Package 'smooth'

July 1, 2025

Type Package

Title Forecasting Using State Space Models

Version 4.3.0

Date 2025-06-30

URL https://github.com/config-i1/smooth

BugReports https://github.com/config-i1/smooth/issues

Language en-GB

Description Functions implementing Single Source of Error state space models for purposes of time series analysis and forecasting.
The package includes ADAM (Svetunkov, 2023, <https://openforecast.org/adam/>), Exponential Smoothing (Hyndman et al., 2008, <doi:10.1007/978-3-540-71918-2>), SARIMA (Svetunkov & Boylan, 2019 <doi:10.1080/00207543.2019.1600764>), Complex Exponential Smoothing (Svetunkov & Kourentzes, 2018, <doi:10.13140/RG.2.2.24986.29123>), Simple Moving Average (Svetunkov & Petropoulos, 2018 <doi:10.1080/00207543.2017.1380326>) and several simulation functions. It also allows dealing with intermittent demand based on the iETS framework (Svetunkov & Boylan, 2019, <doi:10.13140/RG.2.2.35897.06242>).

License LGPL-2.1

Depends R (>= 3.0.2), greybox (>= 2.0.2)

Imports Rcpp (>= 0.12.3), stats, generics (>= 0.1.2), graphics, grDevices, pracma, statmod, MASS, nloptr, utils, xtable, zoo

LinkingTo Rcpp, RcppArmadillo (>= 0.8.100.0.0)

Suggests legion, numDeriv, testthat, knitr, rmarkdown, doMC, doParallel, foreach

VignetteBuilder knitr

RoxygenNote 7.3.2

Encoding UTF-8

ByteCompile true

NeedsCompilation yes

Author Ivan Svetunkov [aut, cre] (Senior Lecturer at Centre for Marketing Analytics and Forecasting, Lancaster University, UK)

Maintainer Ivan Svetunkov <ivan@svetunkov.com>

Repository CRAN

Date/Publication 2025-07-01 10:50:02 UTC

Contents

`accuracy.smooth` *Error measures for an estimated model*

Description

Function produces error measures for the provided object and the holdout values of the response variable. Note that instead of parameters `x`, `test`, the function accepts the vector of values in `holdout`. Also, the parameters `d` and `D` are not supported - MASE is always calculated via division by first differences.

Usage

```
## S3 method for class 'smooth'
accuracy(object, holdout = NULL, ...)

## S3 method for class 'smooth.forecast'
accuracy(object, holdout = NULL, ...)
```

Arguments

`object`	The estimated model or a forecast from the estimated model generated via either `predict()` or `forecast()` functions.
`holdout`	The vector of values of the response variable in the holdout (test) set. If not provided, then the function will return the in-sample error measures. If the `holdout=TRUE` parameter was used in the estimation of a model, the holdout values will be extracted automatically.
`...`	Other variables passed to the `forecast()` function (e.g. `newdata`).

Details

The function is a wrapper for the measures function and is implemented for convenience.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

Examples

```
y <- rnorm(100, 100, 10)
ourModel <- adam(y, holdout=TRUE, h=10)
accuracy(ourModel)
```

adam *ADAM is Augmented Dynamic Adaptive Model*

Description

Function constructs an advanced Single Source of Error model, based on ETS taxonomy and ARIMA elements

Usage

```
adam(data, model = "ZXZ", lags = c(frequency(data)), orders = list(ar =
  c(0), i = c(0), ma = c(0), select = FALSE), constant = FALSE,
  formula = NULL, regressors = c("use", "select", "adapt"),
  occurrence = c("none", "auto", "fixed", "general", "odds-ratio",
  "inverse-odds-ratio", "direct"), distribution = c("default", "dnorm",
  "dlaplace", "ds", "dgnorm", "dlnorm", "dinvgauss", "dgamma"),
  loss = c("likelihood", "MSE", "MAE", "HAM", "LASSO", "RIDGE", "MSEh",
  "TMSE", "GTMSE", "MSCE"), outliers = c("ignore", "use", "select"),
  level = 0.99, h = 0, holdout = FALSE, persistence = NULL,
  phi = NULL, initial = c("backcasting", "optimal", "two-stage",
  "complete"), arma = NULL, ic = c("AICc", "AIC", "BIC", "BICc"),
  bounds = c("usual", "admissible", "none"), silent = TRUE, ...)

## S3 method for class 'adam'
simulate(object, nsim = 1, seed = NULL,
  obs = nobs(object), ...)

auto.adam(data, model = "ZXZ", lags = c(frequency(data)),
  orders = list(ar = c(3, 3), i = c(2, 1), ma = c(3, 3), select = TRUE),
  formula = NULL, regressors = c("use", "select", "adapt"),
  occurrence = c("none", "auto", "fixed", "general", "odds-ratio",
  "inverse-odds-ratio", "direct"), distribution = c("dnorm", "dlaplace",
  "ds", "dgnorm", "dlnorm", "dinvgauss", "dgamma"), outliers = c("ignore",
  "use", "select"), level = 0.99, h = 0, holdout = FALSE,
  persistence = NULL, phi = NULL, initial = c("backcasting", "optimal",
  "two-stage", "complete"), arma = NULL, ic = c("AICc", "AIC", "BIC",
  "BICc"), bounds = c("usual", "admissible", "none"), silent = TRUE,
  parallel = FALSE, ...)

## S3 method for class 'adam'
sm(object, model = "YYY", lags = NULL, orders = list(ar =
  c(0), i = c(0), ma = c(0), select = FALSE), constant = FALSE,
  formula = NULL, regressors = c("use", "select", "adapt"), data = NULL,
  persistence = NULL, phi = NULL, initial = c("optimal", "backcasting"),
  arma = NULL, ic = c("AICc", "AIC", "BIC", "BICc"), bounds = c("usual",
  "admissible", "none"), silent = TRUE, ...)
```

Arguments

`data` Vector, containing data needed to be forecasted. If a matrix (or data.frame / data.table) is provided, then the first column is used as a response variable, while the rest of the matrix is used as a set of explanatory variables. `formula` can be used in the latter case in order to define what relation to have.

`model` The type of ETS model. The first letter stands for the type of the error term ("A" or "M"), the second (and sometimes the third as well) is for the trend ("N", "A", "Ad", "M" or "Md"), and the last one is for the type of seasonality ("N", "A" or "M"). In case of several lags, the seasonal components are assumed to be the same. The model is then printed out as ETS(M,Ad,M)[m1,m2,...], where m1, m2, ... are the lags specified by the `lags` parameter. There are several options for the `model` besides the conventional ones, which rely on information criteria:

1. `model="ZZZ"` means that the model will be selected based on the chosen information criteria type. The Branch and Bound is used in the process.
2. `model="XXX"` means that only additive components are tested, using Branch and Bound.
3. `model="YYY"` implies selecting between multiplicative components.
4. `model="CCC"` triggers the combination of forecasts of models using information criteria weights (Kolassa, 2011).
5. combinations between these four and the classical components are also accepted. For example, `model="CAY"` will combine models with additive trend and either none or multiplicative seasonality.
6. `model="PPP"` will produce the selection between pure additive and pure multiplicative models. "P" stands for "Pure". This cannot be mixed with other types of components.
7. `model="FFF"` will select between all the 30 types of models. "F" stands for "Full". This cannot be mixed with other types of components.
8. `model="SSS"` creates a pool of 19 standard sensible models, which have finite variance. This can be combined with "X" or "Y" to further restrict the pool. For example, `model="SXS"` corresponds to the default pool of 15 models in the `ets()` function from the forecast package.
9. The parameter `model` can also be a vector of names of models for a finer tuning (pool of models). For example, `model=c("ANN","AAA")` will estimate only two models and select the best of them.

Also, `model` can accept a previously estimated adam and use all its parameters.

Keep in mind that model selection with "Z" components uses Branch and Bound algorithm and may skip some models that could have slightly smaller information criteria. If you want to do a exhaustive search, you would need to list all the models to check as a vector.

The default value is set to `"ZXZ"`, because the multiplicative trend is explosive and dangerous. It should be used only for each separate time series, not for the automated predictions for big datasets.

`lags` Defines lags for the corresponding components. All components count, starting from level, so ETS(M,M,M) model for monthly data will have `lags=c(1,1,12)`. However, the function will also accept `lags=c(12)`, assuming that the lags 1

were dropped. In case of ARIMA, lags specify what should be the seasonal component lag. e.g. `lags=c(1,12)` will lead to the seasonal ARIMA with m=12. This can accept several lags, supporting multiple seasonal ETS and ARIMA models.

`orders` The order of ARIMA to be included in the model. This should be passed either as a vector (in which case the non-seasonal ARIMA is assumed) or as a list of a type `orders=list(ar=c(p,P),i=c(d,D),ma=c(q,Q))`, in which case the `lags` variable is used in order to determine the seasonality m. See msarima for details. In addition, `orders` accepts one more parameter: `orders=list(select=FALSE)`. If `TRUE`, then the function will select the most appropriate order using a mechanism similar to `auto.msarima()`, but implemented in `auto.adam()`. The values `list(ar=...,i=...,ma=...)` specify the maximum orders to check in this case.

`constant` Logical, determining, whether the constant is needed in the model or not. This is mainly needed for ARIMA part of the model, but can be used for ETS as well. In case of pure regression, this is completely ignored (use `formula` instead).

`formula` Formula to use in case of explanatory variables. If `NULL`, then all the variables are used as is. Can also include `trend`, which would add the global trend. Only needed if `data` is a matrix or if `trend` is provided.

`regressors` The variable defines what to do with the provided explanatory variables: `"use"` means that all of the data should be used, while `"select"` means that a selection using `ic` should be done, `"adapt"` will trigger the mechanism of time varying parameters for the explanatory variables.

`occurrence` The type of model used in probability estimation. Can be `"none"` - none, `"fixed"` - constant probability, `"general"` - the general Beta model with two parameters, `"odds-ratio"` - the Odds-ratio model with b=1 in Beta distribution, `"inverse-odds-ratio"` - the model with a=1 in Beta distribution, `"direct"` - the TSB-like (Teunter et al., 2011) probability update mechanism a+b=1, `"auto"` - the automatically selected type of occurrence model.

The type of model used in the occurrence is equal to the one provided in the `model` parameter.

Also, a model produced using oes or alm function can be used here.

`distribution` what density function to assume for the error term. The full name of the distribution should be provided, starting with the letter "d" - "density". The names align with the names of distribution functions in R. For example, see dnorm. For detailed explanation of available distributions, see vignette in greybox package: `vignette("greybox","alm")`.

`loss` The type of Loss Function used in optimization. `loss` can be:

- `likelihood` - the model is estimated via the maximisation of the likelihood of the function specified in `distribution`;
- `MSE` (Mean Squared Error),
- `MAE` (Mean Absolute Error),
- `HAM` (Half Absolute Moment),
- `LASSO` - use LASSO to shrink the parameters of the model;
- `RIDGE` - use RIDGE to shrink the parameters of the model;

- TMSE - Trace Mean Squared Error,
- GTMSE - Geometric Trace Mean Squared Error,
- MSEh - optimisation using only h-steps ahead error,
- MSCE - Mean Squared Cumulative Error.

In case of LASSO / RIDGE, the variables are not normalised prior to the estimation, but the parameters are divided by the mean values of explanatory variables.

Note that model selection and combination works properly only for the default loss="likelihood".

Furthermore, just for fun the absolute and half analogues of multistep estimators are available: MAEh, TMAE, GTMAE, MACE, HAMh, THAM, GTHAM, CHAM.

Last but not least, user can provide their own function here as well, making sure that it accepts parameters actual, fitted and B. Here is an example:

lossFunction <- function(actual, fitted, B) return(mean(abs(actual-fitted)))

loss=lossFunction

outliers — Defines what to do with outliers: "ignore", so just returning the model, "use" - detect outliers based on specified level and include dummies for them in the model, or detect and "select" those of them that reduce ic value.

level — What confidence level to use for detection of outliers. The default is 99%. The specific bounds of confidence interval depend on the distribution used in the model.

h — The forecast horizon. Mainly needed for the multistep loss functions.

holdout — Logical. If TRUE, then the holdout of the size h is taken from the data (can be used for the model testing purposes).

persistence — Persistence vector g, containing smoothing parameters. If NULL, then estimated. Can be also passed as a names list of the type: persistence=list(level=0.1, trend=0.05, seasonal=c(0.1,0.2),xreg=c(0.1,0.2)). Dropping some elements from the named list will make the function estimate them. e.g. if you don't specify seasonal in the persistence for the ETS(M,N,M) model, it will be estimated.

phi — Value of damping parameter. If NULL then it is estimated. Only applicable for damped-trend models.

initial — Can be either character or a list, or a vector of initial states. If it is character, then it can be "backcasting", meaning that the initials of dynamic part of the model are produced using backcasting procedure (advised for data with high frequency), or "optimal", meaning that all initial states are optimised, or "two-stage", meaning that optimisation is done after the backcasting, refining the states. In case of backcasting, the parameters of the explanatory variables are optimised. This is recommended for ETSX and ARIMAX models. Alternatively, you can set initial="complete" backcasting, which means that all states (including explanatory variables) are initialised via backcasting.

If a use provides a list of values, it is recommended to use the named one and to provide the initial components that are available. For example: initial=list(level=1000,trend=10,s
If some of the components are needed by the model, but are not provided in the list, they will be estimated. If the vector is provided, then it is expected that the components will be provided in the same order as above, one after another without any gaps.

arma — Either the named list or a vector with AR / MA parameters ordered lag-wise. The number of elements should correspond to the specified orders e.g. `orders=list(ar=c(1,1),ma=c(1,1)) lags=c(1,4), arma=list(ar=c(0.9,0.8),ma=c(-0.3,0.3))`

ic — The information criterion to use in the model selection / combination procedure.

bounds — The type of bounds for the persistence to use in the model estimation. Can be either `admissible` - guaranteeing the stability of the model, `usual` - restricting the values with (0, 1) or `none` - no restrictions (potentially dangerous).

silent — Specifies, whether to provide the progress of the function or not. If `TRUE`, then the function will print what it does and how much it has already done.

... — Other non-documented parameters. For example, `FI=TRUE` will make the function also produce Fisher Information matrix, which then can be used to calculated variances of smoothing parameters and initial states of the model. This is calculated based on the hessian of log-likelihood function and accepts `stepSize` parameter, determining how it is calculated. The default value is `stepSize=.Machine$double.eps^(1/4` This is used in the vcov method. Number of iterations inside the backcasting loop to do is regulated with `nIterations` parameter. By default it is set to 2. Furthermore, starting values of parameters can be passed via `B`, while the upper and lower bounds should be passed in `ub` and `lb` respectively. In this case they will be used for optimisation. These values should have the length equal to the number of parameters to estimate in the following order:

1. All smoothing parameters (for the states and then for the explanatory variables);
2. Damping parameter (if needed);
3. ARMA parameters;
4. All the initial values (for the states and then for the explanatory variables).

You can also pass parameters to the optimiser in order to fine tune its work:

- `maxeval` - maximum number of evaluations to carry out. The default is 40 per estimated parameter for ETS and / or ARIMA and at least 1000 if explanatory variables are introduced in the model (100 per parameter for explanatory variables, but not less than 1000);
- `maxtime` - stop, when the optimisation time (in seconds) exceeds this;
- `xtol_rel` - the relative precision of the optimiser (the default is 1E-6);
- `xtol_abs` - the absolute precision of the optimiser (the default is 1E-8);
- `ftol_rel` - the stopping criterion in case of the relative change in the loss function (the default is 1E-8);
- `ftol_abs` - the stopping criterion in case of the absolute change in the loss function (the default is 0 - not used);
- `algorithm` - the algorithm to use in optimisation (by default, `"NLOPT_LN_NELDERMEAD"` is used);
- `print_level` - the level of output for the optimiser (0 by default). If equal to 41, then the detailed results of the optimisation are returned.

You can read more about these parameters by running the function nloptr.print.options. It is also possible to regulate what smoother to use to get initial seasonal indices from the msdecompose function via the `smoother` parameter. Finally, the parameter `lambda` for LASSO / RIDGE, `alpha` for the Asymmetric Laplace,

shape for the Generalised Normal and nu for Student's distributions can be provided here as well.

`object` The model previously estimated using `adam()` function.

`nsim` Number of series to generate from the model.

`seed` Random seed used in simulation of data.

`obs` Number of observations to produce in the simulated data.

`parallel` If TRUE, the estimation of ADAM models is done in parallel (used in `auto.adam` only). If the number is provided (e.g. `parallel=41`), then the specified number of cores is set up. WARNING! Packages `foreach` and either `doMC` (Linux and Mac only) or `doParallel` are needed in order to run the function in parallel.

Details

Function estimates ADAM in a form of the Single Source of Error state space model of the following type:

$$y_t = o_t(w(v_{t-l}) + h(x_t, a_{t-1}) + r(v_{t-l})\epsilon_t)$$

$$v_t = f(v_{t-l}, a_{t-1}) + g(v_{t-l}, a_{t-1}, x_t)\epsilon_t$$

Where o_t is the Bernoulli distributed random variable (in case of normal data it equals to 1 for all observations), v_t is the state vector and l is the vector of lags, x_t is the vector of exogenous variables. w(.) is the measurement function, r(.) is the error function, f(.) is the transition function, g(.) is the persistence function and a_t is the vector of parameters for exogenous variables. Finally, ϵ_t is the error term.

The implemented model allows introducing several seasonal states and supports intermittent data via the `occurrence` variable.

The error term ϵ_t can follow different distributions, which are regulated via the `distribution` parameter. This includes:

1. `default` - Normal distribution is used for the Additive error models, Gamma is used for the Multiplicative error models.
2. dnorm - Normal distribution,
3. dlaplace - Laplace distribution,
4. ds - S distribution,
5. dgnorm - Generalised Normal distribution,
6. dlnorm - Log-Normal distribution,
7. dgamma - Gamma distribution,
8. dinvgauss - Inverse Gaussian distribution,

For some more information about the model and its implementation, see the vignette: `vignette("adam","smooth")`. The more detailed explanation of ADAM is provided by Svetunkov (2021).

The function `auto.adam()` tries out models with the specified distributions and returns the one with the most suitable one based on selected information criterion.

sm.adam method estimates the scale model for the already estimated adam. In order for ADAM to take the SM model into account, the latter needs to be recorded in the former, amending the likelihood and the number of degrees of freedom. This can be done using implant method.

Value

Object of class "adam" is returned. It contains the list of the following values:

- `model` - the name of the constructed model,
- `timeElapsed` - the time elapsed for the estimation of the model,
- `data` - the in-sample part of the data used for the training of the model. Includes the actual values in the first column,
- `holdout` - the holdout part of the data, excluded for purposes of model evaluation,
- `fitted` - the vector of fitted values,
- `residuals` - the vector of residuals,
- `forecast` - the point forecast for h steps ahead (by default NA is returned). NOTE that these do not always correspond to the conditional expectations for ETS models. See ADAM textbook, Section 6.4. for details (`https://openforecast.org/adam/ETSTaxonomyMaths.html`),
- `states` - the matrix of states with observations in rows and states in columns,
- `persisten` - the vector of smoothing parameters,
- `phi` - the value of damping parameter,
- `transition` - the transition matrix,
- `measurement` - the measurement matrix with observations in rows and state elements in columns,
- `initial` - the named list of initial values, including level, trend, seasonal, ARIMA and xreg components,
- `initialEstimated` - the named vector, defining which of the initials were estimated in the model,
- `initialType` - the type of initialisation used (backcasting/optimal/two-stage/complete/provided),
- `orders` - the orders of ARIMA used in the estimation,
- `constant` - the value of the constant (if it was included),
- `arma` - the list of AR / MA parameters used in the model,
- `nParam` - the matrix of the estimated / provided parameters,
- `occurrence` - the oes model used for the occurrence part of the model,
- `formula` - the formula used for the explanatory variables expansion,
- `loss` - the type of loss function used in the estimation,
- `lossValue` - the value of that loss function,
- `logLik` - the value of the log-likelihood,
- `distribution` - the distribution function used in the calculation of the likelihood,
- `scale` - the value of the scale parameter,

- `lambda` - the value of the parameter used in LASSO / dalaplace / dt,
- `B` - the vector of all estimated parameters,
- `lags` - the vector of lags used in the model construction,
- `lagsAll` - the vector of the internal lags used in the model,
- `profile` - the matrix with the profile used in the construction of the model,
- `profileInitial` - the matrix with the initial profile (for the before the sample values),
- `call` - the call used in the evaluation,
- `bounds` - the type of bounds used in the process,
- `res` - result of the model estimation, the output of the `nloptr()` function, explaining how optimisation went,
- `other` - the list with other parameters, such as shape for distributions or ARIMA polynomials.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov I. (2023) Smooth forecasting with the smooth package in R. arXiv:2301.01790. doi:10.48550/arXiv.2301.01790.
- Svetunkov I. (2015 - Inf) "smooth" package for R - series of posts about the underlying models and how to use them: https://openforecast.org/category/r-en/smooth/.
- Svetunkov, I. (2023). Forecasting and Analytics with the Augmented Dynamic Adaptive Model (ADAM) (1st ed.). Chapman and Hall/CRC. doi:10.1201/9781003452652, online version: https://openforecast.org/adam/.
- Svetunkov, I., 2023. Smooth Forecasting with the Smooth Package in R. arXiv. doi:10.48550/arXiv.2301.01790
- Snyder, R. D., 1985. Recursive Estimation of Dynamic Linear Models. Journal of the Royal Statistical Society, Series B (Methodological) 47 (2), 272-276.
- Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) Forecasting with exponential smoothing: the state space approach, Springer-Verlag. doi:10.1007/9783540719182.
- Svetunkov, I., Boylan, J.E., 2023a. iETS: State Space Model for Intermittent Demand Forecastings. International Journal of Production Economics. 109013. doi:10.1016/j.ijpe.2023.109013
- Teunter R., Syntetos A., Babai Z. (2011). Intermittent demand: Linking forecasting to inventory obsolescence. European Journal of Operational Research, 214, 606-615.
- Croston, J. (1972) Forecasting and stock control for intermittent demands. Operational Research Quarterly, 23(3), 289-303.
- Kolassa, S. (2011) Combining exponential smoothing forecasts using Akaike weights. International Journal of Forecasting, 27, pp 238 - 251.
- Svetunkov, I., Boylan, J.E., 2023b. Staying Positive: Challenges and Solutions in Using Pure Multiplicative ETS Models. IMA Journal of Management Mathematics. p. 403-425. doi:10.1093/imaman/dpad028

- Taylor, J.W. and Bunn, D.W. (1999) A Quantile Regression Approach to Generating Prediction Intervals. Management Science, Vol 45, No 2, pp 225-237.
- Lichtendahl Kenneth C., Jr., Grushka-Cockayne Yael, Winkler Robert L., (2013) Is It Better to Average Probabilities or Quantiles? Management Science 59(7):1594-1611. DOI: doi:10.1287/mnsc.1120.1667

See Also

es, msarima

Examples

```
### The main examples are provided in the adam vignette, check it out via:
## Not run: vignette("adam","smooth")

# Model selection using a specified pool of models
ourModel <- adam(rnorm(100,100,10), model=c("ANN","ANA","AAA"), lags=c(5,10))
adamSummary <- summary(ourModel)
xtable(adamSummary)

forecast(ourModel)
par(mfcol=c(3,4))
plot(ourModel, c(1:11))

# Model combination using a specified pool
ourModel <- adam(rnorm(100,100,10), model=c("ANN","AAN","MNN","CCC"),
                         lags=c(5,10))

# ADAM ARIMA
ourModel <- adam(rnorm(100,100,10), model="NNN",
                         lags=c(1,4), orders=list(ar=c(1,0),i=c(1,0),ma=c(1,1)))

# Fit ADAM to the data
ourModel <- adam(rnorm(100,100,10), model="AAdN")
# Simulate the data
x <- simulate(ourModel)

# Automatic selection of appropriate distribution and orders of ADAM ETS+ARIMA
ourModel <- auto.adam(rnorm(100,100,10), model="ZZN", lags=c(1,4),
                      orders=list(ar=c(2,2),ma=c(2,2),select=TRUE))
```

---

ces *Complex Exponential Smoothing*

Description

Function estimates CES in state space form with information potential equal to errors and returns several variables.

Usage

```
ces(y, seasonality = c("none", "simple", "partial", "full"),
  lags = c(frequency(y)), initial = c("backcasting", "optimal",
  "two-stage", "complete"), a = NULL, b = NULL, loss = c("likelihood",
  "MSE", "MAE", "HAM", "MSEh", "TMSE", "GTMSE", "MSCE"), h = 0,
  holdout = FALSE, bounds = c("admissible", "none"), silent = TRUE,
  model = NULL, xreg = NULL, regressors = c("use", "select", "adapt"),
  initialX = NULL, ...)

auto.ces(y, seasonality = c("none", "simple", "partial", "full"),
  lags = c(frequency(y)), initial = c("backcasting", "optimal",
  "two-stage", "complete"), ic = c("AICc", "AIC", "BIC", "BICc"),
  loss = c("likelihood", "MSE", "MAE", "HAM", "MSEh", "TMSE", "GTMSE",
  "MSCE"), h = 0, holdout = FALSE, bounds = c("admissible", "none"),
  silent = TRUE, xreg = NULL, regressors = c("use", "select", "adapt"),
  ...)

ces_old(y, seasonality = c("none", "simple", "partial", "full"),
  initial = c("backcasting", "optimal"), a = NULL, b = NULL,
  ic = c("AICc", "AIC", "BIC", "BICc"), loss = c("likelihood", "MSE",
  "MAE", "HAM", "MSEh", "TMSE", "GTMSE", "MSCE"), h = 10, holdout = FALSE,
  bounds = c("admissible", "none"), silent = c("all", "graph", "legend",
  "output", "none"), xreg = NULL, regressors = c("use", "select"),
  initialX = NULL, ...)
```

Arguments

y	Vector or ts object, containing data needed to be forecasted.
seasonality	The type of seasonality used in CES. Can be: none - No seasonality; simple - Simple seasonality, using lagged CES (based on t-m observation, where m is the seasonality lag); partial - Partial seasonality with the real seasonal component (equivalent to additive seasonality); full - Full seasonality with complex seasonal component (can do both multiplicative and additive seasonality, depending on the data). First letter can be used instead of full words. In case of the auto.ces() function, this parameter defines which models to try.
lags	Vector of lags to use in the model. Allows defining multiple seasonal models.
initial	Can be either character or a list, or a vector of initial states. If it is character, then it can be "backcasting", meaning that the initials of dynamic part of the model are produced using backcasting procedure (advised for data with high frequency), or "optimal", meaning that all initial states are optimised, or "two-stage", meaning that optimisation is done after the backcasting, refining the states. In case of backcasting, the parameters of the explanatory variables are optimised. Alternatively, you can set initial="complete" backcasting, which means that all states (including explanatory variables) are initialised via backcasting.
a	First complex smoothing parameter. Should be a complex number.

NOTE! CES is very sensitive to a and b values so it is advised either to leave them alone, or to use values from previously estimated model.

`b` Second complex smoothing parameter. Can be real if `seasonality="partial"`. In case of `seasonality="full"` must be complex number.

`loss` The type of Loss Function used in optimization. `loss` can be: `likelihood` (assuming Normal distribution of error term), `MSE` (Mean Squared Error), `MAE` (Mean Absolute Error), `HAM` (Half Absolute Moment), `TMSE` - Trace Mean Squared Error, `GTMSE` - Geometric Trace Mean Squared Error, `MSEh` - optimisation using only h-steps ahead error, `MSCE` - Mean Squared Cumulative Error. If `loss!="MSE"`, then likelihood and model selection is done based on equivalent `MSE`. Model selection in this cases becomes not optimal.

There are also available analytical approximations for multistep functions: `aMSEh`, `aTMSE` and `aGTMSE`. These can be useful in cases of small samples.

Finally, just for fun the absolute and half analogues of multistep estimators are available: `MAEh`, `TMAE`, `GTMAE`, `MACE`, `TMAE`, `HAMh`, `THAM`, `GTHAM`, `CHAM`.

`h` Length of forecasting horizon.

`holdout` If `TRUE`, holdout sample of size `h` is taken from the end of the data.

`bounds` The type of bounds for the persistence to use in the model estimation. Can be either `admissible` - guaranteeing the stability of the model, or `none` - no restrictions (potentially dangerous).

`silent` accepts `TRUE` and `FALSE`. If FALSE, the function will print its progress and produce a plot at the end.

`model` A previously estimated GUM model, if provided, the function will not estimate anything and will use all its parameters.

`xreg` The vector (either numeric or time series) or the matrix (or data.frame) of exogenous variables that should be included in the model. If matrix included than columns should contain variables and rows - observations. Note that `xreg` should have number of observations equal either to in-sample or to the whole series. If the number of observations in `xreg` is equal to in-sample, then values for the holdout sample are produced using es function.

`regressors` The variable defines what to do with the provided xreg: `"use"` means that all of the data should be used, while `"select"` means that a selection using `ic` should be done.

`initialX` The vector of initial parameters for exogenous variables. Ignored if `xreg` is NULL.

`...` Other non-documented parameters. See adam for details. However, there are several unique parameters passed to the optimiser in comparison with `adam`: 1. `algorithm0`, which defines what algorithm to use in nloptr for the initial optimisation. By default, this is "NLOPT_LN_BOBYQA". 2. `algorithm` determines the second optimiser. By default this is "NLOPT_LN_NELDERMEAD". 3. maxeval0 and maxeval, that determine the number of iterations for the two optimisers. By default, `maxeval0=maxeval=40*k`, where k is the number of estimated parameters. 4. xtol_rel0 and xtol_rel, which are 1e-8 and 1e-6 respectively. There are also ftol_rel0, ftol_rel, ftol_abs0 and ftol_abs, which by default are set to values explained in the `nloptr.print.options()` function.

`ic` The information criterion to use in the model selection.

Details

The function estimates Complex Exponential Smoothing in the state space form described in Svetunkov et al. (2022) with the information potential equal to the approximation error.

The `auto.ces()` function implements the automatic seasonal component selection based on information criteria.

`ces_old()` is the old implementation of the model and will be discontinued starting from smooth v4.5.0.

`ces()` uses two optimisers to get good estimates of parameters. By default these are BOBYQA and then Nelder-Mead. This can be regulated via `...` - see details below.

For some more information about the model and its implementation, see the vignette: `vignette("ces","smooth")`

Value

Object of class "adam" is returned with similar elements to the adam function.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov I. (2023) Smooth forecasting with the smooth package in R. arXiv:2301.01790. doi:10.48550/arXiv.2301.01790.
- Svetunkov I. (2015 - Inf) "smooth" package for R - series of posts about the underlying models and how to use them: https://openforecast.org/category/r-en/smooth/.
- Svetunkov, I. (2023). Forecasting and Analytics with the Augmented Dynamic Adaptive Model (ADAM) (1st ed.). Chapman and Hall/CRC. doi:10.1201/9781003452652, online version: https://openforecast.org/adam/.
- Svetunkov, I., 2023. Smooth Forecasting with the Smooth Package in R. arXiv. doi:10.48550/arXiv.2301.01790
- Snyder, R. D., 1985. Recursive Estimation of Dynamic Linear Models. Journal of the Royal Statistical Society, Series B (Methodological) 47 (2), 272-276.
- Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) Forecasting with exponential smoothing: the state space approach, Springer-Verlag. doi:10.1007/9783540719182.
- Svetunkov, I., Kourentzes, N., & Ord, J. K. (2022). Complex exponential smoothing. Naval Research Logistics, 69(8), 1108–1123. https://doi.org/10.1002/nav.22074

See Also

adam, es

Examples

```
y <- rnorm(100,10,3)
ces(y, h=20, holdout=FALSE)

y <- 500 - c(1:100)*0.5 + rnorm(100,10,3)
ces(y, h=20, holdout=TRUE)

ces(BJsales, h=8, holdout=TRUE)

ces(AirPassengers, h=18, holdout=TRUE, seasonality="s")
ces(AirPassengers, h=18, holdout=TRUE, seasonality="p")
ces(AirPassengers, h=18, holdout=TRUE, seasonality="f")

y <- ts(rnorm(100,10,3),frequency=12)
# CES with and without holdout
auto.ces(y,h=20,holdout=TRUE)
auto.ces(y,h=20,holdout=FALSE)


# Selection between "none" and "full" seasonalities
auto.ces(AirPassengers, h=12, holdout=TRUE,
                    seasonality=c("n","f"), ic="AIC")

ourModel <- auto.ces(AirPassengers)

summary(ourModel)
forecast(ourModel, h=12)
```

---

cma *Centered Moving Average*

Description

Function constructs centered moving average based on state space SMA

Usage

```
cma(y, order = NULL, silent = TRUE, ...)
```

Arguments

y	Vector or ts object, containing data needed to be smoothed.
order	Order of centered moving average. If NULL, then the function will try to select order of SMA based on information criteria. See sma for details.
silent	If TRUE, then plot is not produced. Otherwise, there is a plot...
...	Nothing. Needed only for the transition to the new name of variables.

Details

If the order is odd, then the function constructs SMA(order) and shifts it back in time. Otherwise an AR(order+1) model is constructed with the preset parameters:

$$phi_i = 0.5, 1, 1, ..., 0.5/order$$

This then corresponds to the centered MA with 0.5 weight for the first observation and 0.5 weight for an additional one. e.g. if this is monthly data and we use order=12, then half of the first January and half of the new one is taken.

This is not a forecasting tool. This is supposed to smooth the time series in order to find trend. So don't expect any forecasts from this function!

Value

Object of class "smooth" is returned. It contains the list of the following values:

- `model` - the name of the estimated model.
- `timeElapsed` - time elapsed for the construction of the model.
- `order` - order of the moving average.
- `nParam` - table with the number of estimated / provided parameters. If a previous model was reused, then its initials are reused and the number of provided parameters will take this into account.
- `fitted` - the fitted values, shifted in time.
- `forecast` - NAs, because this function does not produce forecasts.
- `residuals` - the residuals of the SMA / AR model.
- `s2` - variance of the residuals (taking degrees of freedom into account) of the SMA / AR model.
- `y` - the original data.
- `ICs` - values of information criteria from the respective SMA or AR model. Includes AIC, AICc, BIC and BICc.
- `logLik` - log-likelihood of the SMA / AR model.
- `lossValue` - Cost function value (for the SMA / AR model).
- `loss` - Type of loss function used in the estimation.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov I. (2023) Smooth forecasting with the smooth package in R. arXiv:2301.01790. doi:10.48550/arXiv.2301.01790.
- Svetunkov I. (2015 - Inf) "smooth" package for R - series of posts about the underlying models and how to use them: https://openforecast.org/category/r-en/smooth/.

See Also

es, ssarima

Examples

```
# CMA of specific order
ourModel <- cma(rnorm(118,100,3),order=12)

# CMA of arbitrary order
ourModel <- cma(rnorm(118,100,3))

summary(ourModel)
```

---

es *Exponential Smoothing in SSOE state space model*

Description

Function constructs ETS model and returns forecast, fitted values, errors and matrix of states. It is a wrapper of adam function.

Usage

```
es(y, model = "ZZZ", lags = c(frequency(y)), persistence = NULL,
  phi = NULL, initial = c("backcasting", "optimal", "two-stage",
  "complete"), initialSeason = NULL, ic = c("AICc", "AIC", "BIC", "BICc"),
  loss = c("likelihood", "MSE", "MAE", "HAM", "MSEh", "TMSE", "GTMSE",
  "MSCE"), h = 10, holdout = FALSE, bounds = c("usual", "admissible",
  "none"), silent = TRUE, xreg = NULL, regressors = c("use", "select"),
  initialX = NULL, ...)

es_old(y, model = "ZZZ", persistence = NULL, phi = NULL,
  initial = c("optimal", "backcasting"), initialSeason = NULL,
  ic = c("AICc", "AIC", "BIC", "BICc"), loss = c("likelihood", "MSE",
  "MAE", "HAM", "MSEh", "TMSE", "GTMSE", "MSCE"), h = 10, holdout = FALSE,
  cumulative = FALSE, interval = c("none", "parametric", "likelihood",
  "semiparametric", "nonparametric"), level = 0.95, bounds = c("usual",
  "admissible", "none"), silent = c("all", "graph", "legend", "output",
  "none"), xreg = NULL, regressors = c("use", "select"), initialX = NULL,
  ...)
```

Arguments

y	Vector or ts object, containing data needed to be forecasted.

`model` The type of ETS model. The first letter stands for the type of the error term ("A" or "M"), the second (and sometimes the third as well) is for the trend ("N", "A", "Ad", "M" or "Md"), and the last one is for the type of seasonality ("N", "A" or "M"). So, the function accepts words with 3 or 4 characters: `ANN`, `AAN`, `AAdN`, `AAA`, `AAdA`, `MAdM` etc. `ZZZ` means that the model will be selected based on the chosen information criteria type. Models pool can be restricted with additive only components. This is done via `model="XXX"`. For example, making selection between models with none / additive / damped additive trend component only (i.e. excluding multiplicative trend) can be done with `model="ZXZ"`. Furthermore, selection between multiplicative models (excluding additive components) is regulated using `model="YYY"`. This can be useful for positive data with low values (for example, slow moving products). Finally, if `model="CCC"`, then all the models are estimated and combination of their forecasts using AIC weights is produced (Kolassa, 2011). This can also be regulated. For example, `model="CCN"` will combine forecasts of all non-seasonal models and `model="CXY"` will combine forecasts of all the models with non-multiplicative trend and non-additive seasonality with either additive or multiplicative error. Not sure why anyone would need this thing, but it is available.

The parameter `model` can also be a vector of names of models for a finer tuning (pool of models). For example, `model=c("ANN","AAA")` will estimate only two models and select the best of them.

Also `model` can accept a previously estimated ES or ETS (from forecast package) model and use all its parameters.

Keep in mind that model selection with "Z" components uses Branch and Bound algorithm and may skip some models that could have slightly smaller information criteria.

`lags` Defines lags for the corresponding components. All components count, starting from level, so ETS(M,M,M) model for monthly data will have `lags=c(1,1,12)`. However, the function will also accept `lags=c(12)`, assuming that the lags 1 were dropped.

`persistence` Persistence vector g, containing smoothing parameters. If `NULL`, then estimated.

`phi` Value of damping parameter. If `NULL` then it is estimated.

`initial` Can be either character or a list, or a vector of initial states. If it is character, then it can be `"backcasting"`, meaning that the initials of dynamic part of the model are produced using backcasting procedure (advised for data with high frequency), or `"optimal"`, meaning that all initial states are optimised, or `"two-stage"`, meaning that optimisation is done after the backcasting, refining the states. In case of backcasting, the parameters of the explanatory variables are optimised. Alternatively, you can set `initial="complete"` backcasting, which means that all states (including explanatory variables) are initialised via backcasting.

`initialSeason` Vector of initial values for seasonal components. If `NULL`, they are estimated during optimisation.

`ic` The information criterion used in the model selection procedure.

`loss` The type of Loss Function used in optimization. `loss` can be: `likelihood` (assuming Normal distribution of error term), `MSE` (Mean Squared Error), `MAE`

(Mean Absolute Error), HAM (Half Absolute Moment), TMSE - Trace Mean Squared Error, GTMSE - Geometric Trace Mean Squared Error, MSEh - optimisation using only h-steps ahead error, MSCE - Mean Squared Cumulative Error. If loss!="MSE", then likelihood and model selection is done based on equivalent MSE. Model selection in this cases becomes not optimal.

There are also available analytical approximations for multistep functions: aMSEh, aTMSE and aGTMSE. These can be useful in cases of small samples.

Finally, just for fun the absolute and half analogues of multistep estimators are available: MAEh, TMAE, GTMAE, MACE, TMAE, HAMh, THAM, GTHAM, CHAM.

h — Length of forecasting horizon.

holdout — If TRUE, holdout sample of size h is taken from the end of the data.

bounds — What type of bounds to use in the model estimation. The first letter can be used instead of the whole word. "usual" implies restrictions on the smoothing parameter, guaranteeing that the exponential smoothing behaves as an averaging model. "admissible" guarantee that the model is stable.

silent — accepts TRUE and FALSE. If FALSE, the function will print its progress and produce a plot at the end.

xreg — The vector (either numeric or time series) or the matrix (or data.frame) of exogenous variables that should be included in the model. If matrix included than columns should contain variables and rows - observations. Note that xreg should have number of observations equal either to in-sample or to the whole series. If the number of observations in xreg is equal to in-sample, then values for the holdout sample are produced using es function.

regressors — The variable defines what to do with the provided xreg: "use" means that all of the data should be used, while "select" means that a selection using ic should be done.

initialX — The vector of initial parameters for exogenous variables. Ignored if xreg is NULL.

... — Other non-documented parameters. For example FI=TRUE will make the function also produce Fisher Information matrix, which then can be used to calculated variances of smoothing parameters and initial states of the model. Parameters B, lb and ub can be passed via ellipsis as well. In this case they will be used for optimisation. B sets the initial values before the optimisation, lb and ub define lower and upper bounds for the search inside of the specified bounds. These values should have length equal to the number of parameters to estimate. You can also pass two parameters to the optimiser: 1. maxeval - maximum number of evaluations to carry on; 2. xtol_rel - the precision of the optimiser. The default values used in es() are maxeval=500 and xtol_rel=1e-8. You can read more about these parameters in the documentation of nloptr function.

cumulative — If TRUE, then the cumulative forecast and prediction interval are produced instead of the normal ones. This is useful for inventory control systems.

interval — Type of interval to construct. This can be:

- "none", aka "n" - do not produce prediction interval.
- "parametric", "p" - use state-space structure of ETS. In case of mixed models this is done using simulations, which may take longer time than for

the pure additive and pure multiplicative models. This type of interval relies on unbiased estimate of in-sample error variance, which divides the sume of squared errors by T-k rather than just T.

- `"likelihood"`, `"l"` - these are the same as `"p"`, but relies on the biased estimate of variance from the likelihood (division by T, not by T-k).
- `"semiparametric"`, `"sp"` - interval based on covariance matrix of 1 to h steps ahead errors and assumption of normal / log-normal distribution (depending on error type).
- `"nonparametric"`, `"np"` - interval based on values from a quantile regression on error matrix (see Taylor and Bunn, 1999). The model used in this process is e[j] = a j^b, where j=1,..,h.

The parameter also accepts `TRUE` and `FALSE`. The former means that parametric interval are constructed, while the latter is equivalent to `none`. If the forecasts of the models were combined, then the interval are combined quantile-wise (Lichtendahl et al., 2013).

`level` Confidence level. Defines width of prediction interval.

Details

Function estimates ETS in a form of the Single Source of Error state space model of the following type:

$$y_t = o_t(w(v_{t-l}) + h(x_t, a_{t-1}) + r(v_{t-l})\epsilon_t)$$

$$v_t = f(v_{t-l}) + g(v_{t-l})\epsilon_t$$

$$a_t = F_X a_{t-1} + g_X \epsilon_t / x_t$$

Where o_t is the Bernoulli distributed random variable (in case of normal data it equals to 1 for all observations), v_t is the state vector and l is the vector of lags, x_t is the vector of exogenous variables. w(.) is the measurement function, r(.) is the error function, f(.) is the transition function, g(.) is the persistence function and h(.) is the explanatory variables function. a_t is the vector of parameters for exogenous variables, F_X is the `transitionX` matrix and g_X is the `persistenceX` matrix. Finally, ϵ_t is the error term.

For the details see Hyndman et al.(2008).

For some more information about the model and its implementation, see the vignette: `vignette("es","smooth")`.

Also, there are posts about the functions of the package smooth on the website of Ivan Svetunkov: https://openforecast.org/category/r-en/smooth/ - they explain the underlying models and how to use the functions.

Value

Object of class "adam" is returned. It contains the list of the following values for classical ETS models:

- `model` - type of constructed model.

- `formula` - mathematical formula, describing interactions between components of es() and exogenous variables.
- `timeElapsed` - time elapsed for the construction of the model.
- `states` - matrix of the components of ETS.
- `persistence` - persistence vector. This is the place, where smoothing parameters live.
- `phi` - value of damping parameter.
- `transition` - transition matrix of the model.
- `measurement` - measurement vector of the model.
- `initialType` - type of the initial values used.
- `initial` - initial values of the state vector (non-seasonal).
- `initialSeason` - initial values of the seasonal part of state vector.
- `nParam` - table with the number of estimated / provided parameters. If a previous model was reused, then its initials are reused and the number of provided parameters will take this into account.
- `fitted` - fitted values of ETS. In case of the intermittent model, the fitted are multiplied by the probability of occurrence.
- `forecast` - the point forecast for h steps ahead (by default NA is returned). NOTE that these do not always correspond to the conditional expectations. See ADAM textbook, Section 4.4. for details (`https://openforecast.org/adam/ETSTaxonomyMaths.html`),
- `lower` - lower bound of prediction interval. When `interval="none"` then NA is returned.
- `upper` - higher bound of prediction interval. When `interval="none"` then NA is returned.
- `residuals` - residuals of the estimated model.
- `errors` - trace forecast in-sample errors, returned as a matrix. Only returned when the multi-step losses are used and semiparametric interval is needed.
- `s2` - variance of the residuals (taking degrees of freedom into account). This is an unbiased estimate of variance.
- `interval` - type of interval asked by user.
- `level` - confidence level for interval.
- `cumulative` - whether the produced forecast was cumulative or not.
- `y` - original data.
- `holdout` - holdout part of the original data.
- `xreg` - provided vector or matrix of exogenous variables. If `regressors="s"`, then this value will contain only selected exogenous variables.
- `initialX` - initial values for parameters of exogenous variables.
- `ICs` - values of information criteria of the model. Includes AIC, AICc, BIC and BICc.
- `logLik` - concentrated log-likelihood of the function.
- `lossValue` - loss function value.
- `loss` - type of loss function used in the estimation.
- `FI` - Fisher Information. Equal to NULL if `FI=FALSE` or when `FI` is not provided at all.

- accuracy - vector of accuracy measures for the holdout sample. In case of non-intermittent data includes: MPE, MAPE, SMAPE, MASE, sMAE, RelMAE, sMSE and Bias coefficient (based on complex numbers). In case of intermittent data the set of errors will be: sMSE, sPIS, sCE (scaled cumulative error) and Bias coefficient. This is available only when holdout=TRUE.
- B - the vector of all the estimated parameters.

If combination of forecasts is produced (using model="CCC"), then a shorter list of values is returned:

- model,
- timeElapsed,
- initialType,
- fitted,
- forecast,
- lower,
- upper,
- residuals,
- s2 - variance of additive error of combined one-step-ahead forecasts,
- interval,
- level,
- cumulative,
- y,
- holdout,
- ICs - combined ic,
- ICw - ic weights used in the combination,
- loss,
- xreg,
- accuracy.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov, I., 2023. Smooth Forecasting with the Smooth Package in R. arXiv. doi:10.48550/arXiv.2301.01790
- Snyder, R. D., 1985. Recursive Estimation of Dynamic Linear Models. Journal of the Royal Statistical Society, Series B (Methodological) 47 (2), 272-276.
- Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) Forecasting with exponential smoothing: the state space approach, Springer-Verlag. doi:10.1007/9783540719182.
- Svetunkov, I., Boylan, J.E., 2023a. iETS: State Space Model for Intermittent Demand Forecastings. International Journal of Production Economics. 109013. doi:10.1016/j.ijpe.2023.109013

- Teunter R., Syntetos A., Babai Z. (2011). Intermittent demand: Linking forecasting to inventory obsolescence. European Journal of Operational Research, 214, 606-615.
- Croston, J. (1972) Forecasting and stock control for intermittent demands. Operational Research Quarterly, 23(3), 289-303.
- Kolassa, S. (2011) Combining exponential smoothing forecasts using Akaike weights. International Journal of Forecasting, 27, pp 238 - 251.
- Svetunkov, I., Boylan, J.E., 2023b. Staying Positive: Challenges and Solutions in Using Pure Multiplicative ETS Models. IMA Journal of Management Mathematics. p. 403-425. doi:10.1093/imaman/dpad028
- Taylor, J.W. and Bunn, D.W. (1999) A Quantile Regression Approach to Generating Prediction Intervals. Management Science, Vol 45, No 2, pp 225-237.
- Lichtendahl Kenneth C., Jr., Grushka-Cockayne Yael, Winkler Robert L., (2013) Is It Better to Average Probabilities or Quantiles? Management Science 59(7):1594-1611. DOI: doi:10.1287/mnsc.1120.1667

See Also

adam, forecast,ts, sim.es

Examples

```
# See how holdout and trace parameters influence the forecast
es(BJsales,model="AAdN",h=8,holdout=FALSE,loss="MSE")
es(AirPassengers,model="MAM",h=18,holdout=TRUE,loss="TMSE")

# Model selection example
es(BJsales,model="ZZN",ic="AIC",h=8,holdout=FALSE,bounds="a")

# Model selection. Compare AICc of these two models:
es(AirPassengers,"ZZZ",h=10,holdout=TRUE)
es(AirPassengers,"MAdM",h=10,holdout=TRUE)

# Model selection, excluding multiplicative trend
es(AirPassengers,model="ZXZ",h=8,holdout=TRUE)

# Combination example
es(BJsales,model="CCN",h=8,holdout=TRUE)

# Model selection using a specified pool of models
ourModel <- es(AirPassengers,model=c("ANN","AAM","AMdA"),h=18)

# Produce forecast and prediction interval
forecast(ourModel, h=18, interval="parametric")

# Semiparametric interval example
forecast(ourModel, h=18, interval="semiparametric")

# This will be the same model as in previous line but estimated on new portion of data
es(BJsales,model=ourModel,h=18,holdout=FALSE)
```

forecast.adam *Forecasting time series using smooth functions*

Description

Function produces conditional expectation (point forecasts) and prediction intervals for the estimated model.

Usage

```
## S3 method for class 'adam'
forecast(object, h = 10, newdata = NULL,
  occurrence = NULL, interval = c("none", "prediction", "confidence",
  "simulated", "approximate", "semiparametric", "nonparametric", "empirical",
  "complete"), level = 0.95, side = c("both", "upper", "lower"),
  cumulative = FALSE, nsim = NULL, scenarios = FALSE, ...)

## S3 method for class 'smooth'
forecast(object, h = 10, interval = c("parametric",
  "semiparametric", "nonparametric", "none"), level = 0.95,
  side = c("both", "upper", "lower"), ...)

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

## S3 method for class 'msdecompose'
forecast(object, h = 10, interval = c("parametric",
  "semiparametric", "nonparametric", "none"), level = 0.95, model = NULL,
  ...)
```

Arguments

object	Time series model for which forecasts are required.
h	Forecast horizon.
newdata	The new data needed in order to produce forecasts.
occurrence	The vector containing the future occurrence variable (values in [0,1]), if it is known.
interval	What type of mechanism to use for interval construction. the recommended option is interval="prediction", which will use analytical solutions for pure additive models and simulations for the others. interval="simulated" is the slowest method, but is robust to the type of model. interval="approximate" (aka interval="parametric") uses analytical formulae for conditional h-steps ahead variance, but is approximate for the non-additive error models. interval="semiparametric" relies on the multiple steps ahead forecast error (extracted via rmultistep method)

and on the assumed distribution of the error term. `interval="nonparametric"` uses Taylor & Bunn (1999) approach with quantile regressions. `interval="empirical"` constructs intervals based on empirical quantiles of multistep forecast errors. `interval="complete"` will call for `reforecast()` function and produce interval based on the uncertainty around the parameters of the model. Finally, `interval="confidence"` tries to generate the confidence intervals for the point forecast based on the `reforecast` method.

`level` Confidence level. Defines width of prediction interval.

`side` Defines, whether to provide `"both"` sides of prediction interval or only `"upper"`, or `"lower"`.

`cumulative` If `TRUE`, then the cumulative forecast and prediction interval are produced instead of the normal ones. This is useful for inventory control systems.

`nsim` Number of iterations to do in cases of `interval="simulated"`, `interval="prediction"` (for mixed and multiplicative model), `interval="confidence"` and `interval="complete"`. The default value for the prediction / simulated interval is 1000. In case of confidence or complete intervals, this is set to 100.

`scenarios` Binary, defining whether to return scenarios produced via simulations or not. Only works if `interval="simulated"`. If `TRUE` the object will contain `scenarios` variable.

`...` Other arguments accepted by either es, ces, gum or ssarima.

`model` The type of ETS model to fit on the decomposed trend. Only applicable to "msdecompose" class. This is then returned in parameter "esmodel". If `NULL`, then it will be selected automatically based on the type of the used decomposition (either among pure additive or among pure multiplicative ETS models).

Details

By default the function will generate conditional expectations from the estimated model and will also produce a variety of prediction intervals based on user preferences.

Value

Returns object of class "smooth.forecast", which contains:

- `model` - the estimated model (ES / CES / GUM / SSARIMA).
- `method` - the name of the estimated model (ES / CES / GUM / SSARIMA).
- `forecast` aka `mean` - point forecasts of the model (conditional mean).
- `lower` - lower bound of prediction interval.
- `upper` - upper bound of prediction interval.
- `level` - confidence level.
- `interval` - binary variable (whether interval were produced or not).
- `scenarios` - in case of `forecast.adam()` and `interval="simulated"` returns matrix with scenarios (future paths) that were used in simulations.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

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.

See Also

forecast

Examples

```
ourModel <- es(rnorm(100,0,1), h=10)
forecast(ourModel, h=10, interval="parametric")
```

---

gum *Generalised Univariate Model*

Description

Function constructs Generalised Univariate Model, estimating matrices F, w, vector g and initial parameters.

Usage

```
gum(y, orders = c(1, 1), lags = c(1, frequency(y)), type = c("additive",
  "multiplicative"), initial = c("backcasting", "optimal", "two-stage",
  "complete"), persistence = NULL, transition = NULL,
  measurement = rep(1, sum(orders)), loss = c("likelihood", "MSE", "MAE",
  "HAM", "MSEh", "TMSE", "GTMSE", "MSCE"), h = 0, holdout = FALSE,
  bounds = c("admissible", "none"), silent = TRUE, model = NULL,
  xreg = NULL, regressors = c("use", "select", "adapt", "integrate"),
  initialX = NULL, ...)

auto.gum(y, orders = 3, lags = frequency(y), type = c("additive",
  "multiplicative", "select"), initial = c("backcasting", "optimal",
  "two-stage", "complete"), ic = c("AICc", "AIC", "BIC", "BICc"),
  loss = c("likelihood", "MSE", "MAE", "HAM", "MSEh", "TMSE", "GTMSE",
  "MSCE"), h = 0, holdout = FALSE, bounds = c("admissible", "none"),
  silent = TRUE, xreg = NULL, regressors = c("use", "select", "adapt",
  "integrate"), ...)

gum_old(y, orders = c(1, 1), lags = c(1, frequency(y)),
  type = c("additive", "multiplicative"), persistence = NULL,
```

```
  transition = NULL, measurement = rep(1, sum(orders)),
  initial = c("optimal", "backcasting"), loss = c("likelihood", "MSE",
  "MAE", "HAM", "MSEh", "TMSE", "GTMSE", "MSCE"), h = 10, holdout = FALSE,
  bounds = c("restricted", "admissible", "none"), silent = c("all",
  "graph", "legend", "output", "none"), xreg = NULL, regressors = c("use",
  "select"), initialX = NULL, ...)

ges(...)
```

Arguments

y	Vector or ts object, containing data needed to be forecasted.
orders	Order of the model. Specified as vector of number of states with different lags. For example, `orders=c(1,1)` means that there are two states: one of the first lag type, the second of the second type. In case of `auto.gum()`, this parameters is the value of the max order to check.
lags	Defines lags for the corresponding orders. If, for example, `orders=c(1,1)` and lags are defined as `lags=c(1,12)`, then the model will have two states: the first will have lag 1 and the second will have lag 12. The length of `lags` must correspond to the length of `orders`. In case of the `auto.gum()`, the value of the maximum lag to check. This should usually be a maximum frequency of the data.
type	Type of model. Can either be `"additive"` or `"multiplicative"`. The latter means that the GUM is fitted on log-transformed data. In case of `auto.gum()`, can also be `"select"`, implying automatic selection of the type.
initial	Can be either character or a list, or a vector of initial states. If it is character, then it can be `"backcasting"`, meaning that the initials of dynamic part of the model are produced using backcasting procedure (advised for data with high frequency), or `"optimal"`, meaning that all initial states are optimised, or `"two-stage"`, meaning that optimisation is done after the backcasting, refining the states. In case of backcasting, the parameters of the explanatory variables are optimised. Alternatively, you can set `initial="complete"` backcasting, which means that all states (including explanatory variables) are initialised via backcasting.
persistence	Persistence vector g, containing smoothing parameters. If `NULL`, then estimated.
transition	Transition matrix F. Can be provided as a vector. Matrix will be formed using the default `matrix(transition,nc,nc)`, where `nc` is the number of components in the state vector. If `NULL`, then estimated.
measurement	Measurement vector w. If `NULL`, then estimated.
loss	The type of Loss Function used in optimization. `loss` can be: `likelihood` (assuming Normal distribution of error term), `MSE` (Mean Squared Error), `MAE` (Mean Absolute Error), `HAM` (Half Absolute Moment), `TMSE` - Trace Mean Squared Error, `GTMSE` - Geometric Trace Mean Squared Error, `MSEh` - optimisation using only h-steps ahead error, `MSCE` - Mean Squared Cumulative Error. If `loss!="MSE"`, then likelihood and model selection is done based on equivalent `MSE`. Model selection in this cases becomes not optimal.

There are also available analytical approximations for multistep functions: `aMSEh`, `aTMSE` and `aGTMSE`. These can be useful in cases of small samples.

Finally, just for fun the absolute and half analogues of multistep estimators are available: `MAEh`, `TMAE`, `GTMAE`, `MACE`, `TMAE`, `HAMh`, `THAM`, `GTHAM`, `CHAM`.

`h` Length of forecasting horizon.

`holdout` If `TRUE`, holdout sample of size `h` is taken from the end of the data.

`bounds` The type of bounds for the parameters to use in the model estimation. Can be either `admissible` - guaranteeing the stability of the model, or `none` - no restrictions (potentially dangerous).

`silent` accepts `TRUE` and `FALSE`. If FALSE, the function will print its progress and produce a plot at the end.

`model` A previously estimated GUM model, if provided, the function will not estimate anything and will use all its parameters.

`xreg` The vector (either numeric or time series) or the matrix (or data.frame) of exogenous variables that should be included in the model. If matrix included than columns should contain variables and rows - observations. Note that `xreg` should have number of observations equal either to in-sample or to the whole series. If the number of observations in `xreg` is equal to in-sample, then values for the holdout sample are produced using es function.

`regressors` The variable defines what to do with the provided xreg: `"use"` means that all of the data should be used, while `"select"` means that a selection using `ic` should be done.

`initialX` The vector of initial parameters for exogenous variables. Ignored if `xreg` is NULL.

`...` Other non-documented parameters. See adam for details. However, there are several unique parameters passed to the optimiser in comparison with `adam`: 1. `algorithm0`, which defines what algorithm to use in nloptr for the initial optimisation. By default, this is "NLOPT_LN_BOBYQA". 2. `algorithm` determines the second optimiser. By default this is "NLOPT_LN_NELDERMEAD". 3. maxeval0 and maxeval, that determine the number of iterations for the two optimisers. By default, `maxeval0=maxeval=40*k`, where k is the number of estimated parameters. 4. xtol_rel0 and xtol_rel, which are 1e-8 and 1e-6 respectively. There are also ftol_rel0, ftol_rel, ftol_abs0 and ftol_abs, which by default are set to values explained in the `nloptr.print.options()` function.

`ic` The information criterion used in the model selection procedure.

Details

The function estimates the Single Source of Error state space model of the following type:

$$y_t = w_t' v_{t-l} + \epsilon_t$$

$$v_t = F v_{t-l} + g_t \epsilon_t$$

where v_t is the state vector (defined using `orders`) and l is the vector of `lags`, w_t is the `measurement` vector (which includes fixed elements and explanatory variables), F is the `transition` matrix, g_t is the `persistence` vector (includes explanatory variables as well if provided), finally, ϵ_t is the error term.

For some more information about the model and its implementation, see the vignette: `vignette("gum","smooth")`

Value

Object of class "adam" is returned with similar elements to the adam function.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov I. (2023) Smooth forecasting with the smooth package in R. arXiv:2301.01790. doi:10.48550/arXiv.2301.01790.
- Svetunkov I. (2015 - Inf) "smooth" package for R - series of posts about the underlying models and how to use them: https://openforecast.org/category/r-en/smooth/.
- Svetunkov, I., 2023. Smooth Forecasting with the Smooth Package in R. arXiv. doi:10.48550/arXiv.2301.01790
- Snyder, R. D., 1985. Recursive Estimation of Dynamic Linear Models. Journal of the Royal Statistical Society, Series B (Methodological) 47 (2), 272-276.
- Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) Forecasting with exponential smoothing: the state space approach, Springer-Verlag. doi:10.1007/9783540719182.

See Also

adam, es, ces

gum, es,ces, sim.es, ssarima

Examples

```
gum(BJsales, h=8, holdout=TRUE)

ourModel <- gum(rnorm(118,100,3), orders=c(2,1), lags=c(1,4), h=18, holdout=TRUE)

# Redo previous model on a new data and produce prediction interval
gum(rnorm(118,100,3), model=ourModel, h=18)

# Produce something crazy with optimal initials (not recommended)
gum(rnorm(118,100,3), orders=c(1,1,1), lags=c(1,3,5), h=18, holdout=TRUE, initial="o")

# Simpler model estimated using trace forecast error loss function and its analytical analogue
gum(rnorm(118,100,3), orders=c(1), lags=c(1), h=18, holdout=TRUE, bounds="n", loss="TMSE")
```

```
x <- rnorm(50,100,3)

# The best GUM model for the data
ourModel <- auto.gum(x, orders=2, lags=4, h=18, holdout=TRUE)

summary(ourModel)
```

`is.smooth` *Smooth classes checkers*

Description

Functions to check if an object is of the specified class

Functions to check if an object is of the specified class

Usage

```
is.smooth(x)

is.smoothC(x)

is.msarima(x)

is.oes(x)

is.oesg(x)

is.smooth.sim(x)

is.smooth.forecast(x)

is.adam(x)

is.adam.sim(x)

is.msdecompose(x)

is.msdecompose.forecast(x)
```

Arguments

`x`	The object to check.

Details

The list of functions includes:

- `is.smooth()` tests if the object was produced by a smooth function (e.g. es / ces / ssarima / gum / sma / msarima);
- `is.msarima()` tests if the object was produced by the msarima function;
- `is.smoothC()` tests if the object was produced by a combination function (currently applies only to smoothCombine);
- `is.oes()` tests if the object was produced by oes function;
- `is.smooth.sim()` tests if the object was produced by simulate functions (e.g. sim.es / sim.ces / sim.ssarima / sim.sma / sim.gum);
- `is.smooth.forecast()` checks if the forecast was produced from a smooth function using forecast() function.

The list of functions includes:

- `is.adam()` tests if the object was produced by a adam function
- `is.adam.sim()` tests if the object was produced by sim.adam() function (not implemented yet);

Value

`TRUE` if this is the specified class and `FALSE` otherwise.

`TRUE` if this is the specified class and `FALSE` otherwise.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

Examples

```
ourModel <- msarima(rnorm(100,100,10))

is.smooth(ourModel)
is.msarima(ourModel)


ourModel <- adam(rnorm(100,100,10))
is.adam(ourModel)
```

`msarima` *Multiple Seasonal ARIMA*

Description

Function constructs Multiple Seasonal State Space ARIMA, estimating AR, MA terms and initial states. It is a wrapper of adam function.

Usage

```
msarima(y, orders = list(ar = c(0), i = c(1), ma = c(1)), lags = c(1),
  constant = FALSE, AR = NULL, MA = NULL, model = NULL,
  initial = c("backcasting", "optimal", "two-stage", "complete"),
  ic = c("AICc", "AIC", "BIC", "BICc"), loss = c("likelihood", "MSE",
  "MAE", "HAM", "MSEh", "TMSE", "GTMSE", "MSCE"), h = 10, holdout = FALSE,
  bounds = c("usual", "admissible", "none"), silent = TRUE, xreg = NULL,
  regressors = c("use", "select", "adapt"), initialX = NULL, ...)

auto.msarima(y, orders = list(ar = c(3, 3), i = c(2, 1), ma = c(3, 3)),
  lags = c(1, frequency(y)), initial = c("backcasting", "optimal",
  "two-stage", "complete"), ic = c("AICc", "AIC", "BIC", "BICc"),
  loss = c("likelihood", "MSE", "MAE", "HAM", "MSEh", "TMSE", "GTMSE",
  "MSCE"), h = 10, holdout = FALSE, bounds = c("usual", "admissible",
  "none"), silent = TRUE, xreg = NULL, regressors = c("use", "select",
  "adapt"), initialX = NULL, ...)

msarima_old(y, orders = list(ar = c(0), i = c(1), ma = c(1)), lags = c(1),
  constant = FALSE, AR = NULL, MA = NULL, initial = c("backcasting",
  "optimal"), ic = c("AICc", "AIC", "BIC", "BICc"), loss = c("likelihood",
  "MSE", "MAE", "HAM", "MSEh", "TMSE", "GTMSE", "MSCE"), h = 10,
  holdout = FALSE, cumulative = FALSE, interval = c("none", "parametric",
  "likelihood", "semiparametric", "nonparametric"), level = 0.95,
  bounds = c("admissible", "none"), silent = c("all", "graph", "legend",
  "output", "none"), xreg = NULL, regressors = c("use", "select"),
  initialX = NULL, ...)
```

Arguments

`y`	Vector or ts object, containing data needed to be forecasted.
`orders`	List of orders, containing vector variables `ar`, `i` and `ma`. Example: `orders=list(ar=c(1,2),i=c(1),ma=` If a variable is not provided in the list, then it is assumed to be equal to zero. At least one variable should have the same length as `lags`. Another option is to specify orders as a vector of a form `orders=c(p,d,q)`. The non-seasonal ARIMA(p,d,q) is constructed in this case. For `auto.msarima` this is the list of maximum orders to check, containing vector variables `ar`, `i` and `ma`. If a variable is not provided in the list, then it is assumed to be equal to zero. At least one variable should have the same length as `lags`.

lags — Defines lags for the corresponding orders (see examples above). The length of lags must correspond to the length of either ar, i or ma in orders variable. There is no restrictions on the length of lags vector. It is recommended to order lags ascending. The orders are set by a user. If you want the automatic order selection, then use auto.msarima function instead.

constant — If TRUE, constant term is included in the model. Can also be a number (constant value). For auto.msarima, if NULL, then the function will check if constant is needed.

AR — Vector or matrix of AR parameters. The order of parameters should be lag-wise. This means that first all the AR parameters of the firs lag should be passed, then for the second etc. AR of another msarima() can be passed here.

MA — Vector or matrix of MA parameters. The order of parameters should be lag-wise. This means that first all the MA parameters of the firs lag should be passed, then for the second etc. MA of another msarima can be passed here.

model — Previously estimated MSARIMA model.

initial — Can be either character or a vector of initial states. If it is character, then it can be "optimal", meaning that all initial states are optimised, or "backcasting", meaning that the initials of dynamic part of the model are produced using backcasting procedure (advised for data with high frequency). In the latter case, the parameters of the explanatory variables are optimised. This is recommended for ARIMAX model. Alternatively, you can set initial="complete" backcasting, which means that all states (including explanatory variables) are initialised via backcasting.

ic — The information criterion used in the model selection procedure.

loss — The type of Loss Function used in optimization. loss can be: likelihood (assuming Normal distribution of error term), MSE (Mean Squared Error), MAE (Mean Absolute Error), HAM (Half Absolute Moment), TMSE - Trace Mean Squared Error, GTMSE - Geometric Trace Mean Squared Error, MSEh - optimisation using only h-steps ahead error, MSCE - Mean Squared Cumulative Error. If loss!="MSE", then likelihood and model selection is done based on equivalent MSE. Model selection in this cases becomes not optimal.

There are also available analytical approximations for multistep functions: aMSEh, aTMSE and aGTMSE. These can be useful in cases of small samples.

Finally, just for fun the absolute and half analogues of multistep estimators are available: MAEh, TMAE, GTMAE, MACE, TMAE, HAMh, THAM, GTHAM, CHAM.

h — Length of forecasting horizon.

holdout — If TRUE, holdout sample of size h is taken from the end of the data.

bounds — What type of bounds to use in the model estimation. The first letter can be used instead of the whole word. "admissible" guarantee that the model is stable. "usual" are not supported due to restrictions in adam().

silent — accepts TRUE and FALSE. If FALSE, the function will print its progress and produce a plot at the end.

xreg — The vector (either numeric or time series) or the matrix (or data.frame) of exogenous variables that should be included in the model. If matrix included than columns should contain variables and rows - observations. Note that xreg

should have number of observations equal either to in-sample or to the whole series. If the number of observations in `xreg` is equal to in-sample, then values for the holdout sample are produced using es function.

`regressors` The variable defines what to do with the provided xreg: `"use"` means that all of the data should be used, while `"select"` means that a selection using `ic` should be done.

`initialX` The vector of initial parameters for exogenous variables. Ignored if `xreg` is NULL.

`...` Other non-documented parameters. see adam for details.

`FI=TRUE` will make the function produce Fisher Information matrix, which then can be used to calculated variances of parameters of the model.

`cumulative` If `TRUE`, then the cumulative forecast and prediction interval are produced instead of the normal ones. This is useful for inventory control systems.

`interval` Type of interval to construct. This can be:

- `"none"`, aka `"n"` - do not produce prediction interval.
- `"parametric"`, `"p"` - use state-space structure of ETS. In case of mixed models this is done using simulations, which may take longer time than for the pure additive and pure multiplicative models. This type of interval relies on unbiased estimate of in-sample error variance, which divides the sume of squared errors by T-k rather than just T.
- `"likelihood"`, `"l"` - these are the same as `"p"`, but relies on the biased estimate of variance from the likelihood (division by T, not by T-k).
- `"semiparametric"`, `"sp"` - interval based on covariance matrix of 1 to h steps ahead errors and assumption of normal / log-normal distribution (depending on error type).
- `"nonparametric"`, `"np"` - interval based on values from a quantile regression on error matrix (see Taylor and Bunn, 1999). The model used in this process is e[j] = a j^b, where j=1,..,h.

The parameter also accepts `TRUE` and `FALSE`. The former means that parametric interval are constructed, while the latter is equivalent to `none`. If the forecasts of the models were combined, then the interval are combined quantile-wise (Lichtendahl et al., 2013).

`level` Confidence level. Defines width of prediction interval.

Details

The model, implemented in this function differs from the one in ssarima function (Svetunkov & Boylan, 2019), but it is more efficient and better fitting the data (which might be a limitation).

The basic ARIMA(p,d,q) used in the function has the following form:

$$(1-B)^d(1-a_1B-a_2B^2-...-a_pB^p)y_{[}t] = (1+b_1B+b_2B^2+...+b_qB^q)\epsilon_{[}t]+c$$

where $y_{[}t]$ is the actual values, $\epsilon_{[}t]$ is the error term, a_i, b_j are the parameters for AR and MA respectively and c is the constant. In case of non-zero differences c acts as drift.

This model is then transformed into ARIMA in the Single Source of Error State space form (based by Snyder, 1985, but in a slightly different formulation):

$$y_t = o_t(w' v_{t-l} + x_t a_{t-1} + \epsilon_t)$$
$$v_t = F v_{t-l} + g \epsilon_t$$
$$a_t = F_X a_{t-1} + g_X \epsilon_t / x_t$$

Where o_t is the Bernoulli distributed random variable (in case of normal data equal to 1), v_t is the state vector (defined based on `orders`) and l is the vector of `lags`, x_t is the vector of exogenous parameters. w is the `measurement` vector, F is the `transition` matrix, g is the `persistence` vector, a_t is the vector of parameters for exogenous variables, F_X is the `transitionX` matrix and g_X is the `persistenceX` matrix. The main difference from ssarima function is that this implementation skips zero polynomials, substantially decreasing the dimension of the transition matrix. As a result, this function works faster than ssarima on high frequency data, and it is more accurate.

Due to the flexibility of the model, multiple seasonalities can be used. For example, something crazy like this can be constructed: SARIMA(1,1,1)(0,1,1)[24](2,0,1)[24*7](0,0,1)[24*30], but the estimation may take some time... Still this should be estimated in finite time (not like with `ssarima`).

The `auto.msarima` function constructs several ARIMA models in Single Source of Error state space form based on `adam` function (see adam documentation) and selects the best one based on the selected information criterion.

For some additional details see the vignettes: `vignette("adam","smooth")` and `vignette("ssarima","smooth")`

Value

Object of class "adam" is returned. It contains the list of the following values:

- `model` - the name of the estimated model.
- `timeElapsed` - time elapsed for the construction of the model.
- `states` - the matrix of the fuzzy components of msarima, where `rows` correspond to time and `cols` to states.
- `transition` - matrix F.
- `persistence` - the persistence vector. This is the place, where smoothing parameters live.
- `measurement` - measurement vector of the model.
- `AR` - the matrix of coefficients of AR terms.
- `I` - the matrix of coefficients of I terms.
- `MA` - the matrix of coefficients of MA terms.
- `constant` - the value of the constant term.
- `initialType` - Type of the initial values used.
- `initial` - the initial values of the state vector (extracted from `states`).
- `nParam` - table with the number of estimated / provided parameters. If a previous model was reused, then its initials are reused and the number of provided parameters will take this into account.
- `fitted` - the fitted values.
- `forecast` - the point forecast.
- `lower` - the lower bound of prediction interval. When `interval="none"` then NA is returned.
- `upper` - the higher bound of prediction interval. When `interval="none"` then NA is returned.

- `residuals` - the residuals of the estimated model.
- `errors` - The matrix of 1 to h steps ahead errors. Only returned when the multistep losses are used and semiparametric interval is needed.
- `s2` - variance of the residuals (taking degrees of freedom into account).
- `interval` - type of interval asked by user.
- `level` - confidence level for interval.
- `cumulative` - whether the produced forecast was cumulative or not.
- `y` - the original data.
- `holdout` - the holdout part of the original data.
- `xreg` - provided vector or matrix of exogenous variables. If `regressors="s"`, then this value will contain only selected exogenous variables.
- `initialX` - initial values for parameters of exogenous variables.
- `ICs` - values of information criteria of the model. Includes AIC, AICc, BIC and BICc.
- `logLik` - log-likelihood of the function.
- `lossValue` - Cost function value.
- `loss` - Type of loss function used in the estimation.
- `FI` - Fisher Information. Equal to NULL if `FI=FALSE` or when `FI` is not provided at all.
- `accuracy` - vector of accuracy measures for the holdout sample. In case of non-intermittent data includes: MPE, MAPE, SMAPE, MASE, sMAE, RelMAE, sMSE and Bias coefficient (based on complex numbers). In case of intermittent data the set of errors will be: sMSE, sPIS, sCE (scaled cumulative error) and Bias coefficient. This is available only when `holdout=TRUE`.
- `B` - the vector of all the estimated parameters.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov, I., 2023. Smooth Forecasting with the Smooth Package in R. arXiv. doi:10.48550/arXiv.2301.01790
- Snyder, R. D., 1985. Recursive Estimation of Dynamic Linear Models. Journal of the Royal Statistical Society, Series B (Methodological) 47 (2), 272-276.
- Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) Forecasting with exponential smoothing: the state space approach, Springer-Verlag. doi:10.1007/9783540719182.
- Svetunkov, I., & Boylan, J. E. (2019). State-space ARIMA for supply-chain forecasting. International Journal of Production Research, 0(0), 1–10. doi:10.1080/00207543.2019.1600764

See Also

`adam`, `orders`,`es`, `auto.ssarima`

Examples

```
x <- rnorm(118,100,3)

# The simple call of ARIMA(1,1,1):
ourModel <- msarima(x,orders=c(1,1,1),lags=1,h=18,holdout=TRUE)

# Example of SARIMA(2,0,0)(1,0,0)[4]
msarima(x,orders=list(ar=c(2,1)),lags=c(1,4),h=18,holdout=TRUE)

# SARIMA of a peculiar order on AirPassengers data
ourModel <- msarima(AirPassengers,orders=list(ar=c(1,0,3),i=c(1,0,1),ma=c(0,1,2)),
                    lags=c(1,6,12),h=10,holdout=TRUE)

# ARIMA(1,1,1) with Mean Squared Trace Forecast Error
msarima(x,orders=c(1,1,1),lags=1,h=18,holdout=TRUE,loss="TMSE")

plot(forecast(ourModel, h=18, interval="approximate"))


x <- rnorm(118,100,3)
# The best ARIMA for the data
ourModel <- auto.msarima(x,orders=list(ar=c(2,1),i=c(1,1),ma=c(2,1)),lags=c(1,12),
                     h=18,holdout=TRUE)

# The other one using optimised states
auto.msarima(x,orders=list(ar=c(3,2),i=c(2,1),ma=c(3,2)),lags=c(1,12),
                      h=18,holdout=TRUE)

# And now combined ARIMA
auto.msarima(x,orders=list(ar=c(3,2),i=c(2,1),ma=c(3,2)),lags=c(1,12),
                      combine=TRUE,h=18,holdout=TRUE)

plot(forecast(ourModel, h=18, interval="simulated"))
```

`msdecompose` *Multiple seasonal classical decomposition*

Description

Function decomposes multiple seasonal time series into components using the principles of classical decomposition.

Usage

```
msdecompose(y, lags = c(12), type = c("additive", "multiplicative"),
  smoother = c("ma", "lowess", "supsmu"), ...)
```

Arguments

y	Vector or ts object, containing data needed to be smoothed.
lags	Vector of lags, corresponding to the frequencies in the data.
type	The type of decomposition. If "multiplicative" is selected, then the logarithm of data is taken prior to the decomposition.
smoother	The type of function used in the smoother of the data to extract the trend and in seasonality smoothing. smoother="ma" relies on the centred moving average and will result in the classical decomposition. smoother="lowess" will use lowess, resulting in a decomposition similar to the STL (stl). Finally, smoother="supsmu" will use the Friedman's super smoother via supsmu.
...	Other parameters passed to smoothers. Only works with lowess/supsmu.

Details

The function applies centred moving averages based on filter function and order specified in lags variable in order to smooth the original series and obtain level, trend and seasonal components of the series.

Value

The object of the class "msdecompose" is return, containing:

- y - the original time series.
- initial - the estimates of the initial level and trend.
- trend - the long term trend in the data.
- seasonal - the list of seasonal parameters.
- lags - the provided lags.
- type - the selected type of the decomposition.
- yName - the name of the provided data.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov I. (2023) Smooth forecasting with the smooth package in R. arXiv:2301.01790. doi:10.48550/arXiv.2301.01790.
- Svetunkov I. (2015 - Inf) "smooth" package for R - series of posts about the underlying models and how to use them: https://openforecast.org/category/r-en/smooth/.

See Also

filter

Examples

```
# Decomposition of multiple frequency data
## Not run: ourModel <- msdecompose(forecast::taylor, lags=c(48,336), type="m")
ourModel <- msdecompose(AirPassengers, lags=c(12), type="m")

plot(ourModel)
plot(forecast(ourModel, model="AAN", h=12))
```

`multicov` *Function returns the multiple steps ahead covariance matrix of forecast errors*

Description

This function extracts covariance matrix of 1 to h steps ahead forecast errors for `adam()`, `ssarima()`, `gum()`, `sma()`, `es()` and `ces()` models.

Usage

```
multicov(object, type = c("analytical", "empirical", "simulated"), h = 10,
  nsim = 1000, ...)

## S3 method for class 'smooth'
multicov(object, type = c("analytical", "empirical",
  "simulated"), h = 10, nsim = 1000, ...)
```

Arguments

`object`	Model estimated using one of the functions of smooth package.
`type`	What method to use in order to produce covariance matrix: 1. `analytical` - based on the state space structure of the model and the one-step-ahead forecast error. This works for pure additive and pure multiplicative models. The values for the mixed models might be off. 2. `empirical` - based on the in-sample 1 to h steps ahead forecast errors (works fine on larger samples); 3. `simulated` - the data is simulated from the estimated model, then the same model is applied to it and then the empirical 1 to h steps ahead forecast errors are produced;
`h`	Forecast horizon to use in the calculations.
`nsim`	Number of iterations to produce in the simulation. Only needed if `type="simulated"`
`...`	Other parameters passed to simulate function (if `type="simulated"` is used). These are `obs` and `seed`. By default `obs=1000`. This approach increases the accuracy of covariance matrix on small samples and intermittent data;

Details

The function returns either scalar (if it is a non-smooth model) or the matrix of (h x h) size with variances and covariances of 1 to h steps ahead forecast errors.

Value

Scalar in cases of non-smooth functions. (h x h) matrix otherwise.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

orders

Examples

```
x <- rnorm(100,0,1)

# A simple example with a 5x5 covariance matrix
ourModel <- ces(x, h=5)
multicov(ourModel)
```

---

oes *Occurrence ETS model*

Description

Function returns the occurrence part of iETS model with the specified probability update and model types.

Usage

```
oes(y, model = "MNN", persistence = NULL, initial = "o",
  initialSeason = NULL, phi = NULL, occurrence = c("fixed", "general",
  "odds-ratio", "inverse-odds-ratio", "direct", "auto", "none"),
  ic = c("AICc", "AIC", "BIC", "BICc"), h = 10, holdout = FALSE,
  bounds = c("usual", "admissible", "none"), silent = c("all", "graph",
  "legend", "output", "none"), xreg = NULL, regressors = c("use",
  "select"), initialX = NULL, ...)
```

Arguments

y	Either numeric vector or time series vector.
model	The type of ETS model used for the estimation. Normally this should be "MNN" or any other pure multiplicative or additive model. The model selection is available here (although it's not fast), so you can use, for example, "YYN" and "XXN" for selecting between the pure multiplicative and pure additive models respectively. Using mixed models is possible, but not recommended.
persistence	Persistence vector g, containing smoothing parameters. If NULL, then estimated.
initial	Can be either character or a vector of initial states. If it is character, then it can be "optimal", meaning that the initial states are optimised, or "backcasting", meaning that the initials are produced using backcasting procedure.
initialSeason	The vector of the initial seasonal components. If NULL, then it is estimated.
phi	The value of the dampening parameter. Used only for damped-trend models.
occurrence	The type of model used in probability estimation. Can be "none" - none, "fixed" - constant probability, "odds-ratio" - the Odds-ratio model with b=1 in Beta distribution, "inverse-odds-ratio" - the model with a=1 in Beta distribution, "direct" - the TSB-like (Teunter et al., 2011) probability update mechanism a+b=1, "auto" - the automatically selected type of occurrence model, "general" - the general Beta model with two parameters. This will call oesg() function with two similar ETS models and the same provided parameters (initials and smoothing).
ic	The information criteria to use in case of model selection.
h	The forecast horizon.
holdout	If TRUE, holdout sample of size h is taken from the end of the data.
bounds	What type of bounds to use in the model estimation. The first letter can be used instead of the whole word.
silent	If silent="none", then nothing is silent, everything is printed out and drawn. silent="all" means that nothing is produced or drawn (except for warnings). In case of silent="graph", no graph is produced. If silent="legend", then legend of the graph is skipped. And finally silent="output" means that nothing is printed out in the console, but the graph is produced. silent also accepts TRUE and FALSE. In this case silent=TRUE is equivalent to silent="all", while silent=FALSE is equivalent to silent="none". The parameter also accepts first letter of words ("n", "a", "g", "l", "o").
xreg	The vector (either numeric or time series) or the matrix (or data.frame) of exogenous variables that should be included in the model. If matrix included than columns should contain variables and rows - observations. Note that xreg should have number of observations equal either to in-sample or to the whole series. If the number of observations in xreg is equal to in-sample, then values for the holdout sample are produced using es function.
regressors	The variable defines what to do with the provided xreg: "use" means that all of the data should be used, while "select" means that a selection using ic should be done.

`initialX` The vector of initial parameters for exogenous variables. Ignored if `xreg` is NULL.

`...` The parameters passed to the optimiser, such as `maxeval`, `xtol_rel`, `algorithm` and `print_level`. The description of these is printed out by `nloptr.print.options()` function from the `nloptr` package. The default values in the oes function are `maxeval=500`, `xtol_rel=1E-8`, `algorithm="NLOPT_LN_NELDERMEAD"` and `print_level=0`.

Details

The function estimates probability of demand occurrence, using the selected ETS state space models.

For the details about the model and its implementation, see the respective vignette: `vignette("oes","smooth")`

Value

The object of class "occurrence" is returned. It contains following list of values:

- `model` - the type of the estimated ETS model;
- `timeElapsed` - the time elapsed for the construction of the model;
- `fitted` - the fitted values for the probability;
- `fittedModel` - the fitted values of the underlying ETS model, where applicable (only for occurrence=c("o","i","d"));
- `forecast` - the forecast of the probability for `h` observations ahead;
- `forecastModel` - the forecast of the underlying ETS model, where applicable (only for occurrence=c("o","i","d"));
- `lower` - the lower bound of the interval if `interval!="none"`;
- `upper` - the upper bound of the interval if `interval!="none"`;
- `lowerModel` - the lower bound of the interval of the underlying ETS model if `interval!="none"`;
- `upperModel` - the upper bound of the interval of the underlying ETS model if `interval!="none"`;
- `states` - the values of the state vector;
- `logLik` - the log-likelihood value of the model;
- `nParam` - the number of parameters in the model (the matrix is returned);
- `residuals` - the residuals of the model;
- `y` - actual values of occurrence (zeros and ones).
- `persistence` - the vector of smoothing parameters;
- `phi` - the value of the damped trend parameter;
- `initial` - initial values of the state vector;
- `initialSeason` - the matrix of initials seasonal states;
- `occurrence` - the type of the occurrence model;
- `updateX` - boolean, defining, if the states of exogenous variables were estimated as well.
- `initialX` - initial values for parameters of exogenous variables.

- `persistenceX` - persistence vector g for exogenous variables.
- `transitionX` - transition matrix F for exogenous variables.
- `accuracy` - The error measures for the forecast (in case of `holdout=TRUE`).
- `B` - the vector of all the estimated parameters (in case of "odds-ratio", "inverse-odds-ratio" and "direct" models).

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov, I., Boylan, J.E., 2023a. iETS: State Space Model for Intermittent Demand Forecastings. International Journal of Production Economics. 109013. doi:10.1016/j.ijpe.2023.109013
- Teunter R., Syntetos A., Babai Z. (2011). Intermittent demand: Linking forecasting to inventory obsolescence. European Journal of Operational Research, 214, 606-615.
- Croston, J. (1972) Forecasting and stock control for intermittent demands. Operational Research Quarterly, 23(3), 289-303.

See Also

`adam`, `oesg`, `es`

Examples

```
y <- rpois(100,0.1)
oes(y, occurrence="auto")

oes(y, occurrence="f")
```

`oesg` *Occurrence ETS, general model*

Description

Function returns the general occurrence model of the of iETS model.

Usage

```
oesg(y, modelA = "MNN", modelB = "MNN", persistenceA = NULL,
  persistenceB = NULL, phiA = NULL, phiB = NULL, initialA = "o",
  initialB = "o", initialSeasonA = NULL, initialSeasonB = NULL,
  ic = c("AICc", "AIC", "BIC", "BICc"), h = 10, holdout = FALSE,
  bounds = c("usual", "admissible", "none"), silent = c("all", "graph",
  "legend", "output", "none"), xregA = NULL, xregB = NULL,
  initialXA = NULL, initialXB = NULL, regressorsA = c("use", "select"),
  regressorsB = c("use", "select"), ...)
```

Arguments

y	Either numeric vector or time series vector.
modelA	The type of the ETS for the model A.
modelB	The type of the ETS for the model B.
persistenceA	The persistence vector g, containing smoothing parameters used in the model A. If NULL, then estimated.
persistenceB	The persistence vector g, containing smoothing parameters used in the model B. If NULL, then estimated.
phiA	The value of the dampening parameter in the model A. Used only for damped-trend models.
phiB	The value of the dampening parameter in the model B. Used only for damped-trend models.
initialA	Either "o" - optimal or the vector of initials for the level and / or trend for the model A.
initialB	Either "o" - optimal or the vector of initials for the level and / or trend for the model B.
initialSeasonA	The vector of the initial seasonal components for the model A. If NULL, then it is estimated.
initialSeasonB	The vector of the initial seasonal components for the model B. If NULL, then it is estimated.
ic	Information criteria to use in case of model selection.
h	Forecast horizon.
holdout	If TRUE, holdout sample of size h is taken from the end of the data.
bounds	What type of bounds to use in the model estimation. The first letter can be used instead of the whole word.
silent	If silent="none", then nothing is silent, everything is printed out and drawn. silent="all" means that nothing is produced or drawn (except for warnings). In case of silent="graph", no graph is produced. If silent="legend", then legend of the graph is skipped. And finally silent="output" means that nothing is printed out in the console, but the graph is produced. silent also accepts TRUE and FALSE. In this case silent=TRUE is equivalent to silent="all", while silent=FALSE is equivalent to silent="none". The parameter also accepts first letter of words ("n", "a", "g", "l", "o").
xregA	The vector or the matrix of exogenous variables, explaining some parts of occurrence variable of the model A.
xregB	The vector or the matrix of exogenous variables, explaining some parts of occurrence variable of the model B.
initialXA	The vector of initial parameters for exogenous variables in the model A. Ignored if xregA is NULL.
initialXB	The vector of initial parameters for exogenous variables in the model B. Ignored if xregB is NULL.

regressorsA	Variable defines what to do with the provided xregA: "use" means that all of the data should be used, while "select" means that a selection using ic should be done.
regressorsB	Similar to the regressorsA, but for the part B of the model.
...	The parameters passed to the optimiser, such as maxeval, xtol_rel, algorithm and print_level. The description of these is printed out by nloptr.print.options() function from the nloptr package. The default values in the oes function are maxeval=500, xtol_rel=1E-8, algorithm="NLOPT_LN_NELDERMEAD" and print_level=0.

Details

The function estimates probability of demand occurrence, based on the iETS_G state-space model. It involves the estimation and modelling of the two simultaneous state space equations. Thus two parts for the model type, persistence, initials etc.

For the details about the model and its implementation, see the respective vignette: vignette("oes","smooth")

The model is based on:

$$o_t \sim Bernoulli(p_t)$$
$$p_t = \frac{a_t}{a_t + b_t}$$

,

where a_t and b_t are the parameters of the Beta distribution and are modelled using separate ETS models.

Value

The object of class "occurrence" is returned. It contains following list of values:

- modelA - the model A of the class oes, that contains the output similar to the one from the oes() function;
- modelB - the model B of the class oes, that contains the output similar to the one from the oes() function.
- B - the vector of all the estimated parameters.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

es, oes

Examples

```
y <- rpois(100,0.1)
oesg(y, modelA="MNN", modelB="ANN")
```

orders *Functions that extract values from the fitted model*

Description

These functions allow extracting orders and lags for ssarima(), gum() and sma(), type of model from es() and ces() and name of model.

Usage

```
orders(object, ...)

lags(object, ...)

modelName(object, ...)

modelType(object, ...)
```

Arguments

object	Model estimated using one of the functions of smooth package.
...	Currently nothing is accepted via ellipsis.

Details

orders() and lags() are useful only for SSARIMA, GUM and SMA. They return NA for other functions. This can also be applied to arima(), Arima() and auto.arima() functions from stats and forecast packages. modelType() is useful only for ETS and CES. They return NA for other functions. This can also be applied to ets() function from forecast package. errorType extracts the type of error from the model (either additive or multiplicative). Finally, modelName returns the name of the fitted model. For example, "ARIMA(0,1,1)". This is purely descriptive and can also be applied to non-smooth classes, so that, for example, you can easily extract the name of the fitted AR model from ar() function from stats package.

Value

Either vector, scalar or list with values is returned. orders() in case of ssarima returns list of values:

- ar - AR orders.
- i - I orders.
- ma - MA orders.

lags() returns the vector of lags of the model. All the other functions return strings of character.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

ssarima, msarima

Examples

```
x <- rnorm(100,0,1)

# Just as example. orders and lags do not return anything for ces() and es(). But modelType() does.
ourModel <- ces(x, h=10)
orders(ourModel)
lags(ourModel)
modelType(ourModel)
modelName(ourModel)

# And as another example it does the opposite for gum() and ssarima()
ourModel <- gum(x, h=10, orders=c(1,1), lags=c(1,4))
orders(ourModel)
lags(ourModel)
modelType(ourModel)
modelName(ourModel)

# Finally these values can be used for simulate functions or original functions.
ourModel <- auto.ssarima(x)
ssarima(x, orders=orders(ourModel), lags=lags(ourModel), constant=ourModel$constant)
sim.ssarima(orders=orders(ourModel), lags=lags(ourModel), constant=ourModel$constant)

ourModel <- es(x)
es(x, model=modelType(ourModel))
sim.es(model=modelType(ourModel))
```

---

plot.adam *Plots for the fit and states*

Description

The function produces diagnostics plots for a smooth model

Usage

```
## S3 method for class 'adam'
plot(x, which = c(1, 2, 4, 6), level = 0.95,
  legend = FALSE, ask = prod(par("mfcol")) < length(which) &&
  dev.interactive(), lowess = TRUE, ...)

## S3 method for class 'smooth'
plot(x, which = c(1, 2, 4, 6), level = 0.95,
  legend = FALSE, ask = prod(par("mfcol")) < length(which) &&
  dev.interactive(), lowess = TRUE, ...)
```

```
## S3 method for class 'msdecompose'
plot(x, which = c(1, 2, 4, 6), level = 0.95,
  legend = FALSE, ask = prod(par("mfcol")) < length(which) &&
  dev.interactive(), lowess = TRUE, ...)
```

Arguments

x	Estimated smooth model.
which	Which of the plots to produce. The possible options (see details for explanations): 1. Actuals vs Fitted values; 2. Standardised residuals vs Fitted; 3. Studentised residuals vs Fitted; 4. Absolute residuals vs Fitted; 5. Squared residuals vs Fitted; 6. Q-Q plot with the specified distribution; 7. Fitted over time; 8. Standardised residuals vs Time; 9. Studentised residuals vs Time; 10. ACF of the residuals; 11. PACF of the residuals; 12. Plot of states of the model; 13. Absolute standardised residuals vs Fitted; 14. Squared standardised residuals vs Fitted; 15. ACF of the squared residuals; 16. PACF of the squared residuals.
level	Confidence level. Defines width of confidence interval. Used in plots (2), (3), (7), (8), (9), (10) and (11).
legend	If TRUE, then the legend is produced on plots (2), (3) and (7).
ask	Logical; if TRUE, the user is asked to press Enter before each plot.
lowess	Logical; if TRUE, LOWESS lines are drawn on scatterplots, see lowess.
...	The parameters passed to the plot functions. Recommended to use with separate plots.

Details

The list of produced plots includes:

1. Actuals vs Fitted values. Allows analysing, whether there are any issues in the fit. Does the variability of actuals increase with the increase of fitted values? Is the relation well captured? They grey line on the plot corresponds to the perfect fit of the model.
2. Standardised residuals vs Fitted. Plots the points and the confidence bounds (red lines) for the specified confidence level. Useful for the analysis of outliers;

3. Studentised residuals vs Fitted. This is similar to the previous plot, but with the residuals divided by the scales with the leave-one-out approach. Should be more sensitive to outliers;
4. Absolute residuals vs Fitted. Useful for the analysis of heteroscedasticity;
5. Squared residuals vs Fitted - similar to (3), but with squared values;
6. Q-Q plot with the specified distribution. Can be used in order to see if the residuals follow the assumed distribution. The type of distribution depends on the one used in the estimation (see `distribution` parameter in alm);
7. ACF of the residuals. Are the residuals autocorrelated? See acf for details;
8. Fitted over time. Plots actuals (black line), fitted values (purple line), point forecast (blue line) and prediction interval (grey lines). Can be used in order to make sure that the model did not miss any important events over time;
9. Standardised residuals vs Time. Useful if you want to see, if there is autocorrelation or if there is heteroscedasticity in time. This also shows, when the outliers happen;
10. Studentised residuals vs Time. Similar to previous, but with studentised residuals;
11. PACF of the residuals. No, really, are they autocorrelated? See pacf function from stats package for details;
12. Plot of the states of the model. It is not recommended to produce this plot together with the others, because there might be several states, which would cause the creation of a different canvas. In case of "msdecompose", this will produce the decomposition of the series into states on a different canvas;
13. Absolute standardised residuals vs Fitted. Similar to the previous, but with absolute values. This is more relevant to the models where scale is calculated as an absolute value of something (e.g. Laplace);
14. Squared standardised residuals vs Fitted. This is an additional plot needed to diagnose heteroscedasticity in a model with varying scale. The variance on this plot will be constant if the adequate model for `scale` was constructed. This is more appropriate for normal and the related distributions.

Which of the plots to produce, is specified via the `which` parameter.

Value

The function produces the number of plots, specified in the parameter `which`.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

plot.greybox

Examples

```
ourModel <- es(c(rnorm(50,100,10),rnorm(50,120,10)), "ANN", h=10)
par(mfcol=c(3,4))
plot(ourModel, c(1:11))
plot(ourModel, 12)
```

---

pls *Prediction Likelihood Score*

Description

Function estimates Prediction Likelihood Score for the provided model

Usage

```
pls(object, holdout = NULL, ...)

## S3 method for class 'smooth'
pls(object, holdout = NULL, ...)
```

Arguments

object	The model estimated using smooth functions. This thing also accepts other models (e.g. estimated using functions from forecast package), but may not always work properly with them.
holdout	The values for the holdout part of the sample. If the model was fitted on the data with the holdout=TRUE, then the parameter is not needed.
...	Parameters passed to multicov function. The function is called in order to get the covariance matrix of 1 to h steps ahead forecast errors.

Details

Prediction likelihood score (PLS) is based on either normal or log-normal distribution of errors. This is extracted from the provided model. The likelihood based on the distribution of 1 to h steps ahead forecast errors is used in the process.

Value

A value of the log-likelihood.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

distribution. IEEE Signal Processing Letters. 13 (5): 300-303. doi:10.1109/LSP.2006.870353 - this is not yet used in the function.

Snyder, R. D., Ord, J. K., Beaumont, A., 2012. Forecasting the intermittent demand for slow-moving inventories: A modelling approach. International Journal of Forecasting 28 (2), 485-496.

- Kolassa, S., 2016. Evaluating predictive count data distributions in retail sales forecasting. International Journal of Forecasting 32 (3), 788-803..

Examples

```
# Generate data, apply es() with the holdout parameter and calculate PLS
x <- rnorm(100,0,1)
ourModel <- es(x, h=10, holdout=TRUE)
pls(ourModel, type="a")
pls(ourModel, type="e")
pls(ourModel, type="s", obs=100, nsim=100)
```

`reapply` *Reapply the model with randomly generated initial parameters and produce forecasts*

Description

`reapply` function generates the parameters based on the values in the provided object and then reapplies the same model with those parameters to the data, getting the fitted paths and updated states. `reforecast` function uses those values in order to produce forecasts for the `h` steps ahead.

Usage

```
reapply(object, nsim = 1000, bootstrap = FALSE, heuristics = NULL, ...)

reforecast(object, h = 10, newdata = NULL, occurrence = NULL,
  interval = c("prediction", "confidence", "none"), level = 0.95,
  side = c("both", "upper", "lower"), cumulative = FALSE, nsim = 100,
  ...)
```

Arguments

`object`	Model estimated using one of the functions of smooth package.
`nsim`	Number of paths to generate (number of simulations to do).
`bootstrap`	The logical, which determines, whether to use bootstrap for the covariance matrix of parameters or not.
`heuristics`	The value for proportion to use for heuristic estimation of the standard deviation of parameters. If `NULL`, it is not used.

...	Other parameters passed to `reapply()` and `mean()` functions in case of `reforecast` (`trim` parameter in `mean()` is set to 0.01 by default) and to `vcov` in case of `reapply`.
h	Forecast horizon.
newdata	The new data needed in order to produce forecasts.
occurrence	The vector containing the future occurrence variable (values in [0,1]), if it is known.
interval	What type of mechanism to use for interval construction. The options include `interval="none"`, `interval="prediction"` (prediction intervals) and `interval="confidence"` (intervals for the point forecast). The other options are not supported and do not make much sense for the refitted model.
level	Confidence level. Defines width of prediction interval.
side	Defines, whether to provide `"both"` sides of prediction interval or only `"upper"`, or `"lower"`.
cumulative	If `TRUE`, then the cumulative forecast and prediction interval are produced instead of the normal ones. This is useful for inventory control systems.

Details

The main motivation of the function is to take the randomness due to the in-sample estimation of parameters into account when fitting the model and to propagate this randomness to the forecasts. The methods can be considered as a special case of recursive bootstrap.

Value

`reapply()` returns object of the class "reapply", which contains:

- `timeElapsed` - Time elapsed for the code execution;
- `y` - The actual values;
- `states` - The array of states of the model;
- `refitted` - The matrix with fitted values, where columns correspond to different paths;
- `fitted` - The vector of fitted values (conditional mean);
- `model` - The name of the constructed model;
- `transition` - The array of transition matrices;
- `measurement` - The array of measurement matrices;
- `persistence` - The matrix of persistence vectors (paths in columns);
- `profile` - The array of profiles obtained by the end of each fit.

`reforecast()` returns the object of the class forecast.smooth, which contains in addition to the standard list the variable `paths` - all simulated trajectories with h in rows, simulated future paths for each state in columns and different states (obtained from `reapply()` function) in the third dimension.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

forecast.smooth

Examples

```
x <- rnorm(100,0,1)

# Just as example. orders and lags do not return anything for ces() and es(). But modelType() does.
ourModel <- adam(x, "ANN")
refittedModel <- reapply(ourModel, nsim=50)
plot(refittedModel)

ourForecast <- reforecast(ourModel, nsim=50)
```

---

rmultistep *Multiple steps ahead forecast errors*

Description

The function extracts 1 to h steps ahead forecast errors from the model.

Usage

```
rmultistep(object, h = 10, ...)
```

Arguments

object	Model estimated using one of the forecasting functions.
h	The forecasting horizon to use.
...	Currently nothing is accepted via ellipsis.

Details

The errors correspond to the error term epsilon_t in the ETS models. Don't forget that different models make different assumptions about epsilon_t and / or 1+epsilon_t.

Value

The matrix with observations in rows and h steps ahead values in columns. So, the first row corresponds to the forecast produced from the 0th observation from 1 to h steps ahead.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

residuals,

Examples

```
x <- rnorm(100,0,1)
ourModel <- adam(x)
rmultistep(ourModel, h=13)
```

`sim.ces` *Simulate Complex Exponential Smoothing*

Description

Function generates data using CES with Single Source of Error as a data generating process.

Usage

```
sim.ces(seasonality = c("none", "simple", "partial", "full"), obs = 10,
  nsim = 1, frequency = 1, a = NULL, b = NULL, initial = NULL,
  randomizer = c("rnorm", "rt", "rlaplace", "rs"), probability = 1, ...)
```

Arguments

`seasonality`	The type of seasonality used in CES. Can be: `none` - No seasonality; `simple` - Simple seasonality, using lagged CES (based on `t-m` observation, where `m` is the seasonality lag); `partial` - Partial seasonality with real seasonal components (equivalent to additive seasonality); `full` - Full seasonality with complex seasonal components (can do both multiplicative and additive seasonality, depending on the data). First letter can be used instead of full words. Any seasonal CES can only be constructed for time series vectors.
`obs`	Number of observations in each generated time series.
`nsim`	Number of series to generate (number of simulations to do).
`frequency`	Frequency of generated data. In cases of seasonal models must be greater than 1.
`a`	First complex smoothing parameter. Should be a complex number. NOTE! CES is very sensitive to a and b values so it is advised to use values from previously estimated model.
`b`	Second complex smoothing parameter. Can be real if `seasonality="partial"`. In case of `seasonality="full"` must be complex number.
`initial`	A matrix with initial values for CES. In case with `seasonality="partial"` and `seasonality="full"` first two columns should contain initial values for non-seasonal components, repeated `frequency` times.

randomizer	Type of random number generator function used for error term. Defaults are: rnorm, rt, rlaplace and rs. rlnorm should be used for multiplicative models (e.g. ETS(M,N,N)). But any function from Distributions will do the trick if the appropriate parameters are passed. For example rpois with lambda=2 can be used as well, but might result in weird values.
probability	Probability of occurrence, used for intermittent data generation. This can be a vector, implying that probability varies in time (in TSB or Croston style).
...	Additional parameters passed to the chosen randomizer. All the parameters should be passed in the order they are used in chosen randomizer. For example, passing just sd=0.5 to rnorm function will lead to the call rnorm(obs, mean=0.5, sd=1).

Details

For the information about the function, see the vignette: vignette("simulate","smooth")

Value

List of the following values is returned:

- model - Name of CES model.
- a - Value of complex smoothing parameter a. If nsim>1, then this is a vector.
- b - Value of complex smoothing parameter b. If seasonality="n" or seasonality="s", then this is equal to NULL. If nsim>1, then this is a vector.
- initial - Initial values of CES in a form of matrix. If nsim>1, then this is an array.
- data - Time series vector (or matrix if nsim>1) of the generated series.
- states - Matrix (or array if nsim>1) of states. States are in columns, time is in rows.
- residuals - Error terms used in the simulation. Either vector or matrix, depending on nsim.
- occurrence - Values of occurrence variable. Once again, can be either a vector or a matrix...
- logLik - Log-likelihood of the constructed model.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov, I., Kourentzes, N., & Ord, J. K. (2022). Complex exponential smoothing. Naval Research Logistics, 69(8), 1108–1123. https://doi.org/10.1002/nav.22074

See Also

sim.es, sim.ssarima,ces, Distributions

Examples

```
# Create 120 observations from CES(n). Generate 100 time series of this kind.
x <- sim.ces("n",obs=120,nsim=100)

# Generate similar thing for seasonal series of CES(s)_4
x <- sim.ces("s",frequency=4,obs=80,nsim=100)

# Estimate model and then generate 10 time series from it
ourModel <- ces(rnorm(100,100,5))
simulate(ourModel,nsim=10)
```

---

sim.es *Simulate Exponential Smoothing*

Description

Function generates data using ETS with Single Source of Error as a data generating process.

Usage

```
sim.es(model = "ANN", obs = 10, nsim = 1, frequency = 1,
  persistence = NULL, phi = 1, initial = NULL, initialSeason = NULL,
  bounds = c("usual", "admissible", "restricted"), randomizer = c("rnorm",
  "rlnorm", "rt", "rlaplace", "rs"), probability = 1, ...)
```

Arguments

model	Type of ETS model according to [Hyndman et. al., 2008] taxonomy. Can consist of 3 or 4 chars: ANN, AAN, AAdN, AAA, AAdA, MAdM etc.
obs	Number of observations in each generated time series.
nsim	Number of series to generate (number of simulations to do).
frequency	Frequency of generated data. In cases of seasonal models must be greater than 1.
persistence	Persistence vector, which includes all the smoothing parameters. Must correspond to the chosen model. The maximum length is 3: level, trend and seasonal smoothing parameters. If NULL, values are generated.
phi	Value of damping parameter. If trend is not chosen in the model, the parameter is ignored.
initial	Vector of initial states of level and trend. The maximum length is 2. If NULL, values are generated.
initialSeason	Vector of initial states for seasonal coefficients. Should have length equal to frequency parameter. If NULL, values are generated.

`bounds` Type of bounds to use for persistence vector if values are generated. `"usual"` - bounds from p.156 by Hyndman et. al., 2008. `"restricted"` - similar to `"usual"` but with upper bound equal to 0.3. `"admissible"` - bounds from tables 10.1 and 10.2 of Hyndman et. al., 2008. Using first letter of the type of bounds also works. These bounds are also used for multiplicative models, but the models are much more restrictive, so weird results might be obtained. Be careful!

`randomizer` Type of random number generator function used for error term. Defaults are: `rnorm`, `rt`, `rlaplace` and `rs`. `rlnorm` should be used for multiplicative models (e.g. ETS(M,N,N)). But any function from Distributions will do the trick if the appropriate parameters are passed. For example `rpois` with `lambda=2` can be used as well, but might result in weird values.

`probability` Probability of occurrence, used for intermittent data generation. This can be a vector, implying that probability varies in time (in TSB or Croston style).

`...` Additional parameters passed to the chosen randomizer. All the parameters should be passed in the order they are used in chosen randomizer. For example, passing just `sd=0.5` to `rnorm` function will lead to the call `rnorm(obs, mean=0.5, sd=1)`. ATTENTION! When generating the multiplicative errors some tuning might be needed to obtain meaningful data. `sd=0.1` is usually already a high value for such models. ALSO NOTE: In case of multiplicative error model, the randomizer will generate `1+e_t` error, not `e_t`. This means that the mean should typically be equal to 1, not zero.

Details

For the information about the function, see the vignette: `vignette("simulate","smooth")`

Value

List of the following values is returned:

- `model` - Name of ETS model.
- `data` - Time series vector (or matrix if `nsim>1`) of the generated series.
- `states` - Matrix (or array if `nsim>1`) of states. States are in columns, time is in rows.
- `persistence` - Vector (or matrix if `nsim>1`) of smoothing parameters used in the simulation.
- `phi` - Value of damping parameter used in time series generation.
- `initial` - Vector (or matrix) of initial values.
- `initialSeason` - Vector (or matrix) of initial seasonal coefficients.
- `probability` - vector of probabilities used in the simulation.
- `intermittent` - type of the intermittent model used.
- `residuals` - Error terms used in the simulation. Either vector or matrix, depending on `nsim`.
- `occurrence` - Values of occurrence variable. Once again, can be either a vector or a matrix...
- `logLik` - Log-likelihood of the constructed model.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov, I., 2023. Smooth Forecasting with the Smooth Package in R. arXiv. doi:10.48550/arXiv.2301.01790
- Snyder, R. D., 1985. Recursive Estimation of Dynamic Linear Models. Journal of the Royal Statistical Society, Series B (Methodological) 47 (2), 272-276.
- Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) Forecasting with exponential smoothing: the state space approach, Springer-Verlag. doi:10.1007/9783540719182.

See Also

es, ts, Distributions

Examples

```
# Create 40 observations of quarterly data using AAA model with errors from normal distribution
ETSAAA <- sim.es(model="AAA",frequency=4,obs=40,randomizer="rnorm",mean=0,sd=100)

# Create 50 series of quarterly data using AAA model
# with 40 observations each with errors from normal distribution
ETSAAA <- sim.es(model="AAA",frequency=4,obs=40,randomizer="rnorm",mean=0,sd=100,nsim=50)

# Create 50 series of quarterly data using AAdA model
# with 40 observations each with errors from normal distribution
# and smoothing parameters lying in the "admissible" range.
ETSAAA <- sim.es(model="AAA",phi=0.9,frequency=4,obs=40,bounds="admissible",
                 randomizer="rnorm",mean=0,sd=100,nsim=50)

# Create 60 observations of monthly data using ANN model
# with errors from beta distribution
ETSANN <- sim.es(model="ANN",persistence=c(1.5),frequency=12,obs=60,
                 randomizer="rbeta",shape1=1.5,shape2=1.5)
plot(ETSANN$states)

# Create 60 observations of monthly data using MAM model
# with errors from uniform distribution
ETSMAM <- sim.es(model="MAM",persistence=c(0.3,0.2,0.1),initial=c(2000,50),
           phi=0.8,frequency=12,obs=60,randomizer="runif",min=-0.5,max=0.5)

# Create 80 observations of quarterly data using MMM model
# with predefined initial values and errors from the normal distribution
ETSMMM <- sim.es(model="MMM",persistence=c(0.1,0.1,0.1),initial=c(2000,1),
           initialSeason=c(1.1,1.05,0.9,.95),frequency=4,obs=80,mean=0,sd=0.01)

# Generate intermittent data using AAdN
iETSAAdN <- sim.es("AAdN",obs=30,frequency=1,probability=0.1,initial=c(3,0),phi=0.8)

# Generate iETS(MNN) with TSB style probabilities
```

```
oETSMNN <- sim.oes("MNN",obs=50,occurrence="d",persistence=0.2,initial=1,
                   randomizer="rlnorm",meanlog=0,sdlog=0.3)
iETSMNN <- sim.es("MNN",obs=50,frequency=12,persistence=0.2,initial=4,
                  probability=oETSMNN$probability)
```

`sim.gum` *Simulate Generalised Exponential Smoothing*

Description

Function generates data using GUM with Single Source of Error as a data generating process.

Usage

```
sim.gum(orders = c(1), lags = c(1), obs = 10, nsim = 1,
  frequency = 1, measurement = NULL, transition = NULL,
  persistence = NULL, initial = NULL, randomizer = c("rnorm", "rt",
  "rlaplace", "rs"), probability = 1, ...)
```

Arguments

`orders`	Order of the model. Specified as vector of number of states with different lags. For example, `orders=c(1,1)` means that there are two states: one of the first lag type, the second of the second type.
`lags`	Defines lags for the corresponding orders. If, for example, `orders=c(1,1)` and lags are defined as `lags=c(1,12)`, then the model will have two states: the first will have lag 1 and the second will have lag 12. The length of `lags` must correspond to the length of `orders`.
`obs`	Number of observations in each generated time series.
`nsim`	Number of series to generate (number of simulations to do).
`frequency`	Frequency of generated data. In cases of seasonal models must be greater than 1.
`measurement`	Measurement vector w. If `NULL`, then estimated.
`transition`	Transition matrix F. Can be provided as a vector. Matrix will be formed using the default `matrix(transition,nc,nc)`, where `nc` is the number of components in state vector. If `NULL`, then estimated.
`persistence`	Persistence vector g, containing smoothing parameters. If `NULL`, then estimated.
`initial`	Vector of initial values for state matrix. If `NULL`, then generated using advanced, sophisticated technique - uniform distribution.
`randomizer`	Type of random number generator function used for error term. Defaults are: `rnorm`, `rt`, `rlaplace` and `rs`. `rlnorm` should be used for multiplicative models (e.g. ETS(M,N,N)). But any function from Distributions will do the trick if the appropriate parameters are passed. For example `rpois` with `lambda=2` can be used as well, but might result in weird values.

probability — Probability of occurrence, used for intermittent data generation. This can be a vector, implying that probability varies in time (in TSB or Croston style).

... — Additional parameters passed to the chosen randomizer. All the parameters should be passed in the order they are used in chosen randomizer. For example, passing just sd=0.5 to rnorm function will lead to the call rnorm(obs, mean=0.5, sd=1).

Details

For the information about the function, see the vignette: vignette("simulate","smooth")

Value

List of the following values is returned:

- model - Name of GUM model.
- measurement - Matrix w.
- transition - Matrix F.
- persistence - Persistence vector. This is the place, where smoothing parameters live.
- initial - Initial values of GUM in a form of matrix. If nsim>1, then this is an array.
- data - Time series vector (or matrix if nsim>1) of the generated series.
- states - Matrix (or array if nsim>1) of states. States are in columns, time is in rows.
- residuals - Error terms used in the simulation. Either vector or matrix, depending on nsim.
- occurrence - Values of occurrence variable. Once again, can be either a vector or a matrix...
- logLik - Log-likelihood of the constructed model.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov I. (2023) Smooth forecasting with the smooth package in R. arXiv:2301.01790. doi:10.48550/arXiv.2301.01790.
- Svetunkov I. (2015 - Inf) "smooth" package for R - series of posts about the underlying models and how to use them: https://openforecast.org/category/r-en/smooth/.

See Also

sim.es, sim.ssarima, sim.ces, gum, Distributions

Examples

```
# Create 120 observations from GUM(1[1]). Generate 100 time series of this kind.
x <- sim.gum(orders=c(1),lags=c(1),obs=120,nsim=100)

# Generate similar thing for seasonal series of GUM(1[1],1[4]])
x <- sim.gum(orders=c(1,1),lags=c(1,4),frequency=4,obs=80,nsim=100,transition=c(1,0,0.9,0.9))

# Estimate model and then generate 10 time series from it
ourModel <- gum(rnorm(100,100,5))
simulate(ourModel,nsim=10)
```

---

sim.oes *Simulate Occurrence Part of ETS model*

Description

Function generates data using ETS with Single Source of Error as a data generating process for the demand occurrence. As the main output it produces probabilities of occurrence.

Usage

```
sim.oes(model = "MNN", obs = 10, nsim = 1, frequency = 1,
  occurrence = c("odds-ratio", "inverse-odds-ratio", "direct", "general"),
  bounds = c("usual", "admissible", "restricted"), randomizer = c("rlnorm",
  "rinvgauss", "rgamma", "rnorm"), persistence = NULL, phi = 1,
  initial = NULL, initialSeason = NULL, modelB = model,
  persistenceB = persistence, phiB = phi, initialB = initial,
  initialSeasonB = initialSeason, ...)
```

Arguments

model	Type of ETS model according to [Hyndman et. al., 2008] taxonomy. Can consist of 3 or 4 chars: ANN, AAN, AAdN, AAA, AAdA, MAdM etc. The conventional oETS model assumes that the error term is positive, so "MZZ" models are recommended for this. If you use additive error models, then the function will exponentiate the obtained values before transforming them and getting the probability. This is the type of model A.
obs	Number of observations in each generated time series.
nsim	Number of series to generate (number of simulations to do).
frequency	Frequency of generated data. In cases of seasonal models must be greater than 1.
occurrence	Type of occurrence model. See vignette("oes","smooth") for details.

bounds	Type of bounds to use for persistence vector if values are generated. "usual" - bounds from p.156 by Hyndman et. al., 2008. "restricted" - similar to "usual" but with upper bound equal to 0.3. "admissible" - bounds from tables 10.1 and 10.2 of Hyndman et. al., 2008. Using first letter of the type of bounds also works. These bounds are also used for multiplicative models, but the models are much more restrictive, so weird results might be obtained. Be careful!
randomizer	Type of random number generator function used for error term. It is advised to use rlnorm() or rinvgauss() in case of multiplicative error models. If a randomiser is used, it is advised to specify the parameters in the ellipsis.
persistence	Persistence vector, which includes all the smoothing parameters. Must correspond to the chosen model. The maximum length is 3: level, trend and seasonal smoothing parameters. If NULL, values are generated.
phi	Value of damping parameter. If trend is not chosen in the model, the parameter is ignored.
initial	Vector of initial states of level and trend. The maximum length is 2. If NULL, values are generated.
initialSeason	Vector of initial states for seasonal coefficients. Should have length equal to frequency parameter. If NULL, values are generated.
modelB	Type of model B. This is used in occurrence="general" and occurrence="inverse-odds-ratio".
persistenceB	The persistence vector for the model B.
phiB	Value of damping parameter for the model B.
initialB	Vector of initial states of level and trend for the model B.
initialSeasonB	Vector of initial states for seasonal coefficients for the model B.
...	Additional parameters passed to the chosen randomizer. All the parameters should be passed in the order they are used in chosen randomizer. Both model A and model B share the same parameters for the randomizer.

Details

For the information about the function, see the vignette: vignette("simulate","smooth")

Value

List of the following values is returned:

- model - Name of ETS model.
- modelA - Model A, generated using sim.es() function;
- modelB - Model B, generated using sim.es() function;
- probability - The value of probability generated by the model;
- occurrence - Type of occurrence used in the model;
- logLik - Log-likelihood of the constructed model.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov, I., 2023. Smooth Forecasting with the Smooth Package in R. arXiv. doi:10.48550/arXiv.2301.01790
- Snyder, R. D., 1985. Recursive Estimation of Dynamic Linear Models. Journal of the Royal Statistical Society, Series B (Methodological) 47 (2), 272-276.
- Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) Forecasting with exponential smoothing: the state space approach, Springer-Verlag. doi:10.1007/9783540719182.

See Also

oes, sim.es, Distributions

Examples

```
# This example uses rinvgauss function from statmod package.
oETSMNNIG <- sim.oes(model="MNN",frequency=12,obs=60,
                               randomizer="rinvgauss",mean=1,dispersion=0.5)

# A simpler example with log normal distribution
oETSMNNlogN <- sim.oes(model="MNN",frequency=12,obs=60,initial=1,
                       randomizer="rlnorm",meanlog=0,sdlog=0.1)
```

---

sim.sma *Simulate Simple Moving Average*

Description

Function generates data using SMA in a Single Source of Error state space model as a data generating process.

Usage

```
sim.sma(order = NULL, obs = 10, nsim = 1, frequency = 1,
  initial = NULL, randomizer = c("rnorm", "rt", "rlaplace", "rs"),
  probability = 1, ...)
```

Arguments

order	Order of the modelled series. If omitted, then a random order from 1 to 100 is selected.
obs	Number of observations in each generated time series.
nsim	Number of series to generate (number of simulations to do).
frequency	Frequency of generated data. In cases of seasonal models must be greater than 1.
initial	Vector of initial states for the model. If NULL, values are generated.

`randomizer` Type of random number generator function used for error term. Defaults are: `rnorm`, `rt`, `rlaplace` and `rs`. `rlnorm` should be used for multiplicative models (e.g. ETS(M,N,N)). But any function from Distributions will do the trick if the appropriate parameters are passed. For example `rpois` with `lambda=2` can be used as well, but might result in weird values.

`probability` Probability of occurrence, used for intermittent data generation. This can be a vector, implying that probability varies in time (in TSB or Croston style).

`...` Additional parameters passed to the chosen randomizer. All the parameters should be passed in the order they are used in chosen randomizer. For example, passing just `sd=0.5` to `rnorm` function will lead to the call `rnorm(obs, mean=0.5, sd=1)`.

Details

For the information about the function, see the vignette: `vignette("simulate","smooth")`

Value

List of the following values is returned:

- `model` - Name of SMA model.
- `data` - Time series vector (or matrix if `nsim>1`) of the generated series.
- `states` - Matrix (or array if `nsim>1`) of states. States are in columns, time is in rows.
- `initial` - Vector (or matrix) of initial values.
- `probability` - vector of probabilities used in the simulation.
- `intermittent` - type of the intermittent model used.
- `residuals` - Error terms used in the simulation. Either vector or matrix, depending on `nsim`.
- `occurrence` - Values of occurrence variable. Once again, can be either a vector or a matrix...
- `logLik` - Log-likelihood of the constructed model.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov, I., 2023. Smooth Forecasting with the Smooth Package in R. arXiv. doi:10.48550/arXiv.2301.01790
- Snyder, R. D., 1985. Recursive Estimation of Dynamic Linear Models. Journal of the Royal Statistical Society, Series B (Methodological) 47 (2), 272-276.
- Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) Forecasting with exponential smoothing: the state space approach, Springer-Verlag. doi:10.1007/9783540719182.

See Also

`es`, `ts`, `Distributions`

Examples

```
# Create 40 observations of quarterly data using AAA model with errors from normal distribution
sma10 <- sim.sma(order=10,frequency=4,obs=40,randomizer="rnorm",mean=0,sd=100)
```

`sim.ssarima` *Simulate SSARIMA*

Description

Function generates data using SSARIMA with Single Source of Error as a data generating process.

Usage

```
sim.ssarima(orders = list(ar = 0, i = 1, ma = 1), lags = 1, obs = 10,
  nsim = 1, frequency = 1, AR = NULL, MA = NULL, constant = FALSE,
  initial = NULL, bounds = c("admissible", "none"),
  randomizer = c("rnorm", "rt", "rlaplace", "rs"), probability = 1, ...)
```

Arguments

`orders`	List of orders, containing vector variables `ar`, `i` and `ma`. Example: `orders=list(ar=c(1,2),i=c(1),ma=` If a variable is not provided in the list, then it is assumed to be equal to zero. At least one variable should have the same length as `lags`.
`lags`	Defines lags for the corresponding orders (see examples above). The length of `lags` must correspond to the length of `orders`. There is no restrictions on the length of `lags` vector. It is recommended to order `lags` ascending.
`obs`	Number of observations in each generated time series.
`nsim`	Number of series to generate (number of simulations to do).
`frequency`	Frequency of generated data. In cases of seasonal models must be greater than 1.
`AR`	Vector or matrix of AR parameters. The order of parameters should be lag-wise. This means that first all the AR parameters of the firs lag should be passed, then for the second etc. AR of another ssarima can be passed here.
`MA`	Vector or matrix of MA parameters. The order of parameters should be lag-wise. This means that first all the MA parameters of the firs lag should be passed, then for the second etc. MA of another ssarima can be passed here.
`constant`	If `TRUE`, constant term is included in the model. Can also be a number (constant value).
`initial`	Vector of initial values for state matrix. If `NULL`, then generated using advanced, sophisticated technique - uniform distribution.
`bounds`	Type of bounds to use for AR and MA if values are generated. `"admissible"` - bounds guaranteeing stability and stationarity of SSARIMA. `"none"` - we generate something, but do not guarantee stationarity and stability. Using first letter of the type of bounds also works.

`randomizer` Type of random number generator function used for error term. Defaults are: `rnorm`, `rt`, `rlaplace` and `rs`. `rlnorm` should be used for multiplicative models (e.g. ETS(M,N,N)). But any function from Distributions will do the trick if the appropriate parameters are passed. For example `rpois` with `lambda=2` can be used as well, but might result in weird values.

`probability` Probability of occurrence, used for intermittent data generation. This can be a vector, implying that probability varies in time (in TSB or Croston style).

`...` Additional parameters passed to the chosen randomizer. All the parameters should be passed in the order they are used in chosen randomizer. For example, passing just `sd=0.5` to `rnorm` function will lead to the call `rnorm(obs, mean=0.5, sd=1)`.

Details

For the information about the function, see the vignette: `vignette("simulate","smooth")`

Value

List of the following values is returned:

- `model` - Name of SSARIMA model.
- `AR` - Value of AR parameters. If `nsim>1`, then this is a matrix.
- `MA` - Value of MA parameters. If `nsim>1`, then this is a matrix.
- `constant` - Value of constant term. If `nsim>1`, then this is a vector.
- `initial` - Initial values of SSARIMA. If `nsim>1`, then this is a matrix.
- `data` - Time series vector (or matrix if `nsim>1`) of the generated series.
- `states` - Matrix (or array if `nsim>1`) of states. States are in columns, time is in rows.
- `residuals` - Error terms used in the simulation. Either vector or matrix, depending on `nsim`.
- `occurrence` - Values of occurrence variable. Once again, can be either a vector or a matrix...
- `logLik` - Log-likelihood of the constructed model.

Author(s)

Ivan Svetunkov, `<ivan@svetunkov.com>`

References

- Svetunkov, I., 2023. Smooth Forecasting with the Smooth Package in R. arXiv. doi:10.48550/arXiv.2301.01790
- Snyder, R. D., 1985. Recursive Estimation of Dynamic Linear Models. Journal of the Royal Statistical Society, Series B (Methodological) 47 (2), 272-276.
- Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) Forecasting with exponential smoothing: the state space approach, Springer-Verlag. doi:10.1007/9783540719182.
- Svetunkov, I., & Boylan, J. E. (2019). State-space ARIMA for supply-chain forecasting. International Journal of Production Research, 0(0), 1–10. doi:10.1080/00207543.2019.1600764

See Also

sim.es, ssarima, Distributions, orders

Examples

```
# Create 120 observations from ARIMA(1,1,1) with drift. Generate 100 time series of this kind.
x <- sim.ssarima(ar.orders=1,i.orders=1,ma.orders=1,obs=120,nsim=100,constant=TRUE)

# Generate similar thing for seasonal series of SARIMA(1,1,1)(0,0,2)_4
x <- sim.ssarima(ar.orders=c(1,0),i.orders=c(1,0),ma.orders=c(1,2),lags=c(1,4),
                 frequency=4,obs=80,nsim=100,constant=FALSE)

# Generate 10 series of high frequency data from SARIMA(1,0,2)_1(0,1,1)_7(1,0,1)_30
x <- sim.ssarima(ar.orders=c(1,0,1),i.orders=c(0,1,0),ma.orders=c(2,1,1),lags=c(1,7,30),
                 obs=360,nsim=10)
```

---

sma *Simple Moving Average*

Description

Function constructs state space simple moving average of predefined order

Usage

```
sma(y, order = NULL, ic = c("AICc", "AIC", "BIC", "BICc"), h = 10,
  holdout = FALSE, silent = TRUE, fast = TRUE, ...)

sma_old(y, order = NULL, ic = c("AICc", "AIC", "BIC", "BICc"), h = 10,
  holdout = FALSE, cumulative = FALSE, interval = c("none", "parametric",
  "likelihood", "semiparametric", "nonparametric"), level = 0.95,
  silent = c("all", "graph", "legend", "output", "none"), ...)
```

Arguments

y	Vector or ts object, containing data needed to be forecasted.
order	Order of simple moving average. If NULL, then it is selected automatically using information criteria.
ic	The information criterion used in the model selection procedure.
h	Length of forecasting horizon.
holdout	If TRUE, holdout sample of size h is taken from the end of the data.
silent	accepts TRUE and FALSE. If FALSE, the function will print its progress and produce a plot at the end.

`fast` if `TRUE`, then the modified Ternary search is used to find the optimal order of the model. This does not guarantee the optimal solution, but gives a reasonable one (local minimum).

`...` Other non-documented parameters. For example parameter `model` can accept a previously estimated SMA model and use its parameters.

`cumulative` If `TRUE`, then the cumulative forecast and prediction interval are produced instead of the normal ones. This is useful for inventory control systems.

`interval` Type of interval to construct. This can be:

- `"none"`, aka `"n"` - do not produce prediction interval.
- `"parametric"`, `"p"` - use state-space structure of ETS. In case of mixed models this is done using simulations, which may take longer time than for the pure additive and pure multiplicative models. This type of interval relies on unbiased estimate of in-sample error variance, which divides the sume of squared errors by T-k rather than just T.
- `"likelihood"`, `"l"` - these are the same as `"p"`, but relies on the biased estimate of variance from the likelihood (division by T, not by T-k).
- `"semiparametric"`, `"sp"` - interval based on covariance matrix of 1 to h steps ahead errors and assumption of normal / log-normal distribution (depending on error type).
- `"nonparametric"`, `"np"` - interval based on values from a quantile regression on error matrix (see Taylor and Bunn, 1999). The model used in this process is e[j] = a j^b, where j=1,..,h.

The parameter also accepts `TRUE` and `FALSE`. The former means that parametric interval are constructed, while the latter is equivalent to `none`. If the forecasts of the models were combined, then the interval are combined quantile-wise (Lichtendahl et al., 2013).

`level` Confidence level. Defines width of prediction interval.

Details

The function constructs AR model in the Single Source of Error state space form based on the idea that:

$y_t = \frac{1}{n} \sum_{j=1}^{n} y_{t-j}$

which is AR(n) process, that can be modelled using:

$y_t = w' v_{t-1} + \epsilon_t$

$v_t = F v_{t-1} + g \epsilon_t$

Where v_t is a state vector.

For some more information about the model and its implementation, see the vignette: `vignette("sma","smooth")`

Value

Object of class "smooth" is returned. It contains the list of the following values:

- `model` - the name of the estimated model.

- timeElapsed - time elapsed for the construction of the model.
- states - the matrix of the fuzzy components of ssarima, where rows correspond to time and cols to states.
- transition - matrix F.
- persistence - the persistence vector. This is the place, where smoothing parameters live.
- measurement - measurement vector of the model.
- order - order of moving average.
- initial - Initial state vector values.
- initialType - Type of initial values used.
- nParam - table with the number of estimated / provided parameters. If a previous model was reused, then its initials are reused and the number of provided parameters will take this into account.
- fitted - the fitted values.
- forecast - the point forecast.
- lower - the lower bound of prediction interval. When interval=FALSE then NA is returned.
- upper - the higher bound of prediction interval. When interval=FALSE then NA is returned.
- residuals - the residuals of the estimated model.
- errors - The matrix of 1 to h steps ahead errors. Only returned when the multistep losses are used and semiparametric interval is needed.
- s2 - variance of the residuals (taking degrees of freedom into account).
- interval - type of interval asked by user.
- level - confidence level for interval.
- cumulative - whether the produced forecast was cumulative or not.
- y - the original data.
- holdout - the holdout part of the original data.
- ICs - values of information criteria of the model. Includes AIC, AICc, BIC and BICc.
- logLik - log-likelihood of the function.
- lossValue - Cost function value.
- loss - Type of loss function used in the estimation.
- accuracy - vector of accuracy measures for the holdout sample. Includes: MPE, MAPE, SMAPE, MASE, sMAE, RelMAE, sMSE and Bias coefficient (based on complex numbers). This is available only when holdout=TRUE.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov I. (2023) Smooth forecasting with the smooth package in R. arXiv:2301.01790. doi:10.48550/arXiv.2301.01790.
- Svetunkov I. (2015 - Inf) "smooth" package for R - series of posts about the underlying models and how to use them: https://openforecast.org/category/r-en/smooth/.
- Svetunkov, I., & Petropoulos, F. (2017). Old dog, new tricks: a modelling view of simple moving averages. International Journal of Production Research, 7543(January), 1-14. doi:10.1080/00207543.2017.1380326

See Also

filter, adam, msarima

Examples

```
# SMA of specific order
ourModel <- sma(rnorm(118,100,3), order=12, h=18, holdout=TRUE)

# SMA of arbitrary order
ourModel <- sma(rnorm(118,100,3), h=18, holdout=TRUE)

plot(forecast(ourModel, h=18, interval="empirical"))
```

---

smooth *Smooth package*

Description

Package contains functions implementing Single Source of Error state space models for purposes of time series analysis and forecasting.

Details

Package:	smooth
Type:	Package
Date:	2016-01-27 - Inf
License:	GPL-2

The following functions are included in the package:

- es - Exponential Smoothing in Single Source of Errors State Space form.
- ces - Complex Exponential Smoothing.
- gum - Generalised Exponential Smoothing.

- ssarima - SARIMA in state space framework.
- auto.ces - Automatic selection between seasonal and non-seasonal CES.
- auto.ssarima - Automatic selection of ARIMA orders.
- sma - Simple Moving Average in state space form.
- smoothCombine - the function that combines forecasts from es(), ces(), gum(), ssarima() and sma() functions.
- cma - Centered Moving Average. This is for smoothing time series, not for forecasting.
- sim.es - simulate time series using ETS as a model.
- sim.ces - simulate time series using CES as a model.
- sim.ssarima - simulate time series using SARIMA as a model.
- sim.gum - simulate time series using GUM as a model.
- sim.sma - simulate time series using SMA.
- sim.oes - simulate time series based on occurrence part of ETS model.
- oes - occurrence part of the intermittent state space model.

There are also several methods implemented in the package for the classes "smooth" and "smooth.sim":

- orders - extracts orders of the fitted model.
- lags - extracts lags of the fitted model.
- modelType - extracts type of the fitted model.
- forecast - produces forecast using provided model.
- multicov - returns covariance matrix of multiple steps ahead forecast errors.
- pls - returns Prediction Likelihood Score.
- nparam - returns number of the estimated parameters.
- fitted - extracts fitted values from provided model.
- getResponse - returns actual values from the provided model.
- residuals - extracts residuals of provided model.
- plot - plots either states of the model or produced forecast (depending on what object is passed).
- simulate - uses sim functions (sim.es, sim.ces, sim.ssarima, sim.gum, sim.sma and sim.oes) in order to simulate data using the provided object.
- summary - provides summary of the object.
- AICc, BICc - return, guess what...

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov, I., 2023. Smooth Forecasting with the Smooth Package in R. arXiv. doi:10.48550/arXiv.2301.01790
- Snyder, R. D., 1985. Recursive Estimation of Dynamic Linear Models. Journal of the Royal Statistical Society, Series B (Methodological) 47 (2), 272-276.
- Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) Forecasting with exponential smoothing: the state space approach, Springer-Verlag. doi:10.1007/9783540719182.
- Svetunkov, I., Boylan, J.E., 2023a. iETS: State Space Model for Intermittent Demand Forecastings. International Journal of Production Economics. 109013. doi:10.1016/j.ijpe.2023.109013
- Teunter R., Syntetos A., Babai Z. (2011). Intermittent demand: Linking forecasting to inventory obsolescence. European Journal of Operational Research, 214, 606-615.
- Croston, J. (1972) Forecasting and stock control for intermittent demands. Operational Research Quarterly, 23(3), 289-303.
- Svetunkov, I., Kourentzes, N., & Ord, J. K. (2022). Complex exponential smoothing. Naval Research Logistics, 69(8), 1108–1123. https://doi.org/10.1002/nav.22074
- Svetunkov I. (2023) Smooth forecasting with the smooth package in R. arXiv:2301.01790. doi:10.48550/arXiv.2301.01790.
- Svetunkov I. (2015 - Inf) "smooth" package for R - series of posts about the underlying models and how to use them: https://openforecast.org/category/r-en/smooth/.
- Kolassa, S. (2011) Combining exponential smoothing forecasts using Akaike weights. International Journal of Forecasting, 27, pp 238 - 251.
- Svetunkov, I., Boylan, J.E., 2023b. Staying Positive: Challenges and Solutions in Using Pure Multiplicative ETS Models. IMA Journal of Management Mathematics. p. 403-425. doi:10.1093/imaman/dpad028
- Taylor, J.W. and Bunn, D.W. (1999) A Quantile Regression Approach to Generating Prediction Intervals. Management Science, Vol 45, No 2, pp 225-237.
- Lichtendahl Kenneth C., Jr., Grushka-Cockayne Yael, Winkler Robert L., (2013) Is It Better to Average Probabilities or Quantiles? Management Science 59(7):1594-1611. DOI: doi:10.1287/mnsc.1120.1667

See Also

forecast, es,ssarima, ces, gum

Examples

```
y <- ts(rnorm(100,10,3), frequency=12)

adam(y, h=20, holdout=TRUE)
es(y, h=20, holdout=TRUE)
gum(y, h=20, holdout=TRUE)
auto.ces(y, h=20, holdout=TRUE)
auto.ssarima(y, h=20, holdout=TRUE)
```

smoothCombine *Combination of forecasts of state space models*

Description

Function constructs ETS, SSARIMA, CES, GUM and SMA and combines their forecasts using IC weights.

Usage

```
smoothCombine(y, models = NULL, initial = c("backcasting", "optimal",
  "two-stage", "complete"), ic = c("AICc", "AIC", "BIC", "BICc"),
  loss = c("MSE", "MAE", "HAM", "MSEh", "TMSE", "GTMSE", "MSCE"), h = 10,
  holdout = FALSE, cumulative = FALSE, interval = c("none", "prediction",
  "confidence", "simulated", "approximate", "semiparametric", "nonparametric",
  "empirical", "complete"), level = 0.95, bins = 200,
  intervalCombine = c("quantile", "probability"), bounds = c("usual",
  "admissible", "none"), silent = TRUE, xreg = NULL,
  regressors = c("use", "select"), initialX = NULL, ...)
```

Arguments

y	Vector or ts object, containing data needed to be forecasted.
models	List of the estimated smooth models to use in the combination. If NULL, then all the models are estimated in the function.
initial	Can be "optimal", meaning that the initial states are optimised, or "backcasting", meaning that the initials are produced using backcasting procedure.
ic	The information criterion used in the model selection procedure.
loss	The type of Loss Function used in optimization. loss can be: likelihood (assuming Normal distribution of error term), MSE (Mean Squared Error), MAE (Mean Absolute Error), HAM (Half Absolute Moment), TMSE - Trace Mean Squared Error, GTMSE - Geometric Trace Mean Squared Error, MSEh - optimisation using only h-steps ahead error, MSCE - Mean Squared Cumulative Error. If loss!="MSE", then likelihood and model selection is done based on equivalent MSE. Model selection in this cases becomes not optimal. There are also available analytical approximations for multistep functions: aMSEh, aTMSE and aGTMSE. These can be useful in cases of small samples. Finally, just for fun the absolute and half analogues of multistep estimators are available: MAEh, TMAE, GTMAE, MACE, TMAE, HAMh, THAM, GTHAM, CHAM.
h	Length of forecasting horizon.
holdout	If TRUE, holdout sample of size h is taken from the end of the data.
cumulative	If TRUE, then the cumulative forecast and prediction interval are produced instead of the normal ones. This is useful for inventory control systems.
interval	Type of interval to construct. This can be:

- "none", aka "n" - do not produce prediction interval.
- "parametric", "p" - use state-space structure of ETS. In case of mixed models this is done using simulations, which may take longer time than for the pure additive and pure multiplicative models. This type of interval relies on unbiased estimate of in-sample error variance, which divides the sume of squared errors by T-k rather than just T.
- "likelihood", "l" - these are the same as "p", but relies on the biased estimate of variance from the likelihood (division by T, not by T-k).
- "semiparametric", "sp" - interval based on covariance matrix of 1 to h steps ahead errors and assumption of normal / log-normal distribution (depending on error type).
- "nonparametric", "np" - interval based on values from a quantile regression on error matrix (see Taylor and Bunn, 1999). The model used in this process is e[j] = a j^b, where j=1,..,h.

The parameter also accepts TRUE and FALSE. The former means that parametric interval are constructed, while the latter is equivalent to none. If the forecasts of the models were combined, then the interval are combined quantile-wise (Lichtendahl et al., 2013).

`level` Confidence level. Defines width of prediction interval.

`bins` The number of bins for the prediction interval. The lower value means faster work of the function, but less precise estimates of the quantiles. This needs to be an even number.

`intervalCombine`
How to average the prediction interval: quantile-wise ("quantile") or probability-wise ("probability").

`bounds` What type of bounds to use in the model estimation. The first letter can be used instead of the whole word. "usual" implies restrictions on the smoothing parameter, guaranteeing that the exponential smoothing behaves as an averaging model. "admissible" guarantee that the model is stable.

`silent` accepts TRUE and FALSE. If FALSE, the function will print its progress and produce a plot at the end.

`xreg` The vector (either numeric or time series) or the matrix (or data.frame) of exogenous variables that should be included in the model. If matrix included than columns should contain variables and rows - observations. Note that xreg should have number of observations equal either to in-sample or to the whole series. If the number of observations in xreg is equal to in-sample, then values for the holdout sample are produced using es function.

`regressors` The variable defines what to do with the provided xreg: "use" means that all of the data should be used, while "select" means that a selection using ic should be done.

`initialX` The vector of initial parameters for exogenous variables. Ignored if xreg is NULL.

`...` This currently determines nothing.

- timeElapsed - time elapsed for the construction of the model.
- initialType - type of the initial values used.

- `fitted` - fitted values of ETS.
- `quantiles` - the 3D array of produced quantiles if `interval!="none"` with the dimensions: (number of models) x (bins) x (h).
- `forecast` - point forecast of ETS.
- `lower` - lower bound of prediction interval. When `interval="none"` then NA is returned.
- `upper` - higher bound of prediction interval. When `interval="none"` then NA is returned.
- `residuals` - residuals of the estimated model.
- `s2` - variance of the residuals (taking degrees of freedom into account).
- `interval` - type of interval asked by user.
- `level` - confidence level for interval.
- `cumulative` - whether the produced forecast was cumulative or not.
- `y` - original data.
- `holdout` - holdout part of the original data.
- `xreg` - provided vector or matrix of exogenous variables. If `regressors="s"`, then this value will contain only selected exogenous variables.
- `ICs` - values of information criteria of the model. Includes AIC, AICc, BIC and BICc.
- `accuracy` - vector of accuracy measures for the holdout sample. In case of non-intermittent data includes: MPE, MAPE, SMAPE, MASE, sMAE, RelMAE, sMSE and Bias coefficient (based on complex numbers). In case of intermittent data the set of errors will be: sMSE, sPIS, sCE (scaled cumulative error) and Bias coefficient.

Details

The combination of these models using information criteria weights is possible because they are all formulated in Single Source of Error framework. Due to the the complexity of some of the models, the estimation process may take some time. So be patient.

The prediction interval are combined either probability-wise or quantile-wise (Lichtendahl et al., 2013), which may take extra time, because we need to produce all the distributions for all the models. This can be sped up with the smaller value for bins parameter, but the resulting interval may be imprecise.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov, I., 2023. Smooth Forecasting with the Smooth Package in R. arXiv. doi:10.48550/arXiv.2301.01790
- Snyder, R. D., 1985. Recursive Estimation of Dynamic Linear Models. Journal of the Royal Statistical Society, Series B (Methodological) 47 (2), 272-276.

- Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) Forecasting with exponential smoothing: the state space approach, Springer-Verlag. doi:10.1007/9783540719182.
- Kolassa, S. (2011) Combining exponential smoothing forecasts using Akaike weights. International Journal of Forecasting, 27, pp 238 - 251.
- Svetunkov, I., Boylan, J.E., 2023b. Staying Positive: Challenges and Solutions in Using Pure Multiplicative ETS Models. IMA Journal of Management Mathematics. p. 403-425. doi:10.1093/imaman/dpad028
- Taylor, J.W. and Bunn, D.W. (1999) A Quantile Regression Approach to Generating Prediction Intervals. Management Science, Vol 45, No 2, pp 225-237.
- Lichtendahl Kenneth C., Jr., Grushka-Cockayne Yael, Winkler Robert L., (2013) Is It Better to Average Probabilities or Quantiles? Management Science 59(7):1594-1611. DOI: doi:10.1287/mnsc.1120.1667

See Also

es, auto.ssarima,auto.ces, auto.gum, sma

Examples

```
## Not run: ourModel <- smoothCombine(BJsales,interval="p")
plot(ourModel)
## End(Not run)

# models parameter accepts either previously estimated smoothCombine
# or a manually formed list of smooth models estimated in sample:
## Not run: smoothCombine(BJsales,models=ourModel)

## Not run: models <- list(es(BJsales), sma(BJsales))
smoothCombine(BJsales,models=models)
## End(Not run)
```

---

sowhat *Function returns the ultimate answer to any question*

Description

You need a description? So what?

Usage

```
sowhat(...)
```

Arguments

...	Any number of variables or string with a question.

Details

You need details? So what?

Value

It doesn't return any value, only messages. So what?

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Sowhat?
- Sowhat?
- 42

See Also

Nowwhat (to be implemented),

Examples

```
x <- rnorm(10000,0,1);
sowhat(x);

sowhat("What's the meaning of life?")

sowhat("I don't have a girlfriend.")
```

---

ssarima *State Space ARIMA*

Description

Function constructs State Space ARIMA, estimating AR, MA terms and initial states.

Function selects the best State Space ARIMA based on information criteria, using fancy branch and bound mechanism. The resulting model can be not optimal in IC meaning, but it is usually reasonable.

Function constructs State Space ARIMA, estimating AR, MA terms and initial states.

Usage

```
ssarima(y, orders = list(ar = c(0), i = c(1), ma = c(1)), lags = c(1),
  constant = FALSE, arma = NULL, model = NULL,
  initial = c("backcasting", "optimal", "two-stage", "complete"),
  loss = c("likelihood", "MSE", "MAE", "HAM", "MSEh", "TMSE", "GTMSE",
  "MSCE"), h = 0, holdout = FALSE, bounds = c("admissible", "usual",
  "none"), silent = TRUE, xreg = NULL, regressors = c("use", "select",
  "adapt"), initialX = NULL, ...)

auto.ssarima(y, orders = list(ar = c(3, 3), i = c(2, 1), ma = c(3, 3)),
  lags = c(1, frequency(y)), fast = TRUE, constant = NULL,
  initial = c("backcasting", "optimal", "two-stage", "complete"),
  loss = c("likelihood", "MSE", "MAE", "HAM", "MSEh", "TMSE", "GTMSE",
  "MSCE"), ic = c("AICc", "AIC", "BIC", "BICc"), h = 0, holdout = FALSE,
  bounds = c("admissible", "usual", "none"), silent = TRUE, xreg = NULL,
  regressors = c("use", "select", "adapt"), ...)

ssarima_old(y, orders = list(ar = c(0), i = c(1), ma = c(1)), lags = c(1),
  constant = FALSE, AR = NULL, MA = NULL, initial = c("backcasting",
  "optimal"), ic = c("AICc", "AIC", "BIC", "BICc"), loss = c("likelihood",
  "MSE", "MAE", "HAM", "MSEh", "TMSE", "GTMSE", "MSCE"), h = 10,
  holdout = FALSE, bounds = c("admissible", "none"), silent = c("all",
  "graph", "legend", "output", "none"), xreg = NULL, regressors = c("use",
  "select"), initialX = NULL, ...)
```

Arguments

y	Vector or ts object, containing data needed to be forecasted.
orders	List of orders, containing vector variables ar, i and ma. Example: orders=list(ar=c(1,2),i=c(1),ma= If a variable is not provided in the list, then it is assumed to be equal to zero. At least one variable should have the same length as lags. Another option is to specify orders as a vector of a form orders=c(p,d,q). The non-seasonal ARIMA(p,d,q) is constructed in this case.
lags	Defines lags for the corresponding orders (see examples above). The length of lags must correspond to the length of either ar, i or ma in orders variable. There is no restrictions on the length of lags vector. It is recommended to order lags ascending. The orders are set by a user. If you want the automatic order selection, then use auto.ssarima function instead.
constant	If TRUE, constant term is included in the model. Can also be a number (constant value).
arma	Either the named list or a vector with AR / MA parameters ordered lag-wise. The number of elements should correspond to the specified orders e.g. orders=list(ar=c(1,1),ma=c(1,1)) lags=c(1,4), arma=list(ar=c(0.9,0.8),ma=c(-0.3,0.3))
model	A previously estimated ssarima model, if provided, the function will not estimate anything and will use all its parameters.

initial — Can be either character or a vector of initial states. If it is character, then it can be "optimal", meaning that the initial states are optimised, or "backcasting", meaning that the initials are produced using backcasting procedure.

loss — The type of Loss Function used in optimization. loss can be: likelihood (assuming Normal distribution of error term), MSE (Mean Squared Error), MAE (Mean Absolute Error), HAM (Half Absolute Moment), TMSE - Trace Mean Squared Error, GTMSE - Geometric Trace Mean Squared Error, MSEh - optimisation using only h-steps ahead error, MSCE - Mean Squared Cumulative Error. If loss!="MSE", then likelihood and model selection is done based on equivalent MSE. Model selection in this cases becomes not optimal.

There are also available analytical approximations for multistep functions: aMSEh, aTMSE and aGTMSE. These can be useful in cases of small samples.

Finally, just for fun the absolute and half analogues of multistep estimators are available: MAEh, TMAE, GTMAE, MACE, TMAE, HAMh, THAM, GTHAM, CHAM.

h — Length of forecasting horizon.

holdout — If TRUE, holdout sample of size h is taken from the end of the data.

bounds — What type of bounds to use in the model estimation. The first letter can be used instead of the whole word. In case of ssarima(), the "usual" means restricting AR and MA parameters to lie between -1 and 1.

silent — accepts TRUE and FALSE. If FALSE, the function will print its progress and produce a plot at the end.

xreg — The vector (either numeric or time series) or the matrix (or data.frame) of exogenous variables that should be included in the model. If matrix included than columns should contain variables and rows - observations. Note that xreg should have number of observations equal either to in-sample or to the whole series. If the number of observations in xreg is equal to in-sample, then values for the holdout sample are produced using es function.

regressors — The variable defines what to do with the provided xreg: "use" means that all of the data should be used, while "select" means that a selection using ic should be done.

initialX — The vector of initial parameters for exogenous variables. Ignored if xreg is NULL.

... — Other non-documented parameters.

Parameter model can accept a previously estimated SSARIMA model and use all its parameters.

FI=TRUE will make the function produce Fisher Information matrix, which then can be used to calculated variances of parameters of the model.

fast — If TRUE, then some of the orders of ARIMA are skipped. This is not advised for models with lags greater than 12.

ic — The information criterion used in the model selection procedure.

AR — Vector or matrix of AR parameters. The order of parameters should be lag-wise. This means that first all the AR parameters of the firs lag should be passed, then for the second etc. AR of another ssarima can be passed here.

MA — Vector or matrix of MA parameters. The order of parameters should be lag-wise. This means that first all the MA parameters of the firs lag should be passed, then for the second etc. MA of another ssarima can be passed here.

Details

The model, implemented in this function, is discussed in Svetunkov & Boylan (2019).

The basic ARIMA(p,d,q) used in the function has the following form:

$$(1-B)^d(1-a_1B-a_2B^2-...-a_pB^p)y_{[}t] = (1+b_1B+b_2B^2+...+b_qB^q)\epsilon_{[}t]+c$$

where $y_{[}t]$ is the actual values, $\epsilon_{[}t]$ is the error term, a_i, b_j are the parameters for AR and MA respectively and c is the constant. In case of non-zero differences c acts as drift.

This model is then transformed into ARIMA in the Single Source of Error State space form (proposed in Snyder, 1985):

$$y_t = w' v_{t-l} + \epsilon_t$$

$$v_t = F v_{t-l} + g_t \epsilon_t$$

where v_t is the state vector (defined based on `orders`) and l is the vector of `lags`, w_t is the `measurement` vector (with explanatory variables if provided), F is the `transition` matrix, g_t is the `persistence` vector (which includes explanatory variables if they were used).

Due to the flexibility of the model, multiple seasonalities can be used. For example, something crazy like this can be constructed: SARIMA(1,1,1)(0,1,1)[24](2,0,1)[24*7](0,0,1)[24*30], but the estimation may take a lot of time... If you plan estimating a model with more than one seasonality, it is recommended to use msarima instead.

The model selection for SSARIMA is done by the auto.ssarima function.

For some more information about the model and its implementation, see the vignette: `vignette("ssarima","smooth")`

The function constructs bunch of ARIMAs in Single Source of Error state space form (see ssarima documentation) and selects the best one based on information criterion. The mechanism is described in Svetunkov & Boylan (2019).

Due to the flexibility of the model, multiple seasonalities can be used. For example, something crazy like this can be constructed: SARIMA(1,1,1)(0,1,1)[24](2,0,1)[24*7](0,0,1)[24*30], but the estimation may take a lot of time... It is recommended to use auto.msarima in cases with more than one seasonality and high frequencies.

For some more information about the model and its implementation, see the vignette: `vignette("ssarima","smooth")`

The model, implemented in this function, is discussed in Svetunkov & Boylan (2019).

The basic ARIMA(p,d,q) used in the function has the following form:

$$(1-B)^d(1-a_1B-a_2B^2-...-a_pB^p)y_{[}t] = (1+b_1B+b_2B^2+...+b_qB^q)\epsilon_{[}t]+c$$

where $y_{[}t]$ is the actual values, $\epsilon_{[}t]$ is the error term, a_i, b_j are the parameters for AR and MA respectively and c is the constant. In case of non-zero differences c acts as drift.

This model is then transformed into ARIMA in the Single Source of Error State space form (proposed in Snyder, 1985):

$$y_t = o_t(w' v_{t-l} + x_t a_{t-1} + \epsilon_t)$$

$$v_t = F v_{t-l} + g \epsilon_t$$

$$a_t = F_X a_{t-1} + g_X \epsilon_t / x_t$$

Where o_t is the Bernoulli distributed random variable (in case of normal data equal to 1), v_t is the state vector (defined based on `orders`) and l is the vector of `lags`, x_t is the vector of exogenous parameters. w is the `measurement` vector, F is the `transition` matrix, g is the `persistence`

vector, a_t is the vector of parameters for exogenous variables, F_X is the `transitionX` matrix and g_X is the `persistenceX` matrix.

Due to the flexibility of the model, multiple seasonalities can be used. For example, something crazy like this can be constructed: SARIMA(1,1,1)(0,1,1)[24](2,0,1)[24*7](0,0,1)[24*30], but the estimation may take some finite time... If you plan estimating a model with more than one seasonality, it is recommended to consider doing it using msarima.

The model selection for SSARIMA is done by the auto.ssarima function.

For some more information about the model and its implementation, see the vignette: `vignette("ssarima","smooth")`

Value

Object of class "adam" is returned with similar elements to the adam function.

Object of class "smooth" is returned. See ssarima for details.

Object of class "smooth" is returned. It contains the list of the following values:

- `model` - the name of the estimated model.
- `timeElapsed` - time elapsed for the construction of the model.
- `states` - the matrix of the fuzzy components of ssarima, where `rows` correspond to time and `cols` to states.
- `transition` - matrix F.
- `persistence` - the persistence vector. This is the place, where smoothing parameters live.
- `measurement` - measurement vector of the model.
- `AR` - the matrix of coefficients of AR terms.
- `I` - the matrix of coefficients of I terms.
- `MA` - the matrix of coefficients of MA terms.
- `constant` - the value of the constant term.
- `initialType` - Type of the initial values used.
- `initial` - the initial values of the state vector (extracted from `states`).
- `nParam` - table with the number of estimated / provided parameters. If a previous model was reused, then its initials are reused and the number of provided parameters will take this into account.
- `fitted` - the fitted values.
- `forecast` - the point forecast.
- `lower` - the lower bound of prediction interval. When `interval="none"` then NA is returned.
- `upper` - the higher bound of prediction interval. When `interval="none"` then NA is returned.
- `residuals` - the residuals of the estimated model.
- `errors` - The matrix of 1 to h steps ahead errors. Only returned when the multistep losses are used and semiparametric interval is needed.
- `s2` - variance of the residuals (taking degrees of freedom into account).
- `interval` - type of interval asked by user.
- `level` - confidence level for interval.

- cumulative - whether the produced forecast was cumulative or not.
- y - the original data.
- holdout - the holdout part of the original data.
- xreg - provided vector or matrix of exogenous variables. If regressors="s", then this value will contain only selected exogenous variables.
- initialX - initial values for parameters of exogenous variables.
- ICs - values of information criteria of the model. Includes AIC, AICc, BIC and BICc.
- logLik - log-likelihood of the function.
- lossValue - Cost function value.
- loss - Type of loss function used in the estimation.
- FI - Fisher Information. Equal to NULL if FI=FALSE or when FI is not provided at all.
- accuracy - vector of accuracy measures for the holdout sample. In case of non-intermittent data includes: MPE, MAPE, SMAPE, MASE, sMAE, RelMAE, sMSE and Bias coefficient (based on complex numbers). In case of intermittent data the set of errors will be: sMSE, sPIS, sCE (scaled cumulative error) and Bias coefficient. This is available only when holdout=TRUE.
- B - the vector of all the estimated parameters.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov I. (2023) Smooth forecasting with the smooth package in R. arXiv:2301.01790. doi:10.48550/arXiv.2301.01790.
- Svetunkov I. (2015 - Inf) "smooth" package for R - series of posts about the underlying models and how to use them: https://openforecast.org/category/r-en/smooth/.
- Svetunkov, I., 2023. Smooth Forecasting with the Smooth Package in R. arXiv. doi:10.48550/arXiv.2301.01790
- Snyder, R. D., 1985. Recursive Estimation of Dynamic Linear Models. Journal of the Royal Statistical Society, Series B (Methodological) 47 (2), 272-276.
- Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) Forecasting with exponential smoothing: the state space approach, Springer-Verlag. doi:10.1007/9783540719182.

- Svetunkov, I., 2023. Smooth Forecasting with the Smooth Package in R. arXiv. doi:10.48550/arXiv.2301.01790
- Snyder, R. D., 1985. Recursive Estimation of Dynamic Linear Models. Journal of the Royal Statistical Society, Series B (Methodological) 47 (2), 272-276.
- Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) Forecasting with exponential smoothing: the state space approach, Springer-Verlag. doi:10.1007/9783540719182.

- Svetunkov, I., Boylan, J.E., 2023a. iETS: State Space Model for Intermittent Demand Forecastings. International Journal of Production Economics. 109013. doi:10.1016/j.ijpe.2023.109013

- Teunter R., Syntetos A., Babai Z. (2011). Intermittent demand: Linking forecasting to inventory obsolescence. European Journal of Operational Research, 214, 606-615.
- Croston, J. (1972) Forecasting and stock control for intermittent demands. Operational Research Quarterly, 23(3), 289-303.
- Svetunkov, I., & Boylan, J. E. (2019). State-space ARIMA for supply-chain forecasting. International Journal of Production Research, 0(0), 1–10. doi:10.1080/00207543.2019.1600764
- Svetunkov, I., 2023. Smooth Forecasting with the Smooth Package in R. arXiv. doi:10.48550/arXiv.2301.01790
- Snyder, R. D., 1985. Recursive Estimation of Dynamic Linear Models. Journal of the Royal Statistical Society, Series B (Methodological) 47 (2), 272-276.
- Hyndman, R.J., Koehler, A.B., Ord, J.K., and Snyder, R.D. (2008) Forecasting with exponential smoothing: the state space approach, Springer-Verlag. doi:10.1007/9783540719182.
- Svetunkov, I., & Boylan, J. E. (2019). State-space ARIMA for supply-chain forecasting. International Journal of Production Research, 0(0), 1–10. doi:10.1080/00207543.2019.1600764

See Also

auto.ssarima, auto.msarima, adam, es, ces

es, ces,sim.es, gum, ssarima

auto.ssarima, orders,msarima, auto.msarima,sim.ssarima, adam

Examples

```
# ARIMA(1,1,1) fitted to some data
ourModel <- ssarima(rnorm(118,100,3),orders=list(ar=c(1),i=c(1),ma=c(1)),lags=c(1))

# Model with the same lags and orders, applied to a different data
ssarima(rnorm(118,100,3),orders=orders(ourModel),lags=lags(ourModel))

# The same model applied to a different data
ssarima(rnorm(118,100,3),model=ourModel)

# Example of SARIMA(2,0,0)(1,0,0)[4]
ssarima(rnorm(118,100,3),orders=list(ar=c(2,1)),lags=c(1,4))

# SARIMA(1,1,1)(0,0,1)[4] with different initialisations
ssarima(rnorm(118,100,3),orders=list(ar=c(1),i=c(1),ma=c(1,1)),
        lags=c(1,4),h=18,holdout=TRUE,initial="backcasting")


set.seed(41)
x <- rnorm(118,100,3)

# The best ARIMA for the data
ourModel <- auto.ssarima(x,orders=list(ar=c(2,1),i=c(1,1),ma=c(2,1)),lags=c(1,12),
                                    h=18,holdout=TRUE)
```

```
# The other one using optimised states
auto.ssarima(x,orders=list(ar=c(3,2),i=c(2,1),ma=c(3,2)),lags=c(1,12),
                       initial="two",h=18,holdout=TRUE)

summary(ourModel)
forecast(ourModel)
plot(forecast(ourModel))


# ARIMA(1,1,1) fitted to some data
ourModel <- ssarima_old(rnorm(118,100,3),orders=list(ar=c(1),i=c(1),ma=c(1)),lags=c(1),h=18,
                            holdout=TRUE)

# Model with the same lags and orders, applied to a different data
ssarima_old(rnorm(118,100,3),orders=orders(ourModel),lags=lags(ourModel),h=18,holdout=TRUE)

# The same model applied to a different data
ssarima_old(rnorm(118,100,3),model=ourModel,h=18,holdout=TRUE)

# SARIMA(0,1,1) with exogenous variables
ssarima_old(rnorm(118,100,3),orders=list(i=1,ma=1),h=18,holdout=TRUE,xreg=c(1:118))

summary(ourModel)
forecast(ourModel)
plot(forecast(ourModel))
```

Index