Onlineforecast: An R package for adaptive and recursive forecasting

Bacher, Peder; Bergsteinsson, Hjörleifur G.; Frölke, Linde; Sørensen, Mikkel L.; Lemos-Vinasco, Julian; Liisberg, Jon; Møller, Jan Kloppenborg; Nielsen, Henrik Aalborg; Madsen, Henrik

Published in:
R Journal

Link to article, DOI:
10.32614/RJ-2023-031

Publication date:
2023

Document Version
Early version, also known as pre-print

Pre-print version: September 28, 2021 - 01:17.

onlineforecast: An R package for adaptive and recursive forecasting

Peder Bacher **Hjörleifur G. Bergsteinsson** **Linde Frölke**

Mikkel L. Sørensen **Julian Lemos-Vinasco** **Jon Liisberg**

Jan Kloppenborg Møller **Henrik Aalborg Nielsen** **Henrik Madsen**

Abstract

Systems that rely on forecasts to make decisions, e.g. control or energy trading systems, require frequent updates of the forecasts. Usually, the forecasts are updated whenever new observations become available, hence in an online setting. We present the R package onlineforecast that provides a generalized setup of data and models for online forecasting. It has functionality for time-adaptive fitting of linear regression-based models. Furthermore, dynamical and non-linear effects can be easily included in the models. The setup is tailored to enable effective use of forecasts as model inputs, e.g. numerical weather forecast. Users can create new models for their particular system applications and run models in an operational online setting. The package also allows users to easily replace parts of the setup, e.g. use kernel or neural network methods for estimation. The package comes with comprehensive vignettes and examples of online forecasting applications in energy systems, but can easily be applied in all fields where online forecasting is used.

Keywords: Recursive estimation, Adaptive, Non-linear transformation, Time series, Energy, Online Forecasting, Prediction, R.

1. Introduction

Time series analysis and forecasting is of indispensable importance to numerous practical fields such as business, finance, science and engineering (Cryer and Chan 2008). Time series analysis is the process of statistical modelling of time series, i.e. data which is sampled at different points in time over a period – often with a constant distance in time, i.e. equidistant. Classical time series models for a single equidistant time series use past values of the response variable (model output) as predictors (inputs). In this way appropriate models describing the inherent auto-correlation structure of the time series can be realized. Such models are exponential smoothing (e.g. Holt-Winters), AutoRegressive (AR) and Moving Average (MA), and usually the combination of the latter two as ARMA models. When multiple correlated times series are at hand, they can be used as model inputs to improve forecasts. They are then called eXogenous variables and the classical model becomes an ARMAX – hence the *X* indicates that input variables are included.

The use of ARMAX models and variations thereof is widespread (De Gooijer and Hyndman 2006), especially in modelling of energy systems due to the high dependency between e.g. weather, load, renewable generation and periodic phenomena. Load forecasting is an obvious example. A nice overview for electric load forecasting is given by Alfares and Nazeeruddin (2002) and Hong and Fan (2016), and for heat load by Dotzauer (2002), who demonstrates the dependency between the response variable, heat load, and the predictor, ambient temperature, using a piecewise linear function. It is also proposed to model the daily and weekly diurnal using hours of the week as inputs. For solar power forecasting (Kleissl 2013) the improvement from an autoregressive (AR) to an AR with exogenous input (ARX), where the ARX model uses numerical weather predictions (NWPs) as inputs, is demonstrated by Bacher, Madsen, and Nielsen (2009). The ARX model uses past observations and NWPs of global irradiance to forecast the power production from PV systems and the ARX model obtains higher accuracy than the AR model. Bacher, Madsen, Nielsen, and Perers (2013) identifies exogenous variables that are suitable for forecasting heat load of a building, with similar models.

Energy systems are time-varying systems as they usually change over time due to wear and contamination, like dirt on solar panels or change in usage. For example, with new tenants in a house the dependency between the heat load and other variables, like calendar and temperature, changes. Therefore, a forecast model needs to adapt – the model coefficients are not optimal if they are constant, they need to be updated and allowed to change over time. The Recursive Least Square (RLS) method provides a recursive estimation scheme for the coefficients in regression models, where they are updated at each step when new data becomes available. Introduction of a forgetting factor in RLS allows control on how fast the coefficients can change over time – this is referred to as adaptive recursive estimation, with exponential forgetting, in linear regression and autoregressive models. The method is described by Ljung and Söderström (1983) and advances has been made since then, see e.g. (Engel, Mannor, and Meir 2004).

A wide range of existing software useful for time series forecasting is currently available – all have their suitable applications (Chatfield and Xing 2019). In the following an overview is given, where we stick to listing the most relevant R packages, which are suited for online forecasting and are available on CRAN.

Exponential smoothing models are popular and simple methods for time series. In the exponential smoothing past observations are exponentially down weighted, thus older observations have less impact than newer. The *Holt-Winters* procedure, where three smoothing constants are used to describe the variation in time of the parameters, is one of the most famous exponential smoothing methods. Winters (1960) extended the double exponential smoothing formulation by Holt to capture the seasonality. The `HoltWinters()` function from the stats package estimates parameters of the *Holt-Winters* procedure. The fable package (O'Hara-Wild, Hyndman, and Wang 2020) provides a state space framework to create exponential smoothing models in the function `ETS()`. The function is based on the exponential smoothing framework presented by Hyndman, Koehler, Ord, and Snyder (2008). The smooth package also provides methods for exponential smoothing.

The classical ARMA models can be estimated with the `arima()` function from the stats package and the `Arima()` function from the forecast package (Hyndman and Khandakar 2008), however they are not suited for fitting ARMAX models, i.e. including exogenous variables. Packages like marima (Spliid 1983), KFAS, sysid and dlm (Petris 2010) can be used for fitting ARMAX models. Spliid (1983) proposed a very fast and simple method for parameter estimation in large multivariate ARMAX models with a pseudo-regression method that repeats the regression estimation until it converges. The other packages represent time series and regression models as state space models and use a Kalman or Bayesian filter to include exogenous variables in the model, and optimally reconstruct and predict the states.

State space modelling is frequently used to describe time series data from a dynamical system, e.g. a falling body, see (Madsen 2007). The dynamical system can in such cases be written as differential equations or difference equations. State space models use filter techniques to optimally reconstruct and predict the states, e.g. the Kalman filter, the extended Kalman filter or other Bayesian filters. This gives the possibility of tracking the coefficients over time, i.e. time-varying parameter estimation. The KFAS package (Helske 2017) provides state space modelling, where the observations come from the exponential family, e.g. Gaussian or Poisson. The ctsm-r package provides a framework for identifying and estimating partially observed continuous-discrete time state space models, referred to as grey-box models. This modelling approach bridges the gap between physical and statistical modelling using Stochastic Differential Equations (SDEs) to model the system equations in continuous time and the measurement equations in discrete time. Packages for discrete time state space modelling are: dlm for Bayesian analysis of dynamic linear models, MARSS and SSsimple for fitting multivariate state space models.

For non-parametric time series models the number of available packages are growing rapidly. NTS provides simulation, estimation, prediction and identification for non-linear time series data. It also includes threshold autoregressive models (e.g. self-exciting threshold autoregressive models) and neural network estimation. tsDyn provides methods for estimating non-parametric times series models, including neural network estimation. Neural network, deep learning and machine learning methods have been more developed for the programming language Python, but equivalent is available in R for most methods. Recurrent neural networks are in the rnn, the keras and tensorflow packages. Additive time series models, where non-linear trends are fitted with seasonality patterns are in prophet.

Some packages can be useful for forecast evaluation, e.g. ForecastTB presented in (Bokde,

Yaseen, and Andersen 2020). Packages like forecastML and modeltime provide functionality that simplifies the process of multi-step-ahead forecasting with standard machine learning algorithms. This purpose of handling multi-step-ahead forecasts is also a key feature of the onlineforecast package. The classical time series models, such as ARMAX and Exponential Smoothing models, are mostly optimal for modelling Linear Time Invariant (LTI) systems, however most systems are not LTI. Furthermore, since a model is always a simplification of reality, optimal multi-step forecasting is often not possible with the classical models, especially when using exogenous inputs. For optimal multi-step ahead forecasting the models must be tuned for each horizon – which is exactly what the onlineforecast package does.

The onlineforecast package builds on an advanced model setup for forecasting. This model setup was developed for applications such as forecasting wind power (Nielsen, Nielsen, and Madsen 2002) and thermal loads in district heating (Nielsen and Madsen 2006). The significance of the package is in the "online" term, indicating that the model is updated when new observations becomes available – recursively updating the coefficients and generating new forecasts at every point in time.

The objective of the package is to make it easy to set up and optimize models for generating online multi-step forecasts. The package contains functionalities not directly available elsewhere:

- Use of forecasts, e.g. NWPs, as input to multi-step forecast models.
- Application of non-linear models with non-parametric and coefficient varying techniques.
- Optimal tuning of models for multi-step horizons.
- Recursive estimation for tracking time-varying systems.

The package also provides a framework for handling data and setting up models, which makes it easy to apply in a wide range of forecasting applications.

A model is an approximation to the real world, thus it will always be a simplification and can never predict perfectly. One of the main challenges of identifying a good forecast model is to find the most informative input variables and the best structure of the model. The package provides functionality for defining, validating and selecting models in a systematic way.

To introduce the onlineforecast models consider the simplest model with one input. It's the linear model for the k'th horizon

$$Y_{t+k|t} = \beta_{0,k} + \beta_{1,k} u_{t+k|t} + \varepsilon_{t+k|t} \tag{1}$$

where $Y_{t+k|t}$ is the response variable and $u_{t+k|t}$ is the input variable. The coefficients are $\beta_{0,k}$ and $\beta_{1,k}$, note that they are subscripted with k to indicate that they are estimated for each horizon. The error $\varepsilon_{t+k|t}$ represents the difference between the model prediction and the observed value for the k-step horizon. The interpretation of the subscript notation $t+k|t$ on a variable, is that it's the k-step prediction calculated using only past information at time t, usually referred to either "conditional on time t" or "given time t".

The package offers to estimate the coefficients using either the Least Squares (LS) or Recursive Least Squares (RLS) method. In the LS method the coefficients are constant, while the in RLS method the coefficients can change over time

$$Y_{t+k|t} = \beta_{0,k,t} + \beta_{1,k,t} u_{t+k|t} + \varepsilon_{t+k|t} \tag{2}$$

as indicated with the t on the coefficients. This allows for tracking changes occurring over time – for example when forecasting the heat load of a building, the occupants can change behaviour and thus affect the heating patterns, which can be adapted to automatically by the RLS method.

The package allows for easy definition of transformations and thus the possibility to fit non-linear models e.g.

$$Y_{t+k|t} = \beta_{0,k,t} + \beta_{1,k,t} f(u_{t+k|t}; \alpha) + \varepsilon_{t+k|t} \tag{3}$$

where the function $f(u_{t+k|t}; \alpha)$ is some non-linear function of the input $u_{t+k|t}$ with parameter α e.g. a low pass filter on the outdoor temperature to model building heat dynamics. The package sets up tuning of the non-linear function parameters, e.g. if the parameter α determines the degree of low-pass filtering it can be tuned with an optimizer to match the system dynamics inherent in the data at hand.

An example of generated forecasts can be appreciated in Figure 1. Hourly forecasts up to 36 steps ahead of heat load in a single building are shown for three consecutive steps. This is the typical structure of forecasts generated with the package. It can be seen how the forecasts change slightly as they are updated in each step, e.g. around 12:00 at day 2, hence horizon $k = 23$ in the upper plot, which corresponds to $k = 21$ in the lower plot.

1.1. Vignettes

A great way to get actual hands-on experience is through vignettes. They are available when installing the package and on the website onlineforecasting.org, where also examples of different forecast applications can be found. The packages vignettes are:

- setup-data covers how data must be set up. The vignette goes into details on how observations and model inputs (forecasts) are set up. The vignette also focus on the importance of aligning forecasts correctly in time.
- setup-and-use-model focus on how to set up a model and use it to generate forecasts.
- model-selection demonstrates how model selection can be carried out.
- forecast-evaluation covers evaluation of forecasts, and how to use this information to improve a model.
- online-updating demonstrates how to update a model in actual operation, when new observations become available. This functionality isn't described in the R examples in the present paper.

Furthermore, one vignette is available only on the website:

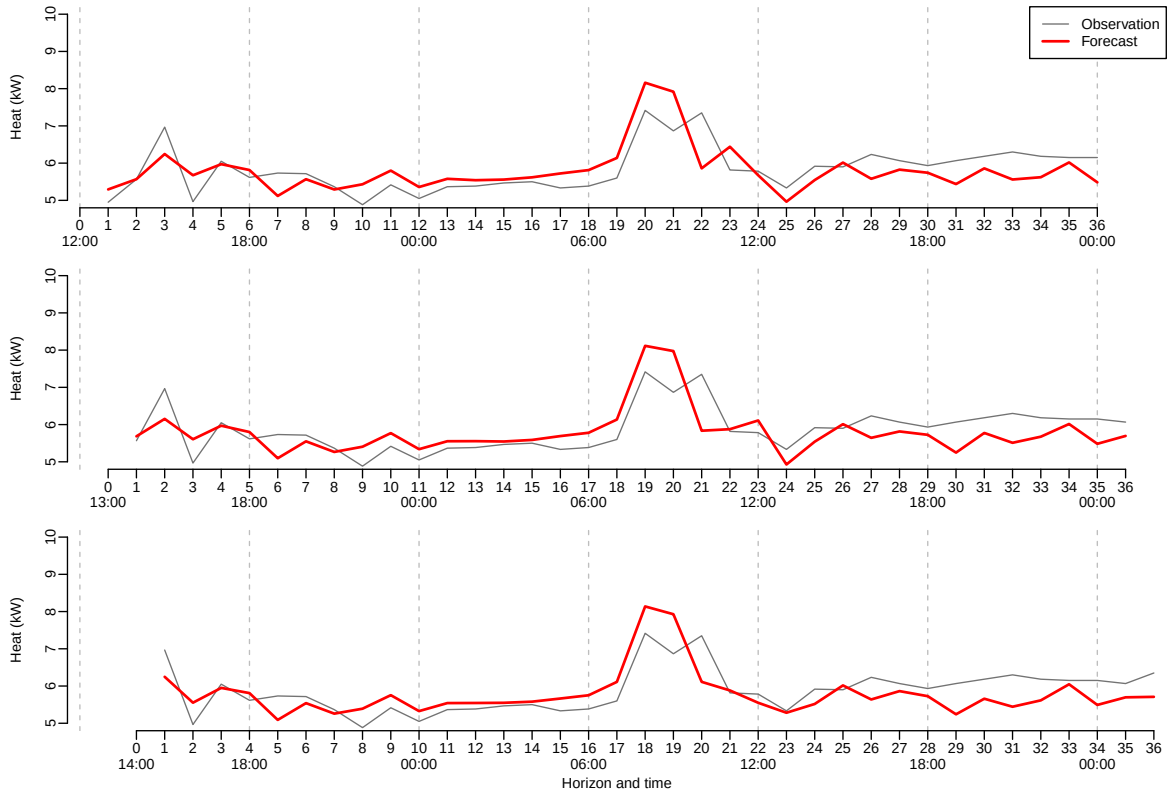


Figure 1: Example of hourly load forecasts at three consecutive time steps. The upper is calculated at 12:00, the middle is calculated at 13:00 and lower at 14:00. It can be seen how the forecasts change slightly as they are updated in each step, most clearly seen around 12:00 on day 2.

- nice-tricks provides some useful tips on how to make the work flow easier with the package.

1.2. Paper structure

The structure of the paper is the following: In Section 2 the notation used in the paper and how to set up data is introduced. The core methodology is presented in Section 3 and important aspects of forecast modelling are outlined in Section 4. In Section 5 examples with R code are presented to provide a short hands-on tutorial. The paper ends with summary and conclusions in Section 6.

In addition, three appendixes are included in the paper. In Appendix A some guidelines on mathematical notation of forecast models are provided. In Appendix B the functions used for transformations are detailed and in Appendix C the regression schemes are covered in full detail.

2. Notation and forecast matrices

The notation in this article follows Madsen (2007) as close as possible. All time series considered are equidistant sampled and the sampling period is normalized to 1. Hence, the time t is simply an integer indexing the value of a variable at time t. The same goes for k which index the forecast horizon k steps ahead.

In the onlineforecast setup, forecasts are calculated at time t for each horizon up to n_k steps ahead. To achieve the desired notation that can deal with overlapping time series, a two dimensional index is required. The notation used is

$$u_{t+k|t} \tag{4}$$

which translates to: the value of variable u at time $t+k$ *conditional* on the information available at time t. The conditional term is indicated by the bar $|$. Thus, for $k > 0$ this is a forecast available at t and k is the horizon.

When writing a forecast model the following convention is used, here a simple example

$$Y_{t+k|t} = \beta_{0,k} + \beta_{1,k} u_{t+k|t} + \varepsilon_{t+k|t} \tag{5}$$

where $Y_{t+k|t}$ is the model output, $\beta_{0,k}$ and $\beta_{1,k}$ are the coefficients and $\varepsilon_{t+k|t}$ with $\mathrm{Var}(\varepsilon_{t+k|t}) = \sigma_k^2$ is the error. The error process and variance σ_k^2 is thus separate for each horizon. Note, that the model is fitted separately for each horizon, so the coefficients take different values for each horizon, and the predictions and errors are separated for each horizon. This was a simplified example, see Appendix A on how to write the full forecast models.

2.1. Forecast matrix

A forecast matrix is the format of forecast data in the onlineforecast setup. See examples in the setup-data vignette. Data must have this format in order to be used as model input, and

the forecasts generated are in this format. The forecast matrix holds for any past time *the latest available forecast along the row* for the corresponding time

$$\boldsymbol{u}_n = \begin{array}{c} \begin{array}{ccccc} \mathbf{k0} & \mathbf{k1} & \mathbf{k2} & \dots & \mathbf{k}n_k \end{array} \\ \begin{pmatrix} u_{1|1} & u_{2|1} & u_{3|1} & \dots & u_{1+n_k|1} \\ u_{2|2} & u_{3|2} & u_{4|2} & \dots & u_{2+n_k|2} \\ \vdots & \vdots & \vdots & & \vdots \\ u_{t-1|t-1} & u_{t|t-1} & u_{t+1|t-1} & \dots & u_{t-1+n_k|t-1} \\ u_{t|t} & u_{t+1|t} & u_{t+2|t} & \dots & u_{t+n_k|t} \\ \vdots & \vdots & \vdots & & \vdots \\ u_{n|n} & u_{n+1|n} & u_{n+2|n} & \dots & u_{n+n_k|n} \end{pmatrix} \end{array} \begin{array}{c} \rightarrow \mathbf{horizon/time} \downarrow \\ 1 \\ 2 \\ \vdots \\ t-1 \\ t \\ \vdots \\ n \end{array} \tag{6}$$

where

- t is the counter of time for equidistant time points with sampling period 1 (note that t is not included in the matrix, it is simply the row number).
- n is the number of time points in the matrix. Hence, the data is available and can be used as model input at time $t = n$.
- n_k is the longest forecasting horizon.
- The column names (in R) are indicated above the matrix, they are simply a '**k**' concatenated with the value of k, e.g. n_k in the last column.

Note, that the **k0** column holds values with forecast horizon $k = 0$, which could be real time observations. Usually, only the horizons to be forecasted should be included, hence often **k0** is not needed.

For example with a prediction horizon $n_k = 24$ at $t = 100$, we will have the forecast matrix

$$\boldsymbol{u}_{100} = \begin{array}{c} \begin{array}{ccccc} \textbf{k0} & \textbf{k1} & \textbf{k2} & \dots & \textbf{k24} \end{array} \\ \begin{pmatrix} u_{1|1} & u_{2|1} & u_{3|1} & \dots & u_{25|1} \\ u_{2|2} & u_{3|2} & u_{4|2} & \dots & u_{26|2} \\ \vdots & \vdots & \vdots & & \vdots \\ u_{99|99} & u_{100|99} & u_{101|99} & \dots & u_{123|99} \\ u_{100|100} & u_{101|100} & u_{102|100} & \dots & u_{124|100} \end{pmatrix} \end{array} \begin{array}{c} \rightarrow \textbf{horizon/time} \downarrow \\ 1 \\ 2 \\ \vdots \\ 99 \\ 100 \end{array} \tag{7}$$

In Section 5.1 examples of how data and forecast matrices are set up in R are given.

3. Two-stage modelling procedure

A widespread approach to model non-linear functional relations between inputs and output is a two-stage modelling procedure. See, e.g. Breiman and Friedman (1985) and Weisberg (2005) for direct transformation of predictor variables, and Hastie, Tibshirani, and Friedman (2009) for non-parametric transformation techniques (basis functions). Using transformations allows for fitting complex models with robust and fast estimation techniques.

In the first stage, the *transformation stage*, the inputs are mapped by some function – potentially into a higher dimensional space. In the second stage, the *regression stage*, a linear regression model[1] is applied between the transformed inputs and the output. An exemplification of this is presented in the following.

As an example a model with two inputs is presented. In this model the transformation stage consists of generating an intercept and mapping the two inputs (they are set up as forecast matrices $\boldsymbol{u}_{1,t}$ and $\boldsymbol{u}_{2,t}$)

$$\text{Intercept:} \quad x_{0,t+k|t} = 1 \tag{8}$$

$$\text{Input 1:} \quad x_{1,t+k|t} = f_1(u_{1,t+k|t}, \boldsymbol{\alpha}_1) \tag{9}$$

$$\text{Input 2:} \quad \boldsymbol{x}_{2,t+k|t} = f_2(u_{2,t+k|t}, \boldsymbol{\alpha}_2) \tag{10}$$

where the f's are transformation functions that map the inputs to regressors. Note, that the intercept is simply a constant passed on to the regression. The transformations result in multiple inputs for the regression – the latter actually as multiple variables indicated by the bold font notation. In the regression stage the linear model

$$Y_{t+k|t} = \beta_{0,k} x_{0,t+k|t} + \beta_{1,k} x_{1,t+k|t} + \boldsymbol{\beta}_{2,k}^T \boldsymbol{x}_{2,t+k|t} + \varepsilon_{t+k|t} \tag{11}$$

[1] In the remaining of the text, when the term "regression" is used it is implicit that it's "linear regression".

is fitted. The regression is carried out separately for each horizon k.

Thus, the combined model has:

- An intercept
- Two inputs: $u_{1,t+k|t}$ and $u_{2,t+k|t}$
- Output: $Y_{t+k|t}$
- Transformation functions: f_1 and f_2
- Transformation parameters: $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$
- Regression coefficients: $\beta_{0,k}$, $\beta_{1,k}$ and $\boldsymbol{\beta}_{2,k}$

Some transformation parameters should be optimized for the data at hand, e.g. a low-pass filter coefficient depends on the system dynamics. The same goes for some parameters related to the regression scheme, e.g. the forgetting factor (introduced below). We will refer to them together as "offline" parameters. The onlineforecast package provides a setup where the offline parameters can be optimized using a heuristic optimization (e.g., a BFGS quasi-Newton method). The default score, which is minimized, is the Root Mean Square Error (RMSE) of the predictions – hence offline parameters in the model above, given data from the period, $t = 1, 2, \ldots, n$, are found by solving

$$\min_{\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2} \quad \frac{1}{n-k} \sum_{t=1}^{n-k} (y_{t+k} - \hat{y}_{t+k|t}(\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2))^2 \tag{12}$$

Naturally, other scores can be minimized (e.g. MAE or the Huber psi-function, however the regression schemes should be modified accordingly, which is not trivial).

The regression coefficients are calculated with a closed-form scheme: either with the Least-Squares (LS) or the Recursive Least-Squares (RLS) scheme – in the latter the coefficients are allowed to vary over time. In both schemes the coefficients are calculated separately for each horizon k. In Appendix C both schemes are presented in full detail.

In the LS scheme the coefficients are gathered in the vector $\hat{\boldsymbol{\beta}}_k$, which is constant in the entire period. The output vector is $\boldsymbol{y}_{k,n}$ and for a given value of the transformation parameters (i.e. here $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$) the transformed data is calculated and set up in the design matrix $\boldsymbol{X}_{k,n}$. The LS coefficients are then calculated by

$$\hat{\boldsymbol{\beta}}_k = (\boldsymbol{X}_{k,n}\boldsymbol{X}_{k,n})^{-1}\boldsymbol{X}_{k,n}\boldsymbol{y}_{k,n} \tag{13}$$

and the predictions by

$$\hat{\boldsymbol{y}}_{k,n} = \boldsymbol{X}_{k,n}\hat{\boldsymbol{\beta}}_k \tag{14}$$

where $\hat{\boldsymbol{y}}_{k,n} = \begin{bmatrix} \hat{y}_{1+k|1} & \hat{y}_{2+k|2} & \ldots & \hat{y}_{n|n-k} \end{bmatrix}^T$ are the predictions. Note, that for the LS scheme the predictions are "in-sample", since from the entire period was used for the coefficient estimation.

In the RLS scheme the coefficients are calculated recursively, meaning that they are updated at each time t when stepping through the period. In each update only the "newly" obtained data at t is used, thus only past data is used in the calculations, which makes the calculated RLS coefficients and predictions "out-of-sample" (opposed to LS). Furthermore, the coefficients vary such that the model adapts to the data over time. When writing the model as in Equation (11) this can be indicated with a t subscript on the coefficients, i.e. $\beta_{0,k,t}$, $\beta_{1,k,t}$ and $\boldsymbol{\beta}_{2,k,t}$. The level of adaptivity can be controlled by setting the forgetting factor λ in the RLS scheme to a value between 0 and 1. For $\lambda = 1$ all past data at t is equally weighted. For $\lambda < 1$ higher weight is put on recent data – the smaller the value the faster the model adapts to data. By optimizing the forgetting factor as an offline parameter the model adaptivity can be tuned to optimally track system changes over time.

An important point to notice is, that the offline parameters are always constant for the given period, hence all predictions are essentially "in-sample". However, depending on the regression scheme there is a difference: with the LS scheme the regression coefficients are constant through the period, thus the predictions are (fully) "in-sample", where as with the RLS scheme they adapt through the period and the predictions are "out-of-sample" (i.e. except for the offline parameters). This makes a difference, since model over-fitting is less of a problem when using the RLS scheme, although still problems can occur, e.g. if the forgetting factor is optimized to be close to 1, because no systematic change occurred in the training period, but changes appear in new data.

The typical onlineforecast setup is to optimize the (usually few) offline parameters in an "offline" setting, but calculate the regression coefficients adaptively with the RLS. This has the advantage that the model adapts and tracks the systematic changes in input-output relations, while keeping the setup computationally very effective – updating the coefficients and calculating a forecast at each time t takes few operations, since only the newly available data is used. The more computationally heavy optimization of offline parameters can be carried when computational resources are available (e.g. every week for hourly forecasts).

In the following sections more information, on the package functions for the two stages, are presented.

3.1. Transformations stage

In the transformation stage the inputs are mapped using some function as demonstrated above, for more examples see the setup-and-use-model vignette. The onlineforecast package has functions available for most common use, however it is easy to write and use new functions as they are simply R functions, which return a forecast matrix (or a list of them), for more details see Section 5.2.1. The currently available transformation functions are:

- Low-pass filtering, `lp()`: A low-pass filtering for modelling linear dynamics as a simple RC-model. See e.g. Nielsen and Madsen (2006) for further information.
- Basis splines, `bspline()`: Use the `bs()` function for calculating regression splines basis functions.
- Periodic basis splines, `pbspline()`: Use the `pbs()` function for calculating periodic

regression splines basis functions.

- Fourier series, `fs()`: Fourier series as periodic regression basis functions.
- Auto-regressive, `AR()`: For including Auto-Regressive (AR) terms.
- Intercept, `one()`: Generates a forecast matrix of ones, i.e. intercept.

In the following subsection, the low-pass filtering is shortly described below. In Appendix B the other transformation function are presented and in Section 5 a full example with R code is provided.

The implementation in onlineforecast allows all parameters, which are used in some way (except the regression coefficients), to be included in an optimization – using any available optimizer i R. This includes e.g. the RLS forgetting factor, knot points or order of splines – hence both continuous and integer variables. This functionality is achieved using a simple syntax as explained in Section 5.

Low-pass filtering

When modelling time series from linear dynamical systems, the classical ARMAX model is often the optimal choice (Madsen 2007). However, for multi-step forecasting this is often not the case, especially for longer horizons. In the onlineforecast setup, where the regression model is fitted for each horizon, a "trick" can be used for modelling linear dynamics: simply apply a filter on the input and then use the filtered input in the regression stage. For example, dynamics between ambient air temperature and indoor temperature are slow due to the thermal mass of the building. Therefore, it is reasonable that the dynamics between the ambient air temperature and the heat demand (when keeping a desired indoor temperature) can be modelled using a low-pass filter. This technique is successfully applied for energy modelling using physical knowledge, see Nielsen and Madsen (2006) for modelling heat load in district heating and Bacher *et al.* (2013) for forecasting single buildings heat load.

In the package the simple low-pass filter

$$x_{t+k|t} = \frac{(1-a)u_{t+k|t}}{1-ax_{t-1+k|t-1}}, \tag{15}$$

is implemented. The filter coefficient a must take a value between 0 and 1 and should be tuned to match the time constant optimal for the particular data. When the current implemented low-pass filter is applied in the transformation stage, on some forecast matrix $\boldsymbol{u}_{t+k|t}$, the filter is applied on each column. Hence, independently for each horizon k. More advanced filters can be implemented. See Appendix B.1 for a more detailed description.

3.2. Regression stage

As described above and in full detail in Appendix C, the regression model takes transformed data from a period $t = 1, 2, \ldots, n$ and is fitted separately for each horizon k, i.e. the model structure remains the same, only the coefficients change with the horizon. In the presented

example in the following the regression model is as stated in Equation (11) – it's implicit that the regression is linear and that the coefficients are calculated using the LS or RLS scheme, both are closed-form minimization of the *RMSE* of the predictions. The main differences between the two schemes have been outlined above and it should be clear that in most settings the RLS is preferable.

Two fundamental assumptions are behind the optimality of least squares predictions, hence the minimizing the *RMSE*. Both assumptions are related to error process $\varepsilon_{t+k|t}$ in such models:

- The one-step error $\varepsilon_{t+1|t}$ is white noise, hence a sequence of independent, identically distributed (i.i.d.) random variables.
- They are drawn from a normal distribution, i.e. $\varepsilon_{t+k|t} \sim N(0, \sigma_k^2)$.

The latter assumption is not very important, since first of all the LS ensures the best and un-biased estimation of the conditional mean, which is often the wanted and optimal point prediction (Madsen 2007). For example, one important feature is that sums of least squares predictions are also un-biased, e.g. the daily sum of hourly LS predictions are good predictions of the daily total. The i.i.d. assumption should be checked during model validation, as described in the following section, with the Auto-Correlation Function (ACF) for the one-step ahead residuals, as well as the Cross-Correlation Function (CCF) between the one-step ahead residuals and the inputs, as demonstrated by Bacher *et al.* (2013).

4. Model selection and validation

In this section the model selection techniques implemented in the package, and how model validation can be carried out, are presented.

4.1. Model selection

In statistics different model selection procedures are used (Madsen and Thyregod 2010). Essentially, a backward or a forward selection procedure can be applied, or some combined approach. In the onlineforecast package both procedures are implemented, as well as a combined approach. The following is a short description of the stepping process of each procedure implemented, for examples of this see the model-selection vignette.

In each step of the selection process two properties of the model can be modified:

- Model inputs: In each step inputs can either be removed or added.
- Integer offline parameters: In each step integer parameters, such as the number of knot points in a basis spline or the number of harmonics in a Fourier series, can be counted one up or down.

In each step of the process the offline parameters are first optimized to minimize the score for each modified model (in most cases the appropriate score is the RMSE in Equation (12)

summed for selected horizons). Then the scores of the modified models are compared with the score of the currently selected model and the model with the lowest score is selected for the next step. This continues until no further improvement of the score is achieved and the model with lowest score is selected. It's important to note, that the implemented procedure should only used with the RLS scheme, with the LS scheme the score is calculated fully in-sample leading to over-fitting. For the LS an F-test should be applied, however that is currently not implemented.

A model must be given to initialize the stepping process. A backward selection will start with the full model and one-by-one remove inputs and count down parameters. A forward selection will start with the null model and, taking from a provided full model, add inputs and count up parameters. If the direction "both" is set then in each step, inputs will either be removed and added, and parameters will be counted both up and down.

4.2. Model validation

The most important aspects on validation of forecast models are discussed in this section, see the forecast-evaluation vignette for examples.

Training and test set

One fundamental caveat in data-driven modelling is over-fitting. This can easily happen when the model is fitted (trained) and evaluated on the same data. There are essentially two ways of dealing with this: Penalize increased model complexity (regularization) or divide the data into a training set and test set (cross-validation) (Tashman 2000).

In most forecasting applications the easiest and most transparent approach is some cross-validation – many methods for dividing into sets are possible. In the onlineforecast setup, when a model is fitted using a recursive estimation method (like the RLS) only past data is used when calculating the regression coefficients, so there is no need for dividing into a training set and a test set.

The offline parameters (like the forgetting factor and low-pass filter coefficients) are optimized on a particular period, hence over-fitting is possible, however it's most often very few parameters compared to the number of observations – so the it's very unlikely to over-fit a recursive fitted model in this setup.

For non-recursive fitting it is naturally important to divide into a training and a test set – which is easily done in the onlineforecast setup using the `scoreperiod` variable as demonstrated in Section 5.

Scoring

Scoring forecasts can be done in many ways, however in the onlineforecast, where the conditional mean is estimated and when using the RLS scheme, it is straight forward to choose the Root Mean Square Error (RMSE) in Equation (12) as the best score to use. When using the LS scheme it can be favourable to include regulation to avoid over-fitting, hence AIC or

BIC is preferable.

One important point when comparing forecasts is to only include the complete cases, i.e. forecasts at time points with no missing values across all horizons and across all evaluated models. A function for easy selection of only complete cases given multiple forecasts is implemented, see the examples in Section 5.4.

Residual analysis

Analysing the residuals is an important way to validate that a model cannot be further improved or learn how it can improved. The main difference from classical time series model validation, where only the one-step ahead error is examined, is that multiple horizons should be included in the analysis.

The two most important analysis:

- Plot residual time series to find where large forecast errors occur. The accumulated residuals are also often useful to examine, e.g. cumulative squared error plot.
- Plot scatter plots of the residuals vs. other variables to see if any apparent dependencies are not described by the model.

In order to dig a bit more into the result of the recursive estimation the regression coefficients can be plotted over time. In this way it is possible to learn how the relations between the variables in the model evolve over time. If drastic changes are found in some periods it might be worthwhile to zoom into those periods to learn what causes these and potentially how to improve the model. In case auto-correlation is left in the residuals, an error model can be used to improve the forecasts by applying an auto-regressive model on the residuals. This is somewhat equivalent to include an MA part in the original model (which is not implemented as it is far from trivial).

As summarizing measures for validation of how well dynamics are modelled:

- Plot the auto-correlation function (ACF) of the one-step residuals.
- Plot cross-correlation functions from one-step residuals to other variables, see (Bacher *et al.* 2013).

Systematic patterns found in these functions lead to direct knowledge on how to improve the model, see for example the table on Page 155 in Madsen (2007) and the examples in Section 5.4 in the present paper.

5. Example with R code

A short introduction to the basic functionalities and steps in setting up a model is given in the following – for more details and functionalities see the vignettes listed in Section 1.1 and the website onlineforecasting.org.

5.1. Setup of data

As input to the model, we will use weather forecasts, which are arranged in forecast matrices, e.g. the ambient temperature

$$\boldsymbol{T}_{\mathrm{a},n} = \begin{array}{c} \begin{array}{ccccc} \textbf{k0} & \textbf{k1} & \textbf{k2} & \dots & \textbf{kxx} \end{array} \\ \begin{pmatrix} T_{\mathrm{a},1|1} & T_{\mathrm{a},2|1} & T_{\mathrm{a},3|1} & \dots & T_{\mathrm{a},1+n_k|1} \\ T_{\mathrm{a},2|2} & T_{\mathrm{a},3|2} & T_{\mathrm{a},4|2} & \dots & T_{\mathrm{a},2+n_k|2} \\ \vdots & \vdots & \vdots & & \vdots \\ T_{\mathrm{a},n-1|n-1} & T_{\mathrm{a},n|n-1} & T_{\mathrm{a},n+1|n-1} & \dots & T_{\mathrm{a},n-1+n_k|n-1} \\ T_{\mathrm{a},n|n} & T_{\mathrm{a},n+1|n} & T_{\mathrm{a},n+2|n} & \dots & T_{\mathrm{a},n+n_k|n} \end{pmatrix} \end{array} \begin{array}{c} \rightarrow \textbf{horizon/time} \downarrow \\ 1 \\ 2 \\ \vdots \\ n-1 \\ n \end{array} \tag{16}$$

Building heat load and weather forecast data is available in `Dbuilding` after loading the package:

```
D <- Dbuilding
str(D, 1)

## List of 7
##  $ t            : POSIXct[1:1824], format: "2010-12-15 01:00:00" ...
##  $ heatload     : num [1:1824] 5.92 5.85 5.85 5.88 5.85 ...
##  $ heatloadtotal: num [1:1824] 4 4.07 4.08 4.01 4.46 ...
##  $ Taobs        : num [1:1824] -6.19 -5.7 -5.05 -5.1 -5.62 ...
##  $ Iobs         : num [1:1824] 0 0 5.41 12.99 13.01 ...
##  $ Ta           :'data.frame': 1824 obs. of  36 variables:
##  $ I            :'data.frame': 1824 obs. of  36 variables:
##  - attr(*, "class")= chr "data.list"

class(D)

## [1] "data.list"
```

The `data.list` class comes with the package, it's actually a `list` with some predefined structure, see `?data.list`.

All inputs to be used in a model must be given as forecast matrices. The ambient temperature forecast is set up in a forecast matrix, which class is just a `data.frame`:

```
class(D$Ta)
```

```
## [1] "data.frame"
```

```
head(D$Ta[ ,1:8], 4)
```

```
##         k1       k2      k3      k4       k5       k6       k7       k8
## 1 -2.82340 -3.20275 -3.1185 -3.0896 -3.13200 -3.16130 -3.16645 -3.08885
## 2 -2.90405 -3.11850 -3.0896 -3.1320 -3.16130 -3.16645 -3.08885 -2.77165
## 3 -2.93590 -3.08960 -3.1320 -3.1613 -3.16645 -3.08885 -2.77165 -2.32185
## 4 -2.89315 -3.11285 -3.0484 -3.1090 -3.11600 -2.80990 -2.36895 -2.00945
```

The time is kept in the vector t where the time stamps are set in the end of the hour:

```
D$t[1:4]
```

```
## [1] "2010-12-15 01:00:00 GMT" "2010-12-15 02:00:00 GMT"
## [3] "2010-12-15 03:00:00 GMT" "2010-12-15 04:00:00 GMT"
```

Observations are simply in a vector:

```
D$heatload[1:4]
```

```
## [1] 5.916667 5.850000 5.850000 5.883333
```

For more details on the `data.list` class, see the setup-data vignette with `vignette("setup-data")` – which demonstrates useful functions for manipulating and exploring forecast data.

5.2. Defining a model

Models are set up using the R6 class `forecastmodel`. An object of the class is instantiated by:

```
model <- forecastmodel$new()
```

It holds variables and functions for representing and manipulating a model.

We want to forecast the `heatload` variable in the data list, so we set that as the model output by:

```
model$output <- "heatload"
```

The model inputs and transformations must then be defined. We can add an input as a linear function by:

```
model$add_inputs(Ta = "Ta")
```

hence the name of the ambient temperature forecast matrix. Then the k horizon column becomes $\beta_k T_{a,t+k|t}$ in the regression for the k horizon.

Adding an intercept to a model can be done by:

```
model$add_inputs(mu = "one()")
```

where the function `one()` simply returns a vector of 1's, which will be inserted in the design matrix, see in Appendix C. The function is run during the transformation, which is carried out with the function `model$transform_data()`. It's actually easy to debug a function used during transformation by:

```
debug(one)
model$transform_data(D)
undebug(one)
```

Most of the time, `model$transform_data()` is called inside a "fitting function" when a model is fitted.

Input transformation

Transformations (or mappings) of inputs are simply R code which returns a forecast matrix or a list of forecast matrices.

Dynamics can be modelled using filters. For example, low-pass filtering of a variable with:

```
model$add_inputs(Ta = "lp(Ta, a1=0.9)")
```

will, when the transformation is run, apply a low-pass filter along each column of the forecast matrix `Ta`. The filter coefficient is set to $a = 0.9$. To illustrate the effect of this, see Appendix B and the vignette setup-and-use-model

Non-linear effects can be modelled using basis functions. For mapping an input to basis splines the function `bspline()` is provided. It's a wrapper of the `bs()` function from the splines package and has the same arguments. To e.g. include a non-linear function of the ambient temperature:

```
model$add_inputs(Ta = "bspline(Ta, df=5)")
```

where `df` is the degrees of freedom of the spline function.

Functions can be nested, e.g. first a low-pass filter before mapping to basis splines:

```
model$add_inputs(Ta = "bspline(lp(Ta, a1=0.9), df=5)")
```

Fourier series can also be used as basis functions. They can be generated with the `fs()` function, e.g. if `tday` is a forecast matrix with the time of day in hours (range from 0 to 24) then:

```
model$add_inputs(mutday = "fs(tday/24, nharmonics=10)")
```

generates the Fourier series for fitting a diurnal curve with 10 harmonics (since there is both a cosine and a sine for each harmonic it results in 20 inputs in the regression model).

Varying-coefficient models can be realized with multiplication of inputs (Hastie and Tibshirani 1993). The multiplication operator `%**%` must be used, see how with `?%**%`. For example, models similar to those proposed by Rasmussen, Frölke, Bacher, Madsen, and Rode (2020), where the coefficient for global radiation change as a function of the time of day `tday`, can be realised by:

```
model$add_inputs(I = "bspline(tday, df=5) %**% I")
```

For more details, see the solar-power-forecasting example on the website.

Finally, it's often useful to add an auto-regressive input. This can be done by:

```
model$add_inputs(AR = "AR(c(0,1))")
```

which adds the output observations at time t and time $t-1$ as regression inputs.

5.3. Model fitting and parameter optimization

After setting up a model it can be fitted to data. This implies carrying out the transformation and regression stages, which is done by passing the model to a fitting function. Different functions implement different regression schemes.

Both the two currently available fitting functions `lm_fit()` and `rls_fit()` takes a the offline parameters as a vector, fits a model and returns the RMSE (summed for all horizons).

To demonstrate this we replace the inputs on the previous defined model with two inputs:

```
model$inputs <- NULL
model$add_inputs(mu = "one()",
                 Ta = "lp(Ta, a1=0.9)")
```

We also have to set the "score period", which is simply a logical vector that specifies the time points to be included in the score calculation. For the linear regression we can include all points by:

```
D$scoreperiod <- rep(TRUE, length(D$t))
```

We can now fit the model and obtain the summed RMSE for the horizons 1 to 6 steps ahead by:

```
model$kseq <- 1:6
lm_fit(c(Ta__a1=0.8), model, D, scorefun=rmse, returnanalysis=FALSE)


## [1] 5.039611
```

The function can be passed to an optimizer, which can then find the parameter value(s) which minimize the score.

To facilitate the optimization wrappers for `optim()` are included. We define parameter(s) to optimize:

```
model$add_prmbounds(Ta__a1 = c(min=0.8, init=0.9, max=0.9999))
```

Note, the double underscore syntax. The double underscore separates the input name and the name of the parameter. So in the case above the value of `a1` in the R code for input `Ta` will be optimized, starting at an initial value of 0.9 and staying within the specified limits.

We can then run the optimization calculating scores (to save computational time run on only a horizons 3 and 18 steps ahead):

```
lm_optim(model, D, kseq=c(3,18))
```

The `lm_optim()` function is a wrapper for the R optimizer function `optim()`. It returns the result from `optim()` and sets optimized parameters in:

```
model$prm


##    Ta__a1
## 0.8926342
```

If we want to carry out the regression using the RLS scheme, we must have a burn-in period, i.e. exclude a period in the beginning of the data from the calculation of the score, e.g. 5 days:

```
D$scoreperiod <- in_range(min(D$t)+5*24*3600, D$t)
```

and we have to set the forgetting factor as a regression parameter:

```
model$add_regprm("rls_prm(lambda=0.9)")
```

We can now optimize with RLS:

```
rls_optim(model, D, kseq=c(3,18))
```

and if we want to also optimize the forgetting factor:

```
model$add_prmbounds(lambda = c(min=0.7, init=0.99, max=0.9999))
rls_optim(model, D, kseq=c(3,18))
```

Calculating forecasts

While developing models it's most convenient to use the fit functions for calculating predictions, e.g.:

```
model$kseq <- 1:24
fit <- rls_fit(model$prm, model, D)
```

will return a list holding the forecasts (in the forecast matrix `fit$Yhat`) and other useful information.

Forecasts can also be calculated directly with a predict function:

```
rls_predict(model, model$transform_data(D))
```

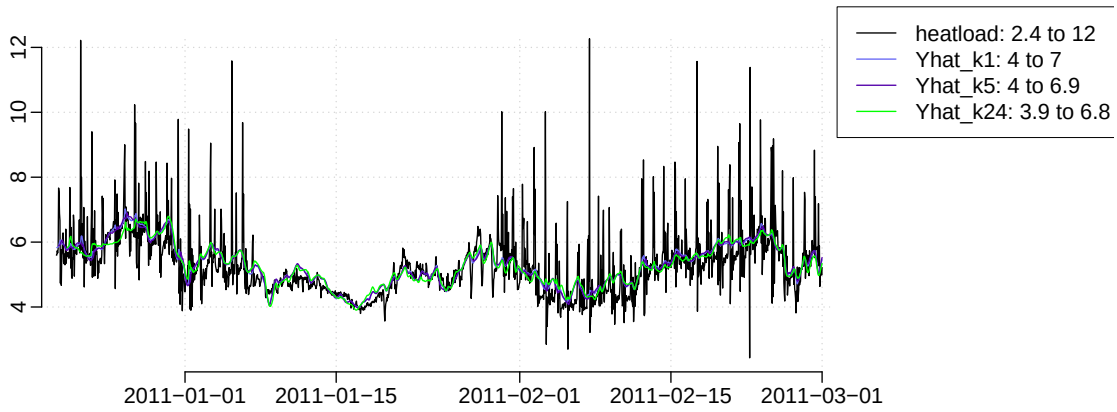
will return a forecast matrix using the input data in `D`.

5.4. Evaluation

Finally, it's time to evaluate the forecasts and potentially get inspired to improve the model. For a comprehensive introduction, see the forecast-evaluation vignette.

First, a plot of the forecasts is always a good idea to learn how the model work:

```
D$Yhat <- fit$Yhat
plot_ts(subset(D,D$scoreperiod), "heatload$|Yhat", kseq=c(1,5,24))
```



We can extract the residuals from the fit:

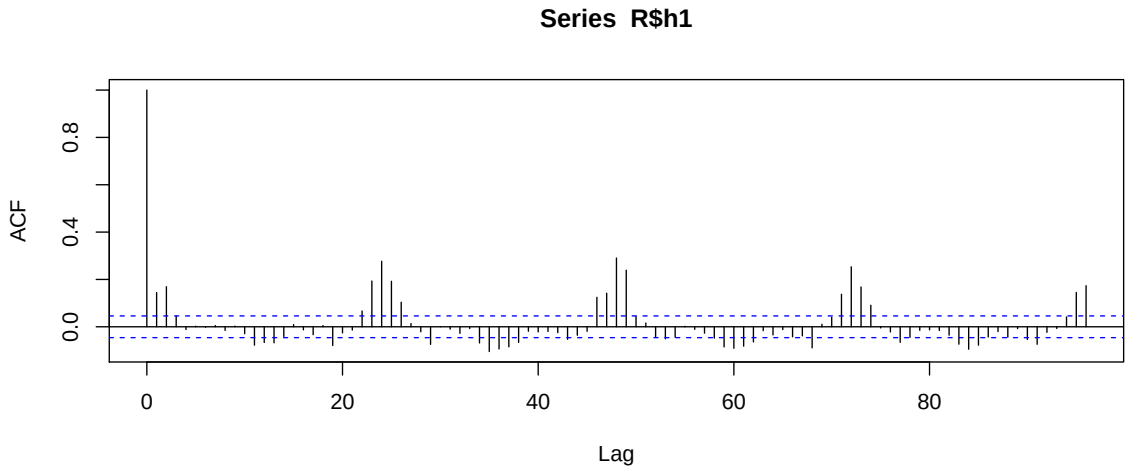
```
R <- residuals(fit)
R[101:105, 1:5]
```

```
##              h1           h2           h3           h4           h5
## 101 -0.30889122 -0.311105984 -0.311287844 -0.317092662 -0.309276247
## 102 -0.15452158 -0.168868282 -0.168640790 -0.171378395 -0.176755885
## 103  0.02755442  0.008792884  0.007182829  0.004900251  0.001789956
## 104  1.78881155  1.768225483  1.768757817  1.763675174  1.762632322
## 105  2.38451273  2.394916757  2.398424406  2.394625320  2.389784248
```

Note, that the column names are with `h` prefix. Hence, it's not a forecast matrix, since each column is a time series aligned with time t. They are observations of the error.

Plot the ACF of the one-step residuals:

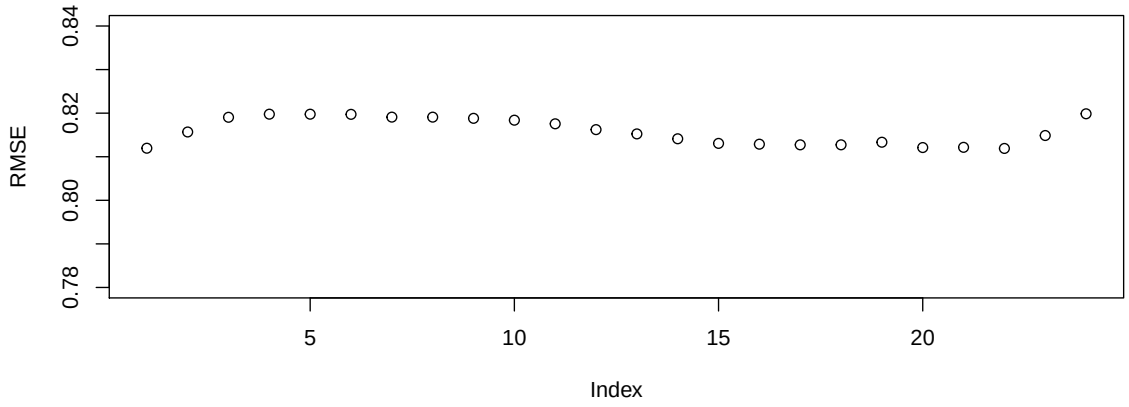
```
acf(R$h1, na.action=na.pass, lag.max=96)
```



The ACF plot suggest that there remains a diurnal pattern to be modelled. It can be achieve by adding a diurnal curve to the model, e.g. with Fourier series basis functions. This is demonstrated in the vignette setup-and-use-model.

We also want to calculate the score as a function of the horizon:

```
inscore <- D$scoreperiod & complete_cases(fit$Yhat)
RMSE <- score(residuals(fit), scoreperiod = inscore)
plot(RMSE, ylim=c(0.78,0.84))
```



Which is relatively constant. The offline parameters were optimized for $k = 3$ and $k = 18$, which can explain why it's not monotonic increasing with the horizon.

6. Discussion and conclusion

A short discussion on extending functionalities and a conclusion is given in this section.

6.1. Extending functionality

The current package is designed to make it easy to implement new transformation functions and regression schemes, as well as using other optimizers for tuning parameters.

Implementing a new transformation function is straight forward. It must receive either a forecast matrix or a list of forecast matrices and return either after processing. Furthermore, when used in an operational online setup, where the transformation is executed when new data arrives, it's possible to save state information in a transformation function, such that next time the function is called, the state can be read and used. See the `lp()` function for inspiration when writing a new transformation function.

A new regression scheme, e.g. a kernel or quantile regression, can be implemented. A fitting function should be implemented in similar way as `lm_fit()` and `rls_fit()`, such that the first argument is the parameter vector and it returns a score value, which can be passed to an optimizer.

It's very easy to use other optimizers. The current fitting functions can simply be passed to any optimizer in R, which follows the `optim()` way of receiving a function for optimization, see the code in `lm_optim()`.

In future versions new regression techniques, e.g. kernel regression (local fitting) and quantile regression, might be added. The latter opens up the possibilities to calculate probabilistic forecasts, see (Nielsen, Madsen, and Nielsen 2006) and (Bjerregøard, Møller, and Madsen 2021), as well as carry out normalization and Copula transformations, which can be very useful for spatio-temporal forecast models, see (Tastu, Pinson, Kotwa, Madsen, and Nielsen 2011) or (Lemos-Vinasco, Bacher, and Møller 2021).

6.2. Summary and conclusion

This paper provides an entry point and reference for working with the onlineforecast package. The paper covers version 1.0 of the package, which has been available on CRAN in almost one year in a <1.0 version. We have added functionalities in version 1.0, but it has remained fully backwards compatible.

The main contribution of the package is to make it easy to generate online multi-step forecasts in a flexible way. The package contains functionalities not directly available elsewhere, such as:

- Enabling the use of input variables given as forecasts, e.g. NWPs, in an easy and flexible way.
- Modelling of dynamics and non-linearities using transformations including tuning the parameters of these transformations.
- Recursive estimation for tracking time-varying systems computationally efficient for multiple horizons.

Furthermore, online operation with computationally effective updating is uncomplicated – so the package is well suited for real-time operational applications.

The onlineforecast package has a significant value for anyone who needs to model operational online forecasting. For example, in energy scheduling, where recursive updated forecasts are needed as input to optimal decision making and real-time control of systems. It can also be very useful for companies that need online forecasts for other monitoring and real-time applications – especially the functionality for model updating with very little computational costs when new data becomes available, is a unique feature of the package.

Computational details

We have tried to make the onlineforecast package depend on as few other packages as possible. Only a few additional packages are used in the core functionalities: **R6** for the "usual" OOP functionalities and **Rcpp** with **RcppArmadillo** for easy integration of fast compiled code. For extending the modelling possibilities the **splines** and **pbs** packages are essential, and for nice caching the **digest** package. We acknowledge the **devtools** and **knitr**, **rmarkdown**, **R.rsp**, **testthat** packages, which are indispensable for developing a package. We acknowledge the R community and the amazing work behind R done by many people over the years!

The results in this paper were obtained using R 4.0.5. R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/.

Acknowledgments

The software has been developed with funding from multiple projects: FLEXCoop (European Union's Horizon 2020, grant agreement No 773909), Flexible Energy Denmark, Heat 4.0 and Decision support tools for smart home energy management systems (Innovation Fund Denmark, No. 9045-00017B, 8090-00046B and 8053-00156B), TOP-UP (Innovation Fund Denmark and ERA-NET, No. 9045-00017B), Digital-twin, IEA Annex 71 and 83 Danish participation (EUDP, No. 64019-0570, 64017-05139 and 64020-1007), and finally SCA+ (EU Interreg, No. 20293290).

References

Alfares HK, Nazeeruddin M (2002). "Electric load forecasting: Literature survey and classification of methods." *International Journal of Systems Science*, **33**(1), 23–34. doi:10.1080/00207720110067421. https://doi.org/10.1080/00207720110067421, URL https://doi.org/10.1080/00207720110067421.

Bacher P, Madsen H, Nielsen HA (2009). "Online short-term solar power forecasting." *Solar Energy*, **83**(10), 1772 – 1783. ISSN 0038-092X. doi:https://doi.org/10.1016/j.solener.2009.05.016.

Bacher P, Madsen H, Nielsen HA, Perers B (2013). "Short-term heat load forecasting for single family houses." *Energy and buildings*, **65**, 101–112.

Bjerregøard MB, Møller JK, Madsen H (2021). "An introduction to multivariate probabilistic forecast evaluation." *Energy and AI*, p. 100058.

Bokde ND, Yaseen ZM, Andersen GB (2020). "Forecasttb—An R package as a test-bench for time series forecasting—Application of wind speed and solar radiation modeling." *Energies*, **13**(10), 2578.

Breiman L, Friedman JH (1985). "Estimating optimal transformations for multiple regression and correlation." *Journal of the American statistical Association*, **80**(391), 580–598.

Chatfield C, Xing H (2019). *The analysis of time series: an introduction with R*. Chapman and hall/CRC.

Cryer JD, Chan KS (2008). *Time series analysis: with applications in R*, volume 2. Springer.

De Gooijer JG, Hyndman RJ (2006). "25 years of time series forecasting." *International journal of forecasting*, **22**(3), 443–473.

Dotzauer E (2002). "Simple model for prediction of loads in district - heating systems." *Applied Energy*, **73**(3-4), 277–284. doi:10.1016/S0306-2619(02)00078-8.

Engel Y, Mannor S, Meir R (2004). "The kernel recursive least-squares algorithm." *IEEE Transactions on signal processing*, **52**(8), 2275–2285.

Hastie T, Tibshirani R (1993). "Varying-coefficient models." *Journal of the Royal Statistical Society: Series B (Methodological)*, **55**(4), 757–779.

Hastie T, Tibshirani R, Friedman J (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

Helske J (2017). "KFAS: Exponential Family State Space Models in R." *Journal of Statistical Software*, **78**(10), 1–39. doi:10.18637/jss.v078.i10.

Hong T, Fan S (2016). "Probabilistic electric load forecasting: A tutorial review." *International Journal of Forecasting*, **32**(3), 914–938.

Hyndman R, Koehler AB, Ord JK, Snyder RD (2008). *Forecasting with exponential smoothing: the state space approach*. Springer Science & Business Media.

Hyndman RJ, Khandakar Y (2008). "Automatic time series forecasting: the forecast package for R." *Journal of Statistical Software*, **26**(3), 1–22. URL https://www.jstatsoft.org/article/view/v027i03.

Kleissl J (2013). *Solar energy forecasting and resource assessment*. Academic Press.

Lemos-Vinasco J, Bacher P, Møller JK (2021). "Probabilistic load forecasting considering temporal correlation: Online models for the prediction of households' electrical load." *Applied Energy*, **303**, 117594.

Ljung L, Söderström T (1983). *Theory and practice of recursive identification*. MIT press.

Madsen H (2007). *Time series analysis*. CRC Press.

Madsen H, Thyregod P (2010). *Introduction to general and generalized linear models*. CRC Press.

Nielsen HA, Madsen H (2006). "Modelling the heat consumption in district heating systems using a grey-box approach." *Energy and buildings*, **38**(1), 63–71.

Nielsen HA, Madsen H, Nielsen TS (2006). "Using quantile regression to extend an existing wind power forecasting system with probabilistic forecasts." *Wind Energy: An International Journal for Progress and Applications in Wind Power Conversion Technology*, **9**(1-2), 95–108.

Nielsen TS, Nielsen HA, Madsen H (2002). "Prediction of wind power using time-varying coefficient functions." In *Proceedings of the XV IFAC World Congress*.

O'Hara-Wild M, Hyndman R, Wang E (2020). *fable: Forecasting Models for Tidy Time Series*. R package version 0.2.1, URL https://CRAN.R-project.org/package=fable.

Petris G (2010). "An R package for dynamic linear models." *Journal of Statistical Software*, **36**(1), 1–16.

Rasmussen C, Frölke L, Bacher P, Madsen H, Rode C (2020). "Semi-parametric modelling of sun position dependent solar gain using B-splines in grey-box models." *Solar Energy*, **195**, 249–258.

Sayed AH, Kailath T (1994). "A state-space approach to adaptive RLS filtering." *IEEE signal processing magazine*, **11**(3), 18–60.

Shah SL, Cluett WR (1991). "Recursive least squares based estimation schemes for self-tuning control." *The Canadian Journal of Chemical Engineering*, **69**(1), 89–96.

Spliid H (1983). "A fast estimation method for the vector autoregressive moving average model with exogenous variables." *Journal of the American Statistical Association*, **78**(384), 843–849.

Tashman LJ (2000). "Out-of-sample tests of forecasting accuracy: an analysis and review." *International journal of forecasting*, **16**(4), 437–450.

Tastu J, Pinson P, Kotwa E, Madsen H, Nielsen HA (2011). "Spatio-temporal analysis and modelling of short-term wind power forecast errors." *Wind Energy*, **14**(1), 43–60.

Weisberg S (2005). *Applied linear regression*, volume 528. John Wiley & Sons.

Winters PR (1960). "Forecasting Sales by Exponentially Weighted Moving Averages." *Management Science*, **6**(3), 324–342. doi:10.1287/mnsc.6.3.324.

A. Forecast model notation

In this section it is shown how to write onlineforecast models in mathematical notation. Both in a full description and how to write a shorter summarized description. Note, that when variables are noted in bold font it indicates that they are multi-variate.

A model can be described in full detail as presented in the following.

The transformation stage

$$\begin{aligned}
\text{Intercept:} \quad & \mu_{t+k|t} && = 1 && (17)\\
\text{Periodic:} \quad & \boldsymbol{x}_{\text{per},t+k|t} && = f_{\text{fs}}(t; n_{\text{har}}) && (18)\\
\text{Part 1:} \quad & x_{1,t+k|t} && = H(B;a)u_{1,t+k|t} && (19)\\
\text{Part 2:} \quad & \boldsymbol{x}_{23,t+k|t} && = f_{\text{bs}}(u_{2,t+k|t}; n_{\text{deg}})u_{3,t+k|t} && (20)\\
\text{Part 3:} \quad & x_{4,t+k|t} && = u_{4,t} && (21)
\end{aligned}$$

and the regression stage

$$Y_{t+k|t} = \beta_{0,k}\mu_{t+k|t} + \boldsymbol{\beta}_{1,k}^T\boldsymbol{x}_{\text{per},t+k|t} + \beta_{2,k}x_{1,t+k|t} + \boldsymbol{\beta}_{3,k}^T\boldsymbol{x}_{23,t+k|t} + \beta_{4,k}x_{4,t+k|t} + \varepsilon_{t+k|t} \quad (22)$$

Thus the model inputs are:

- t is simply the time value.
- $u_{1,t+k|t}$ some forecast input (e.g. NWP variable).
- $u_{2,t+k|t}$ some forecast input (e.g. could be a deterministic value, e.g. time of day which is always know (the $|t$ could be omitted)).
- $u_{3,t+k|t}$ some forecast input (e.g. NWP variable).
- $u_{4,t}$ some value at time t (e.g. an observation variable).

The functions which maps the inputs (u's) to the regression inputs (x's) are:

- $f_{\text{fs}}(t; n_{\text{har}})$ is a function generating Fourier series of some implicit period length.
- $H(B;a)$ is a low-pass filter.
- $f_{\text{bs}}(u_{2,t+k|t}; n_{\text{deg}})$ is a function generating basis splines.

Their parameters are the transformation parameters:

- n_{har} is the number of harmonics.
- a is the low-pass filter coefficient.

- n_{deg} is the degrees of freedom of the spline function.

which must be set or optimized.

The regression coefficients are

$$\boldsymbol{\beta}_k = \left[\beta_{0,k} \;\; \boldsymbol{\beta}_{1,k}^T \;\; \beta_{2,k} \;\; \boldsymbol{\beta}_{3,k}^T \;\; \beta_{4,k}\right]^T \tag{23}$$

$$= \left[\beta_{0,k} \;\; \beta_{1,1,k} \;\; \beta_{1,2,k} \;\; \dots \;\; \beta_{1,2n_{\text{har}},k} \;\; \beta_{2,k} \;\; \beta_{3,1,k} \;\; \beta_{3,2,k} \;\; \dots \;\; \beta_{3,n_{\text{deg}},k} \;\; \beta_{4,k}\right]^T \tag{24}$$

If the model is fitted with a recursive scheme, thus the coefficients change over time, it should be indicated by adding a t to the subscript, e.g. $\beta_{0,k,t}$. Furthermore, other parameters can exist, which can enter an optimization at the transformation stage, e.g. the RLS forgetting factor λ. The parameters which are optimized in the transformation stage should be presented together.

Specifying the model in all details can be cumbersome to include in some texts, so it makes sense to simplify the notation. When using a simpler notation, as suggested below, it should be stated, what is implicit (e.g. the regression stage). Referencing the present text should be sufficient when using a simpler notation. Naturally, all inputs, functions, etc., should be described in some way.

A model can be specified in a simpler way, e.g. the model above in one equation

$$\begin{aligned} Y_{t+k|t} = \beta_{0,k} + \boldsymbol{\beta}_{1,k}^T f_{\text{fs},k}(t; n_{\text{har}}) + \beta_{2,k} H_k(B; a) u_{1,t+k|t} + \boldsymbol{\beta}_{3,k}^T f_{\text{bs},k}(u_{2,t+k}; n_{\text{deg}}) u_{3,t+k|t} \\ + \beta_{4,k} u_{4,t} + \varepsilon_{t+k|t} \end{aligned} \tag{25}$$

or writing the regression stage implicitly by removing the regression coefficients where it's meaningful

$$Y_{t+k|t} = \mu_k + f_{\text{fs},k}(t; n_{\text{har}}) + H_k(B; a) u_{1,t+k|t} + f_{\text{bs},k}(u_{2,t+k|t}; n_{\text{deg}}) u_{3,t+k|t} + \beta_k u_{4,t} + \varepsilon_{t+k|t} \tag{26}$$

It's then implicit that the functions are different from the previous stated functions, since they include the regression coefficients. Again, if fitted with a recursive scheme, then it can be indicated by adding a t subscript, e.g. $f_{\text{fs},k,t}(t; n_{\text{har}})$.

To simplify further the k on the functions can be implicit

$$Y_{t+k|t} = \mu + f_{\text{fs}}(t; n_{\text{har}}) + H(B; a) u_{1,t+k|t} + f_{\text{bs}}(u_{2,t+k|t}; n_{\text{deg}}) u_{3,t+k|t} + \beta u_{4,t} + \varepsilon_{t+k|t} \tag{27}$$

and similarly the transformation parameters can be implicit

$$Y_{t+k|t} = \mu + f_{\text{fs}}(t) + H(B) u_{1,t+k|t} + f_{\text{bs}}(u_{2,t+k|t}) u_{3,t+k|t} + \beta u_{4,t} + \varepsilon_{t+k|t} \tag{28}$$

Then the functions and their parameters, and the fitting scheme (i.e. with either LS or RLS for each horizon) should be described in some other way.

Finally, the most simplified notation would be to even remove the time indexing

$$Y = \mu + f_{\text{fs}}(t) + H(B) u_1 + f_{\text{bs}}(u_2) u_3 + \beta u_4 + \varepsilon \tag{29}$$

after making clear how all the variables are defined.

B. Transformation details

In this section a short introduction is given to the package functions for mapping inputs in the transformation stage.

B.1. Low-pass filtering

When modelling passive dynamical systems low-pass filtering is needed to describe the input-output relation. A simple first-order low-pass filter is equivalent to the transfer function of a single resistor single capacitor system – equivalent to a first order ARX model.

The input time series is filtered with a transfer function

$$x_{t+k|t} = H(B;a)u_{t+k|t} \tag{30}$$

where the simplest first-order low-pass with stationary gain of one (unity DC gain) is

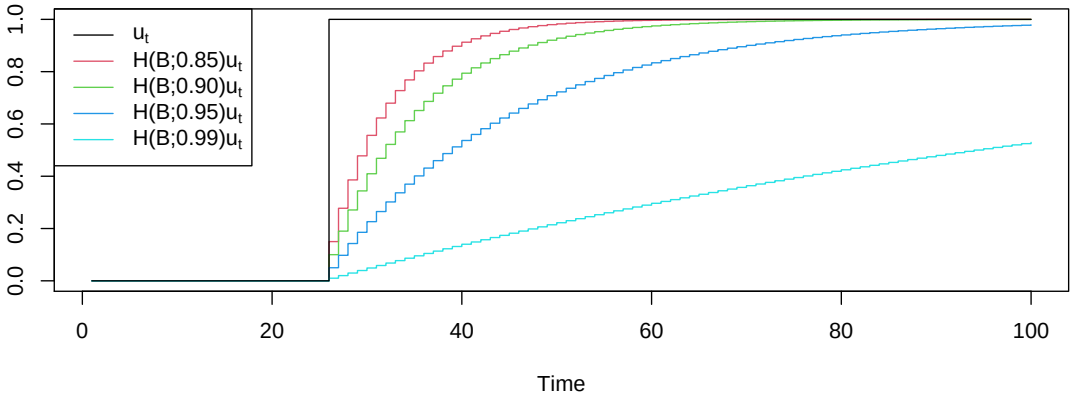
$$H(B;a) = \frac{1-a}{1-aB} \tag{31}$$

thus the low-pass filtered series becomes

$$x_{t+k|t} = \frac{(1-a)u_{t+k|t}}{1-ax_{t-1+k|t-1}} \tag{32}$$

thus we have a filter coefficient a between 0 and 1, which must be tuned to match the particular time constant of the linear system. This filter is implemented in the function `lp()`.

The following plot shows the response to a step function for different filter coefficients a:



It is clearly seen that the responses are exponential functions with different time constants depending on the value of a, such that a higher value results in a slower response.

B.2. Base-splines

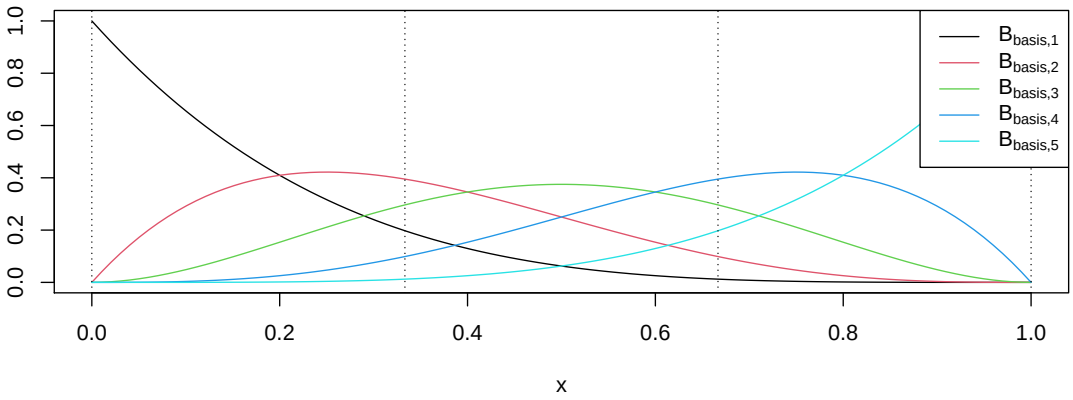
A wide spread approach for modelling non-linear functional relations is to apply spline basis functions (Hastie *et al.* 2009). The basic idea is to "resolve" a single input into multiple inputs – and use the them instead of the single input in the regression. In this way a non-linear function can be fitted between the single input and the model output.

The spline basis functions are calculated separately for each horizon

$$\boldsymbol{x}_{t+k|t} = f_{\text{bs}}(u_{t+k|t}; n_{\text{deg}}) \tag{33}$$

where n_{deg} is the degree of the piecewise polynomial function, hence a higher n_{deg} results in a more "flexible" function. Note, that $u_{t+k|t}$ is a single time series, but the $\boldsymbol{x}_{t+k|t}$ is a matrix of n_{deg} columns, i.e. as many times series all which will be used in the regression for horizon k. The values where the piecewise polynomials meet, are known as knots. The function to be used for transformations is `bspline()`, which builds directly on the `bs()` function in R.

The input is resolved with spline basis functions, e.g. for an input in the interval $[0, 1]$ the basis splines for $n_{\text{deg}} = 4$ are:

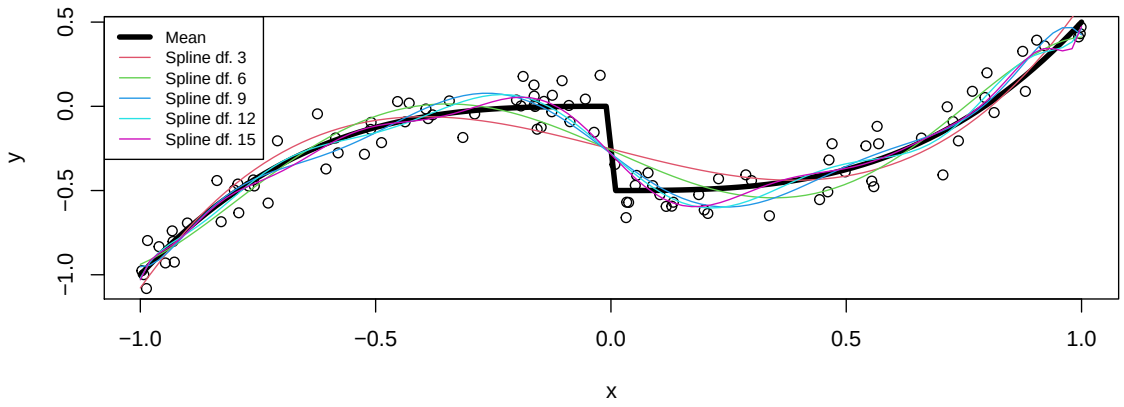


where the vertical dashed horizontal lines marks the knot points (must be set in some way, usually set as equidistant quantiles of values of u_t).

An example of the resulting a spline functions, which can be fitted using spline basis functions presented above is given. First, a non-linear function with some added noise is simulated

$$Y_i = \begin{cases} u_i^3 + \varepsilon_i \text{ for } u_i \leq 0 \\ u_i^3 - 0.5 + \varepsilon_i \text{ for } u_i > 0 \end{cases} \tag{34}$$

where $\varepsilon \sim N(0, 0.1^2)$ and i.i.d.. A sample of 100 observations is simulated and spline basis functions of increasing degree is generated and used as input to a linear regression model. The resulting spline functions modelled is seen in the plot:



It is clear that there is a balance between bias and variance: A too low degree results in an under-fitted model (not able to "bend" enough), while a too high degree results in an over-fitted model (bends to much). The degrees of freedom can be optimized using a cross-validation approach.

B.3. Fourier series

In order to model periodic phenomena a linear combination of Fourier series is a very effective approach.

We use the notation

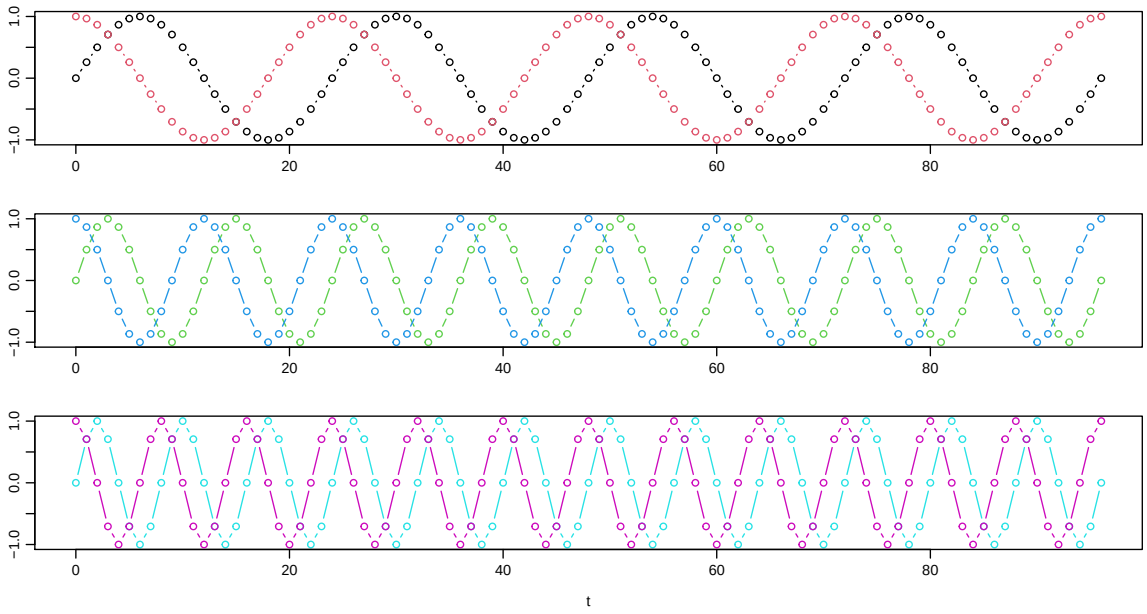
$$\boldsymbol{x}_{t+k|t} = f_{\text{fs}}(u_{t+k|t}; n_{\text{har}}) \tag{35}$$

where n_{har} is the number of harmonic pairs included, hence $\boldsymbol{x}_{t+k|t}$ is vector of length $2n_{\text{har}}$, which is then linearly combined in the regression stage. The package function for calculating Fourier series is `fs()`.

Exemplified with time as input $u_{t+k|t} = t + k$ (which is often done when modelling a periodic phenomena, e.g. diurnal or yearly) the calculated series is

$$\left[\sin\left(\frac{2\pi}{t_{\text{per}}}t\right) \;\; \cos\left(\frac{2\pi}{t_{\text{per}}}t\right) \;\; \sin\left(\frac{2\cdot 2\pi}{t_{\text{per}}}t\right) \;\; \cos\left(\frac{2\cdot 2\pi}{t_{\text{per}}}t\right) \;\; \ldots \;\; \sin\left(\frac{n_{\text{har}}2\pi}{t_{\text{per}}}t\right) \;\; \cos\left(\frac{n_{\text{har}}2\pi}{t_{\text{per}}}t\right)\right] \tag{36}$$

An example of Fourier basis functions is plotted below for $n_{\text{har}} = 3$, hence 3 pairs of harmonics and a period $t_{\text{per}} = 24$.



B.4. Auto-regressive

In classical time series analysis auto-regressive terms are included in models to describe dynamical. This is equivalent to using low-pass filters as described previously, however, it can often be useful to include an auto-regressive term in a model, especially for forecasts a few steps ahead. In the package the function `AR()` can be used to include the latest or lags of the current observed model output. This can also be used for making an error model on residuals.

B.5. Nested transformations

It is perfectly possible to combine transformations to create models involving complicated functions. E.g. first resolve in basis splines and then low-pass each of them

$$\boldsymbol{x}_{t+k|t} = H(B;a) f_{\text{bs}}(u_{t+k|t}; n_{\text{deg}}) \tag{37}$$

This can be implemented in R by nesting the functions, e.g. `lp(bspline())` (of course also providing the function arguments).

Multiplication for coefficient-varying models

In order to build coefficient-varying models, also called interaction effects (Hastie and Tibshirani 1993), it must be possible to multiply inputs. For the forecast models it's carried out for each horizon separately

$$\boldsymbol{x}_{t+k|t} = u_{1,t+k|t} \cdot u_{2,t+k|t} \tag{38}$$

which usually in R can be carried out with the regular multiplication operator '`*`'. However, an operator is needed for forecast matrices, which makes sure that the correct horizons are

multiplied and a bit more. The operator '%**%' handles this and should be used multiplication in the transformations.

C. Regression

In this section the two regression schemes implemented in onlineforecast are described. When fitting a model, thus estimating the regression coefficients, data from a period $t \in (1, 2, \ldots, n)$ is used and passed on to either: the `lm_fit()` function which implements the Least Squares (LS) scheme, or the `rls_fit()` function, which implements the Recursive Least Squares (RLS) scheme.

One important difference between the two implementations is that in the LS the coefficients are estimated using data from the entire period, thus they are constant during the period and the calculated predictions are "in-sample". This is opposed to the RLS, where the coefficients are updated through the period using only past data at each time t. In that case the coefficients vary over time and the calculated predictions are "out-of-sample".

This difference is explained in the following and indicated by subscripting the coefficient vector with t only for the RLS.

C.1. Least squares

The regression coefficients for the k'th horizon is set in the vector

$$\boldsymbol{\beta}_k = \begin{bmatrix} \beta_{0,k} & \beta_{1,k} & \ldots & \beta_{p,k} \end{bmatrix}^T \tag{39}$$

Note, that t is not included in the subscript.

The input data for the k horizon is the design matrix

$$\boldsymbol{X}_{k,t} = \begin{bmatrix} x_{0,1+k|1} & x_{1,1+k|1} & \cdots & x_{p,1+k|1} \\ x_{0,2+k|2} & x_{1,2+k|2} & \cdots & x_{p,2+k|2} \\ \vdots & \vdots & & \vdots \\ x_{0,n-1|n-1-k} & x_{1,n-1|n-1-k} & \cdots & x_{p,n-1|n-1-k} \\ x_{0,n|n-k} & x_{1,n|n-k} & \cdots & x_{p,n|n-k} \end{bmatrix} \tag{40}$$

The output observations are in the vector

$$\boldsymbol{y}_{k,n} = \begin{bmatrix} y_{1+k} & y_{2+k} & \ldots & y_{n-1} & y_n \end{bmatrix}^T \tag{41}$$

The LS estimates of the coefficients are

$$\hat{\boldsymbol{\beta}}_k = (\boldsymbol{X}_{k,n}\boldsymbol{X}_{k,n})^{-1}\boldsymbol{X}_{k,n}\boldsymbol{y}_{k,n} \tag{42}$$

The predictions are "in-sample" and calculated by

$$\hat{\boldsymbol{y}}_{k,n} = \boldsymbol{X}_{k,n}\hat{\boldsymbol{\beta}}_k \tag{43}$$

and returned when fitting a model with `lm_fit()`.

The estimated coefficients may now be used for "out-of-sample" prediction (for $t_{\text{new}} \geq n$), with the input

$$\boldsymbol{x}_{t_{\text{new}}+k|t_{\text{new}}} = \begin{bmatrix} x_{0,t_{\text{new}}+k|t_{\text{new}}} & x_{1,t_{\text{new}}+k|t_{\text{new}}} & \dots & x_{p,t_{\text{new}}+k|t_{\text{new}}} \end{bmatrix}^T \tag{44}$$

by

$$\hat{y}_{t_{\text{new}}+k|t_{\text{new}}} = \boldsymbol{x}_{t_{\text{new}}+k|t_{\text{new}}}\hat{\boldsymbol{\beta}}_k \tag{45}$$

This can be done by providing new data to the `lm_predict()` function.

C.2. Recursive least squares

In the RLS scheme the coefficients are recursively updated through the period. Time t steps from 1 to n and in each step the "newly" obtained data at t is used for calculating updated coefficients. The coefficient vector has the same structure as for LS

$$\boldsymbol{\beta}_{k,t} = \begin{bmatrix} \beta_{0,k,t} & \beta_{1,k,t} & \dots & \beta_{p,k,t} \end{bmatrix}^T \tag{46}$$

The only difference is that we now subscript with t because it varies over time.

Only the most recent input data at t (the row at t from the LS design matrix in Equation (40)) is used in each update

$$\boldsymbol{x}_{k,t} = \begin{bmatrix} x_{0,t|t-k} & x_{1,t|t-k} & \dots & x_{p,t|t-k} \end{bmatrix}^T \tag{47}$$

and similarly the most recent output observation y_t. At each time t the coefficients are updated by

$$\boldsymbol{R}_{k,t} = \lambda \boldsymbol{R}_{k,t-1} + \boldsymbol{x}_{k,t}\boldsymbol{x}_{k,t}^T \tag{48}$$

$$\hat{\boldsymbol{\beta}}_{k,t} = \hat{\boldsymbol{\beta}}_{k,t-1} + \boldsymbol{R}_{k,t}^{-1}\boldsymbol{x}_{k,t}(y_t - \boldsymbol{x}_{k,t}^T\hat{\boldsymbol{\beta}}_{k,t-1}) \tag{49}$$

Hence, when applying RLS for data from the period $t \in (1, 2, \dots, n)$ the RLS provides a new value of the coefficients for each time t (opposed to LS).

The predictions are calculated recursively as well by using the updated coefficients at each time t. Given the inputs

$$\boldsymbol{x}_{t+k|t} = \begin{bmatrix} x_{0,t+k|t} & x_{1,t+k|t} & \dots & x_{p,t+k|t} \end{bmatrix}^T \tag{50}$$

the prediction is

$$\hat{y}_{t+k|t} = \boldsymbol{x}_{t+k|t}\hat{\boldsymbol{\beta}}_{k,t} \tag{51}$$

Only past data has been used when calculating the predictions through the period, hence they are "out-of-sample" predictions (these predictions are returned by `rls_fit()`).

The initial value of R is set simply set to a zero matrix with diagonal $1/10000$ and β set to a zero vector.

An alternative updating scheme, which is actually the implemented scheme (gives the same results as the scheme above), is the Kalman gain scheme (Sayed and Kailath 1994), where matrix inversion is avoided

$$\boldsymbol{K}_{k,t} = \frac{\boldsymbol{P}_{k,t-1}\boldsymbol{x}_{k,t}}{\lambda + \boldsymbol{x}_{k,t}^T \boldsymbol{P}_{k,t-1}\boldsymbol{x}_{k,t}} \tag{52}$$

$$\hat{\boldsymbol{\beta}}_{k,t} = \hat{\boldsymbol{\beta}}_{k,t-1} + \boldsymbol{K}_{k,t}(y_t - \boldsymbol{x}_{k,t}^T \hat{\boldsymbol{\beta}}_{k,t-1}) \tag{53}$$

$$\boldsymbol{P}_{k,t} = \frac{1}{\lambda}\left(\boldsymbol{P}_{k,t-1} - \boldsymbol{K}_{k,t}\boldsymbol{x}_{k,t}^T \boldsymbol{P}_{k,t-1}\right) \tag{54}$$

This actually opens up the possibilities for self-tuned variable forgetting (Shah and Cluett 1991).