Statistical models underlying functions of "smooth" package for R

Ivan Svetunkov[a,*]

[a]*Lancaster Centre for Forecasting*
Lancaster University Management School, Lancaster, LA1 4YX, UK

Abstract

In this paper we describe statistical models underlying functions of "smooth" package for R and show the connection between each of the parameters in functions and variables in those models. Aside from discussing conventional state-space models, we introduce several new models and discuss their properties. This paper should be useful mainly for people who want to understand what is happening in the core of functions implemented in "smooth" package.

Keywords: Forecasting, state-space models, Exponential smoothing, statistics, smoothing

1. Introduction

The package "smooth" has got its name because it's aim is to gather all the essential smoothing techniques used in forecasting. This includes exponential smoothing, several modifications of it using similar principles and ARIMA which can be considered as a smoothing model in some cases. In fact the thing that unites all the functions in the package is the usage of single source of error state-space model. So probably it would be more appropriate to call the package "state-space models" or something along those lines. But "smooth" should also suffice.

The current version of package is 1.4.6 and it includes the following main functions:

*Correspondance: Ivan Svetunkov, Department of Management Science, Lancaster University Management School, Lancaster, Lancashire, LA1 4YX, UK.

Email address: i.svetunkov@lancaster.ac.uk (Ivan Svetunkov)

1. `es()` – Exponential Smoothing (also known in forecasting society as ETS),
2. `ces()` – Complex Exponential Smoothing,
3. `ges()` – Generalised Exponential Smoothing,
4. `ssarima()` – state-space ARIMA,
5. `auto.ssarima()` – automatic order selection of state-space ARIMA,
6. `auto.ces()` – automatic model selection between seasonal and non-seasonal CES,
7. `sim()`-functions: `sim.es()`. The other functions to be implemented at some point: `sim.ces()`, `sim.ges()`, `sim.ssarima` – functions to simulate data from corresponding statistical models.
8. `simulate()`, `forecast()`, `fitted()`, `summary()` and others – methods applied to objects of class "smooth".

The core code of these functions is written in C++ and that is the reason why the package depends on "Rcpp" and "RcppArmadillo" packages. The functions are optimised using R implementation of C++ library "nlopt" (nloptr package), employing optimisation algorithms BOBYQA and Nelder-Mead.

All the forecasting functions produce objects of class "smooth", allowing to apply such methods as: `coef()`, `summary()`, `plot()`, `forecast()`, `fitted()`, `residuals()`. Some of them also use methods: `orders()`, `lags()` and `modelType()`

Overall functions in this package allow to produce forecasts for large variety of data. This includes normal, intermittent and even high frequency data. All the main functions allow to include exogenous variables and even have a mechanism of adaptation of parameters for exogenous variables. All of this will be discuss in details in this paper. We will also study models underlying all these functions and explain how they are estimated in the package. Finally, we will show advantages of the implemented functions in comparison with existing R functions.

The article has the following structure: Section 2 discusses general state-space model, its statistical properties, how estimation, initialisation model selection is done, how forecasts and prediction intervals are constructed and what lies inside of intermittent part of state-space model. Section 3 discusses function specific models, their statistical properties, state-space forms and function specific parameters. It concludes then with examples of usage. This is then followed by conclusions.

2. SSOE state-space models and their implementation

Before starting discussion about general statistical models, we would like to explain what is a state-space model and why we bother at all. The idea behind any state-space model is that the process that we study consists of two parts: measurable part, that we observe, consisting of some components and unobservable part describing the evolution of those components. For example, a so called "local-level" model can be written as:

$$\begin{aligned} y_t &= l_{t-1} + \epsilon_t \\ l_t &= l_{t-1} + \alpha\epsilon_t \end{aligned}, \tag{1}$$

where y_t is actual value of time series, l_t is level component, ϵ_t is error term and α is some parameter. The model in the form (1) is called Single Source of Error model, because we use one and the same ϵ_t in both equations. There are also Multiple Source of Error models, but they are currently neither discussed nor implemented in "smooth".

What we really say by the equation (1) is that there is a component in time series, l_t which changes in time based on errors obtained on new observations. Think of demand on computers, for example. They have some level of sales. For example, on average a company sells 1000 of these things a day. So l_t will be equal to 1000. However, due to prices on competitors' devices, weather conditions and all the other things happening in life, this number, 1000, will not be stable and probably will change in time. So the level of sales on Monday maybe 1000, while it will slowly slide to 999, then to 995 and so on, if our company does not do something in order to stimulate the demand on their computers. So in real life we are dealing with time varying level of sales. This means that our level component needs to be updated somehow when we have new figures, which leads us to the model (1). In the first equation we say that the actual sales consist of the level yesterday plus some random error. In the second equation we say that our new level for today is formed as a sum of yesterday level plus some portion of that random error that we have observed today.

In a similar manner other state-space models are formed. The important thing to note here is that level is unobservable and so is the second equation in (1), but we make up a rule and give it some form. This form, as we can see later in this paper, is very flexible and has good statistical properties. It is in fact one of the most flexible models: almost any other statistical model can be written in state-space form.

2.1. General state-space model

All the models constructed in the package have the following general state-space form with single source of error (similar to **?**):

$$\begin{aligned} y_t &= o_t \left(w(v_{t-l}) + r(v_{t-l})\epsilon_t \right) \\ v_t &= f(v_{t-l}) + g(v_{t-l})\epsilon_t \end{aligned}, \tag{2}$$

where y_t is actual value of time series, o_t is Bernoulli distributed binary variable (which is equal to 1 when y_t is observed), ϵ_t is error term, v_{t-l} is a state vector, $w(.)$ is measurement function, $r(.)$ is error term function, $f(.)$ is transition function and $g(.)$ is persistence function. Introduction of o_t allows to model intermittent data (this will be discussed later in this paper). In cases of non-intermittent data, $o_t = 1$ for all t.

Although this very general state-space model (2) looks complicated, it is not. We can draw the connection between (2) and previously discussed (1) if we set $o_t = 1$, $v_t = l_t$, $w(v_{t-l}) = l_{t-1}$, $r(v_{t-l}) = 1$, $f(v_{t-l}) = v_{t-1}$, $g(v_{t-l}) = 1$. In some other special cases for some other models, these parameters will have different set of values. That's the flexibility of general SSOE state-space model (2). The first equation of (2) is called "measurement equation" (sometimes "observation equation"), while the second is "transition equation". The name of the former indicates that we measure data using this mathematical formula, while the name of the latter demonstrates that there is some sort of transition between components of state-space model.

Functions $w(\cdot)$, $r(\cdot)$, $f(\cdot)$ and $g(\cdot)$ are taken from **?** and allow to switch between additive and multiplicative components of the state-space model. They are written in C++ and are used by other functions internally.

There are two special cases of (2) that are more often used than all the others: pure additive and pure multiplicative model, which are discussed in details in subsections 2.1.1 and 2.1.2. Currently only ETS model implemented in `es()` function uses non-additive models. All the other models and their corresponding functions in the package use pure additive state-space model only.

We need to note at this point that we use v_{t-l}, where l is a vector of lags, instead of v_{t-1} (**?** use x_{t-1} instead, however we prefer to reserve letter x for exogenous variables). For non-seasonal models, there is no difference between (2) and state-space model of **?**. However while **?** model seasonality using dummy variables, we model seasonal components as lagged variables. This will be discussed in details in section 2.2.

We start this section with discussion of cases when $o_t = 1$ for all t (non-intermittent data). Cases when $o_t \neq 1$ are discussed later in this paper, in section 2.3.

2.1.1. Pure additive state-space model

Pure additive models are easy. The local-level model (1) is pure additive, because there is no multiplication of components in the formulae – there is only addition of level component and error term. These models are very popular, easy to construct and they work very well for many time series.

In pure additive cases general state-space model has the following form:

$$\begin{matrix} y_t = w' v_{t-l} + \epsilon_t \\ v_t = F v_{t-l} + g \epsilon_t \end{matrix}, \tag{3}$$

where w is measurement vector, F is transition matrix, g is persistence vector. In this model it is assumed that error term has normal distribution:

$$\epsilon_t = y_t - \mu_{t|t-1} \sim \mathcal{N}(0, \sigma^2), \tag{4}$$

where $\mu_{t|t-1} = w' v_{t-l}$ is conditional expectation.

The local level model, discussed before, corresponds to exponential smoothing with additive error, no trend and no seasonality, ETS(A,N,N). The connection between (3) and (1) becomes obvious if we set: $w = (1)$, $F = (1)$, $g = \alpha$ and $v_t = l_t$. Other examples of pure additive exponential smoothing are discussed in section 3.1.

Now there are several important statistics that can be derived from (3). These are conditional expectation (which shows expected demand level in the future) and conditional variance (it shows the expected future variability of demand around our expected level). If we can show how to calculate them for (3), then we can use the same formulae for all the other pure additive state-space models.

Both of these statistics for some horizon h can be derived using state-space equation (3) if we substitute indices t with $t + h$:

$$\begin{matrix} y_{t+h} = w' v_{t+h-l} + \epsilon_{t+h} \\ v_{t+h-l} = F v_{t+h-2l} + g \epsilon_{t+h-l} \end{matrix}. \tag{5}$$

Substituting previous values of v_{t+h-2l} in transition equation in (5) leads to the following:

$$v_{t+h-l} = F^2 v_{t+h-3l} + F g \epsilon_{t+h-2l} + g \epsilon_{t+h-l}. \tag{6}$$

Repeating this till we get to v_t leads to:

$$v_{t+h-l} = F^{h-1}v_t + \sum_{j=1}^{h-1} F^{j-1} g \epsilon_{t+h-jl}. \tag{7}$$

Finally inserting (7) into measurement equation of (5) leads to the following equation:

$$y_{t+h} = w' F^{h-1} v_t + \sum_{j=1}^{h-1} w' F^{j-1} g \epsilon_{t+h-jl} + \epsilon_{t+h}. \tag{8}$$

This equation shows that the actual value y_{t+h} consists of two parts: component v_t that changes over time due to its multiplication by F^{h-1} and sum of error terms appearing in demand from observation $t+1$ till $t+h$.

Note here that it is not necessary to say that the model (5) is in any sense "true", underlies or even generates the studied process. All we say is that if we use the structure (5) for decomposition of time series into some set of components v_t, then the future actual value of demand can be calculated using (8).

Now expectation of (8) conditional to value of v_t is (because we assume that error term ϵ_t has mean equal to zero):

$$\mu_{t+h|t} = \mathrm{E}(y_{t+h}|v_t) = w' F^{h-1} v_t. \tag{9}$$

Conditional variance of purely additive model can be calculated using:

$$\begin{aligned} &\sigma^2_{t+h|t} = \mathrm{V}(y_{t+h}|v_t) = \\ &\mathrm{V}\left(w' F^{h-1} v_t + \textstyle\sum_{j=1}^{h-1} w' F^{j-1} g \epsilon_{t+h-jl} + \epsilon_{t+h}\right). \end{aligned} \tag{10}$$

Assuming that errors are not autocorrelated (ϵ_{t+h} does not depend on ϵ_{t+1}) and are homoscedastic (variance of ϵ_{t+1} is the same as for ϵ_{t+h}), equation (10) can be rewritten and simplified to:

$$\sigma^2_{t+h|t} = \sigma^2 \left(1 + \sum_{j=1}^{h-1} (w' F^{j-1} g)^2\right). \tag{11}$$

Using (9), (11) and assumption of normality of error term (4), prediction intervals for pure additive state-space models can be constructed using formula:

$$\mu_{t+h|t} + z_{\alpha/2}\sigma_{t+h|t} < y_{t+h} < \mu_{t+h|t} + z_{1-\alpha/2}\sigma_{t+h|t}, \tag{12}$$

where $z_{\alpha/2}$ and $z_{1-\alpha/2}$ are lower and upper quantiles of normal distribution.

Now we can say that future demand will on average be equal to $\mu_{t+h|t}$ and in $1-\alpha$ percent of cases it should vary in a region defined by prediction intervals (12).

2.1.2. Pure multiplicative state-space model

Pure multiplicative models are in a way more natural than additive ones for many time series that we encounter in forecasting. This is because they restrict actuals by positive values. This is useful, for example, for demand forecasting, because in real life negative demand does not make much sense. However in cases with high level of sales, the difference between additive and multiplicative models is negligible and it is much easier to work with former, than with latter. However, there are cases, where pure multiplicative models cannot be substituted by other types. An example of such cases is intermittent demand, which is discussed in section 2.3.2. That is why we need to understand properties of pure multiplicative models.

Pure multiplicative state-space model in our implementation has the following form:

$$\begin{aligned} y_t &= \exp\left(w' \log v_{t-l} + \log(1+\epsilon_t)\right) \\ v_t &= \exp(F \log(v_{t-l}) + \log(1+g\epsilon_t)) \end{aligned}. \tag{13}$$

This linearisation of components using logarithms allows to introduce their multiplication and results in exactly the same models as in **?**. However, we do not assume that error term is distributed normally, but make another assumption – that it has log-normal distribution:

$$(1+\epsilon_t) = \frac{y_t}{\mu_t} \sim \log \mathcal{N}(0, \sigma^2). \tag{14}$$

This means that $\log(1+\epsilon_t)$ in (13) has normal distribution. The form (13) simplifies some derivations and allows construction of non-symmetric intervals in cases, when they are needed.

An example of model (13) is model with multiplicative error and trend, ETS(M,M,N):

$$\begin{aligned} y_t &= l_{t-1} b_{t-1} (1+\epsilon_t) \\ l_t &= l_{t-1} b_{t-1} (1+\alpha\epsilon_t). \\ b_t &= b_{t-1} (1+\beta\epsilon_t) \end{aligned} \tag{15}$$

This model has the following values:

$w = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, $F = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$, $g = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$ and $v_t = \begin{pmatrix} l_t \\ b_t \end{pmatrix}$.

Other examples of pure multiplicative exponential smoothing will be discussed in section 3.1.

In order to produce conditional expectation and variance, we linearise (13) model using natural logarithm:

$$\begin{aligned} \log y_t &= w' \log v_{t-l} + \log(1+\epsilon_t) \\ \log v_t &= F \log(v_{t-l}) + \log(1+g\epsilon_t) \end{aligned}. \tag{16}$$

This model now resembles pure additive model (3), but has some features, one of which is $\log(1+g\epsilon_t)$ instead of just $g\epsilon_t$. This brings some complications in derivations.

In order to calculate conditional expectation and variance, we need to come up with a formula similar to (8):

$$\log y_{t+h} = w' F^{h-1} \log v_t + \sum_{j=1}^{h-1} w' F^{j-1} \log(1+g\epsilon_{t+h-jl}) + \log(1+\epsilon_{t+h}). \tag{17}$$

Now we can take conditional expectation and variance. The former is equal to:

$$\tilde{\mu}_{t+h|t} = \mathrm{E}(\log y_{t+h} | \log v_t) = w' F^{h-1} \log v_t, \tag{18}$$

while the latter is:

$$\begin{aligned} \tilde{\sigma}^2_{t+h|t} = \quad & \mathrm{V}(\log y_{t+h} | \log v_t) = \\ & \sigma^2 \left(1 + \textstyle\sum_{j=1}^{h-1} w' F^{j-1} S_g (F^{j-1})' w\right), \end{aligned} \tag{19}$$

where S_g is covariance matrix calculated as:

$$Sg = \frac{1}{T} \log(1+g\epsilon_t)(\log(1+g\epsilon_t))', \tag{20}$$

Note however that we have taken conditional values of $\log y_{t+h}$ rather than y_{t+h}. We can construct prediction intervals using:

$$\tilde{\mu}_{t+h|t} + z_{\alpha/2} \tilde{\sigma}^2_{t+h|t} < \log(y_{t+h}) < \tilde{\mu}_{t+h|t} + z_{1-\alpha/2} \tilde{\sigma}^2_{t+h|t} \tag{21}$$

and then, after taking exponent of (21), we can return to original scale:

$$\exp(\tilde{\mu}_{t+h|t} + z_{\alpha/2} \tilde{\sigma}^2_{t+h|t}) < y_{t+h} < \exp(\tilde{\mu}_{t+h|t} + z_{1-\alpha/2} \tilde{\sigma}^2_{t+h|t}). \tag{22}$$

Conditional expectation of actual values in normal scale can then be calculated using simple transformation:

$$\mu_{t+h|t} = \exp\left(\tilde{\mu}_{t+h|t}\right), \tag{23}$$

This method of calculation of conditional values and prediction intervals construction works perfectly fine in cases when variance of error is small (for example, smaller than 0.1), because $\log(1+\epsilon_t)$ in this case will be approximately equal to ϵ_t. However, there are special cases with high variance, where some additional transformations are needed in order to obtain correct estimates of mean and variance. They are discussed in section 2.3.

2.1.3. Mixed models

Mixed models represent cases when some components are multiplied, while the others add up. ETS(M,A,M) is an example of such models. It is written as:

$$\begin{matrix} y_t = (l_{t-1} + b_{t-1}) s_{t-m} (1+\epsilon_t) \\ l_t = (l_{t-1} + b_{t-1})(1+\alpha\epsilon_t) \\ b_t = b_{t-1} + (l_{t-1} + b_{t-1})\beta\epsilon_t \\ s_t = s_{t-m}(1+\gamma\epsilon_t) \end{matrix} . \tag{24}$$

Already by looking at (24) it becomes obvious that this is not a simple model to work with: mechanism of adaptation of trend component differs from similar mechanisms for other components. This introduces some changes in the structure of the model and means that neither (3) nor (13) can be used for calculation of conditional expectation and variance.

Due to this complication, conditional expectations for these models are currently taken from **?**, so they are not discussed here. As for conditional variances, they are currently not calculated properly for mixed models. Variances for models with multiplicative errors and non-multiplicative other components (for example, ETS(M,A,A)) can be approximated by variances of similar pure additive models (for example, ETS(A,A,A)). This can be done only when variance of error term is small (smaller than 0.1), which is very common for these models if data is not intermittent. In all the other cases in order to produce prediction intervals simulations can be used.

The simulations for mixed models in "smooth" package are done using `sim.es()` function (discussed in section 3.6). As a result of this 10000 samples with possible forecast trajectories are produced, after which quantiles of data are taken for each horizon using `quantile()` function in R ("stats" package). This method works very well for non-intermittent data and is currently

used by default as "parametric" method of prediction intervals construction for these models.

2.2. Estimation of the model

In order to construct a state-space model and estimate its parameters, vector of states needs to be initialised somehow. There are several possible ways of initialisation of state-space models. We initialise them before the sample starts, which means that v_t should be defined for $t < 1$. This way the calculations (update of components using transition equation and one step ahead forecasts using measurement equation) can start from the first available actual value y_1.

Note. In R states contain matrix with observations in rows and components in columns. So when a model produces state vector, it will have number of rows, covering the initialisation part, number of observations and forecast horizon. So, for example, ETS(A,A,A) fitted on 36 observations of monthly data with forecast horizon of 18 will have $36 + 12 + 18 = 66$ observations of state vector. However there is an exception in this rule: if forecast horizon is lower than frequency of the data, than the number of rows will be calculated as sum of number of observations and two frequencies of the data. So, for example, the very same ETS(A,A,A) on the same data but with horizon of 6 instead of 18 will have $36 + 12 + 12 = 60$ observations of state vector.

The structure of state vector in our implementation differs from conventional structures (for example, in **?**). This needs to be explained in details.

2.2.1. State vector

Any state vector in our implementation can be represented as:

$$v_{t-l} = Lv_t = \begin{pmatrix} B^{m_1} & 0 & \dots & 0 \\ 0 & B^{m_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & B^{m_k} \end{pmatrix} v_t, \tag{25}$$

where L is matrix of lags, B is back shift operator (variable satisfying equation $y_{t-1} = By_t$) and m_1, m_2, ..., m_k are lags of components. For example, ETS(A,A,A) for monthly data will have the following matrix of lags:

$$L = \begin{pmatrix} B & 0 & 0 \\ 0 & B & 0 \\ 0 & 0 & B^{12} \end{pmatrix},$$

meaning that first and second components (level and trend) are taken with lag one, while the last one (seasonal) is taken with lag 12. State vector for this model is usually set as $v_t' = \begin{pmatrix} l_t & b_t & s_t \end{pmatrix}$. If we multiply it by the matrix of lags we will have:

$$v_{t-l} = Lv_t = \begin{pmatrix} B & 0 & 0 \\ 0 & B & 0 \\ 0 & 0 & B^{12} \end{pmatrix} \begin{pmatrix} l_t \\ b_t \\ s_t \end{pmatrix} = \begin{pmatrix} Bl_t \\ Bb_t \\ B^{12}s_t \end{pmatrix} = \begin{pmatrix} l_{t-1} \\ b_{t-1} \\ s_{t-12} \end{pmatrix}$$

Inserting this value into pure additive state-space model (3), leads to the well-known system of equations for ETS(A,A,A):

$$\begin{array}{l} y_t = l_{t-1} + b_{t-1} + s_{t-m} + \epsilon_t \\ l_t = l_{t-1} + b_{t-1} + \alpha\epsilon_t \\ b_t = b_{t-1} + \beta\epsilon_t \\ s_t = s_{t-m} + \gamma\epsilon_t \end{array}, \tag{26}$$

where $m = 12$ in our example.

Note that due to a different structure of state vector (in comparison with **?**) we also define transition matrix and measurement vector differently: $F = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, $w = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$.

2.2.2. Initialisation of state vector, `initial`

Taking this feature of state vector into account, its initialisation should start not on observation $t = 0$, but on observation $t = 1 - m$, where m is the maximum lag in the model. So the initialisation part of state vector takes m observations. For components with lag 1, only $t = 0$ is needed, all the other observations are unimportant and can be substituted by values on observation $t = 0$. For models with multiple seasonal patterns, each j-th component will take places from $t = 1 - m_j$ to $t = 0$, where m_j is lag for respective component.

"smooth" package allows to initialise state vector in several ways:

1. Optimisation. This means that the initial values are set by optimiser during the estimation of model along with smoothing parameters. This initialisation method works very well for simple models with no more than approximately 20 parameters. Otherwise optimisation may take too much time and may also result in not accurate estimates of parameters.

2. Backcasting. This is iterative procedure. On first iteration initial values are set heuristically. The model then is fitted till the end of data, after which on additional m observations of state vector are calculated using transition equation only and taking that $\epsilon_t = 0$. Then the same model is fitted in reverse order till the very first observations and finally initial values are produced using transition equation and the same $\epsilon_t = 0$. The process repeats several times (4 times in current implementation) in order to obtain stable estimates of initial values.
   For example, model ETS(A,A,A) fitted to the data in reverse will have form:
   $$\begin{matrix} y_t = l_{t+1} + b_{t+1} + s_{t+m} + \epsilon_t \\ l_t = l_{t+1} + b_{t+1} + \alpha\epsilon_t \\ b_t = b_{t+1} + \beta\epsilon_t \\ s_t = s_{t+m} + \gamma\epsilon_t \end{matrix} , \tag{27}$$
   while in the tails of the data the following set of equations is used for the components:
   $$\begin{matrix} l_t = l_{t-1} + b_{t-1} \\ b_t = b_{t-1} \\ s_t = s_{t-m} \end{matrix} \text{ and } \begin{matrix} l_t = l_{t+1} + b_{t+1} \\ , b_t = b_{t+1} \\ s_t = s_{t+m} \end{matrix} . \tag{28}$$
   Backcasting asymptotically converges to least squares estimates of parameters and is advised for models with high number of parameters. Examples of such models include seasonal models on weekly data or multi-seasonal models for high frequency data. Optimisation in these cases may become very slow and less efficient than for smaller models. Backcasting however produces decent estimates of initials in both cases of small and large models.
3. Preset values. It is also possible to define initial parameters using some heuristics or use estimated parameters from some previous model. This can be useful when one and the same model needs to be applied to different sample sizes of the same data or when some heuristic method of initialisation needs to be tested.

In R `es()`, `ces()`, `ges()`, `ssarima()` and `auto()` functions allow to make selection between these values. This is done using parameter `initial`, which can accept either `"optimal"` (or `"o"`), or `"backcasting"` (`"b"`) or vector of initial values. `auto()` functions however accept only first two options.

2.2.3. Cost functions, `cfType`

In order to estimate parameters with selected initialisation method, residuals of the model should be calculated. They are then used in some selected objective function. Residuals for model with additive errors are estimated using:

$$e_{t+1|t} = y_{t+1} - \mu_{t+1|t}, \tag{29}$$

while for multiplicative:

$$\tilde{e}_{t+1|t} = \frac{y_{t+1} - \mu_{t+1|t}}{\mu_{t+1|t}}, \tag{30}$$

where $\mu_{t+1|t}$ is one-step-ahead forecast.

However during optimisation of models with multiplicative errors term $e_{t+1|t} = \log(1+\tilde{e}_{t+1|t})$ is used instead of (30) due to assumption of log-normal distribution of residuals ϵ_t.

Instead of calculating one-step-ahead forecasts only, we can produce forecasts for h steps ahead from each observation t, $\mu_{t+h|t}$ and calculate conditional errors $e_{t+h|t} = y_{t+h} - \mu_{t+h|t}$. This then can be used by some cost functions for optimisation. But note the difference between $e_{t+h|t}$ and ϵ_{t+h}. Using pure additive model (3) we can show that:

$$\begin{aligned} e_{t+h|t} = \quad & y_{t+h|t} - \mu_{t+h|t} = \\ & w'F^{h-1}v_t + \sum_{j=1}^{h-1} w'F^{j-1}g\epsilon_{t+h-jl} + \epsilon_{t+h} - w'F^{h-1}v_t = \\ & \sum_{j=1}^{h-1} w'F^{j-1}g\epsilon_{t+h-jl} + \epsilon_{t+h} \end{aligned}. \tag{31}$$

So several steps ahead error $e_{t+h|t}$ includes ϵ_{t+h}. This also means that while there is an assumption about independence of ϵ_{t+i} and ϵ_{t+j} for different i and j (no autocorrelation in the model), there is always a correlation between $e_{t+i|t}$ and $e_{t+j|t}$ for any i and j, unless $w'F^{j-1}g = 0$ in (31). This can happen, for example, when all the smoothing parameters are equal to zero.

The following objective functions are available in "smooth" for parameters estimation:

1. `"MSE"` – Mean Squared Error. This is calculated as:

$$\text{MSE} = \frac{1}{T}\sum_{t=1}^{T} e_{t+1|t}^2, \tag{32}$$

where T is number of observations in sample. It can be shown that minimisation of MSE leads to the same estimates of parameters as likelihood maximisation in case when residuals are distributed normally (or log-normally in cases of multiplicative error term). MSE is known to produce mean estimates of parameters. This is the default objective function for all the forecast functions.

2. `"MAE"` – Mean Absolute Error:

$$\text{MAE} = \frac{1}{T}\sum_{t=1}^{T} |e_{t+1|t}|. \tag{33}$$

Due to its robustness to outliers (because we do not square errors), this cost function is said to produce median estimates of parameters.

3. `"HAM"` – Half Absolute Moment:

$$\text{HAM} = \frac{1}{T}\sum_{t=1}^{T} \sqrt{|e_{t+1|t}|}. \tag{34}$$

This is even more robust estimator, which produces mode estimates in cases of count data and close to mode in continuous data. This is the fastest (in terms of optimisation time) estimator, because of the square root function that allows to reach minimum of the function fast.

4. `"MSEh"` – Mean Squared h steps ahead Error:

$$\text{MSE}_h = \frac{1}{T}\sum_{t=1}^{T} e_{t+h|t}^2. \tag{35}$$

This is one of the cost functions using multiple steps ahead errors (31). This cost function leads to shrinkage of parameters of models (excluding initial states though). This means that with increase of h, smoothing parameters will tend to become closer to zero, making underlying state-space model deterministic. **Reference on TFL paper here!**

5. `"MSTFE"` – Mean Squared Trace Forecast Error:

$$\text{MSTFE} = \sum_{j=1}^{h}\frac{1}{T}\sum_{t=1}^{T} e_{t+j|t}^2. \tag{36}$$

The term "trace forecast" means that values are produced for 1 to h steps ahead. This cost function also shrinks parameters but slower than MSE_h. **Reference on TFL paper here!**

6. `"MLSTFE"` – Mean Logarithmic Squared Trace Forecast Error.

$$\text{MLSTFE} = \sum_{j=1}^{h} \log\left(\frac{1}{T}\sum_{t=1}^{T} e_{t+j|t}^2\right). \tag{37}$$

This cost function allows to bring variances of conditional errors for different horizons to one level, so one-step-ahead error becomes as important as h steps ahead one. This means that shrinkage effect in (37) is weaker than in MSE_h and MSTFE. **Reference on TFL paper here!**

Cost functions (4) – (6) take more optimisation time than (1) – (3), because trace forecasts need to be produced for each observation. They are advised in cases of high frequency data and large sample sizes (1000 observations and more) because of the shrinkage effect, allowing models to become more robust. However calculations in this case will take substantial amount of time. So there are also analytical analogues of cost functions (4) – (6), which can be used either on small samples or in cases when time of estimation is crucial. For example, `"aMSTFE"` produces estimates asymptotically similar to MSTFE, but takes less calculation time, because it is based on statistical properties of state-space model and MSTFE rather than generation of trace forecasts. In order to use these analytical cost functions, user needs to add "a" in front of names. For example, `cfType="aMSEh"` will produce analytical Mean Squared h steps ahead Error.

2.2.4. Parameters space, `bounds`

In general state-space model does not need restrictions on parameter space and anything can be constructed with any parameters. However in forecasting we prefer for models to guarantee that newer observations would have at least not smaller weight than older ones. Situations when some past sale, that happened several years ago, would determine forecast for tomorrow is ridiculous. So in order to have this property we need to make sure that state-space models we estimate are "stable" or "invertible" (the latter is term from ARIMA models, but they are the same). This is in fact the most important property: our state-space models may be non-stationary and can produce variety of forecasting trajectories, depending on selected model and parameters used, but it needs to guarantee that the importance of old information deflates.

So here comes question of what bounds to use for parameters of models.

There are several solutions to this question and several options in "smooth" that correspond to them.

There is a variable `bounds` that can be either `"admissible"`, `"none"` or `"usual"` (for `es()` only). `bounds="a"` guarantee that the estimated model is stable. This is the default value for all the functions except for `es()`. In case of `ssarima()` this value of parameter `bounds` will also guarantee that estimated ARMA is stationary.

If `bounds="n"`, then there is no restriction on parameter space. This is not advised, because models may become unstable (there will be a warning if they do), but this can be used for exploration purposes.

Finally, `es()` function has `bounds="u"`, which ensures that parameter space is restricted in a way, that the resulting exponential smoothing model has a property of "averaging" model (see `?` for details). However it should be noted that if smoothing parameters reach boundary values (for example, α becomes equal to 1), then it is advised to use admissible bounds instead. The reason for that is because an artificial restriction of parameter space by usual bounds may lead to underestimation of uncertainty, which leads to narrower than needed prediction intervals.

2.2.5. Optimisation

Taking into account selected initialisation method, cost function and parameter space restriction, we can find appropriate values that will lead to the lowest value of the objective. This is done via optimisation in two steps. On the first step "BOBYQA" algorithm is used with maximum of 1000 evaluations and relative tolerance of 1e-8. On the second step the parameters are optimised further on using "Nelder-Mead" algorithm with maximum of 1000 evaluations and relative tolerance of 1e-6. In both steps all the parameters are optimised at the same time. In case of `es()` function there may be a step between these two, when on the first step optimiser returns exactly the same parameters as the provided to it in the beginning. This may happen for some mixed state-space models. In this case "BOBYQA" is used again but with other initials.

This two-steps optimisation allows obtaining good estimates of parameters even for large models. While "BOBYQA" gets close to global minimum of cost function, "Nelder-Mead" allows estimating it with higher precision.

2.3. Intermittent data

There is no good, concise and full definition of what is intermittent data. The general property of it is presence of natural zero values, caused by some objective reasons. For example, in case of demand forecasting the reason can be just the absence of demand on the product. A typical example of intermittent demand process is demand on aeroplanes: the daily data of demand on such product will contain a lot of zeroes, because this is not a product of daily consumption.

Standard models do not work in case of intermittent data, so special methods have been developed. Professor John Boylan and I have managed to derive a state-space model, that underlies those methods and showed flexibility of this model. This is done via introduction of binary variable o_t corresponding to data occurrences. This allows using any state-space model for intermittent data.

The main assumption about o_t is that it is distributed Bernoulli with some probability p_t that can vary in time. There are several ways of modelling this probability (for example, **?**, **?**), and they are implemented in "smooth". They have been developed mainly for intermittent demand, however principles described in this section and the implementation in the package can be applied to other types of data with meaningful zeroes. For example, forecasting of solar irradiation data can also benefit from this model.

Occurrence variable is introduced in measurement equation only: $y_t = o_t \left(w(v_{t-l}) + r(v_{t-l})\epsilon_t\right)$, but it allows using all the available state-space models (additive, multiplicative and mixed). This is important, because it has been argued in several papers (**??**) that statistical model for intermittent data must be multiplicative (in order not to have negative values), but noone has figured out before how to use such a model with series with zeroes.

In "smooth" all the state-space models constructed for intermittent data have a name starting with letter "i". For example, iETS(M,N,N) means that ETS(M,N,N) was constructed for intermittent data.

General properties of intermittent state-space model correspond to the ones discussed in section 2. However conditional expectation and variance need to be calculated differently in order to take future probability of occurrence into account.

2.3.1. Pure additive intermittent state-space models

Pure additive intermittent state-space model can be applied only for data with both positive an negative values. This can be, for example, a series of

changes in sales (so differenced data). Although, there is not many examples of such series in business and this model is now a bad substitution of multiplicative model (because of low level of data), pure additive intermittent state-space model still can be useful in some situations.

This model has the following form:

$$\begin{matrix} y_t = o_t \left(w' v_{t-l} + \epsilon_t \right) \\ v_t = F v_{t-l} + g \epsilon_t \end{matrix}, \tag{38}$$

where $o_t \sim \text{Bernoulli}(p_t)$.

What this model assumes is that states may evolve in time even when we do not observe sales. States in this case represent desire of customers to buy a product, which may change in time for different set of reasons. This introduction of o_t does not change properties of the underlying pure additive model substantially, however there are some differences.

First of all measurement equation for h steps ahead (as discussed in section 2.1.1) is modified for intermittent data to:

$$y_{t+h} = o_{t+h} \left(w' F^{h-1} v_t + \sum_{j=1}^{h-1} w' F^{j-1} g \epsilon_{t+h-jl} + \epsilon_{t+h} \right). \tag{39}$$

It is fair to assume that occurrences are not correlated with sizes of data (if this assumption does not hold, then another formula, not discussed here, should be used), so conditional expectation of (39) is:

$$\mu_{t+h|t} = \text{E}(y_{t+h|t}) = \text{E}(o_{t+h|t}) \left(w' F^{h-1} v_t \right) = \mu_{p,t+h|t} \left(w' F^{h-1} v_t \right), \tag{40}$$

where $\mu_{p,t+h|t}$ is conditional expectation of probability of non-zero values occurrence. The conditional expectation (40) means that future demand level is not constant and can be interpreted as average level of sales over a time unit.

As for conditional variance, the formula of variance of product of two independent random variables a and b can be used in order to derive it:

$$\text{V}(ab) = \text{V}(a)\text{V}(b) + \text{V}(a)\text{E}(b)^2 + \text{E}(a)^2\text{V}(b). \tag{41}$$

Variance of (39) is then:

$$\begin{matrix} \sigma^2_{t+h|t} = & \sigma^2 \left(1 + \sum_{j=1}^{h-1} (w' F^{j-1} g)^2 \right) \left(\sigma^2_{p,t+h|t} + \mu^2_{p,t+h|t} \right) + \\ & \left(w' F^{h-1} v_t \right)^2 \sigma^2_{p,t+h|t} \end{matrix}, \tag{42}$$

where $\sigma^2_{p,t+h|t}$ is conditional variance of probability p_t. For Bernoulli distribution this variance will be equal to:

$$\sigma^2_{p,t+h|t} = \mu_{p,t+h|t}(1 - \mu_{p,t+h|t}). \tag{43}$$

Note that in case of continuous data $p_t = 1$ for all t, so the conditional expectation (40) transforms into (9), while conditional variance (42) becomes equal to (10).

After that prediction intervals can easily be constructed using the conventional formula:

$$\mu_{t+h|t} + z_{\alpha/2}\sigma_{t+h|t} < y_{t+h} < \mu_{t+h|t} + z_{1-\alpha/2}\sigma_{t+h|t}, \tag{44}$$

2.3.2. Pure multiplicative intermittent state-space models

As it was mentioned above pure multiplicative models make more sense for intermittent demand and can be considered as more natural than additive or mixed ones. In our model it can be written as:

$$\begin{matrix} y_t = o_t \exp\left(w' \log v_{t-l} + \log(1 + \epsilon_t)\right) \\ v_t = \exp(F \log(v_{t-l}) + \log(1 + g\epsilon_t)) \end{matrix}, \tag{45}$$

where once again $o_t \sim \text{Bernoulli}(p_t)$.

Measurement equation for pure multiplicative models is:

$$y_{t+h} = o_{t+h} \exp\left(w' F^{h-1} \log v_t + \sum_{j=1}^{h-1} w' F^{j-1} \log(1 + g\epsilon_{t+h-jl}) + \log(1 + \epsilon_{t+h})\right). \tag{46}$$

Conditional expectation of (46) is tricky because of connection between mean of normal and mean of log-normal distributions. However conditional median is much easier. From the formula: $\text{Md}(\exp(\tilde{y}_t)) = \exp(\text{E}(\tilde{y}_t))$ – it follows that:

$$\text{Md}(y_{t+h|t}) = \mu_{p,t+h|t} \exp\left(w' F^{h-1} \log v_t\right) = \mu_{p,t+h|t} \exp\left(\tilde{\mu}_{t+h|t}\right). \tag{47}$$

This follows from assumption of log-normal distribution of error term in multiplicative models.

If we really need conditional expectation then following formula should be used instead:

$$\mu_{t+h|t} = \mu_{p,t+h|t} \exp\left(\tilde{\mu}_{t+h|t} + \frac{\tilde{\sigma}^2_{t+h|t}}{2}\right), \tag{48}$$

where $\tilde{\mu}_{t+h|t}$ and $\tilde{\sigma}^2_{t+h|t}$ are conditional mean and variance discussed in section 2.1.2. Note that the smaller conditional variance $\tilde{\sigma}^2_{t+h|t}$ is, the closer final conditional mean and median are to each other. However in cases with intermittent data, variance can reach much higher values than for continuous data (for example, higher than one), so the modification (48) may be necessary if mean is needed rather than median. However we do not make this modification and return **median** instead of mean.

Now calculation of variance for original values of y_t is connected with some complications and does not allow to construct prediction intervals, because in order to find out quantiles of log-normal distribution we would still need to know mean and standard deviation of $\log y_t$. This means that we would need to switch from normal to log-normal distribution and back to normal again several times in order to calculate prediction intervals. So instead of using the whole conditional distribution of y_{t+h} we analyse distribution of residuals only and use that in prediction intervals construction.

First, we find quantiles of log-normal distribution $\tilde{q}_{h,\alpha/2}$ and $\tilde{q}_{h,1-\alpha/2}$ for each h based on $\tilde{\sigma}^2_{t+h|t}$ and zero mean, calculated using formulae from section 2.1.2.

Second, using formula for connection of variance of log-normal variable with parameters of normal distribution (taking expectation equal to zero):

$$\hat{\sigma}^2_{t+h|t} = \left(\exp\left(\tilde{\sigma}^2_{t+h|t}\right) - 1\right) \exp\left(\tilde{\sigma}^2_{t+h|t}\right), \tag{49}$$

we calculate variance $\hat{\sigma}^2_{t+h|t}$ for log-normal distribution. After that we standardise the quantiles produced on the first step using formula:

$$q_h = \frac{(\tilde{q}_h - 1)}{\hat{\sigma}_{t+h|t}} \tag{50}$$

Then we take variance of occurrences (similar to (42)) into account by modifying variances $\hat{\sigma}^2_{t+h|t}$:

$$\sigma^2_{t+h|t} = \hat{\sigma}^2_{t+h|t}\left(\sigma^2_{p,t+h|t} + \mu^2_{p,t+h|t}\right) + \sigma^2_{p,t+h|t}. \tag{51}$$

Finally we produce prediction intervals using (48) and (51):

$$\mu_{t+h|t}(1 - q_{h,\alpha/2}\sigma_{t+h|t}) < y_{t+h} < \mu_{t+h|t}(1 + q_{h,1-\alpha/2}\sigma_{t+h|t}) \tag{52}$$

2.3.3. Mixed intermittent state-space models

As noted in section 2.1.3 prediction intervals for mixed models are currently done using simulations. However estimates of quantiles of simulated values may be incorrect in case of intermittent data. This is because we do not take probability into account in the same way as for pure additive and pure multiplicative models. So it is recommended to use other types of prediction intervals for mixed models (for example, "semiparameteric" and "non-parameteric"). They are discussed in section 2.6.

Now that we have covered size of data parts of intermittent state-space model, we can discuss the occurrences part. There are several ways to model and forecast probability of occurrences. We will start from the simplest one.

2.3.4. Model with fixed probability

This is the simplest model. We assume that probability of occurrences is fixed and does not vary in time, meaning that $\mathrm{E}(p_t) = \mu_{p,t+h|t} = p$ for all t. Expectation for additive model transforms in this case into:

$$\mu_{t+h|t} = \mathrm{E}(y_{t+h|t}) = p \left(w' F^{h-1} v_t \right), \tag{53}$$

while for multiplicative it is:

$$\mu_{t+h|t} = \mathrm{E}(y_{t+h|t}) = p \exp \left(w' F^{h-1} \log v_t \right). \tag{54}$$

Similarly substituting variance of occurrences in (42) and (51) by $\sigma^2_{p,t+h|t} = p(1-p)$ allows calculating conditional variances.

Probability p can be estimated using likelihood, discussed in section 2.4. It can be shown that it is equal to:

$$p = \frac{T_1}{T}, \tag{55}$$

where T_1 is number of non-zero observations and T is number of all the observations.

In R this model can be called using `intermittent="fixed"`.

2.3.5. Croston's model

In the original Croston's method it is assumed that probability varies in time and changes in reverse proportion to number of zero observations between demand occurrences:

$$p_t = \frac{1}{q_t}, \tag{56}$$

where q_t is number of zeroes between demands. Calculation of q_t starts from 1 (for consequent demands $q_t = 1$). The probability of occurrence in this case is updated on each non-zero observation, so there is some sort of stepwise change of p_t in the method. The longer periods without non-zero values lead to lower probability of having demand at all.

In the original paper of **?** simple exponential smoothing (SES) is used for forecasting of q_{t+h}. **?** show that SES has two underlying statistical model: ETS(A,N,N) and ETS(M,N,N). Taking that $p_t \in [0, 1]$, multiplicative model for q_t should be preferred, because in this case q_t is bounded by values greater than zero. So the model underlying occurrences in Croston's method can be formulated as:

$$\begin{aligned} q_t &= l_{q,t-1}(1 + \varepsilon_t) \\ l_{q,t} &= l_{q,t-1}(1 + \alpha_q \varepsilon_t) \end{aligned}. \tag{57}$$

This is pure multiplicative model with $1 + \varepsilon_t$ distributed log-normally, so its properties have already been discussed in section 2.1.2.

Conditional expectation of the probability p_{t+h} however depends on $\frac{1}{q_{t+h}}$ and as was shown in **?** should be calculated using the following formula:

$$\mu_{p,t+h|t} = \left(1 - \frac{\alpha_q}{2}\right) \frac{1}{\mu_{q,t+h|t}}, \tag{58}$$

where $\mu_{q,t+h|t}$ is conditional expectation of the model (57) for h steps ahead. Conditional variance of Croston's model in this case (taking into account (43)) becomes:

$$\sigma^2_{p,t+h|t} = \mu_{p,t+h|t}(1 - \mu_{p,t+h|t}) = \left(1 - \frac{\alpha_q}{2}\right) \frac{\mu_{q,t+h|t} - \left(1 - \frac{\alpha_q}{2}\right)}{\mu^2_{q,t+h|t}}. \tag{59}$$

Values (58) and (59) can then be used for the final point and interval forecasts construction.

In R this model can be constructed using `intermittent="croston"`. The model (57) is then constructed using `es()` function.

2.3.6. TSB model

? proposed a different approach to probability of occurrences estimation. Their method is usually called "TSB" and it uses SES for probability estimation and update. In our case ETS(M,N,N) once again makes more sense, for the same reasons as in Croston's model:

$$\begin{aligned} p_t &= l_{p,t-1}(1 + \xi_t) \\ l_{p,t} &= l_{p,t-1}(1 + \alpha_p \xi_t) \end{aligned}. \tag{60}$$

However assumptions about distribution of p_t need to be different, log-normal distribution is bounded only from one side, which may lead to strange situations with p_t becoming greater than one. So there should be used some other distributions bounded by [0, 1] region. We prefer Beta distribution, because it is flexible enough to cover different possible distributions of probability. Note however that Beta distribution does not allow random variable to take boundary values (0 and 1), so we transform the actual probability the following way:

$$\begin{aligned} p'_t &= l_{p,t-1}(1 + \xi_t) \\ l_{p,t} &= l_{p,t-1}(1 + \alpha_p \xi_t) \end{aligned}, \tag{61}$$

where $p'_t = (1 - 2k)p_t + k$ and k is some small number (we use 10^{-5}). This allows to say that $p'_t \sim \text{Beta}(a, b)$ and use statistical inferences.

The important thing is how to estimate actual probability, and **?** propose to use Naive approach. This means that if non-zero value occurs, then the probability is equal to one, otherwise it is equal to zero. This approach may cause some problems and is hard to use in inferences, but it gives TSB its unique property of updating probability when no demand occurs.

Conditional expectation of TSB is trivial and is equal to $\mu_{p,t+h|t} = l_t$, while conditional variance is equal to $\sigma^2_{p,t+h|t} = \sigma^2_p(1 + (h - 1)\alpha^2_p)$.

However estimation of the model (61) in "smooth" is done in two steps. On the first step the parameters a and b of Beta distribution with some heuristic values of $l_{p,0}$ and α are estimated. This is possible, because value of Beta function is much bigger than values of p_t, so changes in initial and smoothing parameter almost do not change the cost function value. On the second step the parameters of the model (61) are estimated using formulae discussed later in section 2.4.

In R this model can be constructed using `intermittent="tsb"`.

2.3.7. Model with provided data

Finally, in cases, when future probabilities or occurrences are known, they can be provided to "smooth" functions and then will be used as $\mu_{p,t+h|t}$ in order to produce point and intervals forecasts. Model with fixed probability 2.3.4 is built in this case without production of future probabilities. This is done via the very same parameter `intermittent=pForecast`, where `pForecast` needs to be a vector with values lying in [0, 1] region. The length of `pForecast` needs to correspond to the forecast horizon. This functionality can be useful if a researcher knows for some reasons that the product will be bought in some specific days.

2.4. Likelihood and model selection

One of the main elements of modelling using state-space approach is likelihood function. As an estimator it has good statistical properties, allows selecting the most appropriate model among the pool of models or producing combinations of models and forecasts.

2.4.1. Likelihood function for state-space models

We have already discussed that there are two types of models: with additive and multiplicative error terms – which are assumed to have either normal or log-normal distribution. So the very general forms of likelihood functions differ from conventional ones used in **?**, firstly because we assume that multiplicative errors have log-normal distribution and secondly because we also take probabilities of occurrences into account.

In case of additive errors concentrated log-likelihood is:

$$\begin{aligned} \ell(\theta, \hat{\sigma}^2|Y) = & -\tfrac{T_1}{2}\left(\log(2\pi e) + \log(\hat{\sigma}^2)\right) \\ & + \sum_{o_t=1} \log(p_t) + \sum_{o_t=0} \log(1-p_t), \end{aligned} \tag{62}$$

where Y is a vector of all the actual values and T_1 is number of non-zero observations.

If error has multiplicative form, then from the assumption of log-normality the concentrated log-likelihood is:

$$\begin{aligned} \ell(\theta, \hat{\sigma}^2|Y) = & -\tfrac{T_1}{2}\left(\log(2\pi e) + \log(\hat{\sigma}^2)\right) - \sum_{o_t=1} \log(y_t) \\ & + \sum_{o_t=1} \log(p_t) + \sum_{o_t=0} \log(1-p_t) \end{aligned} . \tag{63}$$

For the continuous data ($o_t = 1$ for all t and $T_1 = T$), terms with $o_t = 0$ in (62) and (63) are dropped, so we end up with conventional normal and log-normal likelihood functions.

2.4.2. Model selection

Using likelihoods (62) and (63) information criteria can be calculated. For example, AIC will be:

$$\text{AIC} = 2k - 2\ell(\theta, \hat{\sigma}^2|Y), \tag{64}$$

where k is number of parameters (including variance of errors). In case when initials are found during the optimisation, k includes the number of initials. In case of backcasting, this number is not included. In case of intermittent models k increases by one, reflecting that the probability needs to be estimated. However, probability estimation methods themselves do not reduce parameter space, because they use occurrences rather than values of demand, so there is no difference in number of parameters between fixed probability model, Croston's model and TSB. This allows to select the most appropriate intermittent data model between the three and conventional model with no intermittency.

The default information criterion used in the package is AICc, user however can define a different one using parameter `ic`, which can take values: `"AIC"`, `"AICc"` and `"BIC"`.

Model selection is currently implemented in functions `auto.ssarima()`, `auto.ces()` and `es()`. In addition setting parameter `intermittent` to `"auto"` for any model allows to select the most appropriate type of intermittent model amongst the ones discussed in the section 2.3 using an information criterion.

However, all of that can be done only when MSE is used as a cost function. In cases when some other `cfType` is defined, the corresponding MSE is calculated and then used in likelihood and criteria calculation. This is not completely correct, because the likelihood function is not maximised in this case and becomes not as efficient as it should be. So the results of model selection in this case may be wrong. In these cases user is warned about that but it is his decision to make.

2.4.3. Combinations of forecasts

Some functions allow to combine forecasts of models using information criteria weights, discussed by `?` and in `?` (currently only `es()` has it, but this is to be implemented for `ssarima()` and some other functions as well). These are defined using:

$$w_j = \frac{\exp(-\frac{1}{2}(\text{AIC}_j - \min(\text{AIC})))}{\sum_{i=1}^{m} \exp(-\frac{1}{2}(\text{AIC}_i - \min(\text{AIC})))}, \tag{65}$$

where m is number of models in the pool, AIC_j is AIC for j^{th} model and min(AIC) is the value of the smallest AIC in the pool. These weights are then used for combination of forecasts, prediction intervals and fitted values.

Similar weights can be calculated for AICc and BIC, this is defined by user via parameter `ic`.

2.5. Exogenous variables

Flexibility of state-space models allows introducing exogenous variables, which in general can be done the following way:

$$\begin{aligned} y_t &= o_t \left(w(v_{t-l}) + a'_{t-1} x_t + r(v_{t-l})\epsilon_t\right) \\ v_t &= f(v_{t-l}) + g(v_{t-l})\epsilon_t \\ a_t &= F_x a_{t-1} + g_x {x_t}^{-1} \epsilon_t \end{aligned}, \tag{66}$$

where a_t is vector of time varying parameters for vector of exogenous variables x_t, F_x is transition matrix for exogenous variables, g_x is persistence vector for exogenous variables and ${x_t}^{-1}$ is vector of inverted exogenous variables. If it is assumed that the parameters should not change over time, then $g_x = 0$. Transition matrix F_x in the simplest case is identity matrix, although this condition can be relaxed in order to introduce complex interactions between parameters. Vector x_t contains k_x elements. When $x_t = 0$, adaptation is not happening, meaning that $a_t = a_{t-1}$. For example model ETS(A,N,N), based on (66) with 2 exogenous variables can be written as a system:

$$\begin{aligned} y_t &= l_{t-1} + a_{1,t-1} x_{1,t} + a_{2,t-1} x_{2,t} + \epsilon_t \\ l_t &= l_{t-1} + \alpha \epsilon_t \\ a_{1,t} &= \begin{cases} a_{1,t-1} + \delta_1 \frac{\epsilon_t}{x_{1,t}}, & \text{when } x_{1,t} \neq 0 \\ a_{1,t-1}, & \text{when } x_{1,t} = 0 \end{cases} \\ a_{2,t} &= \begin{cases} a_{2,t-1} + \delta_2 \frac{\epsilon_t}{x_{2,t}}, & \text{when } x_{2,t} \neq 0 \\ a_{2,t-1}, & \text{when } x_{2,t} = 0 \end{cases} \end{aligned}. \tag{67}$$

The adaptation scheme for exogenous variables suggested in (66) is called "Non-Uniform Smoothing" and is based on modification of method of stochastic approximation. State-space models in this case become non-stable, but still can be forecastable. This condition is checked during estimation of parameters. The mechanism of adaptation allows efficiently updating parameters of models with very different exogenous variables, including dummy variables. By default this mechanism is switched off, so F_x is unity matrix, while all elements of g_x are set to be equal to zero.

Exogenous variables are included in "smooth" functions via parameter `xreg`, which can accept either vector, or matrix, or data frame. Note that `xreg` needs to contain number of observations either equal to in-sample or to

the length of y_t. If the values for the forecast period are not provided, then `es()` with automatic selection of iETS is used. This may increase time of calculations.

In case user needs to estimate F_x together with g_x, it can be done by setting parameter `updateX=TRUE`. If either F_x or g_x does not need to be updated, then user needs to provide predefined values via parameters `transitionX` and `persistenceX` respectively.

Initial parameters for exogenous variables are estimated using Least Squares applied to the model $Y = A'X$, where Y is vector of all the actual values, A is vector of parameters and X is matrix of all provided exogenous variables. After that they are re-optimised during estimation of state-space model (or produced using backcasting). However if user provides initials via `initialX` parameter, then those values are taken as a_0 and are not re-optimised.

An important note on exogenous variables usage is that they are currently developed for additive models, so some problems may rise with multiplicative and mixed state-space models.

2.6. Prediction intervals

We have already discussed how to construct parametric prediction intervals for pure additive and multiplicative models (sections 2.1.1 and 2.1.2) and how to take occurrences into account (sections 2.3.1 and 2.3.2). However "smooth" functions can also produce other types of intervals discussed in this subsection.

By default all the main "smooth" functions do not produce any prediction intervals, but this can be changed if parameter `intervals` is passed with some preferred type of intervals. In that case any of the aforementioned functions can construct one of the following types of prediction intervals:

1. Parametric. These have already been covered in sections 2.1.1, 2.1.2, 2.1.3, 2.3.1, 2.3.2 and 2.3.3. They are default for all the functions and correspond to parameter `intervals="parametric"` or `intervals="p"`. The other way to construct parametric prediction intervals is to pass `intervals=TRUE`.
2. Semiparameteric. In order to produce these intervals a forecast for h steps ahead from each observation is done and multiple steps ahead prediction errors are gathered (similar to how it is done for cost functions discussed in section 2.2.3). After that variances of $1, 2, .., h$ steps ahead forecast errors are calculated and then used in order to produce

symmetric intervals based on standard normal or log-normal distribution quantiles. This is called using `intervals="semiparametric"` or `intervals="sp"`.

3. Non-parameteric. These use principles described in **?** and are based on quantile regressions of several steps ahead errors (gather in the same manner as for semiparametric intervals) of the following form: $e_j = aj^b$, where $j = 1, ..h$. Although **?** propose to use polynomials, we found that they are not stable and sometimes produce meaningless intervals (for example with growth and then decline). Using power function however allows covering all the necessary types of intervals (linear, quadratic, square root). Note that these non-parameteric intervals are usually asymmetric and may look strange. However they work very well when key assumptions of modelling are violated (for example, normally distributed residuals). They also seem to perform very well for intermittent data and are recommended in cases of mixed state-space models. These can be constructed with `intervals="nonparametric"` or `intervals="np"`.

Width of prediction intervals is defined via parameter `level`. Although the correct value to pass to `level` should lie between 0 and 1 (for example, 0.95), it will also accept values between 0 and 100 (e.g. 95).

3. Main "smooth" functions

In this section we discuss specific models underlying different functions and see how they work in R. All the examples in this section use package `Mcomp` in R.

There are several parameters that are accepted by all the "smooth" forecasting functions:

1. `h` – forecasting horizon;
2. `holdout` – binary for whether the holdout from the provided data needs to be taken or not. This can be handy if you want to see accuracy of forecasts on the provided data;
3. `silent` – this parameter determines what should not be produced by the function. By default (`silent="none"`) all the messages and texts are printed out and graph with legend is produced. This parameter allows to silent either `"all"` (nothing is printed out or plotted), or `"graph"`, or `"legend"`, or `"output"`.

There are also several hidden parameters that can be used, but are not included in the list:

1. `model` – this parameter allows to pass previously estimated model of the same type to the function. Only `es()` has this parameter as "unhidden";
2. `FI` – if this parameter is `TRUE`, then Fisher Information is returned. This can only be done if package "numDeriv" is installed. Fisher Information then can be used for calculation of variances of parameters.

We start by looking at ETS and end this section with several examples.

3.1. Exponential Smoothing, `es()`

All the 30 variants of exponential smoothing models discussed in `?` are implemented in `es()` function. Any specific model is defined via parameter `model`. For example, ETS(A,Ad,M) can be constructed with `model="AAdM"`.

There are several arguments specific to `es()` that can be passed. These are `initial`, `initialSeason`, `persistence` and `phi`. `initial` cannot have length greater than 2 and allows defining values for level and trend components only. If seasonal coefficients need to be set, then `initialSeason` should be used instead. There is no restriction on number of seasonal coefficients. However, having ETS models with large seasonal lags may lead to optimisation difficulties. If values of any of these parameters are provided, then they are not re-optimised and are used in model construction. `persistence` parameter accepts vector of smoothing parameters. Finally `phi` defines damping parameter and works only for models with damped additive or multiplicative trends.

In addition parameter `model` can accept either specific model (as noted above), or vector of model names, from which the best model needs to be selected, or a previously estimated model. So for example, if we have estimated some model for time series once, we can reuse it:

```
y1 <- rnorm(100,100,10)
esModel <- es(y, h=10, holdout=TRUE)
es(y, model=esModel, h=10)
```

Note that it is not advised to apply one and the same model to different time series, because `es()` will reuse in this case `initial` and `initialSeason` along with `persistence` and `phi`. However different data should have different initials of state vector.

3.1.1. Normalisation of seasonal components

Due to a different structure of ETS models implemented in "smooth" (using lags instead of dummy variables), normalisation of seasonal components cannot be done using conventional methods. As a result we have implemented a different mechanism: instead of updating components of ETS on each observation, we update groups of them each season. So, for example, for monthly data this update will happen every 12^{th} observation. The formulae used in the process are shown in the table 1.

	Additive seasonality	Multiplicative seasonality
Normaliser	$a_j = \frac{1}{m}\sum_{i=1}^{m} s_{m(j-1)+i}$	$a_j = \sqrt[m]{\prod_{i=1}^{m} s_{m(j-1)+i}}$
Seasonal component	$s'_t = s_t - a_j$	$s'_t = \frac{s_t}{a_j}$
T=N, level component	$l'_t = l_t + a_j$	$l'_t = l_t \cdot a_j$
T=A, level component	$l'_t = l_t + a_j$	$l'_t = l_t \cdot a_j$
T=A, trend component	—	$b'_t = b_t \cdot a_j$
T=M, level component	$l'_t = l_t + \frac{a_j}{b_t}$	$l'_t = l_t \cdot a_j$

Table 1: Normalisation formulae for seasonal ETS models.

As can be seen from the table above in case of additive seasonality, we use simple mean in order to estimate the normalising factor, while in case of multiplicative – geometric mean. After applying formulae from table 1, the additive seasonal components add up to zero each season (from January to December, for example) and the multiplication of multiplicative components each season results in one.

Normalisation does not change point forecasts of ETS, however it may slightly change prediction intervals and is needed for analysis purposes (when seasonal components themselves are of the main interest).

3.1.2. Model selection and combinations

Model selection in `es()` can be done in several ways. The simplest one is defining `"Z"` in `model` call for the component that needs to be selected. For example, `model="ZZN"` will select the best non-seasonal model, while `model="MAZ"` will select the most appropriate seasonality for additive trend model with multiplicative error.

In order to decrease time of calculations branch and bound mechanism is used. In general, for `model="ZZZ"`, it is done in the following steps:

1. ETS(A,N,N) is fitted to the data.
2. Seasonal ETS(A,N,A) is fitted to the data. If AIC for this model is lower than for (1), then seasonal model should be used. In this case we move to step 3. Otherwise we move to step 4.
3. Seasonal ETS(A,N,M) is fitted. If AIC for this model is lower than for (2), then we deal with multiplicative seasonal model. Otherwise we have additive seasonality.
4. Additive trend model is fit to the data. The type of model depends on steps (1) - (3). If there is no seasonality, then ETS(A,A,N) is constructed. If seasonality is additive, then we construct ETS(A,A,A). Finally, with multiplicative seasonality we end up with ETS(A,A,M). If the model on this step is better than the one on the previous step (either (1), (2) or (3)), then trend is needed.

After that the pool of models that need to be estimated is defined, models in the pool are estimated and the one with the lowest information criterion is selected. The longest pool of models in this case is when trend is needed – it includes 8 models: for example, for non-seasonal model, excluding already estimated ETS(A,N,N) and ETS(A,A,N) we need to estimate (A,Ad,N), (A,M,N), (A,Md,N), (M,N,N), (M,A,N), (M,Ad,N), (M,M,N) and (M,Md,N).

In cases when not all the components need to be checked (for example, for `model="ZZN"`), some steps of the described branch and bound algorithm are skipped.

If user wants to select a model out of his favourite list of models, he can provide that list as vector of names in the following way:

```
model=c("ANN","AAN","AAdN","ANA","AAA","AAdA")
```

All of these models will then be estimated and the one with the lowest information criterion will be returned.

Finally `es()` allows to produce combinations of forecasts based on AIC weights (see section 2.4.3 for the details). This is defined with letter `"C"` instead of `"Z"`. So `model="CCN"` will produce combined forecasts of all the trend models with both additive and multiplicative errors, but no seasonality. If something like `model="CCZ"` is provided, then `"Z"` is ignored and considered

as yet another `"C"`. In general combination mechanism is slower than model selection because it needs to fit all the models to the data. For example, `model="CCC"` means that all the 30 models are fitted to the data.

3.2. Complex Exponential Smoothing, `ces()`

Complex Exponential Smoothing was proposed by **?** and has its name because it is based on complex variables. The original method is formulated as:

$$\hat{y}_{t+1} + i\hat{p}_{t+1} = (\alpha_0 + i\alpha_1)(y_t + ip_t) + (1 - \alpha_0 + i - i\alpha_1)(\hat{y}_t + i\hat{p}_t), \qquad (68)$$

where i is imaginary unit, number satisfying equation $i^2 = -1$, p_t is potential information variable and α_0 and α_1 are smoothing parameters. The idea behind this method is that there may be some unidentifiable components in time series that cannot be captured using conventional level-trend-seasonal decomposition. So we claim that p_t contains that useful information and can be used in forecasting. However p_t is unobservable so in order to use it, we need to define proxy for it. The model implemented in "smooth" uses the following proxy: $p_t = \epsilon_t$. So the state-space model underlying (68) in this case can be written as (**?**):

$$\begin{aligned} y_t &= l_{t-1} + \epsilon_t \\ l_t &= l_{t-1} - (1 - \alpha_1)c_{t-1} + (\alpha_0 - \alpha_1)\epsilon_t, \\ c_t &= l_{t-1} + (1 - \alpha_0)c_{t-1} + (\alpha_0 + \alpha_1)\epsilon_t \end{aligned} \qquad (69)$$

where c_t is information potential component. The model (69) can be written in compact form of pure additive state-space model discussed in section 2.1.1. This means that it inherits all the properties of that model. However the main difference between CES and ETS is that trend in the former is defined by value of complex smoothing parameter $\alpha_0 + i\alpha_1$. α_1 in this case defines if trend is going upwards or downwards (if it is greater than 1, then it is upwards), while α_0 defines variability of the data. **?** explain statistical properties of CES, so we won't discuss them here.

Main advantage of CES is good estimation of long-term trends. As a result it performs better than ETS for long-term horizons.

There are no "traditional" or "usual" bounds for CES, so only admissible bounds are available for model construction.

There are also several seasonal modifications of CES available for construction, which can be selected using parameter `seasonality`. Here is the brief explanation.

Full seasonal CES model is defined as:

$$\begin{aligned} y_t &= l_{0,t-1} + l_{1,t-m} + \epsilon_t \\ l_{0,t} &= l_{0,t-1} - (1-\alpha_1)c_{0,t-1} + (\alpha_0 - \alpha_1)\epsilon_t \\ c_{0,t} &= l_{0,t-1} + (1-\alpha_0)c_{0,t-1} + (\alpha_0 + \alpha_1)\epsilon_t \\ l_{1,t} &= l_{1,t-m} - (1-\beta_1)c_{1,t-m} + (\beta_0 - \beta_1)\epsilon_t \\ c_{1,t} &= l_{1,t-m} + (1-\beta_0)c_{1,t-m} + (\beta_0 + \beta_1)\epsilon_t \end{aligned} . \tag{70}$$

This model contains two groups of components: with lag 1 and with lag m. So the latter group allows to model wide variety of seasonal patterns. This includes additive and multiplicative types and also introduces new ones (for example, when level of series does not grow, but amplitude of seasonality increases). Number of parameters in the model (70) is high: $k = 4+2+2m$ (4 smoothing parameters, 2 initial values for $l_{0,0}$ and $c_{0,0}$ and $2m$ initial seasonal components.

Partial seasonal CES model includes conventional additive seasonal component:

$$\begin{aligned} y_t &= l_{t-1} + s_{t-m} + \epsilon_t \\ l_t &= l_{t-1} - (1-\alpha_1)c_{t-1} + (\alpha_0 - \alpha_1)\epsilon_t \\ c_t &= l_{t-1} + (1-\alpha_0)c_{t-1} + (\alpha_0 + \alpha_1)\epsilon_t \\ s_t &= s_{t-m} + \gamma\epsilon_t \end{aligned} . \tag{71}$$

The model (71) is not as universal as (70), but maybe sufficient in some cases. Number of parameters in this case is $k = 3 + 2 + m$.

Finally, there is also so called **Simple seasonal CES** model:

$$\begin{aligned} y_t &= l_{t-m} + \epsilon_t \\ l_t &= l_{t-m} - (1-\beta_1)c_{t-m} + (\beta_0 - \beta_1)\epsilon_t \\ c_t &= l_{t-m} + (1-\beta_0)c_{t-m} + (\beta_0 + \beta_1)\epsilon_t \end{aligned} . \tag{72}$$

This model has $k = 2 + 2m$ parameters and seems to perform well in cases of seasonal time series with zeroes. For example, solar irradiation data is one of such time series. However we have not yet conducted an appropriate research on this topic.

An automatic selection between these four models (non-seasonal and three seasonal ones) can be done using `auto.ces()` function.

Due to the flexibility and simple form of the original model (no multiplication of components), `ces()` is a very fast function. Because of that and ability of CES to model different patterns in data, it is recommended for

data with large seasonal frequencies. However, the recommended (and default) initialisation method for CES is backcasting, which allows to decrease number of parameters by m or $2m$ depending on the chosen seasonality type. CES may perform poorly when `initial="optimal"` is used, because parameter space in this case may increase substantially (see details in section 2.2.2).

CES does not accept `persistence` vector directly. It uses `A` and `B` instead, which correspond to non-seasonal and seasonal complex smoothing parameters. These must be complex numbers. A user may also use a hidden parameter `model` that allows applying the same CES model to a different data.

3.3. State-Space ARIMA

? showed that several ARIMA models can be written in state space form (3). Since then this idea has not developed substantially in forecasting literature but **?** discuss connection between any ARIMA and pure additive state-space model (3), describing how to construct ARIMA in that case. We decided to use this idea and these derivations in order to construct State-Space ARIMA with possible multiple seasonalities (Several Seasonals ARIMA). Thus name of SSARIMA stands for both State-Space and Several Seasonals.

In general multiple seasonal ARIMA can be written as:

$$\Phi_{m_1} \cdot \dots \cdot \Phi_{m_n} \Delta_{m_1} \cdot \dots \cdot \Delta_{m_n} y_t = c + \Theta_{m_1} \cdot \dots \cdot \Theta_{m_n} \epsilon_t, \tag{73}$$

where Φ_{m_i} is $\text{SAR}(\text{P}_i)_{m_i}$ polynomial (seasonal autoregressive part), Δ_{m_i} is $\text{SI}(\text{D}_i)_{m_i}$ polynomial (seasonal differences) and finally Θ_{m_i} is $\text{SMA}(\text{Q}_i)_{m_i}$ polynomial (seasonal moving average) with seasonal frequencies m_i where $i = 1, .., n$ is number of seasonal parts and c is constant

However in our implementation a slightly different definition of SARIMA then in **?** is used. The main difference is in MA polynomials that are defined the following way: $\Theta_{m_i} = 1 + \theta_{i,1} B^{m_i} + \theta_{i,2} B^{2m_i} + ...$ So for example $\text{SARIMA}(0,1,1)(1,1,1)_{12}$ in our notations is written as:

$$(1 - \Phi_1 B^{12})(1 - B^{12})(1 - B) y_t = c + (1 + \Theta_1 B^{12})(1 + \theta_1 B) \epsilon_t.$$

Note that while **?** dropped constant term "in accordance with common usage" we do not do that and allow for our SSARIMA to have constant even with differences, which usually results in model with drift.

Derivations of state-space ARIMA are similar to **?**. Firstly multiplications of polynomials for both sides of (73) are done in R using "polynom" package. This leads to the following equation:

$$y_t = c + \sum_{i=1}^{n} \eta_i y_{t-i} + \sum_{i=1}^{n} \theta_i \epsilon_{t-i} + \epsilon_t. \tag{74}$$

Then (skipping some derivations, which can be found in (**?**, p.173)) for each j of n components we have:

$$v_{j,t} = \eta_j v_{1,t-1} + v_{j+1,t-1} + (\eta_j + \theta_j)\epsilon_t, \tag{75}$$

and an additional component for constant term is defined as:

$$v_{n+1,t} = c. \tag{76}$$

However the first component in SSARIMA is updated using slightly different formula than (75), taking constant term into account:

$$v_{1,t} = \eta_1 v_{1,t-1} + v_{2,t-1} + v_{n+1,t} + (\eta_1 + \theta_1)\epsilon_t. \tag{77}$$

After all of that transition matrix, persistence and measurement vectors are formed:

$$F = \begin{pmatrix} \eta_1 & I_{n-1} & 1 \\ \vdots & & \vdots \\ \eta_n & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, g = \begin{pmatrix} \eta_1 + \theta_1 \\ \vdots \\ \eta_n + \theta_n \\ 0 \end{pmatrix}, w = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{78}$$

In case when $c = 0$ (no constant term is needed), SSARIMA becomes exactly the same as in **?**. In the other cases it either acts as a simple constant term, or as drift (depending on differences of original SARIMA).

The estimation of the model after definition of (78) is done in a similar manner as for purely additive models. Furthermore all the properties of pure additive model (3) can be applied to State-Space ARIMA (including conditional expectation, variance, prediction intervals and model selection using information criteria).

The method of transforming of conventional ARIMA into state-space form described here is universal and allows constructing models with any number of seasonal elements. That is why `ssarima()` has several parameters

for orders instead of one: `ar.orders`, `i.orders`, `ma.orders` and `lags`, which accept integer values. A simple ARIMA(0,1,1), for example, can be constructed when `ar.orders=0`, `i.orders=1`, `ma.orders=1` and `lags=1`, while a more complicated SSARIMA$(0,1,1)(1,1,2)_{12}$ is called with `ar.orders=c(0,1)`, `i.orders=c(1,1)`, `ma.orders=c(1,2)` and `lags=c(1,12)`. Inclusion of constant term in the model is regulated with parameter `constant`, which accepts either `TRUE` or `FALSE`.

Due to usage of State-Space model for ARIMA we do not loose any number of observations and estimate any type of ARIMA on one and the same sample, but the initial values need to be estimated. This can be done using either optimisation or backcasting. The latter is recommended for more complicated SARIMA models, because number of initials to estimate may increase substantially. Number of parameters of SARIMA is equal to number of AR and MA terms, plus one for constant and one for standard deviation: $k = \sum_{j=1}^{n} P_j + \sum_{j=1}^{n} Q_j + 1 + 1$. In case of optimisation of initial values, number of parameters increases by the number of components.

In order to ensure that estimated ARIMA is both stationary and invertible, there are two checks during estimation of the model. This is done only when `bounds="admissible"`. If for some reason the estimated model is unstable (non-invertible) or not stationary, then the warning is printed out. However, this may happen mainly in cases when `bounds="none"`.

Finally information criteria can be calculated for any SSARIMA of any order without restrictions, because now all the models are constructed on one and the same sample, and the only difference between them is number of parameters to estimate. "smooth" package has `auto.ssarima()` function that selects order of SSARIMA using minimisation of information criteria. It also uses branch and bound mechanism for order selection, which is done in the following steps for general SSARIMA:

1. Selection of the most appropriate differences;
2. Selection of the most appropriate MA term;
3. Selection of the most appropriate AR term;
4. Check for the constant.

The order selection is done sequentially – model on step two is estimated on residuals of step one model, while step three model is estimated on residuals of step two model. This is done for each differences combination and looks tedious, but in fact allows increasing speed of calculation while preserving accuracy of model selection.

Step one is done with constant term included, so model with first differences will have drift. This in a way allows defining if there is trend in time series and whether we need to take differences without conducting any statistical tests.

On each step (2) – (3) selection starts from lower lag and finishes with the highest, meaning that non-seasonal part is estimated first. Then for each lag there is a loop, where orders of either AR or MA are checked. This is done from higher order to lower. Motivation here is that if, for example, there is AR(1) in some time series, then AR(3) will be better than AR(0) (in terms of information criteria), because former includes that AR(1). AR(2) in its turn will have lower information criterion than AR(3) and finally AR(1) will be the best. However if the higher order does not decrease information criterion in comparison with no order at all, then we can skip the check of orders for the current lag. This allows to speed up the order selection process.

Maximum orders to check are defined in `auto.ssarima()` with a list `order`, which should contain `ar`, `i` and `ma`. The length of at least one of them must correspond to length of `lags` vector.

As an example, let's say that we want to estimate SSARIMA(p,d,q)(P,D,Q)$_{12}$ on some time series, setting `ar=c(3,2)`, `i=c(2,1)`, `ma=c(3,2)` and `lags=c(1,12)`. So the process of SARIMA order selection is done in the following steps:

1. Check differences d and D:
   (a) Some values of d= $\{0, 1, 2\}$ and D= $\{0, 1\}$ are selected with non-zero constant and all the other orders of the model being zero.
   (b) The very first model that is checked is SSARIMA(0,0,0)(0,0,0)$_{12}$, so there is no need in checking zero orders of other elements on other steps.
   (c) Residuals of that model are taken to the next step.
2. Check moving average order, q and Q:
   (a) Model with q= $\{3, 2, 1\}$ with zero constant is fitted to the residuals produced from step 1. if current q leads to the higher information criterion value than on a previous step (either step (1) or the previous value of q), then we move to the step (2,b).
   (b) Find the appropriate Q* using the same principle as in (2,a).
3. Check autoregressive parts, p and P:
   (a) Model with p= $\{3, 2, 1\}$ with zero constant is fitted to the residuals produced from step 2. if current p leads to the higher information criterion value than on a previous step (either step (2) or the previous value of p), then we move to the step (3,b).

(b) Check P= $\{2, 1\}$ in a similar manner as in (3,a). Find the appropriate P*.

4. Move to the step 1 and select next differences to check. When, there is nothing else to check, move to step 5.
5. Compare SSARIMA(p*,d*,q*)(P*,D*,Q*)$_{12}$ with constant and the same model without constant term on the original data.

Using such an exhaustive search of appropriate differences ensures that we select the correct ones with the correct MA and AR terms.

This is not ideal algorithm, however it gives decent results. Currently the mechanism is slow mainly because of polynomial terms multiplication. When this routine is moved to C++, the speed of selection will increase.

3.4. Generalised Exponential Smoothing, `ges()`

This is a next step from CES model. Here we assume that all the components that we use are fuzzy and that the most important thing for us is interaction between them, which leads to non-linear forecasts.

GES is pure additive model, so it has all the properties discussed in section 2.1.1. In a way GES is just a state-space model, where transition matrix, measurement and persistence vectors are estimated. In general it is still a simple model defined by matrix equations:

$$\begin{aligned} y_t &= w' v_{t-l} + \epsilon_t \\ v_t &= F v_{t-l} + g\epsilon_t \end{aligned},$$

where $w = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}$, $F = \begin{pmatrix} f_{1,1} & f_{1,2} & \dots & f_{1,n} \\ f_{2,1} & f_{2,2} & \dots & f_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{n,1} & f_{n,2} & \dots & f_{n,n} \end{pmatrix}$, $g = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{pmatrix}$, $v_t = \begin{pmatrix} v_{1,t} \\ v_{2,t} \\ \vdots \\ v_{n,t} \end{pmatrix}$.

GES is defined by number of components and number of lags, which can be written down as GES($k_1^{m_1}, k_2^{m_2}, ...$). For example, model GES($2^1, 1^4, 1^{12}$) has two components with lag one, one component with lag 4 and one component with lag 12. Such a model can be written in state-space form as:

$$\begin{aligned} y_t &= w_1 v_{1,t-1} + w_2 v_{2,t-1} + w_3 v_{3,t-4} + w_4 v_{4,t-12} + \epsilon_t \\ v_{1,t} &= f_{1,1} v_{1,t-1} + f_{1,2} v_{2,t-1} + f_{1,3} v_{3,t-4} + f_{1,4} v_{4,t-12} + g_1 \epsilon_t \\ v_{2,t} &= f_{2,1} v_{1,t-1} + f_{2,2} v_{2,t-1} + f_{2,3} v_{3,t-4} + f_{2,4} v_{4,t-12} + g_2 \epsilon_t. \\ v_{3,t} &= f_{3,1} v_{1,t-1} + f_{3,2} v_{2,t-1} + f_{3,3} v_{3,t-4} + f_{3,4} v_{4,t-12} + g_3 \epsilon_t \\ v_{4,t} &= f_{4,1} v_{1,t-1} + f_{4,2} v_{2,t-1} + f_{4,3} v_{3,t-4} + f_{4,4} v_{4,t-12} + g_4 \epsilon_t \end{aligned} \tag{79}$$

v_{t-l} in this case is equal to $Lv_t = \begin{pmatrix} B^1 & 0 & 0 & 0 \\ 0 & B^1 & 0 & 0 \\ 0 & 0 & B^4 & 0 \\ 0 & 0 & 0 & B^{12} \end{pmatrix} \begin{pmatrix} v_{1,t} \\ v_{2,t} \\ v_{3,t} \\ v_{4,t} \end{pmatrix}$.

As it can be seen from example (79), components interact with each other and influence each other on every observation. This gives non-linearity and flexibility to the model, so it can produce any forecasting trajectory and any type of seasonality, becoming in a way a perfect approximator, if number of components is large enough.

In R GES is defined using two parameters: `order` and `lags`, which accept vectors of integer numbers. So, for example, the aforementioned GES$(2^1, 1^4, 1^{12})$ can be constructed when `order=c(2,1,1)` and `lags=c(1,4,12)`.

The only problem of GES is number of parameters to estimate. If initial values are not taken into account, then number of parameters will be $2n+n^2$, where n is number of all the components. So GES$(2^1, 1^4, 1^{12})$ has $k = 2 \cdot 4 + 4^2 = 24$ parameters to estimate. In case when initials need to be estimated, number of parameters increases by $m = \sum_{j=1}^{n} m_j$, where m_j is lag for each component. So for our example $m = 1+1+4+12 = 18$ and in this case GES will have $k = 24 + 18 = 42$ parameters to estimate. This is not a trivial task, so "backcasting" is recommended for initialisation of such models. This also means that not all GES models can be used in cases of small samples.

It can be shown that GES is model encompassing any ARIMA model and any pure additive exponential smoothing model (including multiple seasonal additive models), but not all GES models have underlying ARIMA. Connections between GES and ARIMA become obvious from state-space form of ARIMA discussed in section 3.3. This makes GES more general than any other model.

An example, of conventional model written in GES framework is double seasonal additive exponential smoothing applied to daily data with frequencies 7 and 365, which corresponds to GES$(1^1, 1^7, 1^{365})$:

$$\begin{aligned} y_t &= v_{1,t-1} + v_{2,t-7} + v_{3,t-365} + \epsilon_t \\ v_{1,t} &= v_{1,t-1} + g_1 \epsilon_t \\ v_{2,t} &= v_{2,t-7} + g_2 \epsilon_t \\ v_{3,t} &= v_{3,t-365} + g_3 \epsilon_t \end{aligned}, \tag{80}$$

which actually means predefined values for measurement vector and transi-

tion matrix: $w = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ and $F = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, and $L = \begin{pmatrix} B^1 & 0 & 0 \\ 0 & B^7 & 0 \\ 0 & 0 & B^{365} \end{pmatrix}$.

Order selection for GES can be done using information criteria, however we currently do not have a function that would do this automatically. But even the default values of `orders` (`c(1,1)`) and `lags` (`c(1,m)`) for GES should suffice for many cases of simpler time series.

`ges()` function allows to predefine such parameters as `transition` (matrix F), `measurement` (vector w) and `persistence` (vector g). A hidden parameter `model` allows to apply the same GES model to different data.

3.5. Simple Moving Average, `sma()`

Simple Moving Average is a very simple method that has the following formula:

$$\hat{y}_t = \frac{1}{p} \sum_{j=1}^{p} y_{t-j}. \tag{81}$$

Note that we discuss here **simple**, not centred moving average. It has been thought for a long time that this method does not have an underlying statistical model. However it is very easy to show the opposite. In order to do that we move the sum in (84) to the left hand side of the equation and substitute $\hat{y}_t = y_t - \epsilon_t$. So we end up with something looking like AR(p) process:

$$y_t - \frac{1}{p} \sum_{j=1}^{p} y_{t-j} = \epsilon_t. \tag{82}$$

Which in conventional notations of ARIMA can be written as:

$$\left(1 - \frac{1}{p}B - \frac{1}{p}B^2 - ... - \frac{1}{p}B^p\right) y_t = \epsilon_t. \tag{83}$$

So for any simple moving average there is an underlying AR(p) process with predefined parameters. We however estimate ARIMA in state-space form, so in the case of SMA we simply use the same mechanism as in 3.3. Note that having a state space model for SMA allows constructing prediction intervals, selecting the most appropriate order of SMA and also produce values in the beginning of the series. So there are no missing values for SMA in state-space form. This however holds only when we use backcasting as initialisation of the model. Otherwise we loose those first values, because SMA has a short

memory and optimiser ends up with over-fitting first p actual values. This is the reason why we switched off initialisation selection for SMA and produce initials using backcasting only.

The only issue with SMA is how to calculate number of parameters used. The original moving average has $T-1$ degrees of freedom, so the number of parameters of SMA in state-space form should be two (the average + standard deviation of residuals). In case with optimisation of initial parameters, they should be added to number of parameters. But in case when initial values are produced using backcasting all the SMA models will have exactly the same number of parameters. This also means that AIC is influenced only by the fit of SMA and will select the one that has the lowest Mean Squared Error.

There is only one important parameter in `sma()` function – `order`. If it is `NULL`, then the order is selected using information criteria.

An underlying statistical model for Centred Moving Average is more complicated to derive, because it depends on past and future observations at the same time. The formula for CMA(p) method is:

$$\tilde{y}_t = \frac{1}{p} \sum_{j=-k}^{k} y_{t-j}, \tag{84}$$

where $p = 2k - 1$. If we note that $\tilde{y}_t$ in CMA is equal to $\hat{y}_{t+k+1}$ in SMA, then we can set $y_t = \tilde{y}_t + e_t = \hat{y}_{t+k+1} + e_t$. This allows using the same state-space model as used in SMA for conditional expectation of CMA. However errors need to be calculated differently and variance of CMA should differ from variance of SMA. So what we in fact could do, is just to shift the fitted values and calculate errors differently. However this functionality is not yet implemented in "smooth".

3.6. Simulation functions

There is currently only one function in "smooth" that allows simulating data from a selected state-space model. This is `sim.es()`. It generates data using state-space ETS (with all the properties discussed in previous sections) with chosen `model` and preset `initial`, `initialSeason`, `persistence`, `phi` and selected random numbers generator via `randomizer` parameter. If `initial`, `initialSeason` and `persistence` are not specified then the values are generated from uniform distribution. `persistence` in this case is restricted by selected `bounds` (which are either `"usual"`, `"admissible"` or

"restricted" with latter meaning that smoothing parameters should not exceed 0.3). Parameter phi is needed for damped-trend models and is ignored for the others.

randomizer is needed in order to produce error term, the default value for it is rnorm(). In cases with multiplicative error models the generated values are automatically transformed into log-normally distributed. sim.es() allows providing any randomizer with any parameters, but only few of them really make sense. For example, generating log-normal errors with zero mean and standard deviation of 0.1 can be done this way:

```
sim.es("MMN", obs=100, randomizer="rlnorm", meanlog=0, sdlog=0.1)
```

Note that we always need to pass both meanlog and sdlog even if we are fine with the default value of meanlog.

sim.es() also accepts parameter iprob producing intermittent data with fixed probability. If the probability is provided, then data is rounded to integer values.

3.7. Miscellaneous functions

There are several other smaller, but not less important function in "smooth" package. They are available to user, but are needed for main forecasting functions. We briefly discuss them in this subsection.

- iss() – this function produces forecasts for occurrences of time series and is used by other functions, when intermittent parameter is not "none". It allows to construct one of the models mentioned in section 2.3. In case of Croston's model (section 2.3.5) user can also select any ETS model and play around with it. Note that multiplicative error models are advised for intermittent data. User can also provide a desired value of smoothing parameters via persistence variable to both Croston's and TSB models;

- graphmaker() – function creates plot based on passed actuals, forecast, fitted, lower and upper. It also allows defining if legend is needed, the width of prediction interval (if lower and upper are not NULL) and what to print as a title for the graph;

- Error measures, which include: 1. MPE(), Mean Percentage Error; 2. MAPE(), Mean Absolute Percentage Error; 3. SMAPE(), Symmetric

MAPE; 4. `MASE()`, Mean Absolute Scaled Error, equivalent to scaled Mean Absolute Error, sMAE; 5. `RelMAE()`, Relative MAE; 6. `sMSE()`, scaled Mean Squared Error. This variety of errors is needed in order to have better picture of performance of models. For example, errors based on absolute values do not work for intermittent demand, so square errors are needed. All these errors are reported when a model with `holdout=TRUE` is constructed. Literature on error measures is extensive (**?????**), so we are not discussing their properties here;

- `hm()` and `cbias()` – half moment of distributions and complex bias. The former is not used directly by any function, while the second one is reported as "Bias" along with other error measures. "Bias" may take values from -1 to 1. The perfect model should have Bias = 0. **We will provide reference to a paper about Bias soon...**;

3.8. Examples of usage

The detailed examples with graphs and explained outputs can be found in the official vignette of "smooth" package. Here we show some simple examples.

3.8.1. Exponential Smoothing, `es()`

For some random time series from M3 dataset:

```
y <- ts(c(M3$N2457$x,M3$N2457$xx), frequency=frequency(M3$N2457$x),
        start=start(M3$N2457$x))
h <- M3$N2457$h
```

`es()` can be constructed using:

```
es(y, h=h, holdout=TRUE)
```

The function in this case uses the discussed branch and bound algorithm and selects ETS(M,N,N). It also produces an output with brief information about the model and a graph with series, fitted values and point forecasts. If we need prediction intervals, then we run:

```
es(y, h=h, holdout=TRUE, intervals=TRUE)
```

Due to multiplicative error in the model, the intervals will be asymmetric. This is the expected behaviour.

If we save the model (and let's say we want it to work silently):

```
ourModel <- es(y, h=h, holdout=TRUE, silent="all")
```

we can then reuse it for different purposes:

```
es(y, model=ourModel, h=h, holdout=FALSE, intervals=TRUE,
   intervals="np", level=0.93)
```

Or we can just use persistence or initials from one model to construct the other one:

```
es(y, model="MNN", h=h, holdout=FALSE, initial=ourModel$initial)
es(y, model="MNN", h=h, holdout=FALSE, persistence=ourModel$persistence)
```

or provide some arbitrary values:

```
es(y, model="MNN", h=h, holdout=TRUE, initial=1500)
```

3.8.2. Complex Exponential Smoothing, `ces()`, `auto.ces()`

For the same random series from M3 dataset `ces()` can be constructed using:

```
ces(y, h=h, holdout=TRUE)
```

If we want automatic model selection, then we use `auto.ces()` function:

```
auto.ces(y, h=h, holdout=TRUE)
```

If for some reason we want to optimise initial values then we call:

```
auto.ces(y, h=h, holdout=TRUE, initial="o")
```

Or we can call some specific model. For example, with simple seasonality (72):

```
ces(y, seasonality="s", h=h, holdout=TRUE)
```

`ces()` allows use exogenous variables and different types of prediction intervals in exactly the same manner as `es()`. So we can fit a model with exogenous without update first:

```
auto.ces(y, h=h, holdout=TRUE, xreg=x, intervals=TRUE)
```

and then with the update:

```
auto.ces(y, h=h, holdout=TRUE, xreg=x, updateX=TRUE, intervals=TRUE)
```

3.8.3. SSARIMA, `ssarima(), auto.ssarima()`

Simple ARIMA(0,1,1) is constructed by `ssarima()` function by default:

```
ssarima(y, h=h, holdout=TRUE)
```

We could try selecting orders manually, but this can also be done automatically via `auto.ssarima()` function:

```
auto.ssarima(y, h=h, holdout=TRUE)
```

Automatic order selection in SSARIMA with optimised initials does not work well, but we can still ask for it:

```
auto.ssarima(y, h=h, holdout=TRUE, initial="o")
```

This can be seen on example of another time series (which has complicated seasonality):

```
auto.ssarima(M3$N1683, h=18, initial="backcasting")
auto.ssarima(M3$N1683, h=18, initial="optimal")
```

If we save model:

```
ourModel <- auto.ssarima(y, h=h, holdout=TRUE, xreg=x, updateX=TRUE)
```

we can then reuse it:

```
ssarima(y, model=ourModel, h=h, holdout=FALSE, xreg=x, updateX=TRUE)
```

3.8.4. GES, `ges()`

By default `ges()` fits a model GES($1^1, 1^{12}$):

```
ges(y, h=h, holdout=TRUE)
```

But some different orders and lags can be specified. For example:

```
ges(y, h=h, holdout=TRUE, orders=c(2,1), lags=c(1,12))
```

Function `auto.ges()` is not yet implemented in "smooth", but manual selection allows to conclude that GES($2^1, 1^{12}$) has the lowest AIC amongst other possible GES models. In theory inclusion of more orders and lags should lead to decrease of MSE. However this is not the case in our implementation, because currently we use linear programming optimisers, which may leave us with suboptimal values in cases when parameter space is wide.

In addition to standard values that other functions accept, GES accepts predefined values for transition matrix, measurement and persistence vectors. For example, something more common can be passed to the function:

```
transition <- matrix(c(1,0,0,1,1,0,0,0,1),3,3)
measurement <- c(1,1,1)
ges(y, h=h, holdout=TRUE, orders=c(2,1), lags=c(1,12),
    transition=transition, measurement=measurement)
```

The resulting model will be equivalent to ETS(A,A,A). However due to different initialisation of optimisers and different method of number of parameters calculation, `ges()` above and `es(y, "AAA", h=h, holdout=TRUE)` will lead to different models.

3.8.5. Simulate functions, `sim.es()`*,* `sim.ssarima()`*,* `simulate()`

Let's start from something simple. For example, monthly data generated from ETS(A,N,N), 120 observations:

```
ourSimulation <- sim.es("ANN", frequency=12, obs=120)
```

The resulting `ourSimulation` object contains: `ourSimulation$model` – name of ETS model used in simulation; `ourSimulation$data` – vector of simulated data; `ourSimulation$states` – matrix of states, where columns contain different states and rows corresponds to time; `ourSimulation$persistence` – vector of smoothing parameters used in simulation (in our case generated randomly); `ourSimulation$residuals` – vector of errors generated in the simulation; `ourSimulation$occurrences` – vector of demand occurrences (zeroes and ones, in our case only ones); `ourSimulation$likelihood` – true likelihood function for the used generating model.

We can plot produced data, states or residuals in order to see what was generated. This is done using `plot(ourSimulation)`.

`sim.ssarima()` function allows defining any AR, MA parameters, constant and initial values. In cases of non-zero differences, constant is treated as drift value. This is a very flexible tool, which has some advantages over `sim.arima()` function from stats package. The possibility of defining whatever you want however you want is one of those. The function also allows generating data with multiple seasonalities and producing intermittent data.

The simplest example of usage is:

```
ourSimulation <- sim.ssarima(frequency=12, obs=120)
```

This generates data from ARIMA(0,1,1).

If we have estimated some model on data and want to generate something using exactly the same model, we can use `simulate()` function:

```
ourModel <- es(y, h=h, holdout=TRUE)
ourSimulation <- simulate(ourModel, nsim=1000)
```

Similarly we can do things with SSARIMA:

```
ourModel <- auto.ssarima(y, h=h, holdout=TRUE)
ourSimulation <- simulate(ourModel, nsim=1000)
```

For more examples, see vignettes of the package.

3.9. Methods for the class "smooth"

There are several functions that can be used together with `es()`, `ces()`, `ssarima()` and `ges()` models. So when a model is saved to some object `ourModel`, these function will do some things. Here's the list with some explanations:

1. `summary(ourModel)` – function prints brief output with explanation of what was fitted, with what parameters and errors;
2. `fitted(ourModel)` – fitted values;
3. `forecast(ourModel)` – point and interval forecasts. This is needed for compatibility with Rob Hyndman's "forecast" package, however "smooth" does not include that package in dependencies and uses a forecast function similar to Rob's. So when you attach both "smooth" and "forecast" you may see some warnings about masking "forecast" function. But nothing bad happens here. `forecast(ourModel)` returns object of class "forecastSmooth";
4. `residuals(ourModel)` – residuals of constructed model;
5. `AIC(ourModel)`, `BIC(ourModel)` and `AICc(ourModel)` – information criteria of the constructed model. `AICc()` function is not a standard "stats" function and is introduced by "smooth";
6. `plot(ourModel)` – plots states of constructed model;
7. `simulate(ourModel)` – produces data simulated from provided model. Currently only available for ETS;
8. `summary(forecast(ourModel))` – prints point and interval forecasts;
9. `plot(forecast(ourModel))` – produces graph with actuals, forecast, fitted and intervals using `graphmaker()` function.

3.10. Returned values

For the list of returned values with explanation, see help pages in R.

4. Conclusions

Package "smooth" has several functions that are based on state-space models. It supports additive, multiplicative and mixed models, which are currently implemented in exponential smoothing, represented by `es()` function and several other functions.

`es()` function is flexible and has features that no package in R has yet introduced. These include: exogenous variables together with any ETS model, intermittent state-space model using Croston's and TSB, cost functions based on multiple steps ahead errors, different types of intervals, including **?**, model selection and forecasts combination using information criteria, different types of initialisation and more.

"smooth" package also introduces State-space SARIMA model with possible multiple seasonalities and automatic SARIMA order selection mechanism. Implementation of `ssarima()` allows to avoid using any statistical tests, thus decreasing uncertainty about the correct model. Although order selection mechanism in `auto.ssarima()` could still be optimised, it already returns decent results.

"smooth" also introduces brand new Complex Exponential Smoothing and Generalised Exponential Smoothing, that make life of forecaster if not easier than definitely more exciting.

In addition advanced simulation functions are included in the package. They allow full control over parameters of models, which leads to larger variety of data to generate.

All the functions in "smooth" are flexible and allow users who read the manual doing things they want in a way they desire. The package is still developing and we hope to introduce new features in upcoming releases.