),
  model = NULL,
  x = y,
  ...
)
```

Arguments

`y`	a numeric vector or univariate time series of class `ts`
`order`	a specification of the non-seasonal part of the ARIMA model: the three integer components (p, d, q) are the AR order, the degree of differencing, and the MA order.
`seasonal`	a specification of the seasonal part of the ARIMA model, plus the period (which defaults to `frequency(x)`). This may be a `list` with components `order` and `period`, or just a numeric vector of length 3 which specifies the seasonal `order`. In the latter case the default period is used.
`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. Can be abbreviated.

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 stats::arima().

Details

The fitted model is a regression with ARIMA(p,d,q) errors

$$y_t = c + \beta' x_t + z_t$$

where x_t is a vector of regressors at time t and z_t is an ARMA(p,d,q) error process. If there are no regressors, and $d = 0$, then c is an estimate of the mean of y_t. For more information, see Hyndman & Athanasopoulos (2018). For details of the estimation algorithm, see the stats::arima() function in the stats package.

Value

See the stats::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

References

Hyndman, R.J. and Athanasopoulos, G. (2018) "Forecasting: principles and practice", 2nd ed., OTexts, Melbourne, Australia. https://OTexts.com/fpp2/.

See Also

auto.arima(), 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).

Value

A `ts` object

Author(s)

Rob J Hyndman

See Also

`residuals.Arima()`.

---

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`, `ar` or `fracdiff`. Usually the result of a call to `stats::arima()`, `Arima()`, `auto.arima()`, `stats::ar()`, `arfima()` or `fracdiff::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

`stats::ar()`, auto.arima, `Arima()`, `stats::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 numeric vector or univariate time series of class `ts`
`d`	Order of first-differencing. If missing, will choose a value based on `test`.
`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	Additional arguments to be passed to the seasonal unit root test. See nsdiffs() for details.
allowdrift	If TRUE, models with drift terms are considered.
allowmean	If TRUE, models with a non-zero mean are considered.
lambda	Box-Cox transformation parameter. 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 via parallel::mclapply(). This can give a significant speedup on multicore machines. On Windows, this option always fails because forking is not supported.
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 stats::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

Hyndman, RJ and Khandakar, Y (2008) "Automatic time series forecasting: The forecast package for R", *Journal of Statistical Software*, **26**(3).

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

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

Other plotting automation topics: `automatic_plotting`, `autoplot()`, `fortify()`

autolayer.mts — *Automatically create a ggplot for time series objects*

Description

`autoplot` takes an object of type `ts` or `mts` and creates a ggplot object suitable for usage with `stat_forecast`.

Usage

```
## 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 = deparse1(substitute(object)),
  main = NULL,
  ...
)

## S3 method for class 'mts'
autoplot(
  object,
  colour = TRUE,
  facets = FALSE,
  xlab = "Time",
  ylab = deparse1(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.
facets	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 ggplot2::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

stats::plot.ts(), ggplot2::fortify()

Examples

```
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

```
## 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,
```

```
  lag.max = NULL,
  type = c("correlation", "partial"),
  plot = TRUE,
  calc.ci = TRUE,
  level = 95,
  nsim = 100,
  ...
)

ggtaperedpacf(x, ...)
```

Arguments

`object`	Object of class `acf`.
`ci`	coverage probability for confidence interval. Plotting of the confidence interval is suppressed if ci is zero or negative.
`...`	Other plotting parameters to affect the plot.
`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.
`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 `stats::na.contiguous()`. Useful alternatives are `stats::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

If `autoplot` is given an `acf` or `mpacf` object, then an appropriate ggplot object will be created.

ggtaperedpacf

Value

A ggplot object.

Author(s)

Mitchell O'Hara-Wild

See Also

stats::plot.acf() Acf(), [stats::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 `stats::stl()`, class `seas` (created with `seasonal::seas()`), or class `decomposed.ts` (created with `stats::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

`seasonal::seas()`, `stats::stl()`, `stats::decompose()`, `stats::StructTS()`, `stats::plot.stl()`.

Examples

```
library(ggplot2)
co2 |>
  decompose() |>
  autoplot()
nottem |>
  stl(s.window = "periodic") |>
  autoplot()
## Not run:
library(seasonal)
seas(USAccDeaths) |> autoplot()

## End(Not run)
```

`autoplot.mforecast` *Multivariate forecast plot*

Description

Plots historical data with multivariate forecasts and prediction intervals.

Usage

```
## S3 method for class 'mforecast'
autoplot(object, PI = TRUE, facets = TRUE, colour = FALSE, ...)

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

## S3 method for class 'mforecast'
plot(x, main = paste("Forecasts from", unique(x$method)), xlab = "time", ...)
```

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

Hyndman and Athanasopoulos (2018) *Forecasting: principles and practice*, 2nd edition, OTexts: Melbourne, Australia. https://otexts.com/fpp2/

See Also

plot.forecast(), stats::plot.ts()

Examples

```
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

```
baggedModel(y, bootstrapped_series = bld.mbb.bootstrap(y, 100), fn = ets, ...)

baggedETS(y, bootstrapped_series = bld.mbb.bootstrap(y, 100), ...)
```

Arguments

y	A numeric vector or univariate 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 forecast.baggedModel() can then be used to calculate forecasts.

baggedETS is a wrapper for baggedModel, 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

Bergmeir, C., R. J. Hyndman, and J. M. Benitez (2016). Bagging Exponential Smoothing Methods using STL Decomposition and Box-Cox Transformation. International Journal of Forecasting 32, 303-312.

Examples

```
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

```
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 `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 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.
- `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

References

De Livera, A.M., Hyndman, R.J., & Snyder, R. D. (2011), Forecasting time series with complex seasonal patterns using exponential smoothing, *Journal of the American Statistical Association*, **106**(496), 1513-1527.

Examples

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

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

## End(Not run)
```

---

bizdays *Number of trading days in each season*

Description

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

Usage

```
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 `timeDate::isBizday()`.

Value

Time series

Author(s)

Earo Wang

See Also

`monthdays()`

Examples

```
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

```
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

Bergmeir, C., R. J. Hyndman, and J. M. Benitez (2016). Bagging Exponential Smoothing Methods using STL Decomposition and Box-Cox Transformation. International Journal of Forecasting 32, 303-312.

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. InvBox-Cox() 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

$$f_\lambda(x) = (sign(x)|x|^\lambda - 1)/\lambda$$

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

Box, G. E. P. and Cox, D. R. (1964) An analysis of transformations. *JRSS B* **26** 211–246. Bickel, P. J. and Doksum K. A. (1981) An Analysis of Transformations Revisited. *JASA* **76** 296-311.

See Also

`BoxCox.lambda()`

Examples

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

BoxCox.lambda *Automatic selection of Box Cox transformation parameter*

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.

Value

a number indicating the Box-Cox transformation parameter.

Author(s)

Leanne Chhay and Rob J Hyndman

References

Box, G. E. P. and Cox, D. R. (1964) An analysis of transformations. *JRSS B* **26** 211–246.

Guerrero, V.M. (1993) Time-series analysis supported by power transformations. *Journal of Forecasting*, **12**, 37–48.

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))
```

---

checkresiduals — *Check that residuals from a time series model look like white noise*

Description

If `plot = TRUE`, produces a time plot of the residuals, the corresponding ACF, and a histogram. If `test` is not `FALSE`, the output from either a Ljung-Box test or Breusch-Godfrey test is printed.

Usage

```
checkresiduals(object, lag, test, plot = TRUE, ...)
```

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 `min(10, n/5)` for non-seasonal data, and `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.
`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.
`...`	Other arguments are passed to `ggtsdisplay()`.

Value

None

Author(s)

Rob J Hyndman

See Also

`ggtsdisplay()`, `stats::Box.test()`, [lmtest::bgtest()

Examples

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

`croston_model` *Croston forecast model*

Description

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. Returns a model object that can be used to generate forecasts using Croston's method for intermittent demand time series. It isn't a true statistical model in that it doesn't describe a data generating process that would lead to the forecasts produced using Croston's method.

Usage

```
croston_model(y, alpha = 0.1, type = c("croston", "sba", "sbj"))
```

Arguments

`y`	a numeric vector or univariate time series of class `ts`
`alpha`	Value of alpha. Default value is 0.1.
`type`	Which variant of Croston's method to use. Defaults to `"croston"` for Croston's method, but can also be set to `"sba"` for the Syntetos-Boylan approximation, and `"sbj"` for the Shale-Boylan-Johnston method.

Details

Note that prediction intervals are not computed as Croston's method has no underlying stochastic model.

There are two variant methods available which apply multiplicative correction factors to the forecasts that result from the original Croston's method. For the Syntetos-Boylan approximation (`type = "sba"`), this factor is $1 - \alpha/2$, and for the Shale-Boylan-Johnston method (`type = "sbj"`), this factor is $1 - \alpha/(2 - \alpha)$, where α is the smoothing parameter for the interval SES application.

Value

An object of class `croston_model`

Author(s)

Rob J Hyndman

References

Croston, J. (1972) "Forecasting and stock control for intermittent demands", *Operational Research Quarterly*, **23**(3), 289-303.

Shale, E.A., Boylan, J.E., & Johnston, F.R. (2006). Forecasting for intermittent demand: the estimation of an unbiased average. *Journal of the Operational Research Society*, **57**(5), 588-592.

Shenstone, L., and Hyndman, R.J. (2005) "Stochastic models underlying Croston's method for intermittent demand forecasting". *Journal of Forecasting*, **24**, 389-402.

Syntetos A.A., Boylan J.E. (2001). On the bias of intermittent demand estimates. *International Journal of Production Economics*, **71**, 457–466.

Examples

```
y <- rpois(20, lambda = 0.3)
fit <- croston_model(y)
forecast(fit) |> autoplot()
```

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 `stats::lm()` or `tslm()`.

Value

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

Author(s)

Rob J Hyndman

See Also

`stats::AIC()`

Examples

```
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 — *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

```
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

References

Bergmeir, C., Hyndman, R.J., Koo, B. (2018) A note on the validity of cross-validation for evaluating time series prediction. *Computational Statistics & Data Analysis*, **120**, 70-83. https://robjhyndman.com/publications/cv-time-series/.

See Also

CV(), tsCV().

Examples

```
modelcv <- CVar(lynx, k = 5, lambda = 0.15)
print(modelcv)
print(modelcv$fold1)

library(ggplot2)
autoplot(lynx, series = "Data") +
  autolayer(modelcv$testfit, series = "Fits") +
  autolayer(modelcv$residuals, series = "Residuals")
ggAcf(modelcv$residuals)
```

---

dm.test *Diebold-Mariano test for predictive accuracy*

Description

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.
h	The forecast horizon used in calculating e1 and e2.
power	The power used in the loss function. Usually 1 or 2.
varestimator	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 varestimator = "acf", or the estimator based on Bartlett weights varestimator = "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

Diebold, F.X. and Mariano, R.S. (1995) Comparing predictive accuracy. *Journal of Business and Economic Statistics*, **13**, 253-263.

Harvey, D., Leybourne, S., & Newbold, P. (1997). Testing the equality of prediction mean squared errors. *International Journal of forecasting*, **13**(2), 281-291.

Examples

```
# 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

Returns forecasts using Taylor's (2003) Double-Seasonal Holt-Winters method.

Usage

```
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`.

Author(s)

Rob J Hyndman

References

Taylor, J.W. (2003) Short-term electricity demand forecasting using double seasonal exponential smoothing. *Journal of the Operational Research Society*, **54**, 799-805.

Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) *Forecasting with exponential smoothing: the state space approach*, Springer-Verlag. https://robjhyndman.com/expsmooth/.

See Also

`stats::HoltWinters()`, `ets()`.

Examples

```
## 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, 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

```
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

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

ets *Exponential smoothing state space model*

Description

Returns ets model applied to y.

Usage

```
ets(
  y,
  model = "ZZZ",
  damped = NULL,
  alpha = NULL,
  beta = NULL,
  gamma = NULL,
  phi = NULL,
  additive.only = FALSE,
  lambda = NULL,
  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,
  ...
)
```

Arguments

`y` a numeric vector or univariate 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. When lambda is specified, additive.only is set to TRUE.

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.

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.

...
: Other arguments are ignored.

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

Hyndman, R.J., Koehler, A.B., Snyder, R.D., and Grose, S. (2002) "A state space framework for automatic forecasting using exponential smoothing methods", *International J. Forecasting*, **18**(3), 439–454.

Hyndman, R.J., Akram, Md., and Archibald, B. (2008) "The admissible parameter space for exponential smoothing models". *Annals of Statistical Mathematics*, **60**(2), 407–426.

Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) *Forecasting with exponential smoothing: the state space approach*, Springer-Verlag. https://robjhyndman.com/expsmooth/.

See Also

stats::HoltWinters(), rwf(), Arima().

Examples

```
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

```
## 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, bats, tbats, 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

See Also

forecast.Arima(), forecast.bats(), forecast.tbats(), forecast.ets(), forecast.nnetar(), residuals.Arima(), residuals.bats() residuals.tbats(), residuals.ets(), residuals.nnetar().

Examples

```
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)
```

---

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 = if (frequency(object$y) > 1) 2 * frequency(object$y) else 10,
  ...
)
```

Arguments

`object`	An object of class `baggedModel` resulting from a call to `baggedModel()`.
`h`	Number of periods for forecasting. Default value is twice the largest seasonal period (for seasonal data) or ten (for non-seasonal data).
`...`	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.

Value

An object of class `forecast`.

forecast class

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.

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)

The function `summary` can be used to obtain and print a summary of the results, while the functions `plot` and `autoplot` produce plots of the forecasts and prediction intervals. The generic accessors functions `fitted.values` and `residuals` extract various useful features from the underlying model.

Author(s)

Christoph Bergmeir, Fotios Petropoulos

References

Bergmeir, C., R. J. Hyndman, and J. M. Benitez (2016). Bagging Exponential Smoothing Methods using STL Decomposition and Box-Cox Transformation. International Journal of Forecasting 32, 303-312.

See Also

baggedModel().

Examples

```
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)
```

---

forecast.bats *Forecasting using BATS and TBATS models*

Description

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

Usage

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

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

```
  innov = NULL,
  npaths = 5000,
  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 levels 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 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 are ignored.
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). Ignored if innov is not NULL.
innov	Optional matrix of future innovations to be used in simulations. Ignored if simulate = FALSE. If provided, this overrides the bootstrap argument. The matrix should have h rows and npaths columns.
npaths	Number of sample paths used in computing simulated prediction intervals.

Value

An object of class forecast.

forecast class

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.

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)

The function `summary` can be used to obtain and print a summary of the results, while the functions `plot` and `autoplot` produce plots of the forecasts and prediction intervals. The generic accessors functions `fitted.values` and `residuals` extract various useful features from the underlying model.

Author(s)

Slava Razbash and Rob J Hyndman

References

De Livera, A.M., Hyndman, R.J., & Snyder, R. D. (2011), Forecasting time series with complex seasonal patterns using exponential smoothing, *Journal of the American Statistical Association*, **106**(496), 1513-1527.

See Also

`bats()`, `tbats()`, `forecast.ets()`.

Examples

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

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

## End(Not run)
```

---

forecast.croston_model *Forecasts for intermittent demand using Croston's method*

Description

Returns forecasts and other information for Croston's forecasts applied to y.

Usage

```
## S3 method for class 'croston_model'
forecast(object, h = 10, ...)

croston(y, h = 10, alpha = 0.1, type = c("croston", "sba", "sbj"), x = y)
```

Arguments

object	An object of class croston_model as returned by croston_model().
h	Number of periods for forecasting. Default value is twice the largest seasonal period (for seasonal data) or ten (for non-seasonal data).
...	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.
y	a numeric vector or univariate time series of class ts
alpha	Value of alpha. Default value is 0.1.
type	Which variant of Croston's method to use. Defaults to "croston" for Croston's method, but can also be set to "sba" for the Syntetos-Boylan approximation, and "sbj" for the Shale-Boylan-Johnston method.
x	Deprecated. Included for backwards compatibility.

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.

forecast class

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.

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)

The function `summary` can be used to obtain and print a summary of the results, while the functions `plot` and `autoplot` produce plots of the forecasts and prediction intervals. The generic accessors functions `fitted.values` and `residuals` extract various useful features from the underlying model.

Author(s)

Rob J Hyndman

References

Croston, J. (1972) "Forecasting and stock control for intermittent demands", *Operational Research Quarterly*, **23**(3), 289-303.

Shale, E.A., Boylan, J.E., & Johnston, F.R. (2006). Forecasting for intermittent demand: the estimation of an unbiased average. *Journal of the Operational Research Society*, **57**(5), 588-592.

Shenstone, L., and Hyndman, R.J. (2005) "Stochastic models underlying Croston's method for intermittent demand forecasting". *Journal of Forecasting*, **24**, 389-402.

Syntetos A.A., Boylan J.E. (2001). On the bias of intermittent demand estimates. *International Journal of Production Economics*, **71**, 457–466.

See Also

`ses()`.

Examples

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

`forecast.ets` *Forecasting using ETS models*

Description

Returns forecasts and other information for univariate ETS models.

Usage

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

```
    bootstrap = FALSE,
    innov = NULL,
    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. Default value is twice the largest seasonal period (for seasonal data) or ten (for non-seasonal data).
level	Confidence levels 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). Ignored if innov is not NULL.
innov	Optional matrix of future innovations to be used in simulations. Ignored if simulate = FALSE. If provided, this overrides the bootstrap argument. The matrix should have h rows and npaths columns.
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 are ignored.

Value

An object of class forecast.

forecast class

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.

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)

The function `summary` can be used to obtain and print a summary of the results, while the functions `plot` and `autoplot` produce plots of the forecasts and prediction intervals. The generic accessors functions `fitted.values` and `residuals` extract various useful features from the underlying model.

Author(s)

Rob J Hyndman

See Also

`ets()`, `ses()`, `holt()`, `hw()`.

Examples

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

`forecast.fracdiff` *Forecasting using ARIMA or ARFIMA models*

Description

Returns forecasts and other information for univariate ARIMA models.

Usage

```
## S3 method for class 'fracdiff'
forecast(
  object,
  h = 10,
  level = c(80, 95),
  fan = FALSE,
  simulate = FALSE,
  bootstrap = FALSE,
  innov = NULL,
```

```
  npaths = 5000,
  lambda = object$lambda,
  biasadj = attr(lambda, "biasadj"),
  ...
)

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

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

Arguments

`object`	An object of class `Arima`, `ar` or `fracdiff`. Usually the result of a call to `stats::arima()`, `auto.arima()`, `stats::ar()`, `arfima()` or `fracdiff::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 levels 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). Ignored if `innov` is not `NULL`.
`innov`	Optional matrix of future innovations to be used in simulations. Ignored if `simulate = FALSE`. If provided, this overrides the `bootstrap` argument. The matrix should have `h` rows and `npaths` columns.
`npaths`	Number of sample paths used in computing simulated prediction intervals.
`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 are ignored.
`xreg`	Future values of any regression variables. A numerical vector or matrix of external regressors; it should not be a data frame.

Details

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

Value

An object of class `forecast`.

forecast class

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.

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)

The function `summary` can be used to obtain and print a summary of the results, while the functions `plot` and `autoplot` produce plots of the forecasts and prediction intervals. The generic accessors functions `fitted.values` and `residuals` extract various useful features from the underlying model.

Author(s)

Rob J Hyndman

References

Peiris, M. & Perera, B. (1988), On prediction with fractionally differenced ARIMA models, *Journal of Time Series Analysis*, **9**(3), 215-220.

See Also

`stats::predict.Arima()`, `stats::predict.ar()`, `auto.arima()`, `Arima()`, `stats::arima()`, `stats::ar()`, `arfima()`.

Examples

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

library(fracdiff)
x <- fracdiff.sim(100, ma = -0.4, d = 0.3)$series
fit <- arfima(x)
plot(forecast(fit, h = 30))
```

`forecast.HoltWinters` *Forecasting using Holt-Winters objects*

Description

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

Usage

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

Arguments

`object`	An object of class `HoltWinters`. Usually the result of a call to `stats::HoltWinters()`.
`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 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.
`...`	Other arguments are ignored.

Details

This function calls `stats::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 stats::HoltWinters.

Value

An object of class `forecast`.

forecast class

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.

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)

The function `summary` can be used to obtain and print a summary of the results, while the functions `plot` and `autoplot` produce plots of the forecasts and prediction intervals. The generic accessors functions `fitted.values` and `residuals` extract various useful features from the underlying model.

Author(s)

Rob J Hyndman

See Also

stats::predict.HoltWinters, `stats::HoltWinters()`.

Examples

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

`forecast.lm` *Forecast a linear model with possible time series components*

Description

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

Usage

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

Arguments

`object`	Object of class "lm", usually the result of a call to `stats::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 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.
`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 `stats::predict.lm()`.

Details

`forecast.lm` is largely a wrapper for `stats::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`.

forecast class

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.

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)

The function `summary` can be used to obtain and print a summary of the results, while the functions `plot` and `autoplot` produce plots of the forecasts and prediction intervals. The generic accessors functions `fitted.values` and `residuals` extract various useful features from the underlying model.

Author(s)

Rob J Hyndman

See Also

tslm(), stats::lm().

Examples

```
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))
```

---

forecast.mean_model   *Mean Forecast*

Description

Returns forecasts and prediction intervals for a Gaussian iid model. meanf() is a convenience function that combines mean_model() and forecast().

Usage

```
## S3 method for class 'mean_model'
forecast(
  object,
  h = 10,
  level = c(80, 95),
  fan = FALSE,
  lambda = object$lambda,
  biasadj = attr(object$lambda, "biasadj"),
  bootstrap = FALSE,
  npaths = 5000,
  ...
)

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

Arguments

`object`	An object of class `mean_model` as returned by `mean_model()`.
`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 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`, then prediction intervals are produced by simulation using resampled errors (rather than normally distributed errors). Ignored if `innov` is not `NULL`.
`npaths`	Number of sample paths used in computing simulated prediction intervals.
`...`	Additional arguments not used.
`y`	a numeric vector or univariate time series of class `ts`
`x`	Deprecated. Included for backwards compatibility.

Details

The model assumes that the data are independent and identically distributed

$$Y_t \sim N(\mu, \sigma^2)$$

Forecasts are given by

$$Y_{n+h|n} = \mu$$

where μ is estimated by the sample mean.

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 `stats::fitted()` and `stats::residuals()` extract useful features of the object returned by `mean_model()`.

forecast class

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.

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)

The function `summary` can be used to obtain and print a summary of the results, while the functions `plot` and `autoplot` produce plots of the forecasts and prediction intervals. The generic accessors functions `fitted.values` and `residuals` extract various useful features from the underlying model.

Author(s)

Rob J Hyndman

See Also

`mean_model()`

Examples

```
fit_nile <- mean_model(Nile)
fit_nile |> forecast(h = 10) |> autoplot()
nile.fcast <- meanf(Nile, h = 10)
```

`forecast.mlm` *Forecast a multiple linear model with possible time series components*

Description

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

Usage

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

Arguments

`object`	Object of class "mlm", usually the result of a call to `stats::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 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.
`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(), stats::lm().

Examples

```
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))
```

---

forecast.modelAR *Forecasting using user-defined model*

Description

Returns forecasts and other information for user-defined models.

Usage

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

Arguments

object	An object of class modelAR resulting from a call to modelAR().
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 levels 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 any regression variables. A numerical vector or matrix of external regressors; it should not be a data frame.
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 are produced by simulation using resampled errors (rather than normally distributed errors). Ignored if innov is not NULL.
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.
npaths	Number of sample paths used in computing simulated prediction intervals.
...	Additional arguments passed to simulate.nnetar().

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.

forecast class

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.

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)

The function `summary` can be used to obtain and print a summary of the results, while the functions `plot` and `autoplot` produce plots of the forecasts and prediction intervals. The generic accessors functions `fitted.values` and `residuals` extract various useful features from the underlying model.

Author(s)

Rob J Hyndman and Gabriel Caceres

See Also

`nnetar()`.

`forecast.mts` *Forecasting time series*

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

```
## S3 method for class 'mts'
forecast(
  object,
  h = if (frequency(object) > 1) 2 * frequency(object) else 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. Default value is twice the largest seasonal period (for seasonal data) or ten (for non-seasonal data).

level	Confidence levels 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().

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()`.

`forecast.nnetar` *Forecasting using neural network models*

Description

Returns forecasts and other information for univariate neural network models.

Usage

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

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 levels 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 any regression variables. A numerical vector or matrix of external regressors; it should not be a data frame.
`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 are produced by simulation using resampled errors (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`.

forecast class

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.

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)

The function `summary` can be used to obtain and print a summary of the results, while the functions `plot` and `autoplot` produce plots of the forecasts and prediction intervals. The generic accessors functions `fitted.values` and `residuals` extract various useful features from the underlying model.

Author(s)

Rob J Hyndman and Gabriel Caceres

See Also

`nnetar()`.

Examples

```
## 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)
```

`forecast.rw_model` *Naive and Random Walk Forecasts*

Description

Returns forecasts and prediction intervals for a generalized random walk model. `rwf()` is a convenience function that combines `rw_model()` and `forecast()`. `naive()` is a wrapper to `rwf()` with `drift=FALSE` and `lag=1`, while `snaive()` is a wrapper to `rwf()` with `drift=FALSE` and `lag=frequency(y)`.

Usage

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

```
rwf(
  y,
  h = 10,
  drift = FALSE,
  level = c(80, 95),
  fan = FALSE,
  lambda = NULL,
  biasadj = FALSE,
  lag = 1,
  ...,
  x = y
)

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

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

Arguments

`object`	An object of class `rw_model` returned by `rw_model()`.
`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 levels 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). Ignored if `innov` is not `NULL`.

npaths	Number of sample paths used in computing simulated prediction intervals.
innov	Optional matrix of future innovations to be used in simulations. Ignored if `simulate = FALSE`. If provided, this overrides the `bootstrap` argument. The matrix should have `h` rows and `npaths` columns.
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 not used.
y	a numeric vector or univariate time series of class `ts`
drift	Logical flag. If `TRUE`, fits a random walk with drift model.
lag	Lag parameter. `lag = 1` corresponds to a standard random walk (giving naive forecasts if `drift = FALSE` or drift forecasts if `drift = TRUE`), while `lag = m` corresponds to a seasonal random walk where m is the seasonal period (giving seasonal naive forecasts if `drift = FALSE`).
x	Deprecated. Included for backwards compatibility.

Details

The model assumes that

$$Y_t = Y_{t-p} + c + \varepsilon_t$$

where p is the lag parameter, c is the drift parameter, and $\varepsilon_t \sim N(0, \sigma^2)$ are iid.

The model without drift has $c = 0$. In the model with drift, c is estimated by the sample mean of the differences $Y_t - Y_{t-p}$.

If $p = 1$, this is equivalent to an ARIMA(0,1,0) model with an optional drift coefficient. For $p > 1$, it is equivalent to an ARIMA(0,0,0)(0,1,0)p model.

The forecasts are given by

$$Y_{T+h|T} = Y_{T+h-p(k+1)} + ch$$

where k is the integer part of $(h-1)/p$. For a regular random walk, $p = 1$ and $c = 0$, so all forecasts are equal to the last observation. Forecast standard errors allow for uncertainty in estimating the drift parameter (unlike the corresponding forecasts obtained by fitting an ARIMA model directly).

The generic accessor functions `stats::fitted()` and `stats::residuals()` extract useful features of the object returned.

Value

An object of class `forecast`.

forecast class

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.

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)

The function `summary` can be used to obtain and print a summary of the results, while the functions `plot` and `autoplot` produce plots of the forecasts and prediction intervals. The generic accessors functions `fitted.values` and `residuals` extract various useful features from the underlying model.

Author(s)

Rob J Hyndman

See Also

`rw_model()`, `Arima()`

Examples

```
# Three ways to do the same thing
gold_model <- rw_model(gold)
gold_fc1 <- forecast(gold_model, h = 50)
gold_fc2 <- rwf(gold, h = 50)
gold_fc3 <- naive(gold, h = 50)

# Plot the forecasts
autoplot(gold_fc1)

# Drift forecasts
rwf(gold, drift = TRUE) |> autoplot()

# Seasonal naive forecasts
snaive(wineind) |> autoplot()
```

forecast.spline_model *Returns local linear forecasts and prediction intervals using cubic smoothing splines estimated with spline_model().*

Description

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.

Usage

```
## S3 method for class 'spline_model'
forecast(
  object,
  h = 10,
  level = c(80, 95),
  fan = FALSE,
  lambda = object$lambda,
  biasadj = attr(lambda, "biasadj"),
  simulate = FALSE,
  bootstrap = FALSE,
  innov = NULL,
  npaths = 5000,
  ...
)

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

Arguments

object	An object of class spline_model, produced using spline_model().
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 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.
`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). Ignored if `innov` is not `NULL`.
`innov`	Optional matrix of future innovations to be used in simulations. Ignored if `simulate = FALSE`. If provided, this overrides the `bootstrap` argument. The matrix should have `h` rows and `npaths` columns.
`npaths`	Number of sample paths used in computing simulated prediction intervals.
`...`	Other arguments are ignored.
`y`	a numeric vector or univariate time series of class `ts`
`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.
`x`	Deprecated. Included for backwards compatibility.

Value

An object of class `forecast`.

forecast class

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.

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)

The function `summary` can be used to obtain and print a summary of the results, while the functions `plot` and `autoplot` produce plots of the forecasts and prediction intervals. The generic accessors functions `fitted.values` and `residuals` extract various useful features from the underlying model.

Author(s)

Rob J Hyndman

References

Hyndman, King, Pitrun and Billah (2005) Local linear forecasts using cubic smoothing splines. *Australian and New Zealand Journal of Statistics*, **47**(1), 87-99. https://robjhyndman.com/publications/splinefcast/.

See Also

spline_model()

Examples

```
fit <- spline_model(uspop)
fcast <- forecast(fit)
autoplot(fcast)
summary(fcast)
```

---

forecast.stl *Forecasting using stl objects*

Description

Forecasts of STL objects are obtained by applying a non-seasonal forecasting method to the seasonally adjusted data and re-seasonalizing using the last year of the seasonal component.

Usage

```
## 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 = FALSE,
  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 = attr(lambda, "biasadj"),
  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 `stats::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 adjusted series. If `forecastfunction` is not `NULL`, then `method` is ignored. Otherwise `method` is used to specify the forecasting method to be used.
- `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 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 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 forecast.stl, modelfunction or forecastfunction.
y	a numeric vector or univariate time series of class ts
s.window	Either the character string "periodic" or the span (in lags) of the loess window for seasonal extraction.
t.window	A number to control the smoothness of the trend. See stats::stl() for details.
robust	If TRUE, robust fitting will used in the loess procedure within stats::stl().
x	Deprecated. Included for backwards compatibility.

Details

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

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

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

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

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

Value

stlm returns an object of class stlm. The other functions return objects of class forecast.

There are many methods for working with forecast() objects including summary to obtain and print a summary of the results, while plot produces a plot of the forecasts and prediction intervals. The generic accessor functions fitted.values and residuals extract useful features.

Author(s)

Rob J Hyndman

See Also

stats::stl(), forecast.ets(), forecast.Arima().

Examples

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

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

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

---

forecast.StructTS — *Forecasting using Structural Time Series models*

Description

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

Usage

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

Arguments

object	An object of class StructTS. Usually the result of a call to stats::StructTS().
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 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.
...	Other arguments are ignored.

Details

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

Value

An object of class `forecast`.

forecast class

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.

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)

The function `summary` can be used to obtain and print a summary of the results, while the functions `plot` and `autoplot` produce plots of the forecasts and prediction intervals. The generic accessors functions `fitted.values` and `residuals` extract various useful features from the underlying model.

Author(s)

Rob J Hyndman

See Also

`stats::StructTS()`.

Examples

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

forecast.theta_model *Theta method forecasts.*

Description

Returns forecasts and prediction intervals for a theta method forecast. `thetaf()` is a convenience function that combines `theta_model()` and `forecast.theta_model()`. The theta method of Assimakopoulos and Nikolopoulos (2000) is equivalent to simple exponential smoothing with drift (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.

Usage

```
## S3 method for class 'theta_model'
forecast(
  object,
  h = if (frequency(object$y) > 1) 2 * frequency(object$y) else 10,
  level = c(80, 95),
  fan = FALSE,
  lambda = object$lambda,
  biasadj = FALSE,
  ...
)

thetaf(
  y,
  h = if (frequency(y) > 1) 2 * frequency(y) else 10,
  level = c(80, 95),
  fan = FALSE,
  lambda = NULL,
  biasadj = FALSE,
  x = y,
  ...
)
```

Arguments

`object`	An object of class `theta_model` created by `theta_model()`.
`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 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.
...	Other arguments passed to forecast.ets.
y	a numeric vector or univariate time series of class ts
x	Deprecated. Included for backwards compatibility.

Details

More general theta methods are available in the **forecTheta** package.

Value

An object of class forecast.

forecast class

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.

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)

The function summary can be used to obtain and print a summary of the results, while the functions plot and autoplot produce plots of the forecasts and prediction intervals. The generic accessors functions fitted.values and residuals extract various useful features from the underlying model.

Author(s)

Rob J Hyndman

References

Assimakopoulos, V. and Nikolopoulos, K. (2000). The theta model: a decomposition approach to forecasting. *International Journal of Forecasting* **16**, 521-530.

Hyndman, R.J., and Billah, B. (2003) Unmasking the Theta method. *International J. Forecasting*, **19**, 287-290.

See Also

stats::arima(), meanf(), rwf(), ses()

Examples

```
nile_fit <- theta_model(Nile)
forecast(nile_fit) |> autoplot()
```

---

forecast.ts *Forecasting time series*

Description

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

Usage

```
## S3 method for class 'ts'
forecast(
  object,
  h = if (frequency(object) > 1) 2 * frequency(object) else 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. Default value is twice the largest seasonal period (for seasonal data) or ten (for non-seasonal data).
`level`	Confidence levels 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`, `bats`, 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 `stats::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`.

forecast class

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.

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)

The function `summary` can be used to obtain and print a summary of the results, while the functions `plot` and `autoplot` produce plots of the forecasts and prediction intervals. The generic accessors functions `fitted.values` and `residuals` extract various useful features from the underlying model.

Author(s)

Rob J Hyndman

See Also

Other functions which return objects of class `forecast` are `forecast.ets()`, `forecast.Arima()`, `forecast.HoltWinters()`, `forecast.StructTS()`, `meanf()`, `rwf()`, `splinef()`, `thetaf()`, `croston()`, `ses()`, `holt()`, `hw()`.

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
deaths.model <- auto.arima(
  USAccDeaths,
  xreg = fourier(USAccDeaths, K = 5),
  seasonal = FALSE
```

```
)
deaths.fcast <- forecast(
  deaths.model,
  xreg = fourier(USAccDeaths, K = 5, h = 36)
)
autoplot(deaths.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)
```

---

gas *Australian monthly gas production*

Description

Australian monthly gas production: 1956–1995.

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'
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,
  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 `grDevices::nclass.FD()`
`boundary`	A boundary between two bins.

Value

None.

Author(s)

Rob J Hyndman

See Also

`graphics::hist()`, `ggplot2::geom_histogram()`

Examples

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

gglagplot *Time series lag ggplots*

Description

Plots a lag plot using ggplot.

Usage

```
gglagplot(
  x,
  lags = if (frequency(x) > 9) 16 else 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 = if (frequency(x) > 1) min(12, frequency(x)) else 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.

Value

None.

Author(s)

Mitchell O'Hara-Wild

See Also

stats::lag.plot()

Examples

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

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

gglagchull(woolyrnq)
```

---

ggmonthplot *Create a seasonal subseries ggplot*

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)

Details

The `ggmonthplot` function is simply a wrapper for `ggsubseriesplot` as a convenience for users familiar with `stats::monthplot()`.

Value

Returns an object of class `ggplot`.

Author(s)

Mitchell O'Hara-Wild

See Also

`stats::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,
  s,
  season.labels = NULL,
  year.labels = FALSE,
  year.labels.left = FALSE,
  type = "o",
  main,
  xlab = NULL,
  ylab = "",
  col = 1,
  labelgap = 0.1,
  ...
)
```

Arguments

x	a numeric vector or time series 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 graphics::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 graphics::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

Hyndman and Athanasopoulos (2018) *Forecasting: principles and practice*, 2nd edition, OTexts: Melbourne, Australia. https://otexts.com/fpp2/

See Also

`stats::monthplot()`

Examples

```
ggseasonplot(AirPassengers, col = rainbow(12), year.labels = TRUE)
ggseasonplot(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 stats::na.contiguous(). Useful alternatives are stats::na.pass() and na.interp().
theme	Adds a ggplot element to each plot, typically a theme.
...	additional arguments to stats::acf().
ci.type	type of confidence limits for ACF that is passed to stats::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

Hyndman and Athanasopoulos (2018) *Forecasting: principles and practice*, 2nd edition, OTexts: Melbourne, Australia. https://otexts.com/fpp2/

See Also

stats::plot.ts(), Acf(), stats::spec.ar()

Examples

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

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

---

gold *Daily morning gold prices*

Description

Daily morning gold prices in US dollars. 1 January 1985 – 31 March 1989.

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.modelAR(x)

is.stlm(x)

is.nnetar(x)

is.nnetarmodels(x)
```

Arguments

x	object to be tested

---

is.constant — *Is an object constant?*

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 an object a particular forecast type?*

Description

Returns true if the forecast object is of a particular type

Usage

```
is.forecast(x)

is.mforecast(x)

is.splineforecast(x)
```

Arguments

`x`	object to be tested

`ma` *Moving-average smoothing*

Description

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

Usage

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

Arguments

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

`stats::decompose()`

Examples

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

`mean_model` *Mean Forecast Model*

Description

Fits a Gaussian iid model to a univariate time series.

Usage

```
mean_model(y, lambda = NULL, biasadj = FALSE)
```

Arguments

`y`	a numeric vector or univariate time series 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.

Details

The model assumes that the data are independent and identically distributed

$$Y_t \sim N(\mu, \sigma^2)$$

Forecasts are given by

$$Y_{n+h|n} = \mu$$

where μ is estimated by the sample mean.

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 `stats::fitted()` and `stats::residuals()` extract useful features of the object returned by `mean_model()`.

Value

An object of class `mean_model`.

forecast class

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.

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)

The function `summary` can be used to obtain and print a summary of the results, while the functions `plot` and `autoplot` produce plots of the forecasts and prediction intervals. The generic accessors functions `fitted.values` and `residuals` extract various useful features from the underlying model.

Author(s)

Rob J Hyndman

See Also

`forecast.mean_model()`, `meanf()`

Examples

```
fit_nile <- mean_model(Nile)
fit_nile |> forecast(h = 10) |> autoplot()
```

---

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

Description

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

Usage

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

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 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 mtomPwherem = frequency(y). If xregis provided, its columns are also used as inputs. If there are missing values inyorx 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 modelAR.

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

```
## Set up functions
my_lm <- function(x, y) {
 structure(lsfit(x,y), class = "lsfit")
}
predict.lsfit <- function(object, newdata = NULL) {
  n <- length(object$qr$qt)
  if(is.null(newdata)) {
    z <- numeric(n)
    z[seq_len(object$qr$rank)] <- object$qr$qt[seq_len(object$qr$rank)]
    as.numeric(qr.qy(object$qr, z))
  } else {
    sum(object$coefficients * c(1, newdata))
  }
}
# Fit an AR(2) model
fit <- modelAR(
  y = lynx,
  p = 2,
  FUN = my_lm,
  predict.FUN = predict.lsfit,
  lambda = 0.5,
  scale.inputs = TRUE
)
forecast(fit, h = 20) |> autoplot()
```

`modeldf` *Compute model degrees of freedom*

Description

Compute model degrees of freedom

Usage

```
modeldf(object, ...)
```

Arguments

`object`	A time series model.
`...`	Other arguments currently ignored.

`monthdays` *Number of days in each season*

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, `stats::supsmu()` is used to estimate the trend. Optionally, the time series may be Box-Cox transformed before decomposition. Unlike `stats::stl()`, `mstl` is completely automated.

Usage

```
mstl(
  x,
  lambda = NULL,
  biasadj = FALSE,
  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.
`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.
`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 `stats::stl()`.

See Also

`stats::stl()`, `stats::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)
```

na.interp — *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

`tsoutliers()`

Examples

```
data(gold)
plot(na.interp(gold))
```

ndiffs *Number of differences required for a stationary series*

Description

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

Usage

```
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

Dickey DA and Fuller WA (1979), "Distribution of the Estimators for Autoregressive Time Series with a Unit Root", *Journal of the American Statistical Association* **74**:427-431.

Kwiatkowski D, Phillips PCB, Schmidt P and Shin Y (1992) "Testing the Null Hypothesis of Stationarity against the Alternative of a Unit Root", *Journal of Econometrics* **54**:159-178.

Osborn, D.R. (1990) "A survey of seasonality in UK macroeconomic variables", *International Journal of Forecasting*, **6**:327-336.

Phillips, P.C.B. and Perron, P. (1988) "Testing for a unit root in time series regression", *Biometrika*, **72**(2), 335-346.

Said E and Dickey DA (1984), "Testing for Unit Roots in Autoregressive Moving Average Models of Unknown Order", *Biometrika* **71**:599-607.

See Also

`auto.arima()` and `ndiffs()`

Examples

```
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

```
nnetar(
  y,
  p,
  P = 1,
  size = NULL,
  repeats = 20,
  xreg = NULL,
  lambda = NULL,
  model = NULL,
  subset = NULL,
  scale.inputs = TRUE,
  parallel = FALSE,
  num.cores = 2,
  x = y,
  ...
)
```

Arguments

y	a numeric vector or univariate 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). If set to zero to indicate that no non-seasonal lags should be included, then P must be at least 1 and a model with only seasonal lags will be fit.
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 numerical vector or matrix of external regressors, which must have the same number of rows as y. It should not be a data frame.
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.
parallel	If TRUE, then the specification search is done in parallel via parallel::parLapply(). 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. If NULL, then the number of logical cores is automatically detected and all available cores are used.
x	Deprecated. Included for backwards compatibility.
...	Other arguments passed to nnet::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

```
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 for consumption", *Oxford Bulletin of Economics and Statistics* **50**(4):361-377.

Canova F and Hansen BE (1995) "Are Seasonal Patterns Constant over Time? A Test for Seasonal Stability", *Journal of Business and Economic Statistics* **13**(3):237-252.

Hylleberg S, Engle R, Granger C and Yoo B (1990) "Seasonal integration and cointegration.", *Journal of Econometrics* **44**(1), pp. 215-238.

See Also

`auto.arima()`, `ndiffs()`, `ocsb.test()`, `uroot::hegy.test()`, and `uroot::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

```
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

Osborn DR, Chui APL, Smith J, and Birchenhall CR (1988) "Seasonality and the order of integration for consumption", *Oxford Bulletin of Economics and Statistics* **50**(4):361-377.

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

```
## 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, ...)
```

Arguments

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

Examples

```
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

```
## 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 `graphics::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

```
## 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

```
## 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()`

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,
  xlab = "",
  ylab = "",
  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 `graphics::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.

plot.splineforecast autoplot.splineforecast

Value

None.

Author(s)

Rob J Hyndman & Mitchell O'Hara-Wild

References

Hyndman and Athanasopoulos (2018) *Forecasting: principles and practice*, 2nd edition, OTexts: Melbourne, Australia. https://otexts.com/fpp2/

See Also

`stats::plot.ts()`

Examples

```
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

```
## 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.
h	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

```
fit <- Arima(lynx, order = c(4, 0, 0), lambda = 0.5)

plot(residuals(fit))
plot(residuals(fit, type = "response"))
```

---

rw_model *Random walk model*

Description

Fit a generalized random walk with Gaussian errors (and optional drift) to a univariate time series.

Usage

```
rw_model(y, lag = 1, drift = FALSE, lambda = NULL, biasadj = FALSE)
```

Arguments

y	a numeric vector or univariate time series of class ts
lag	Lag parameter. lag = 1 corresponds to a standard random walk (giving naive forecasts if drift = FALSE or drift forecasts if drift = TRUE), while lag = m corresponds to a seasonal random walk where m is the seasonal period (giving seasonal naive forecasts if drift = FALSE).
drift	Logical flag. If TRUE, fits a random walk with drift model.
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.

Details

The model assumes that

$$Y_t = Y_{t-p} + c + \varepsilon_t$$

where p is the lag parameter, c is the drift parameter, and $\varepsilon_t \sim N(0, \sigma^2)$ are iid.

The model without drift has $c = 0$. In the model with drift, c is estimated by the sample mean of the differences $Y_t - Y_{t-p}$.

If $p = 1$, this is equivalent to an ARIMA(0,1,0) model with an optional drift coefficient. For $p > 1$, it is equivalent to an ARIMA(0,0,0)(0,1,0)p model.

The forecasts are given by

$$Y_{T+h|T} = Y_{T+h-p(k+1)} + ch$$

where k is the integer part of $(h-1)/p$. For a regular random walk, $p = 1$ and $c = 0$, so all forecasts are equal to the last observation. Forecast standard errors allow for uncertainty in estimating the drift parameter (unlike the corresponding forecasts obtained by fitting an ARIMA model directly).

The generic accessor functions stats::fitted() and stats::residuals() extract useful features of the object returned.

Value

An object of class rw_model.

See Also

forecast.rw_model(), rwf(), naive(), snaive()

Examples

```
model <- rw_model(gold)
forecast(model, h = 50) |> autoplot()
```

---

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, ...)
```

Arguments

object	Object created by `stats::decompose()`, `stats::stl()` or `tbats()`.
...	Other arguments not currently used.

Value

Univariate time series.

Author(s)

Rob J Hyndman

See Also

`stats::stl()`, `stats::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 `stats::decompose()`, `stats::stl()` or `tbats()`.

Value

Univariate time series.

Author(s)

Rob J Hyndman

See Also

`stats::stl()`, `stats::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
) +
  labs(x = "Year", y = "")
```

---

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.

Value

Numerical matrix.

Author(s)

Rob J Hyndman

See Also

fourier()

Examples

```
plot(ldeaths)

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

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

---

ses *Exponential smoothing forecasts*

Description

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

Usage

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

Arguments

y	a numeric vector or univariate time series of class ts
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 levels for prediction intervals.
fan	If TRUE, level is set to seq(51, 99, by = 3). This is suitable for fan plots.
initial	Method used for selecting initial state values. If optimal, the initial values are optimized along with the smoothing parameters using ets(). If simple, the initial values are set to values obtained using simple calculations on the first few observations. See Hyndman & Athanasopoulos (2014) for details.
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 trans-

	formation 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 `"multiplicative"`.
`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`.

forecast class

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.

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)

The function `summary` can be used to obtain and print a summary of the results, while the functions `plot` and `autoplot` produce plots of the forecasts and prediction intervals. The generic accessors functions `fitted.values` and `residuals` extract various useful features from the underlying model.

Author(s)

Rob J Hyndman

References

Hyndman, R.J., Koehler, A.B., Ord, J.K., Snyder, R.D. (2008) *Forecasting with exponential smoothing: the state space approach*, Springer-Verlag: New York. https://robjhyndman.com/expsmooth/.

Hyndman and Athanasopoulos (2018) *Forecasting: principles and practice*, 2nd edition, OTexts: Melbourne, Australia. https://otexts.com/fpp2/

See Also

ets(), stats::HoltWinters(), rwf(), stats::arima().

Examples

```
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

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

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

```
    lambda = object$lambda,
    ...
  )

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

  ## S3 method for class 'rw_model'
  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$n,
    seed = NULL,
    future = TRUE,
    bootstrap = FALSE,
    innov = NULL,
    lambda = object$lambda,
    ...
  )

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

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

Arguments

object	An object representing a fitted time series model. For example, it may be of class ets, Arima, ar, nnetar, etc.
nsim	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).
seed	Either NULL or an integer that will be used in a call to set.seed() before simu-

	lating the time series. The default, NULL, will not change the random generator state.
future	Produce sample paths that are future to and conditional on the data in object. Otherwise simulate unconditionally.
bootstrap	Do simulation using resampled errors rather than normally distributed errors or errors provided as innov.
innov	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.
...	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 stats::arima(). By default, the error series is assumed normally distributed and generated using stats::rnorm(). If innov is present, it is used instead. If bootstrap = TRUE and innov = NULL, the residuals are resampled instead.

When future = TRUE, the sample paths are conditional on the data. When future = FALSE and the model is stationary, the sample paths do not depend on the data at all. When future = FALSE and the model is non-stationary, the location of the sample paths is arbitrary, so they all start at the value of the first observation.

Value

An object of class ts.

Author(s)

Rob J Hyndman

See Also

ets(), Arima(), auto.arima(), ar(), arfima(), nnetar().

Examples

```
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 stats::decompose() or stats::stl().
h	Number of periods ahead to forecast.

Value

Time series

Author(s)

Rob J Hyndman

Examples

```
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")
```

`spline_model` *Cubic spline stochastic model*

Description

Fits a state space model based on cubic smoothing splines. 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.

Usage

```
spline_model(y, method = c("gcv", "mle"), lambda = NULL, biasadj = FALSE)
```

Arguments

- `y` — a numeric vector or univariate time series of class `ts`
- `method` — Method for selecting the smoothing parameter. If `method = "gcv"`, the generalized cross-validation method from `stats::smooth.spline()` is used. If `method = "mle"`, the maximum likelihood method from Hyndman et al (2002) is used.
- `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.

Value

An object of class `spline_model`.

Author(s)

Rob J Hyndman

References

Hyndman, King, Pitrun and Billah (2005) Local linear forecasts using cubic smoothing splines. *Australian and New Zealand Journal of Statistics*, **47**(1), 87-99. https://robjhyndman.com/publications/splinefcast/.

See Also

`stats::smooth.spline()`, `stats::arima()`, `holt()`.

Examples

```
fit <- spline_model(uspop)
fit
fit |> forecast() |> autoplot()
```

`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

geom_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()`. 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 `ggplot2::ggplot()`.

A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `ggplot2::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. `ggplot2::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 `ggplot2::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

`generics::forecast()`, `ggplot2::ggproto()`

Examples

```
## 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)
```

---

stlm *Forecasting model using STL with a generative time series model*

Description

Forecasts of STL objects are obtained by applying a non-seasonal forecasting model to the seasonally adjusted data and re-seasonalizing using the last year of the seasonal component. `stlm` takes a time series `y`, applies an STL decomposition, and models the seasonally adjusted data using the model passed as `modelfunction` or specified using `method`. It returns an object that includes the original STL decomposition and a time series model fitted to the seasonally adjusted data. This object can be passed to the `forecast.stlm` for forecasting.

Usage

```
stlm(
  y,
  s.window = 7 + 4 * seq(6),
```

```
  t.window = NULL,
  robust = FALSE,
  method = c("ets", "arima"),
  modelfunction = NULL,
  model = NULL,
  etsmodel = "ZZN",
  lambda = NULL,
  biasadj = FALSE,
  xreg = NULL,
  allow.multiplicative.trend = FALSE,
  x = y,
  ...
)
```

Arguments

y	a numeric vector or univariate time series of class ts
s.window	Either the character string "periodic" or the span (in lags) of the loess window for seasonal extraction.
t.window	A number to control the smoothness of the trend. See stats::stl() for details.
robust	If TRUE, robust fitting will used in the loess procedure within stats::stl().
method	Method to use for forecasting the seasonally adjusted series.
modelfunction	An alternative way of specifying the function for modelling the seasonally adjusted series. If modelfunction is not NULL, then method is ignored. Otherwise method is used to specify the time series model to be used.
model	Output from a previous call to stlm. If a stlm model is passed, this same model is fitted to y without re-estimating any parameters.
etsmodel	The ets model specification passed to ets(). By default it allows any non-seasonal model. If method != "ets", this argument is 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.
xreg	Historical regressors to be used in auto.arima() when method = "arima".
allow.multiplicative.trend	If TRUE, then ETS models with multiplicative trends are allowed. Otherwise, only additive or no trend ETS models are permitted.
x	Deprecated. Included for backwards compatibility.
...	Other arguments passed to modelfunction.

Details

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

Value

An object of class `stlm`.

Author(s)

Rob J Hyndman

See Also

`stats::stl()`, `ets()`, `Arima()`.

Examples

```
tsmod <- stlm(USAccDeaths, modelfunction = ar)
forecast(tsmod, h = 36) |> autoplot()

decomp <- stl(USAccDeaths, s.window = "periodic")
forecast(decomp) |> autoplot()
```

`subset.ts` *Subsetting a time series*

Description

Various types of subseting of a time series. Allows subsetting by index values (unlike `stats::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

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

```
)

## 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()`, `stats::window()`

Examples

```
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

Taylor, J.W. (2003) Short-term electricity demand forecasting using double seasonal exponential smoothing. *Journal of the Operational Research Society*, **54**, 799-805.

Examples

```
plot(taylor)
```

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

Description

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

Usage

```
tbats(
  y,
  use.box.cox = NULL,
  use.trend = NULL,
  use.damped.trend = NULL,
  seasonal.periods = NULL,
  use.arma.errors = TRUE,
  use.parallel = 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 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.
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

De Livera, A.M., Hyndman, R.J., & Snyder, R. D. (2011), Forecasting time series with complex seasonal patterns using exponential smoothing, *Journal of the American Statistical Association*, **106**(496), 1513-1527.

See Also

`tbats.components()`.

Examples

```
## 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

De Livera, A.M., Hyndman, R.J., & Snyder, R. D. (2011), Forecasting time series with complex seasonal patterns using exponential smoothing, *Journal of the American Statistical Association*, **106**(496), 1513-1527.

See Also

tbats().

Examples

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

## End(Not run)
```

---

theta_model *Theta model*

Description

The theta method of Assimakopoulos and Nikolopoulos (2000) is equivalent to simple exponential smoothing with drift (Hyndman and Billah, 2003). This function fits the theta model to a time series. 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 fitting the theta model.

Usage

```
theta_model(y, lambda = NULL, biasadj = FALSE)
```

Arguments

y — a numeric vector or univariate time series 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.

Details

More general theta methods are available in the **forecTheta** package.

Value

An object of class `theta_model`.

Author(s)

Rob J Hyndman

References

Assimakopoulos, V. and Nikolopoulos, K. (2000). The theta model: a decomposition approach to forecasting. *International Journal of Forecasting* **16**, 521-530.

Hyndman, R.J., and Billah, B. (2003) Unmasking the Theta method. *International J. Forecasting*, **19**, 287-290.

See Also

`thetaf()`

Examples

```
nile_fit <- theta_model(Nile)
forecast(nile_fit) |> autoplot()
```

`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/.

See Also

`na.interp()`, `tsoutliers()`, `stats::supsmu()`

Examples

```
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

```
tsCV(y, forecastfunction, h = 1, window = NULL, xreg = NULL, initial = 0, ...)
```

Arguments

y	a numeric vector or univariate time series of class ts
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	Number of periods for forecasting. Default value is twice the largest seasonal period (for seasonal data) or ten (for non-seasonal data).
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 $y_1, \dots, y_T$. Then forecastfunction is applied successively to the time series $y_1, \dots, y_t$, for $t = 1, \dots, T - h$, 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, $e_1, \dots, e_T$. 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

CV(), CVar(), residuals.Arima(), https://robjhyndman.com/hyndsight/tscv/.

Examples

```
#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), xreg = newxreg)
}

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

```
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 `stats::lm()`.

Details

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

Value

Returns an object of class "lm".

Author(s)

Mitchell O'Hara-Wild and Rob J Hyndman

See Also

`forecast.lm()`, `stats::lm()`.

Examples

```
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

```
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/.

See Also

`na.interp()`, `tsclean()`

Examples

```
data(gold)
tsoutliers(gold)
```

`wineind` *Australian total wine sales*

Description

Australian total wine sales by wine makers in bottles <= 1 litre. Jan 1980 – Aug 1994.

Usage

```
wineind
```

Format

Time series data

Source

Time Series Data Library. https://pkg.yangzhuoranyang.com/tsdl/

Examples

```
tsdisplay(wineind)
```

---

woolyrnq *Quarterly production of woollen yarn in Australia*

Description

Quarterly production of woollen yarn in Australia: tonnes. Mar 1965 – Sep 1994.

Usage

```
woolyrnq
```

Format

Time series data

Source

Time Series Data Library. https://pkg.yangzhuoranyang.com/tsdl/

Examples

```
tsdisplay(woolyrnq)
```

Index