Onlineforecast: An R Package for Adaptive and Recursive Forecasting

by Peder Bacher, Hjörleifur G. Bergsteinsson, Linde Frölke, Mikkel L. Sørensen, Julian Lemos-Vinasco, Jon Liisberg, Jan Kloppenborg Møller, Henrik Aalborg Nielsen and 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 dynamical and non-linear models. The setup is tailored to enable the effective use of forecasts as model inputs, e.g. numerical weather forecast. Users can create new models for their particular applications and run models in an operational setting. The package also allows users to easily replace parts of the setup, e.g. using new methods for estimation. The package comes with comprehensive vignettes and examples of online forecasting applications in energy systems, but can easily be applied for online forecasting in all fields.

1 Introduction

Time series analysis and forecasting are indispensable to numerous applied 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 increment between the time-points, i.e. equidistant. Classical time series models for a single equidistant time series use past values of the response variable (model output) as the predictors (inputs). In this way, appropriate models describing the inherent auto-correlation structure of the time series can be realized. Examples of these models include exponential smoothing (e.g. Holt-Winters), AutoRegressive (AR), Moving Average (MA), and the combination of the latter two known as ARMA models. When multiple correlated time series are available, they can be used as simultaneous model inputs to improve the forecast. They are then called exogenous variables and the classical model becomes an ARMAX – hence the X indicates that exogenous input variables are included. ARMAX models are optimal for forecasting the output of linear time invariant (LTI) systems, however for most forecasting applications models that can handle non-linear systems are needed. A wide range of techniques for modelling non-linear systems exists, either based on input transformations or local fitting methods. The onlineforecast package implements an advanced model setup for modelling and forecasting the output of non-linear time varying systems. The setup was developed for applications such as forecasting wind power (Nielsen et al., 2002) and thermal loads in district heating (Nielsen and Madsen, 2006). The significance of the package is in the "online" term, indicating that at each sampling point the model parameter estimates are updated in an effective way for generating multi-step forecasts.

The use of ARMAX models and their variations for forecasting is still widespread (De Gooijer and Hyndman, 2006), especially for energy systems due to the high dependency between variables such as weather, load, renewable generation, and periodic phenomena. Load forecasting is an obvious example. A nice overview of 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.

Bacher et al. (2009) demonstrated that solar power forecasting (Kleissl, 2013) can be improved by moving from a standard AR model to an AR model with an exogenous input (ARX), specifically by using numerical weather predictions (NWPs) as the exogenous inputs. 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 et al. (2013) identified exogenous variables that are suitable for forecasting the heat load of a building, via 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 changes in usage. For example, with new tenants in a house, the dependency between heat load and other variables, such as calendar time 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. A forgetting factor can be introduced to RLS to allow for the control of 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), for advances that has been made since then see e.g. (Engel et al., 2004).

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

- Use of forecasts, e.g. NWPs, as input to multi-step forecast models.
- Optimal tuning of models for multi-step horizons.
- Recursive estimation for tracking time-varying systems.

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

Time series modelling and forecasting in R

A wide range of existing software for time series forecasting is currently available (Chatfield and Xing, 2019; Siebert et al., 2021). Below, an overview of the currently most relevant R packages for forecasting is given – generally, the same functionalities are available in Python packages.

Classical ARMAX models can be fitted with the `arima()` function from the stats package and the `Arima()` function from the forecast package (Hyndman and Khandakar, 2008) provides automatic model selection with `arima()`. R Packages like marima (Spliid, 1983), KFAS, sysid and dlm (Petris, 2010) can also 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. Compared to these classical ARMAX models, onlineforecast models offers several advantages, first and foremost the recursive fitting scheme which allows for much faster and adaptive fitting. Furthermore, model coefficients are tuned as a function of the forecast horizon. This optimize the use of multi-step forecasts as models inputs, such functionality is not available for ARMAX models.

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 a system of differential equations or difference equations. State-space models use filter techniques to optimally reconstruct and predict the states, with examples including the Kalman filter, the extended Kalman filter, and 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. The onlineforecast models are basically fitted using a Kalman filter, as explained in Section Regression, thus existing packages could be applied. However, the use of forecasts as model inputs would be very cumbersome and is made very easy with the onlineforecast setup.

For non-parametric time series models, the number of available packages is 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 time series models, including neural network estimation. Neural network, deep learning and machine learning methods are available in R. Recurrent neural networks are available in the rnn, the keras and tensorflow packages. Additive time series models, where non-linear trends are fitted with seasonality patterns, are available in prophet. Time adaptive neural networks, i.e. with recursive updating, can be implemented in various ways (Yang et al., 2019), however currently no effective implementation is available.

Some packages can be useful for forecast evaluation, e.g. ForecastTB presented in (Bokde et al., 2020). Packages like forecastML and modeltime (Alexandrov et al., 2020) provide functionality that simplifies the process of multi-step-ahead forecasting with machine learning algorithms. The handling of 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.

Functionality of onlineforecast

A model is an approximation to the real world, thus it will always be a simplification and can never predict real-world data. 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 onlineforecast 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 is 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 individually for every 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 is the k-step prediction calculated using only available information at time t, usually referred to either "conditional on time t" or "given time t".

The options for estimating the coefficients in the package are 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 by the subscript t on the coefficients. This allows for tracking changes occurring over time.

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 dynamics of the system 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 the second day, hence horizon $k = 23$ in the upper plot, which corresponds to $k = 21$ in the lower plot.

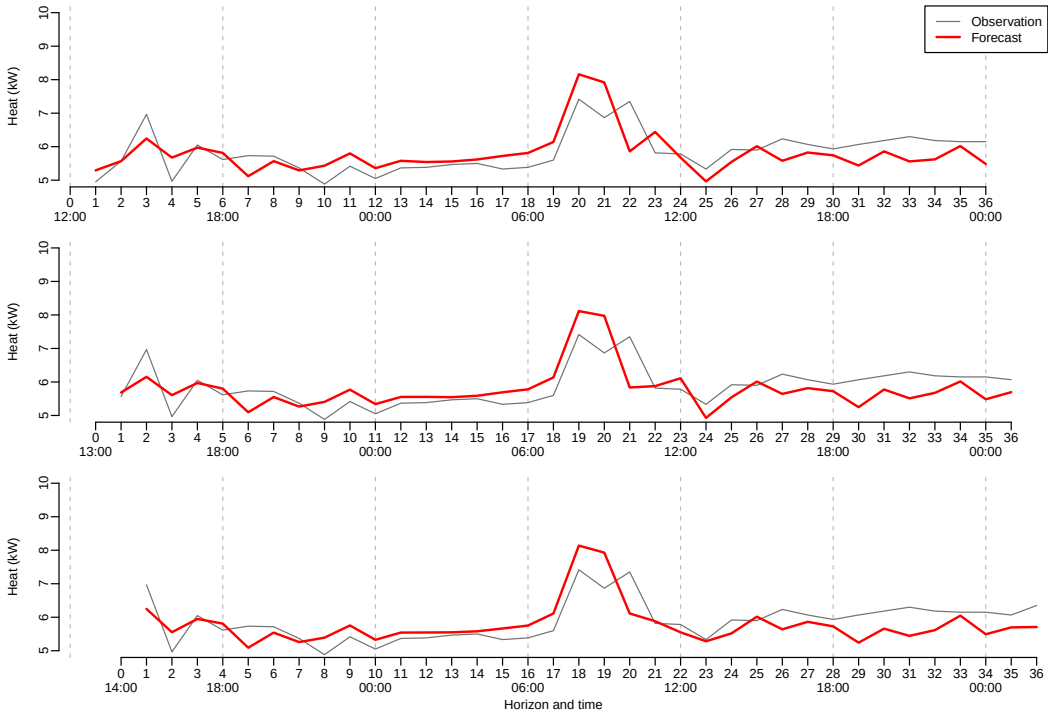


Figure 1: Example of hourly load forecasts at three consecutive time steps. The upper is calculated at 12:00, the middle is calculated at 13:00 and the lower at 14:00. It can be seen how the forecasts change slightly as they are updated in each step, most clearly seen around 12:00 on the second day.

Vignettes

A great way to get hands-on experience with the package 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 package vignettes are:

- setup-data covers how data must be set up. The vignette goes into detail on how observations and model inputs (forecasts) are set up. The vignette also focuses 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 the evaluation of forecasts, and how to use this information to improve a model.
- online-updating demonstrates how to update an operational model when new observations become available. This functionality is not covered in the R examples in the present paper.

Furthermore, one vignette is available only on the website:

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

Paper structure

The paper is structured as follows: In Section Notation and forecast matrices the notation used in the paper and how to set up data is introduced. The core methodology is presented in Section Two-stage modelling procedure and important aspects of forecast modelling are outlined in Section Model selection and validation. In Section Example with R code examples with R code are presented to provide a short hands-on tutorial. The paper ends with a summary and conclusions in Section Discussion and conclusion.

In addition, three appendices are included with the paper. In Appendix Forecast model notation some guidelines on the mathematical notation of forecast models are provided. In Appendix Regression 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 equidistantly 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 indexes 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

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

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 Forecast model notation on how to write the full forecast models.

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

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

where

- t is the counter of time for equidistant time points with sampling period of 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

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

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

3 Two-stage modelling procedure

Two-stage modelling procedure is a widespread approach to modelling non-linear functional relations between inputs and outputs (see e.g. Breiman and Friedman (1985) and Weisberg (2005) for direct transformation of predictor variables, and Hastie et al. (2009) for non-parametric transformation techniques). 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.

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

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

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

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

the model above, given data from the period, $t = 1, 2, \ldots, n$, are optimized 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 gathered in the vector $\boldsymbol{\beta}_k$ and calculated separately for each horizon k. In Appendix Regression both schemes are presented in full detail.

In the LS scheme the coefficients are constant during 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 calculated by

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

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

In the RLS scheme the coefficients are calculated recursively, meaning that they are updated in every time step – the RLS is actually a Kalman filter with the model coefficients in the state vector. At each time t the coefficients are updated by

$$\boldsymbol{R}_{k,t} = \lambda \boldsymbol{R}_{k,t-1} + \boldsymbol{x}_{k,t}\boldsymbol{x}_{k,t}^T \tag{15}$$

$$\hat{\boldsymbol{\beta}}_{k,t} = \hat{\boldsymbol{\beta}}_{k,t-1} + \boldsymbol{R}_{k,t}^{-1}\boldsymbol{x}_{k,t}(y_t - \boldsymbol{x}_{k,t}^T\hat{\boldsymbol{\beta}}_{k,t-1}) \tag{16}$$

and the predictions by

$$\hat{y}_{t+k|t} = \boldsymbol{x}_{t+k|t}\hat{\boldsymbol{\beta}}_{k,t} \tag{17}$$

where $\boldsymbol{x}_{k,t}$ is the data available for horizon k at time t (a row in the design matrix $\boldsymbol{X}_{k,n}$), and $\boldsymbol{x}_{t+k|t}$ is the k'th horizon transformed input forecast available at time t, see the appendix for all details. The coefficients adapts to data over time and the level of adaptivity is controlled by the forgetting factor λ (with value between 0 and 1). When $\lambda = 1$, all past data at t is equally weighted. When $\lambda < 1$, higher weight is put on recent data – the smaller the value, the faster the model adapts to recent data. By optimizing the forgetting factor as an offline parameter, the model adaptivity can be tuned.

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 calculated once using all data, 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" (except for the offline parameters). This makes a difference, since model overfitting is less of a problem when using the RLS scheme.

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. The optimization of offline parameters can be carried out when computational resources are available (e.g. every week for hourly forecasts).

Transformations

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. The main functionality they have to fulfil is to return a forecast matrix (or a list of them), which is also the reason why some of the regular R functions and operators has been extended for the

multi-horizon setup, e.g. for splines as explained below. For R examples, see Section Defining a model and the vignettes. 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 section, the low-pass filtering is shortly described below. For more examples of transformations, see the package vignettes.

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, i.e. both continuous and integer variables. This functionality is achieved using a simple syntax as explained in Section Example with R code.

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 heat demand are slow due to the thermal mass of the building. Thus they can be modelled using a low-pass filter, 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

$$x_{t+k|t} = \frac{(1-a)u_{t+k|t}}{1-ax_{t-1+k|t-1}} \tag{18}$$

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, i.e. independently for each horizon k. More advanced filters can also be implemented.

4 Model selection and validation

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 (see the model-selection vignette for examples).

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 incremented or decremented.

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 the lowest score is selected. It is important to note, that the implemented procedure should only be used with the RLS scheme, with the LS scheme the score is calculated fully in-sample leading to overfitting.

Model validation

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

Training and test set

One fundamental problem in data-driven modelling is overfitting. 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 cross-validation – many methods for dividing into sets are possible. In the onlineforecast setup, when a model is fitted recursively using the RLS, only past data is used when calculating the regression coefficients, so there is little need for dividing into a training and a test set.

The offline parameters (like the forgetting factor and low-pass filter coefficients) are optimized on a particular period, hence overfitting is possible, however, typically, only very few parameters compared with the number of observations are estimated – so it is very unlikely that a recursively-fitted model will overfit.

Scoring

The scoring of forecasts can be done in many ways, however in the onlineforecast package, where the conditional mean is estimated and when using the RLS scheme, we recommend choosing 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 regularization penalty to avoid overfitting, 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 Evaluation.

Residual analysis

Analysing the residuals is an important way of validating that a model cannot be further improved or learning 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 ways of analyzing the residuals are to:

- Plot residual time series to find where large forecast errors occur.
- Plot scatter plots of the residuals vs. other variables to see if there are any apparent dependencies not captured by the model.

In order to dig a bit more into the result of a 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 large changes are found in some periods it might be worthwhile to zoom into those periods to learn what causes these changes and how to potentially 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.

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).

5 Example with R code

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

First, a few remarks on the implementation. onlineforecast models are set up using an object-oriented R6 class. The main reason for this is that R6 objects use reference pointers, which allows to make minimum changes in computer memory when updating a model fit with new data – this would not be possible with the regular S3 class objects, as they are always copied in memory when updated by a function.

Furthermore, it is noted, that model inputs and transformations simply are defined using R code. The regular `formula` class is not used, since it cannot operate as needed on the multi-horizon forecast matrices. The provided code for the inputs defines the transformations etc. and is executed for each input to generate the data used for regression.

Setup of data

Data must be set as variables in a list, here we have loaded `D` with the data for the examples:

```
class(D)


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

As seen its class is `data.list`, which is inherited from the `list` class. Hence, it is simply a `list` extended with some modified and new functions (can be listed with `methods(class="data.list")`).

All inputs to be used must be formatted as forecast matrices and set in the list as `data.frames`. For example the ambient temperature forecasts:

```
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 must be in a `POSIXct` vector named `t`:

```
D$t[1:4]


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

Observations must be in a numeric 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 – which demonstrates useful functions for manipulating, validating and exploring forecast data.

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.

If we want to forecast the observed `heatload` variable in the data list `D`, we set that as the model output by:

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

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

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

The code given as text simply evaluates into the `Ta` forecast matrix, which will lead to the k-step forecast of ambient temperature (i.e. a column in `Ta`) will be set directly into the design matrix for the k horizon regression (more explanation of this is given in the end of the current section).

Adding an intercept to a model can be done by:

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

where the function `one()` evaluates into a forecast matrix of 1's, which will be inserted in the design matrix, see details in Appendix Regression.

Functions for a range of useful transformations were already listed in Section Transformations. 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 apply a low-pass filter along each column of `Ta` and return a forecast matrix with the modified data. The filter coefficient is set to $a = 0.9$. To illustrate the effect of this, see 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 is 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)")
```

Varying-coefficient models can be realized with multiplication of inputs (Hastie and Tibshirani, 1993). For more details, see the solar-power-forecasting example on the website. For more examples of input transformations, e.g. fourier-series and auto-regressive inputs, see the setup-and-use-model vignette.

Execution of the input transformations

As seen above, R code is given for each input. The code is given as text and will simply be executed to calculate the data for regression. This is carried out with the function `model$transform_data(data)`, inside which:

```
eval(parse(text=frml), data)
```

is executed for each input. The `frml` is the R code for the input (as a character e.g. as `"Ta"` or `"bspline(Ta,df=5)"` in the examples) and `data` is the list containing the variables used in the `frml` code (as `D` in the examples).

This way of defining the input formulas simply as code is very flexible. It also allows for easy debugging, for example a function used in the code can be set for debug and it is possible to step through its execution during the transformation – or even by setting `"browser();"` directly into in the input code to stop and step through the execution.

The only constraint to an inputs code is that it must just return a forecast matrix as a `data.frame` (or list of them). The regression can then be carried out *separately for every horizon* by, for the k horizon taking the k column from each of the returned matrices and bind these together into a design matrix, on which LS or RLS can be applied to calculate the k horizon forecasts.

Model fitting and offline parameter tuning

After setting up a model it can be fitted to data by carrying out the transformation and regression steps. To simply the code, functions of fitting the model are provided. Different functions implement different regression schemes. The two currently available fitting functions are `lm_fit()` and `rls_fit()` – they take offline parameters as a vector, fit a model and return the RMSE score (summed across all fitted horizons).

To demonstrate the model-fitting process, we first replace the inputs on the model defined above with two inputs by:

```
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 observations to be included in the score calculation. It is useful for defining a burn-in period and dividing the data into test and training sets. For the current linear regression we simply include all points by:

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

Now the summed RMSE for the horizons 1 to 6 steps ahead can be obtained 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 minimizes the score.

Any optimizer function in R can be used, but, again to simplify the code, wrappers for `optim()` are included – similar wrappers can easily be made for other optimizers. The parameter(s) to optimize within the wrapper function is defined by:

```
model$add_prmbounds(Ta__a1 = c(min=0.8, init=0.95, 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 0.95 and staying within the specified bounds.

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
## 0.8926346
```

i.e. a lower low-pass coefficient than the initial value of 0.95.

For more details, see the setup-and-use-model vignette.

Calculating forecasts

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

```
model$kseq <- 1:24
fit <- lm_fit(model$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 the predict function:

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

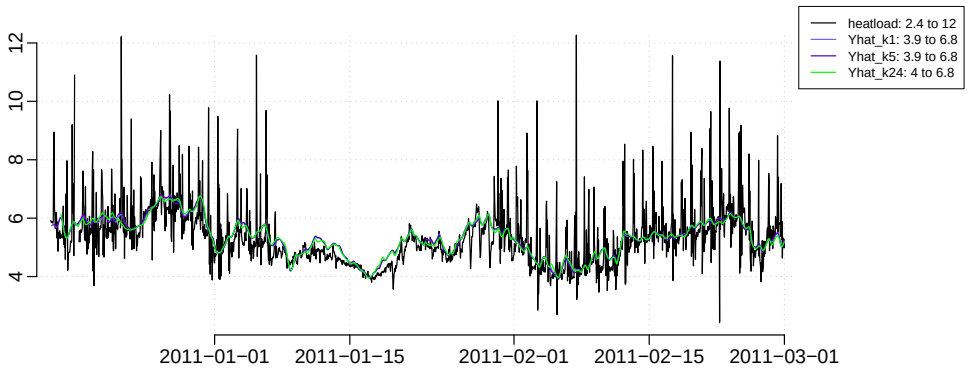
will return a forecast matrix using the input data in `D`.

Evaluation

Finally, it can be worthwhile to evaluate the forecasts and inspect the model for potential improvements. For a more comprehensive introduction, see the forecast-evaluation vignette.

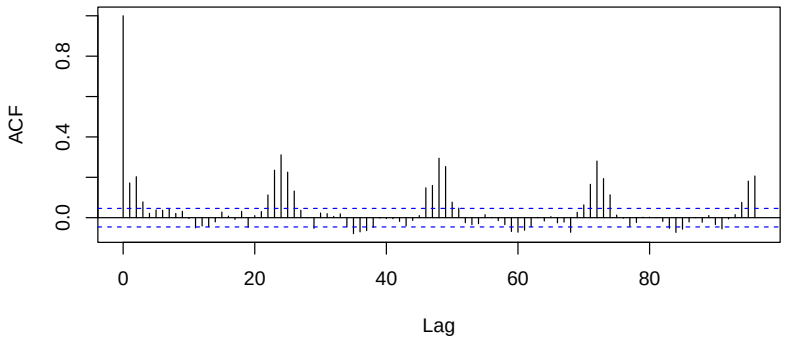
First, it is always a good idea to plot the model's forecasts over time and see how well it predicts:

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



We can plot the ACF of the one-step residuals by:

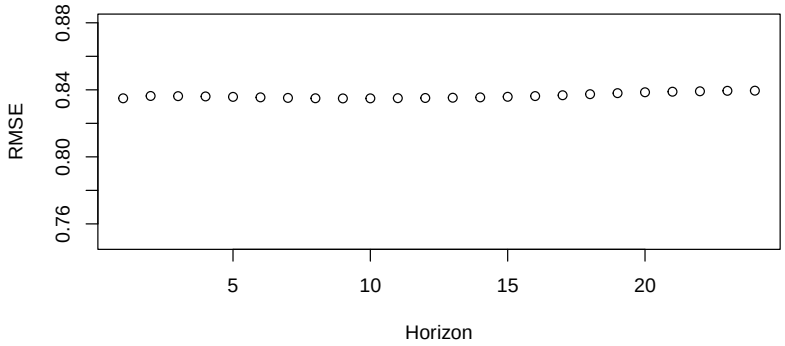
```
acf(residuals(fit)$h1, na.action=na.pass, lag.max=96, main="")
```



The ACF plot suggests that there remains a diurnal pattern to be modelled. It can be achieved 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 may 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(0.75,0.88), xlab="Horizon")
```



The trend is relatively constant, which makes sense since the model is very simple. The offline parameters were optimized for $k = 3$ and $k = 18$, which can explain why it is not monotonic increasing with the horizon.

6 Discussion and conclusion

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 straightforward. The function must receive either a forecast matrix or a list of forecast matrices and return either after processing. Furthermore, when used in an online operational setup, where the transformation is executed whenever new data arrives, it is possible to save state information inside 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 also 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 is 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 of the package, 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 et al., 2006) and (Bjerregaard et al., 2021), as well as carry out normalization and Copula transformations, which can be very useful for spatio-temporal forecast models, see (Tastu et al., 2011) or (Lemos-Vinasco et al., 2021).

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 for almost one year at the time of writing.

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.
- Optimal tuning of non-linear models for multi-step horizons.
- Recursive estimation for tracking time-varying systems computationally efficient for multiple horizons.

The onlineforecast package has a significant value for anyone who needs to carry out 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 – specifically, 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** (Eddelbuettel and Balamuta, 2018) along with **RcppArmadillo** (Eddelbuettel and Sanderson, 2014) 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 was used. We acknowledge the **devtools** and **knitr** (Xie, 2015), **rmarkdown** (Xie et al., 2018), **R.rsp**, **testthat** (Wickham, 2011) 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 4.3.1. 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: PTXHeatUtilisation (Energy Cluster Denmark), 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).

Bibliography

A. Alexandrov, K. Benidis, M. Bohlke-Schneider, V. Flunkert, J. Gasthaus, T. Januschowski, D. C. Maddix, S. S. Rangapuram, D. Salinas, J. Schulz, et al. Gluonts: Probabilistic and neural time series modeling in python. *J. Mach. Learn. Res.*, 21(116):1–6, 2020. [p175]

H. K. Alfares and M. Nazeeruddin. Electric load forecasting: Literature survey and classification of methods. *International Journal of Systems Science*, 33(1):23–34, 2002. doi: 10.1080/00207720110067421. URL https://doi.org/10.1080/00207720110067421. [p173]

P. Bacher, H. Madsen, and H. A. Nielsen. Online short-term solar power forecasting. *Solar Energy*, 83 (10):1772 – 1783, 2009. ISSN 0038-092X. doi: https://doi.org/10.1016/j.solener.2009.05.016. [p173]

P. Bacher, H. Madsen, H. A. Nielsen, and B. Perers. Short-term heat load forecasting for single family houses. *Energy and buildings*, 65:101–112, 2013. [p173, 180, 182]

M. B. Bjerregaard, J. K. Møller, and H. Madsen. An introduction to multivariate probabilistic forecast evaluation. *Energy and AI*, page 100058, 2021. [p187]

N. D. Bokde, Z. M. Yaseen, and G. B. Andersen. Forecasttb—an r package as a test-bench for time series forecasting—application of wind speed and solar radiation modeling. *Energies*, 13(10):2578, 2020. [p175]

L. Breiman and J. H. Friedman. Estimating optimal transformations for multiple regression and correlation. *Journal of the American statistical Association*, 80(391):580–598, 1985. [p178]

C. Chatfield and H. Xing. *The analysis of time series: an introduction with R*. Chapman and hall/CRC, 2019. [p174]

J. D. Cryer and K.-S. Chan. *Time series analysis: with applications in R*, volume 2. Springer, 2008. [p173]

J. G. De Gooijer and R. J. Hyndman. 25 years of time series forecasting. *International journal of forecasting*, 22(3):443–473, 2006. [p173]

E. Dotzauer. Simple model for prediction of loads in district - heating systems. *Applied Energy*, 73(3-4): 277–284, 2002. doi: 10.1016/S0306-2619(02)00078-8. [p173]

D. Eddelbuettel and J. J. Balamuta. Extending extitR with extitC++: A Brief Introduction to extitRcpp. *The American Statistician*, 72(1):28–36, 2018. doi: 10.1080/00031305.2017.1375990. [p187]

D. Eddelbuettel and C. Sanderson. Rcpparmadillo: Accelerating r with high-performance c++ linear algebra. *Computational Statistics and Data Analysis*, 71:1054–1063, March 2014. URL http://dx.doi.org/10.1016/j.csda.2013.02.005. [p187]

Y. Engel, S. Mannor, and R. Meir. The kernel recursive least-squares algorithm. *IEEE Transactions on signal processing*, 52(8):2275–2285, 2004. [p174]

T. Hastie and R. Tibshirani. Varying-coefficient models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 55(4):757–779, 1993. [p183]

T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009. [p178]

J. Helske. KFAS: Exponential family state space models in R. *Journal of Statistical Software*, 78(10):1–39, 2017. doi: 10.18637/jss.v078.i10. [p174]

T. Hong and S. Fan. Probabilistic electric load forecasting: A tutorial review. *International Journal of Forecasting*, 32(3):914–938, 2016. [p173]

R. J. Hyndman and Y. Khandakar. Automatic time series forecasting: the forecast package for R. *Journal of Statistical Software*, 26(3):1–22, 2008. URL https://www.jstatsoft.org/article/view/v027i03. [p174]

J. Kleissl. *Solar energy forecasting and resource assessment*. Academic Press, 2013. [p173]

J. Lemos-Vinasco, P. Bacher, and J. K. Møller. Probabilistic load forecasting considering temporal correlation: Online models for the prediction of households' electrical load. *Applied Energy*, 303: 117594, 2021. [p187]

L. Ljung and T. Söderström. *Theory and practice of recursive identification*. MIT press, 1983. [p174]

H. Madsen. *Time series analysis*. CRC Press, 2007. [p174, 177, 180, 182]

H. Madsen and P. Thyregod. *Introduction to general and generalized linear models*. CRC Press, 2010. [p180]

H. A. Nielsen and H. Madsen. Modelling the heat consumption in district heating systems using a grey-box approach. *Energy and buildings*, 38(1):63–71, 2006. [p173, 180]

H. A. Nielsen, H. Madsen, and T. S. Nielsen. 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, 2006. [p187]

T. S. Nielsen, H. A. Nielsen, and H. Madsen. Prediction of wind power using time-varying coefficient functions. In *Proceedings of the XV IFAC World Congress*, 2002. [p173]

G. Petris. An r package for dynamic linear models. *Journal of Statistical Software*, 36(1):1–16, 2010. [p174]

A. H. Sayed and T. Kailath. A state-space approach to adaptive rls filtering. *IEEE signal processing magazine*, 11(3):18–60, 1994. [p193]

S. L. Shah and W. R. Cluett. Recursive least squares based estimation schemes for self-tuning control. *The Canadian Journal of Chemical Engineering*, 69(1):89–96, 1991. [p193]

J. Siebert, J. Groß, and C. Schroth. A systematic review of python packages for time series analysis. *arXiv preprint arXiv:2104.07406*, 2021. [p174]

H. Spliid. A fast estimation method for the vector autoregressive moving average model with exogenous variables. *Journal of the American Statistical Association*, 78(384):843–849, 1983. [p174]

L. J. Tashman. Out-of-sample tests of forecasting accuracy: an analysis and review. *International journal of forecasting*, 16(4):437–450, 2000. [p181]

J. Tastu, P. Pinson, E. Kotwa, H. Madsen, and H. A. Nielsen. Spatio-temporal analysis and modelling of short-term wind power forecast errors. *Wind Energy*, 14(1):43–60, 2011. [p187]

S. Weisberg. *Applied linear regression*, volume 528. John Wiley & Sons, 2005. [p178]

H. Wickham. testthat: Get started with testing. *The R Journal*, 3:5–10, 2011. URL https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf. [p187]

Y. Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition, 2015. URL https://yihui.org/knitr/. ISBN 978-1498716963. [p187]

Y. Xie, J. Allaire, and G. Grolemund. *R Markdown: The Definitive Guide*. Chapman and Hall/CRC, Boca Raton, Florida, 2018. URL https://bookdown.org/yihui/rmarkdown. ISBN 9781138359338. [p187]

C. Yang, J. Qiao, Z. Ahmad, K. Nie, and L. Wang. Online sequential echo state network with sparse rls algorithm for time series prediction. *Neural Networks*, 118:32–42, 2019. [p174]

1 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

$$\begin{aligned}
\text{Intercept:} \quad & \mu_{t+k|t} && = 1 && (19)\\
\text{Periodic:} \quad & \boldsymbol{x}_{\text{per},t+k|t} && = f_{\text{fs}}(t; n_{\text{har}}) && (20)\\
\text{Part 1:} \quad & x_{1,t+k|t} && = H(B;a)u_{1,t+k|t} && (21)\\
\text{Part 2:} \quad & \boldsymbol{x}_{23,t+k|t} && = f_{\text{bs}}(u_{2,t+k|t}; n_{\text{deg}})u_{3,t+k|t} && (22)\\
\text{Part 3:} \quad & x_{4,t+k|t} && = u_{4,t} && (23)
\end{aligned}$$

and the regression stage

$$Y_{t+k|t} = \beta_{0,k}\mu_{t+k|t} + \boldsymbol{\beta}_{1,k}^T\boldsymbol{x}_{\text{per},t+k|t} + \beta_{2,k}x_{1,t+k|t} + \boldsymbol{\beta}_{3,k}^T\boldsymbol{x}_{23,t+k|t} + \beta_{4,k}x_{4,t+k|t} + \varepsilon_{t+k|t} \quad (24)$$

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

$$\begin{aligned}
\boldsymbol{\beta}_k &= \begin{bmatrix}\beta_{0,k} & \boldsymbol{\beta}_{1,k}^T & \beta_{2,k} & \boldsymbol{\beta}_{3,k}^T & \beta_{4,k}\end{bmatrix}^T && (25)\\
&= \begin{bmatrix}\beta_{0,k} & \beta_{1,1,k} & \beta_{1,2,k} & \cdots & \beta_{1,2n_{\text{har}},k} & \beta_{2,k} & \beta_{3,1,k} & \beta_{3,2,k} & \cdots & \beta_{3,n_{\text{deg}},k} & \beta_{4,k}\end{bmatrix}^T && (26)
\end{aligned}$$

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

$$Y_{t+k|t} = \beta_{0,k} + \boldsymbol{\beta}_{1,k}^T f_{\text{fs},k}(t; n_{\text{har}}) + \beta_{2,k} H_k(B; a) u_{1,t+k|t} + \boldsymbol{\beta}_{3,k}^T f_{\text{bs},k}(u_{2,t+k}; n_{\text{deg}}) u_{3,t+k|t} + \beta_{4,k} u_{4,t} + \varepsilon_{t+k|t} \tag{27}$$

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

$$Y_{t+k|t} = \mu_k + f_{\text{fs},k}(t; n_{\text{har}}) + H_k(B; a) u_{1,t+k|t} + f_{\text{bs},k}(u_{2,t+k|t}; n_{\text{deg}}) u_{3,t+k|t} + \beta_k u_{4,t} + \varepsilon_{t+k|t} \tag{28}$$

It is 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

$$Y_{t+k|t} = \mu + f_{\text{fs}}(t; n_{\text{har}}) + H(B; a) u_{1,t+k|t} + f_{\text{bs}}(u_{2,t+k|t}; n_{\text{deg}}) u_{3,t+k|t} + \beta u_{4,t} + \varepsilon_{t+k|t} \tag{29}$$

and similarly the transformation parameters can be implicit

$$Y_{t+k|t} = \mu + f_{\text{fs}}(t) + H(B) u_{1,t+k|t} + f_{\text{bs}}(u_{2,t+k|t}) u_{3,t+k|t} + \beta u_{4,t} + \varepsilon_{t+k|t} \tag{30}$$

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

$$Y = \mu + f_{\text{fs}}(t) + H(B) u_1 + f_{\text{bs}}(u_2) u_3 + \beta u_4 + \varepsilon \tag{31}$$

after making clear how all the variables are defined.

2 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.

Least squares

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

$$\boldsymbol{\beta}_k = \begin{bmatrix} \beta_{0,k} & \beta_{1,k} & \cdots & \beta_{p,k} \end{bmatrix}^T \tag{32}$$

Note, that t is not included in the subscript.

The input data for the k horizon is the design matrix

$$\boldsymbol{X}_{k,t} = \begin{bmatrix} x_{0,1+k|1} & x_{1,1+k|1} & \cdots & x_{p,1+k|1} \\ x_{0,2+k|2} & x_{1,2+k|2} & \cdots & x_{p,2+k|2} \\ \vdots & \vdots & & \vdots \\ x_{0,n-1|n-1-k} & x_{1,n-1|n-1-k} & \cdots & x_{p,n-1|n-1-k} \\ x_{0,n|n-k} & x_{1,n|n-k} & \cdots & x_{p,n|n-k} \end{bmatrix} \tag{33}$$

The output observations are in the vector

$$\boldsymbol{y}_{k,n} = \begin{bmatrix} y_{1+k} & y_{2+k} & \cdots & y_{n-1} & y_n \end{bmatrix}^T \tag{34}$$

The LS estimates of the coefficients are

$$\hat{\boldsymbol{\beta}}_k = (\boldsymbol{X}_{k,n}\boldsymbol{X}_{k,n})^{-1}\boldsymbol{X}_{k,n}\boldsymbol{y}_{k,n} \tag{35}$$

The predictions are "in-sample" and calculated by

$$\hat{\boldsymbol{y}}_{k,n} = \boldsymbol{X}_{k,n}\hat{\boldsymbol{\beta}}_k \tag{36}$$

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

$$\boldsymbol{x}_{t_{\text{new}}+k|t_{\text{new}}} = \begin{bmatrix} x_{0,t_{\text{new}}+k|t_{\text{new}}} & x_{1,t_{\text{new}}+k|t_{\text{new}}} & \cdots & x_{p,t_{\text{new}}+k|t_{\text{new}}} \end{bmatrix}^T \tag{37}$$

by

$$\hat{y}_{t_{\text{new}}+k|t_{\text{new}}} = \boldsymbol{x}_{t_{\text{new}}+k|t_{\text{new}}}\hat{\boldsymbol{\beta}}_k \tag{38}$$

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

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

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

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 (33)) is used in each update

$$\boldsymbol{x}_{k,t} = \begin{bmatrix} x_{0,t|t-k} & x_{1,t|t-k} & \cdots & x_{p,t|t-k} \end{bmatrix}^T \tag{40}$$

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

$$\boldsymbol{R}_{k,t} = \lambda\boldsymbol{R}_{k,t-1} + \boldsymbol{x}_{k,t}\boldsymbol{x}_{k,t}^T \tag{41}$$

$$\hat{\boldsymbol{\beta}}_{k,t} = \hat{\boldsymbol{\beta}}_{k,t-1} + \boldsymbol{R}_{k,t}^{-1}\boldsymbol{x}_{k,t}(y_t - \boldsymbol{x}_{k,t}^T\hat{\boldsymbol{\beta}}_{k,t-1}) \tag{42}$$

Hence, when applying RLS for data from the period $t \in (1, 2, \ldots, n)$ 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

$$\boldsymbol{x}_{t+k|t} = \begin{bmatrix} x_{0,t+k|t} & x_{1,t+k|t} & \dots & x_{p,t+k|t} \end{bmatrix}^T \tag{43}$$

the prediction is

$$\hat{y}_{t+k|t} = \boldsymbol{x}_{t+k|t}\hat{\boldsymbol{\beta}}_{k,t} \tag{44}$$

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

$$\boldsymbol{K}_{k,t} = \frac{\boldsymbol{P}_{k,t-1}\boldsymbol{x}_{k,t}}{\lambda + \boldsymbol{x}_{k,t}^T\boldsymbol{P}_{k,t-1}\boldsymbol{x}_{k,t}} \tag{45}$$

$$\hat{\boldsymbol{\beta}}_{k,t} = \hat{\boldsymbol{\beta}}_{k,t-1} + \boldsymbol{K}_{k,t}(y_t - \boldsymbol{x}_{k,t}^T\hat{\boldsymbol{\beta}}_{k,t-1}) \tag{46}$$

$$\boldsymbol{P}_{k,t} = \frac{1}{\lambda}\left(\boldsymbol{P}_{k,t-1} - \boldsymbol{K}_{k,t}\boldsymbol{x}_{k,t}^T\boldsymbol{P}_{k,t-1}\right) \tag{47}$$

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

Peder Bacher
Dynamical Systems, Department of Applied Mathematics and Computer Science, Technical University of Denmark
Asmussens Allé, Building 303B
2800 Kgs. Lyngby, Denmark
E-mail: pbac@dtu.dk
URL: https://www.compute.dtu.dk/english/research/research-sections/dynsys/

Hjörleifur G. Bergsteinsson
Dynamical Systems, Department of Applied Mathematics and Computer Science, Technical University of Denmark
Asmussens Allé, Building 303B
2800 Kgs. Lyngby, Denmark
E-mail: hgbe@dtu.dk
URL: https://www.compute.dtu.dk/english/research/research-sections/dynsys/

Linde Frölke
Dynamical Systems, Department of Applied Mathematics and Computer Science, Technical University of Denmark
Asmussens Allé, Building 303B
2800 Kgs. Lyngby, Denmark
E-mail: hgbe@dtu.dk
URL: https://www.compute.dtu.dk/english/research/research-sections/dynsys/

Mikkel L. Sørensen
Dynamical Systems, Department of Applied Mathematics and Computer Science, Technical University of Denmark
Asmussens Allé, Building 303B
2800 Kgs. Lyngby, Denmark
E-mail: mliso@dtu.dk
URL: https://www.compute.dtu.dk/english/research/research-sections/dynsys/

Julian Lemos-Vinasco
Dynamical Systems, Department of Applied Mathematics and Computer Science, Technical University of Denmark
Asmussens Allé, Building 303B
2800 Kgs. Lyngby, Denmark

E-mail: jlvi@dtu.dk
URL: https://www.compute.dtu.dk/english/research/research-sections/dynsys/

Jon Liisberg
Dynamical Systems, Department of Applied Mathematics and Computer Science, Technical University of Denmark
Asmussens Allé, Building 303B
2800 Kgs. Lyngby, Denmark
E-mail: jlvi@dtu.dk
URL: https://www.compute.dtu.dk/english/research/research-sections/dynsys/

Jan Kloppenborg Møller
Dynamical Systems, Department of Applied Mathematics and Computer Science, Technical University of Denmark
Asmussens Allé, Building 303B
2800 Kgs. Lyngby, Denmark
E-mail: jkmo@dtu.dk
URL: https://www.compute.dtu.dk/english/research/research-sections/dynsys/

Henrik Aalborg Nielsen
ENFOR A/S
Røjelskær 11, 3.
2840 Holte, Denmark
E-mail: han@enfor.dk
URL: https://www.enfor.dk

Henrik Madsen
Dynamical Systems, Department of Applied Mathematics and Computer Science, Technical University of Denmark
Asmussens Allé, Building 303B
2800 Kgs. Lyngby, Denmark
E-mail: hmad@dtu.dk
URL: https://www.compute.dtu.dk/english/research/research-sections/dynsys/