

# Package ‘forecast’

September 30, 2013

**Version** 4.8

**Date** 2013-09-30

**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 (>= 2.14.0), stats, graphics

**Imports** tseries, fracdiff, zoo, Rcpp (>= 0.9.10), RcppArmadillo (>= 0.2.35), nnet, colorspace, parallel

**Suggests** Rmalschains

**LinkingTo** Rcpp, RcppArmadillo

**LazyData** yes

**ByteCompile** TRUE

**Author** Rob J Hyndman <Rob.Hyndman@monash.edu> with contributions from George Athanasopoulos, Slava Razbash, Drew Schmidt, Zhenyu Zhou, Yousaf Khan, Christoph Bergmeir

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

**License** GPL (>= 2)

**URL** <http://robjhyndman.com/software/forecast/>

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2013-09-30 08:07:30

**R topics documented:**

accuracy . . . . .	3
Acf . . . . .	4
arfima . . . . .	5
Arima . . . . .	6
arima.errors . . . . .	9
auto.arima . . . . .	10
bats . . . . .	12
BoxCox . . . . .	13
BoxCox.lambda . . . . .	14
croston . . . . .	15
CV . . . . .	17
dm.test . . . . .	17
dshw . . . . .	19
ets . . . . .	21
fitted.Arima . . . . .	23
forecast . . . . .	23
forecast.Arima . . . . .	25
forecast.bats . . . . .	27
forecast.ets . . . . .	28
forecast.HoltWinters . . . . .	30
forecast.lm . . . . .	31
forecast.stl . . . . .	33
forecast.StructTS . . . . .	35
gas . . . . .	36
getResponse . . . . .	37
gold . . . . .	37
logLik.ets . . . . .	38
ma . . . . .	39
meanf . . . . .	39
monthdays . . . . .	41
msts . . . . .	42
na.interp . . . . .	43
naive . . . . .	43
ndiffs . . . . .	45
nnetar . . . . .	46
plot.bats . . . . .	48
plot.ets . . . . .	49
plot.forecast . . . . .	49
rwf . . . . .	51
seasadj . . . . .	52
seasonaldummy . . . . .	53
seasonplot . . . . .	54
ses . . . . .	55
simulate.ets . . . . .	57
sindexf . . . . .	59
splinef . . . . .	59

accuracy 3

subset.ts . . . . .	61
taylor . . . . .	62
tbats . . . . .	63
tbats.components . . . . .	64
thetaf . . . . .	65
tsdisplay . . . . .	66
tslm . . . . .	67
wineind . . . . .	68
woolrymq . . . . .	69

**Index** 70

accuracy	<i>Accuracy measures for forecast model</i>
----------	---

### Description

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

### Usage

```
accuracy(f, x, test="all")
```

### Arguments

<i>f</i>	An object of class "forecast", or a numerical vector containing forecasts. It will also work with Arima, ets and lm objects if <i>x</i> is omitted – in which case in-sample accuracy measures are returned.
<i>x</i>	An optional numerical vector containing actual values of the same length as object, or a time series overlapping with the times of <i>f</i> .
<i>test</i>	Indicator of which elements of <i>x</i> and <i>f</i> to test. If <i>test</i> =="all", all elements are used. Otherwise <i>test</i> is a numeric vector containing the indices of the elements to use in the test.

### Details

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

### 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).

**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 Function Estimation

---

**Description**

Largely wrappers for the [acf](#) function in the stats package. The main difference is that Acf does not plot a spike at lag 0 (which is redundant). Pacf is included for consistency.

**Usage**

```
Acf(x, lag.max=NULL, type=c("correlation", "partial"),
    plot=TRUE, main=NULL, ylim=NULL, ...)
Pacf(x, main=NULL, ...)
```

**Arguments**

x	a univariate time series
lag.max	maximum lag at which to calculate the acf. Default is $10 \cdot \log_{10}(N/m)$ where N is the number of observations and m the number of series. Will be automatically limited to one less than the number of observations in the series.
type	character string giving the type of acf to be computed. Allowed values are "correlation" (the default) or "partial".
plot	logical. If TRUE (the default) the acf is plotted.
main	Title for plot
ylim	The y limits of the plot
...	Additional arguments passed to <a href="#">acf</a> .

**Details**

See the [acf](#) function in the stats package.

**Value**

See the [acf](#) function in the stats package.

**Author(s)**

Rob J Hyndman

**See Also**

[acf](#)

**Examples**

```
Acf(wineind)
Pacf(wineind)
```

---

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

**Arguments**

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

## Details

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

## Value

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

## Author(s)

Rob J Hyndman and Farah Yasmeeen

## 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`, `auto.arima`, `forecast.fracdiff`.

## Examples

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

---

Arima

---

*Fit ARIMA model to univariate time series*

---

## Description

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

**Usage**

```
Arima(x, order=c(0,0,0), seasonal=list(order=c(0,0,0), period=NA),
      xreg=NULL, include.mean=TRUE, include.drift=FALSE,
      include.constant, lambda=model$lambda, transform.pars=TRUE,
      fixed=NULL, init=NULL, method=c("CSS-ML", "ML", "CSS"), n.cond,
      optim.control=list(), kappa=1e6, model=NULL)
```

**Arguments**

<code>x</code>	a univariate time series of class <code>ts</code> .
<code>order</code>	A specification of the non-seasonal part of the ARIMA model: the three components (p, d, q) are the AR order, the degree of differencing, and the MA order.
<code>seasonal</code>	A specification of the seasonal part of the ARIMA model, plus the period (which defaults to <code>frequency(x)</code> ). This should be a list with components <code>order</code> and <code>period</code> , but a specification of just a numeric vector of length 3 will be turned into a suitable list with the specification as the <code>order</code> .
<code>xreg</code>	Optionally, a vector or matrix of external regressors, which must have the same number of rows as <code>x</code> .
<code>include.mean</code>	Should the ARIMA model include a mean term? The default is <code>TRUE</code> for undifferenced series, <code>FALSE</code> for differenced ones (where a mean would not affect the fit nor predictions).
<code>include.drift</code>	Should the ARIMA model include a linear drift term? (i.e., a linear regression with ARIMA errors is fitted.) The default is <code>FALSE</code> .
<code>include.constant</code>	If <code>TRUE</code> , then <code>include.mean</code> is set to be <code>TRUE</code> for undifferenced series and <code>include.drift</code> is set to be <code>TRUE</code> 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.
<code>lambda</code>	Box-Cox transformation parameter. Ignored if <code>NULL</code> . Otherwise, data transformed before model is estimated.
<code>transform.pars</code>	Logical. If true, the AR parameters are transformed to ensure that they remain in the region of stationarity. Not used for <code>method="CSS"</code> .
<code>fixed</code>	optional numeric vector of the same length as the total number of parameters. If supplied, only NA entries in <code>fixed</code> will be varied. <code>transform.pars=TRUE</code> will be overridden (with a warning) if any AR parameters are fixed. It may be wise to set <code>transform.pars=FALSE</code> when fixing MA parameters, especially near non-invertibility.
<code>init</code>	optional numeric vector of initial parameter values. Missing values will be filled in, by zeroes except for regression coefficients. Values already specified in <code>fixed</code> will be ignored.
<code>method</code>	Fitting method: maximum likelihood or minimize conditional sum-of-squares. The default (unless there are missing values) is to use conditional-sum-of-squares to find starting values, then maximum likelihood.

n.cond	Only used if fitting by conditional-sum-of-squares: the number of initial observations to ignore. It will be ignored if less than the maximum lag of an AR term.
optim.control	List of control parameters for optim.
kappa	the prior variance (as a multiple of the innovations variance) for the past observations in a differenced model. Do not reduce this.
model	Output from a previous call to Arima. If model is passed, this same model is fitted to x without re-estimating any parameters.

### Details

See the [arima](#) function in the stats package.

### Value

See the [arima](#) function in the stats package. The additional objects returned are

x	The time series data
xreg	The regressors used in fitting (when relevant).

### Author(s)

Rob J Hyndman

### See Also

[arima](#)

### Examples

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

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

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

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



---

arima.errors	<i>ARIMA errors</i>
--------------	---------------------

---

## Description

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

## Usage

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

## Arguments

<code>z</code>	Fitted ARIMA model from <a href="#">arima</a>
----------------	---

## Value

A time series containing the "errors".

## Author(s)

Rob J Hyndman

## See Also

[arima](#), [residuals](#)

## Examples

```
ukdeaths.fit <- Arima(UKDriverDeaths,c(1,0,1),c(0,1,1),xreg=Seatbelts[, "law"])
ukdeaths.errors <- arima.errors(ukdeaths.fit)
par(mfrow=c(2,1))
plot(UKDriverDeaths)
plot(ukdeaths.errors)
```

auto.arima

*Fit best ARIMA model to univariate time series***Description**

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

**Usage**

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

**Arguments**

x	a univariate time series
d	Order of first-differencing. If missing, will choose a value based on KPSS test.
D	Order of seasonal-differencing. If missing, will choose a value based on OCSB test.
max.p	Maximum value of p
max.q	Maximum value of q
max.P	Maximum value of P
max.Q	Maximum value of Q
max.order	Maximum value of p+q+P+Q if model selection is not stepwise.
start.p	Starting value of p in stepwise procedure.
start.q	Starting value of q in stepwise procedure.
start.P	Starting value of P in stepwise procedure.
start.Q	Starting value of Q in stepwise procedure.
stationary	If TRUE, restricts search to stationary models.
seasonal	If FALSE, restricts search to non-seasonal models.
ic	Information criterion to be used in model selection.
stepwise	If TRUE, will do stepwise selection (faster). Otherwise, it searches over all models. Non-stepwise selection can be very slow, especially for seasonal models.
trace	If TRUE, the list of ARIMA models considered will be reported.
approximation	If TRUE, estimation is via conditional sums of squares and the information criteria used for model selection are approximated. The final model is still computed using maximum likelihood estimation. Approximation should be used for long time series or a high seasonal period to avoid excessive computation times.

xreg	Optionally, a vector or matrix of external regressors, which must have the same number of rows as x.
test	Type of unit root test to use. See <a href="#">ndiffs</a> for details.
seasonal.test	This determines which seasonal unit root test is used. See <a href="#">nsdiffs</a> for details.
allowdrift	If TRUE, models with drift terms are considered.
lambda	Box-Cox transformation parameter. Ignored if NULL. Otherwise, data transformed before model is estimated.
parallel	If TRUE and stepwise = FALSE, then the specification search is done in parallel. This can give a significant speedup on multicore machines.
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.

### Details

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

### Value

Same as for [arima](#)

### Author(s)

Rob J Hyndman

### References

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

### See Also

[Arima](#)

### Examples

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

---

bats	<i>BATS model (Exponential smoothing state space model with Box-Cox transformation, ARMA errors, Trend and Seasonal components)</i>
------	---

---

## 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=TRUE,
     num.cores=NULL, bc.lower=0, bc.upper=1, ...)
```

## Arguments

<code>y</code>	The time series to be forecast. Can be numeric, <code>msts</code> or <code>ts</code> . Only univariate time series are supported.
<code>use.box.cox</code>	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.
<code>use.trend</code>	TRUE/FALSE indicates whether to include a trend or not. If NULL then both are tried and the best fit is selected by AIC.
<code>use.damped.trend</code>	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.
<code>seasonal.periods</code>	If <code>y</code> is a numeric then seasonal periods can be specified with this parameter.
<code>use.arma.errors</code>	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.
<code>use.parallel</code>	TRUE/FALSE indicates whether or not to use parallel processing.
<code>num.cores</code>	The number of parallel processes to be used if using parallel processing. If NULL then the number of logical cores is detected.
<code>bc.lower</code>	The lower limit (inclusive) for the Box-Cox transformation.
<code>bc.upper</code>	The upper limit (inclusive) for the Box-Cox transformation.
<code>...</code>	Additional parameters to be passed to <code>auto.arima</code> when choose an ARMA(p, q) model for the errors.

## 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.

**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**

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

## End(Not run)
```

---

BoxCox

*Box Cox Transformation*


---

**Description**

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

**Usage**

```
BoxCox(x, lambda)
InvBoxCox(x, lambda)
```

**Arguments**

x	a numeric vector or time series
lambda	transformation parameter

**Details**

The Box-Cox transformation is given by

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

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

**References**

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

**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.

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

**Usage**

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

**Arguments**

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

**Value**

a number indicating the Box-Cox transformation parameter.

**Author(s)**

Leanne Chhay and Rob J Hyndman

**References**

- 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))
```

---

croston

---

*Forecasts for intermittent demand using Croston's method*


---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

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

**Value**

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

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

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

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

**Author(s)**

Rob J Hyndman

**References**

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

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

**See Also**

[ses](#).

**Examples**

```
x <- rpois(20, lambda=.3)
fcast <- croston(x)
plot(fcast)
```



---

CV	<i>Cross-validation statistic</i>
----	-----------------------------------

---

**Description**

Computes cross-validation statistic, AIC, corrected AIC, BIC and adjusted  $R^2$  values for a linear model.

**Usage**

```
CV(obj)
```

**Arguments**

obj                      output from `lm` or `tslm`

**Value**

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

**Author(s)**

Rob J Hyndman

**See Also**

[AIC](#)

**Examples**

```
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)
```

---

dm.test	<i>Diebold-Mariano test for predictive accuracy</i>
---------	---

---

**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)
```

**Arguments**

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

**Details**

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

**Value**

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

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

**Author(s)**

George Athanasopoulos, Yousaf Khan and Rob Hyndman

**References**

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

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

## Arguments

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

## Details

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

**Value**

An object of class "forecast".

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

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

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

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

**Author(s)**

Rob J Hyndman

**References**

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. <http://www.exponentials smoothing.net>.

**See Also**

[HoltWinters](#), [ets](#).

**Examples**

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

## End(Not run)

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

ets

*Exponential smoothing state space model***Description**

Returns ets model applied to y.

**Usage**

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

**Arguments**

y	a numeric vector or time series
model	Usually a three-character string identifying method using the framework terminology of Hyndman et al. (2002) and Hyndman et al. (2008). The first letter denotes the error type ("A", "M" or "Z"); the second letter denotes the trend type ("N", "A", "M" or "Z"); and the third letter denotes the season type ("N", "A", "M" or "Z"). In all cases, "N"=none, "A"=additive, "M"=multiplicative and "Z"=automatically selected. So, for example, "ANN" is simple exponential smoothing with additive errors, "MAM" is multiplicative Holt-Winters' method with multiplicative errors, and so on. It is also possible for the model to be equal to the output from a previous call to ets. In this case, the same model is fitted to y without re-estimating any parameters.
damped	If TRUE, use a damped trend (either additive or multiplicative). If NULL, both damped and non-damped trends will be tried and the best model (according to the information criterion ic) returned.
alpha	Value of alpha. If NULL, it is estimated.
beta	Value of beta. If NULL, it is estimated.
gamma	Value of gamma. If NULL, it is estimated.
phi	Value of phi. If NULL, it is estimated.
additive.only	If TRUE, will only consider additive models. Default is FALSE.
lambda	Box-Cox transformation parameter. Ignored if NULL. Otherwise, data transformed before model is estimated. When lambda=TRUE, additive.only is set to FALSE.
lower	Lower bounds for the parameters (alpha, beta, gamma, phi)
upper	Upper bounds for the parameters (alpha, beta, gamma, phi)

<code>opt.crit</code>	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).
<code>nmse</code>	Number of steps for average multistep MSE ( $1 \leq \text{nmse} \leq 10$ ).
<code>bounds</code>	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.
<code>ic</code>	Information criterion to be used in model selection.
<code>restrict</code>	If TRUE, the models with infinite variance will not be allowed.
<code>...</code>	Other undocumented arguments.

### Details

Based on the classification of methods as described in Hyndman et al (2008).

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

### Value

An object of class "ets".

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

### Author(s)

Rob J Hyndman

### References

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. <http://www.exponentialsMOOTHING.net>.

### See Also

[HoltWinters](#), [rwf](#), [arima](#).

### Examples

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

---

fitted.Arima	<i>One-step in-sample forecasts using ARIMA models</i>
--------------	--

---

**Description**

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

**Usage**

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

**Arguments**

object	An object of class "Arima". Usually the result of a call to <a href="#">arima</a> .
...	Other arguments.

**Value**

An time series of the one-step forecasts.

**Author(s)**

Rob J Hyndman

**See Also**

[forecast.Arima](#).

**Examples**

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

---

forecast	<i>Forecasting time series</i>
----------	--------------------------------

---

**Description**

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

For example, the function [forecast.Arima](#) makes forecasts based on the results produced by [arima](#).

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

**Usage**

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

**Arguments**

<code>object</code>	a time series or time series model for which forecasts are required
<code>h</code>	Number of periods for forecasting
<code>level</code>	Confidence level for prediction intervals.
<code>fan</code>	If TRUE, level is set to seq(50,99,by=1). This is suitable for fan plots.
<code>...</code>	Additional arguments affecting the forecasts produced. <code>forecast.ts</code> passes these to <code>forecast.ets</code> or <code>stlf</code> depending on the frequency of the time series.

**Value**

An object of class "forecast".

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

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

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

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

**Author(s)**

Rob J Hyndman

**See Also**

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



forecast.Arima

*Forecasting using ARIMA or ARFIMA models***Description**

Returns forecasts and other information for univariate ARIMA models.

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

An object of class "forecast".

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

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

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

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

**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**

[predict.Arima](#), [predict.ar](#), [auto.arima](#), [Arima](#), [arima](#), [ar](#), [arfima](#).

**Examples**

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

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

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

**Arguments**

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

**Value**

An object of class "forecast".

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

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

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

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

**Author(s)**

Slava Razbash and Rob J Hyndman

**References**

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**

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

## End(Not run)
```

---

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=ifelse(object$m>1, 2*object$m, 10),
         level=c(80,95), fan=FALSE, simulate=FALSE, bootstrap=FALSE,
         npaths=5000, PI=TRUE, lambda=object$lambda, ...)
```

**Arguments**

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

npaths	Number of sample paths used in computing simulated prediction intervals.
PI	If TRUE, prediction intervals are produced, otherwise only point forecasts are calculated. If PI is FALSE, then level, fan, simulate, bootstrap and npaths are all ignored.
lambda	Box-Cox transformation parameter. Ignored if NULL. Otherwise, forecasts back-transformed via an inverse Box-Cox transformation.
...	Other arguments.

**Value**

An object of class "forecast".

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

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

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

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

**Author(s)**

Rob J Hyndman

**See Also**

[ets](#), [ses](#), [holt](#), [hw](#).

**Examples**

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

---

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=ifelse(frequency(object$x)>1,2*frequency(object$x),10),
         level=c(80,95),fan=FALSE,lambda=NULL,...)
```

## Arguments

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

## Details

This function calls [predict.HoltWinters](#) and constructs an object of class "forecast" from the results.

It is included for completeness, but the [ets](#) is recommended for use instead of [HoltWinters](#).

## Value

An object of class "forecast".

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

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

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

model	A list containing information about the fitted model
method	The name of the forecasting method as a character string
mean	Point forecasts as a time series
lower	Lower limits for prediction intervals
upper	Upper limits for prediction intervals

level	The confidence values associated with the prediction intervals
x	The original time series (either object itself or the time series used to create the model stored as object).
residuals	Residuals from the fitted model. That is x minus fitted values.
fitted	Fitted values (one-step forecasts)

**Author(s)**

Rob J Hyndman

**See Also**

[predict.HoltWinters](#), [HoltWinters](#).

**Examples**

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

---

forecast.lm	<i>Forecast a linear model with possible time series components</i>
-------------	---

---

**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, ts=TRUE, ...)
```

**Arguments**

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

## Details

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

## Value

An object of class "forecast".

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

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

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

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

## Author(s)

Rob J Hyndman

## See Also

[tslm](#), [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.stl

Forecasting using stl objects

**Description**

Returns forecasts obtained by either ETS or ARIMA models applied to the seasonally adjusted data from an STL decomposition.

**Usage**

```
## S3 method for class 'stl'
forecast(object, method=c("ets", "arima", "naive", "rwdrift"),
         etsmodel="ZZN", h=frequency(object$time.series)*2, level=c(80,95),
         fan=FALSE, lambda=NULL, xreg=NULL, newxreg=NULL, ...)
stlf(x, h=frequency(x)*2, s.window=7, robust=FALSE,
     method=c("ets", "arima", "naive", "rwdrift"), etsmodel="ZZN",
     level=c(80,95), fan=FALSE, lambda=NULL, xreg=NULL, newxreg=NULL, ...)
```

**Arguments**

object	An object of class "stl". Usually the result of a call to <a href="#">stl</a> .
x	A univariate numeric time series of class "ts"
method	Method to use for forecasting the seasonally adjusted series.
etsmodel	The ets model specification passed to <a href="#">ets</a> . By default it allows any non-seasonal model. If method!="ets", this argument is ignored.
h	Number of periods for forecasting.
level	Confidence level for prediction intervals.
fan	If TRUE, level is set to seq(50,99,by=1). This is suitable for fan plots.
lambda	Box-Cox transformation parameter. Ignored if NULL. Otherwise, data transformed before model is estimated and back-transformed after forecasts are computed.
s.window	Either the character string "periodic" or the span (in lags) of the loess window for seasonal extraction.
robust	If TRUE, robust fitting will used in the loess procedure within <a href="#">stl</a> .
xreg	Historical regressors to be used in <a href="#">auto.arima()</a> .
newxreg	Future regressors to be used in <a href="#">forecast.Arima()</a> .
...	Other arguments passed to <a href="#">ets()</a> or <a href="#">auto.arima()</a> .

**Details**

`forecast.stl` seasonally adjusts the data from an STL decomposition, then uses either ETS or ARIMA models to forecast the result. The seasonal component from the last year of data is added back in to the forecasts. Note that the prediction intervals ignore the uncertainty associated with the seasonal component.

`stlf` takes a `ts` argument and applies a `stl` decomposition before calling `forecast.stl`.

**Value**

An object of class "forecast".

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

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

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

<code>model</code>	A list containing information about the fitted model
<code>method</code>	The name of the forecasting method as a character string
<code>mean</code>	Point forecasts as a time series
<code>lower</code>	Lower limits for prediction intervals
<code>upper</code>	Upper limits for prediction intervals
<code>level</code>	The confidence values associated with the prediction intervals
<code>x</code>	The original time series (either object itself or the time series used to create the model stored as object).
<code>residuals</code>	Residuals from the fitted model. That is (possibly transformed) $x$ minus fitted values.
<code>fitted</code>	Fitted values (one-step forecasts) on transformed scale if <code>lambda</code> is not NULL.

**Author(s)**

Rob J Hyndman

**See Also**

[forecast.ets](#), [forecast.Arima](#).

**Examples**

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

plot(stlf(AirPassengers, lambda=BoxCox.lambda(AirPassengers)))
```

---

forecast.StructTS	<i>Forecasting using Structural Time Series models</i>
-------------------	--

---

## Description

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

## Usage

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

## Arguments

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

## Details

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

## Value

An object of class "forecast".

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

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

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

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

x	The original time series (either object itself or the time series used to create the model stored as object).
residuals	Residuals from the fitted model. That is x minus fitted values.
fitted	Fitted values (one-step forecasts)

**Author(s)**

Rob J Hyndman

**See Also**

[StructTS](#).

**Examples**

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

---

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	<i>Get response variable from time series model.</i>
-------------	--

---

**Description**

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

**Usage**

```
getResponse(object,...)
```

**Arguments**

object	a time series model or forecast object.
...	Additional arguments that are ignored.

**Value**

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

**Author(s)**

Rob J Hyndman

---

gold	<i>Daily morning gold prices</i>
------	----------------------------------

---

**Description**

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

**Usage**

```
data(gold)
```

**Format**

Time series data

**Source**

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

**Examples**

```
tsdisplay(gold)
```

---

`logLik.ets`*Log-Likelihood of an ets object*

---

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

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

**Author(s)**

Rob J Hyndman

**References**

Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) *Forecasting with exponential smoothing: the state space approach*, Springer-Verlag. <http://www.exponentialsMOOTHING.net>.

**See Also**

[ets](#)

**Examples**

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

---

ma	<i>Moving-average smoothing</i>
----	---------------------------------

---

**Description**

Computes a simple moving average smoother.

**Usage**

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

**Arguments**

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

**Value**

Numerical time series object containing the smoothed values.

**Author(s)**

Rob J Hyndman

**See Also**

[ksmooth](#), [decompose](#)

**Examples**

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

---

meanf	<i>Mean Forecast</i>
-------	----------------------

---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

The iid model is

$$Y_t = \mu + Z_t$$

where  $Z_t$  is a normal iid error. Forecasts are given by

$$Y_n(h) = \mu$$

where  $\mu$  is estimated by the sample mean.

**Value**

An object of class "forecast".

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

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

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

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

**Author(s)**

Rob J Hyndman

**See Also**

[rwf](#)



**Examples**

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

---

monthdays	<i>Number of days in each season</i>
-----------	--------------------------------------

---

**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

**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)")
```

msts

*Multi-Seasonal Time Series***Description**

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

**Usage**

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

**Arguments**

data	A numeric vector, ts object, matrix or data frame. It is intended that the time series data is univariate, otherwise treated the same as ts().
seasonal.periods	A vector of the seasonal periods of the msts.
ts.frequency	The seasonal periods that should be used as frequency of the underlying ts object. The default value is max(seasonal.periods).
...	Arguments to be passed to the underlying call to ts(). For example start=c(1987,5).

**Value**

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

**Author(s)**

Slava Razbash and Rob J Hyndman

**Examples**

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

---

na.interp	<i>Interpolate missing values in a time series</i>
-----------	--

---

**Description**

Uses linear interpolation to replace missing values.

**Usage**

```
na.interp(x)
```

**Arguments**

x                      time series

**Details**

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

**Value**

Time series

**Author(s)**

Rob J Hyndman

**Examples**

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

---

naive	<i>Naive forecasts</i>
-------	------------------------

---

**Description**

`naive()` returns forecasts and prediction intervals for an ARIMA(0,1,0) random walk model applied to `x`. `snaive()` returns forecasts and prediction intervals from an ARIMA(0,0,0)(0,1,0)<sub>m</sub> model where `m` is the seasonal period.

**Usage**

```
naive(x, h=10, level=c(80,95), fan=FALSE, lambda=NULL)
snaive(x, h=2*frequency(x), level=c(80,95), fan=FALSE, lambda=NULL)
```

**Arguments**

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

**Details**

These functions are simply convenient wrappers to [Arima](#) with the appropriate arguments to return naive and seasonal naive forecasts.

**Value**

An object of class "forecast".

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

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

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

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

**Author(s)**

Rob J Hyndman

**See Also**

[Arima](#), [rwf](#)

**Examples**

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

---

ndiffs	<i>Number of differences required for a stationary series</i>
--------	---

---

## Description

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

## Usage

```
ndiffs(x, alpha=0.05, test=c("kpss","adf", "pp"))
nsdiffs(x, m=frequency(x), test=c("ocsb","ch"))
```

## Arguments

<code>x</code>	A univariate time series
<code>alpha</code>	Level of the test
<code>m</code>	Length of seasonal period
<code>test</code>	Type of unit root test to use

## Details

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

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

## Value

An integer.

## Author(s)

Rob J Hyndman and Slava Razbash

## References

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

## Examples

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

---

nnetar

---

*Neural Network Time Series Forecasts*


---

## Description

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

## Usage

```
nnetar(x, p, P=1, size, repeats=20, lambda=NULL)
## S3 method for class 'nnetar'
forecast(object, h=ifelse(object$m > 1, 2 * object$m, 10),
         lambda=object$lambda, ...)
```

## Arguments

- |   |   |
|---|---|
| x | a numeric vector or time series   |
| p | Embedding dimension for non-seasonal time series. Number of non-seasonal lags used as inputs. For non-seasonal time series, the default is the optimal number of lags (according to the AIC) for a linear AR(p) model. For seasonal time series, the same method is used but applied to seasonally adjusted data (from an stl decomposition). |

P	Number of seasonal lags used as inputs.
size	Number of nodes in the hidden layer. Default is half of the number of input nodes plus 1.
repeats	Number of networks to fit with different random starting weights. These are then averaged when producing forecasts.
lambda	Box-Cox transformation parameter.
object	An object of class nnetar generated by <a href="#">nnetar</a> .
h	Number of periods for forecasting.
...	Other arguments.

### Details

A feed-forward neural network is fitted with lagged values of  $x$  as inputs and a single hidden layer with  $size$  nodes. The inputs are for lags 1 to  $p$ , and lags  $m$  to  $mP$  where  $m=frequency(x)$ . A total of  $repeats$  networks are fitted, each with random starting weights. These are then averaged when computing forecasts. The network is trained for one-step forecasting. Multi-step forecasts are computed recursively. The fitted model is called an NNAR( $p,P$ ) model and is analogous to an ARIMA( $p,0,0$ )( $P,0,0$ ) model but with nonlinear functions.

### Value

`nnetar` returns an object of class "nnetar". `forecast.nnetar` returns an object of class "forecast".

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

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

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

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

### Author(s)

Rob J Hyndman

### Examples

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

---

`plot.bats`*Plot components from BATS model*

---

### Description

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

### Usage

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

### Arguments

<code>x</code>	Object of class “ets”.
<code>main</code>	Main title for plot.
<code>...</code>	Other plotting parameters passed to <a href="#">par</a> .

### Value

None. Function produces a plot

### Author(s)

Rob J Hyndman

### See Also

[bats](#), [tbats](#)

### Examples

```
fit <- bats(USAccDeaths)
plot(fit)

fit2 <- tbats(USAccDeaths)
plot(fit2)
```



---

plot.ets	<i>Plot components from ETS model</i>
----------	---------------------------------------

---

**Description**

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

**Usage**

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

**Arguments**

x	Object of class “ets”.
...	Other plotting parameters passed to <a href="#">par</a> .

**Value**

None. Function produces a plot

**Author(s)**

Rob J Hyndman

**See Also**

[ets](#)

**Examples**

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

---

plot.forecast	<i>Forecast plot</i>
---------------	----------------------

---

**Description**

Plots historical data with forecasts and prediction intervals.

**Usage**

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

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

**Arguments**

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

**Value**

None.

**Author(s)**

Rob J Hyndman

**References**

Hyndman and Athanasopoulos (2012) *Forecasting: principles and practice*, OTexts: Melbourne, Australia. <http://otexts.com/fpp/>

**See Also**

[plot.ts](#)

**Examples**

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

---

rwf

*Random Walk Forecast*


---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

The random walk with drift model is

$$Y_t = c + Y_{t-1} + Z_t$$

where  $Z_t$  is a normal iid error. Forecasts are given by

$$Y_n(h) = ch + Y_n$$

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

**Value**

An object of class "forecast".

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

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

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

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

**Author(s)**

Rob J Hyndman

**See Also**

[arima](#), [meanf](#)

**Examples**

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

---

seasadj

*Seasonal adjustment*

---

**Description**

Returns seasonally adjusted data constructed by removing the seasonal component.

**Usage**

```
seasadj(object)
```

**Arguments**

object                      Object created by [decompose](#), [stl](#) or [tbats](#).

**Value**

Univariate time series.

**Author(s)**

Rob J Hyndman

**See Also**

[stl](#), [decompose](#), [tbats](#).

**Examples**

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

---

seasonaldummy

*Seasonal dummy variables*


---

**Description**

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

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

**Usage**

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

**Arguments**

$x$                       Seasonal time series  
 $h$                       Number of periods ahead to forecast  
 $K$                       Maximum order of Fourier terms

**Value**

Numerical matrix with number of rows equal to the `length(x)` and number of columns equal to `frequency(x)-1` (for `seasonaldummy` and `seasonaldummyf` or  $2*K$  (for `fourier` or `fourierf`)).

**Author(s)**

Rob J Hyndman

**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(seasonaldummyf(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)

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

```

seasonplot

*Seasonal plot***Description**

Plots a seasonal plot as described in Hyndman and Athanasopoulos (2012, chapter 2).

**Usage**

```

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

```

**Arguments**

<code>x</code>	a numeric vector or time series.
<code>s</code>	seasonal frequency of <code>x</code>
<code>season.labels</code>	Labels for each season in the "year"
<code>year.labels</code>	Logical flag indicating whether labels for each year of data should be plotted on the right.

<code>year.labels.left</code>	Logical flag indicating whether labels for each year of data should be plotted on the left.
<code>type</code>	plot type (as for <code>plot</code> )
<code>main</code>	Main title.
<code>ylab</code>	Y-axis label
<code>xlab</code>	X-axis label
<code>col</code>	Colour
<code>labelgap</code>	Distance between year labels and plotted lines
<code>...</code>	additional arguments to <code>plot</code> .

**Value**

None.

**Author(s)**

Rob J Hyndman

**References**

Hyndman and Athanasopoulos (2012) *Forecasting: principles and practice*, OTexts: Melbourne, Australia. <http://otexts.com/fpp/>

**See Also**

`monthplot`

**Examples**

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

---

ses	<i>Exponential smoothing forecasts</i>
-----	--

---

**Description**

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

**Usage**

```
ses(x, h=10, level=c(80,95), fan=FALSE,
    initial=c("optimal","simple"), alpha=NULL, ...)
holt(x, h=10, damped=FALSE, level=c(80,95), fan=FALSE,
    initial=c("optimal","simple"), exponential=FALSE,
    alpha=NULL, beta=NULL, ...)
hw(x, h=2*frequency(x), seasonal="additive", damped=FALSE,
    level=c(80,95), fan=FALSE, initial=c("optimal","simple"),
    exponential=FALSE, alpha=NULL, beta=NULL, gamma=NULL, ...)
```

### Arguments

<code>x</code>	a numeric vector or time series
<code>h</code>	Number of periods for forecasting.
<code>damped</code>	If TRUE, use a damped trend.
<code>seasonal</code>	Type of seasonality in hw model. "additive" or "multiplicative"
<code>level</code>	Confidence level for prediction intervals.
<code>fan</code>	If TRUE, level is set to seq(50,99,by=1). This is suitable for fan plots.
<code>initial</code>	Method used for selecting initial state values. If optimal, the initial values are optimized along with the smoothing parameters using <a href="#">ets</a> . If simple, the initial values are set to values obtained using simple calculations on the first few observations. See Hyndman & Athanasopoulos (2012) for details.
<code>exponential</code>	If TRUE, an exponential trend is fitted. Otherwise, the trend is (locally) linear.
<code>alpha</code>	Value of smoothing parameter for the level. If NULL, it will be estimated.
<code>beta</code>	Value of smoothing parameter for the trend. If NULL, it will be estimated.
<code>gamma</code>	Value of smoothing parameter for the seasonal component. If NULL, it will be estimated.
<code>...</code>	Other arguments passed to <code>forecast.ets</code> .

### Details

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

### Value

An object of class "forecast".

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

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

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

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



**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. <http://www.exponentialsmoothing.net>.

Hyndman, R.J., Athanasopoulos (2012) *Forecasting: principles and practice*, OTexts: Melbourne, Australia. <http://otexts.com/fpp>.

**See Also**

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

**Arguments**

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

**Details**

With `simulate.Arima()`, the object should be produced by [Arima](#) or [auto.arima](#), rather than [arima](#). By default, the error series is assumed normally distributed and generated using [rnorm](#). If `innov` is present, it is used instead. If `bootstrap=TRUE` and `innov=NULL`, the residuals are resampled instead.

**Value**

An object of class "ts".

**Author(s)**

Rob J Hyndman

**See Also**

[ets](#), [Arima](#), [auto.arima](#), [ar](#), [arfima](#).

**Examples**

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

---

sindexf	<i>Forecast seasonal index</i>
---------	--------------------------------

---

**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 <a href="#">decompose</a> or <a href="#">stl</a> .
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")
```

---

splinef	<i>Cubic Spline Forecast</i>
---------	------------------------------

---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

An object of class "forecast".

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

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

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

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

**Author(s)**

Rob J Hyndman

**References**

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

**See Also**

[smooth.spline](#), [arima](#), [holt](#).

**Examples**

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

---

subset.ts

---

*Subsetting a time series*


---

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

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

**Author(s)**

Rob J Hyndman

**See Also**

[subset](#)

**Examples**

```
plot(subset(gas,month="November"))  
subset(woolyrnq,quarter=3)
```

---

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.

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

---

## Description

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

## Usage

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

## Arguments

<code>y</code>	The time series to be forecast. Can be numeric, <code>msts</code> or <code>ts</code> . Only univariate time series are supported.
<code>use.box.cox</code>	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.
<code>use.trend</code>	TRUE/FALSE indicates whether to include a trend or not. If NULL then both are tried and the best fit is selected by AIC.
<code>use.damped.trend</code>	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.
<code>seasonal.periods</code>	If <code>y</code> is numeric then seasonal periods can be specified with this parameter.
<code>use.arma.errors</code>	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.
<code>use.parallel</code>	TRUE/FALSE indicates whether or not to use parallel processing.
<code>num.cores</code>	The number of parallel processes to be used if using parallel processing. If NULL then the number of logical cores is detected.
<code>bc.lower</code>	The lower limit (inclusive) for the Box-Cox transformation.
<code>bc.upper</code>	The upper limit (inclusive) for the Box-Cox transformation.
<code>...</code>	Additional parameters to be passed to <code>auto.arima</code> when choose an ARMA(p, q) model for the errors.

## 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.

**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**

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

## End(Not run)
```

---

tbats.components

*Extract components of a TBATS model*

---

**Description**

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

**Usage**

```
tbats.components(x)
```

**Arguments**

x                      A tbats object created by [tbats](#).

**Value**

A multiple time series (mts) object.

**Author(s)**

Slava Razbash and Rob J Hyndman

**References**

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**

```
fit <- tbats(USAccDeaths, use.parallel=FALSE)
components <- tbats.components(fit)
plot(components)
```

thetaf

*Theta method forecast***Description**

Returns forecasts and prediction intervals for a theta method forecast.

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

An object of class "forecast".

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

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

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

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

x	The original time series (either object itself or the time series used to create the model stored as object).
residuals	Residuals from the fitted model. That is x minus fitted values.
fitted	Fitted values (one-step forecasts)

**Author(s)**

Rob J Hyndman

**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**

[arima](#), [meanf](#), [rwf](#), [ses](#)

**Examples**

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

---

tsdisplay

*Time series display*


---

**Description**

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

**Usage**

```
tsdisplay(x, plot.type="partial", points=TRUE, ci.type="white",
  lag.max, na.action=na.interp,
  main=NULL, ylab="", xlab="", pch=1, cex=0.5, ...)
```

**Arguments**

x	a numeric vector or time series.
plot.type	type of plot to include in lower right corner. Possible values are "partial", "scatter" or "spectrum".
points	logical flag indicating whether to show the individual points or not in the time plot.
ci.type	type of confidence limits for ACF. Possible values are as for <a href="#">acf</a> .

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	how to handle missing values. Default is to use linear interpolation.
main	Main title.
ylab	Y-axis label
xlab	X-axis label
pch	Plotting character
cex	Character size
...	additional arguments to <a href="#">acf</a> .

**Value**

None.

**Author(s)**

Rob J Hyndman

**References**

Hyndman and Athanasopoulos (2012) *Forecasting: principles and practice*, OTexts: Melbourne, Australia. <http://otexts.com/fpp/>

**See Also**

[plot.ts](#), [acf](#)

**Examples**

```
tsdisplay(diff(WWWusage))
```

---

tslm

*Fit a linear model with time series components*

---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

Returns an object of class "lm".

**Author(s)**

Rob J Hyndman

**See Also**

[forecast.lm](#), [lm](#).

**Examples**

```
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))
```

---

wineind

*Australian total wine sales*

---

**Description**

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

**Usage**

wineind

woolyrnq

69

### Format

Time series data

### Source

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

### 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. <http://data.is/TSDLdemo>

### Examples

```
tsdisplay(woolyrnq)
```

# Index

- \*Topic **datasets**
  - gas, [36](#)
  - gold, [37](#)
  - taylor, [62](#)
  - wineind, [68](#)
  - woolynrq, [69](#)
- \*Topic **hplot**
  - plot.bats, [48](#)
  - plot.ets, [49](#)
- \*Topic **htest**
  - dm.test, [17](#)
- \*Topic **models**
  - CV, [17](#)
- \*Topic **stats**
  - forecast.lm, [31](#)
  - tslm, [67](#)
- \*Topic **ts**
  - accuracy, [3](#)
  - Acf, [4](#)
  - arfima, [5](#)
  - Arima, [6](#)
  - arma.errors, [9](#)
  - auto.arima, [10](#)
  - bats, [12](#)
  - BoxCox, [13](#)
  - BoxCox.lambda, [14](#)
  - croston, [15](#)
  - dm.test, [17](#)
  - dshw, [19](#)
  - ets, [21](#)
  - fitted.Arima, [23](#)
  - forecast, [23](#)
  - forecast.Arima, [25](#)
  - forecast.bats, [27](#)
  - forecast.ets, [28](#)
  - forecast.HoltWinters, [30](#)
  - forecast.stl, [33](#)
  - forecast.StructTS, [35](#)
  - getResponse, [37](#)
  - logLik.ets, [38](#)
  - ma, [39](#)
  - meanf, [39](#)
  - monthdays, [41](#)
  - msts, [42](#)
  - na.interp, [43](#)
  - naive, [43](#)
  - ndiffs, [45](#)
  - nnetar, [46](#)
  - plot.forecast, [49](#)
  - rwf, [51](#)
  - seasadj, [52](#)
  - seasonaldummy, [53](#)
  - seasonplot, [54](#)
  - ses, [55](#)
  - simulate.ets, [57](#)
  - sindexf, [59](#)
  - splinef, [59](#)
  - subset.ts, [61](#)
  - tbats, [63](#)
  - tbats.components, [64](#)
  - thetaf, [65](#)
  - tsdisplay, [66](#)
- accuracy, [3](#)
- Acf, [4](#)
- acf, [4](#), [5](#), [66](#), [67](#)
- AIC, [17](#)
- ar, [25](#), [26](#), [58](#)
- arfima, [5](#), [25](#), [26](#), [58](#)
- Arima, [6](#), [11](#), [26](#), [44](#), [58](#)
- arma, [5](#), [6](#), [8](#), [9](#), [11](#), [22](#), [23](#), [25](#), [26](#), [52](#), [53](#), [57](#), [58](#), [61](#), [66](#)
- arma.errors, [9](#)
- auto.arima, [5](#), [6](#), [10](#), [25](#), [26](#), [33](#), [46](#), [58](#)
- bats, [12](#), [27](#), [28](#), [48](#)
- best.arima (auto.arima), [10](#)
- BoxCox, [13](#), [15](#)
- BoxCox.lambda, [14](#), [14](#)

- croston, 15, 24
- CV, 17
- decompose, 39, 53, 59
- dm.test, 17
- dshw, 19
- ets, 19, 20, 21, 23, 28–30, 33, 38, 49, 56–58
- fitted.Arima, 23
- forecast, 23, 50
- forecast.ar (forecast.Arima), 25
- forecast.Arima, 23, 24, 25, 33, 34
- forecast.bats, 27
- forecast.ets, 24, 28, 28, 34
- forecast.fracdiff, 6, 25
- forecast.fracdiff (forecast.Arima), 25
- forecast.HoltWinters, 24, 30
- forecast.lm, 31, 68
- forecast.nnetar (nnetar), 46
- forecast.stl, 33
- forecast.StructTS, 24, 35
- forecast.tbats (forecast.bats), 27
- forecast.ts, 23
- fourier (seasonaldummy), 53
- fourierf (seasonaldummy), 53
- fracdiff, 6, 25
- gas, 36
- getResponse, 37
- gold, 37
- holt, 24, 61
- holt (ses), 55
- HoltWinters, 20, 22, 30, 31, 57
- hw, 24
- hw (ses), 55
- InvBoxCox (BoxCox), 13
- ksmooth, 39
- lm, 17, 31, 32, 53, 68
- logLik.ets, 38
- ma, 39
- meanf, 24, 39, 52, 66
- monthdays, 41
- monthplot, 55
- msts, 19, 42
- na.interp, 43
- naive, 43
- ndiffs, 11, 45
- nnetar, 46, 47
- nsdiffs, 11
- nsdiffs (ndiffs), 45
- Pacf (Acf), 4
- par, 48, 49
- plot, 50, 55
- plot.bats, 48
- plot.default, 50
- plot.ets, 49
- plot.forecast, 49
- plot.splineforecast (plot.forecast), 49
- plot.tbats (plot.bats), 48
- plot.ts, 51, 67
- predict.ar, 25, 26
- predict.Arima, 25, 26
- predict.HoltWinters, 30, 31
- predict.lm, 31, 32
- print.forecast (forecast), 23
- residuals, 9
- rnorm, 58
- rwf, 22, 24, 40, 44, 51, 57, 66
- seasadj, 52
- seasonaldummy, 53
- seasonaldummyf (seasonaldummy), 53
- seasonplot, 54
- ses, 16, 24, 55, 66
- set.seed, 58
- simulate.ar (simulate.ets), 57
- simulate.Arima (simulate.ets), 57
- simulate.ets, 57
- simulate.fracdiff (simulate.ets), 57
- sindexf, 59
- smooth.spline, 60, 61
- snaive (naive), 43
- splinef, 24, 59
- stl, 33, 53, 59
- stl (forecast.stl), 33
- stlf, 23, 24
- stlf (forecast.stl), 33
- StructTS, 35, 36
- subset, 62
- subset.ts, 61
- summary.forecast (forecast), 23

taylor, [62](#)  
tbats, [28](#), [48](#), [53](#), [63](#), [64](#)  
tbats.components, [64](#)  
thetaf, [24](#), [65](#)  
tsdisplay, [66](#)  
tslm, [17](#), [31](#), [32](#), [53](#), [67](#)  
  
wineind, [68](#)  
woolynrq, [69](#)