House of Energy Markets & Finance

UNIVERSITÄT DUISBURG ESSEN
Open-Minded

Prof. Dr. Florian Ziel
Environmental Economics, esp. Economics of Renewable Energy
House of Energy Markets and Finance
University Duisburg-Essen

florian.ziel@uni-due.de

December 13, 2023

Econometrics of Electricity Markets

course material 1.0

Contents

1 Introduction to European Electricity Markets

A few decades ago power markets were monopolistic and government controlled almost all over the world. This has changed since the 1990s when some electricity markets become deregulated and liberalized. Nowadays, most of the European and American electricity markets are liberalized. Electricity is traded on the spot and derivative markets. However, electricity is a very special commodity. It is physically not storable, but it can be converted to other types of energy that can be stored in an energy storage. Moreover, power systems require that production and consumption matches to have stable electricity networks. These properties, the non-storability, and system balance (production equals consumption) determine the very specific properties of electricity markets. The behaviour distinguishes clearly from standard financial markets like stock markets, see [Wer14].

Electricity is traded in Europe on different electricity markets but also directly by bilateral contracts resp. over-the-counter (OTC) contracts. An OTC contract is usually made between two parties, an electricity supplier that produces the electricity and an electricity consumer who uses the electricity. About OTC contracts there is not that much information as the contract details are usually secret. However, OTC contracts are still a reasonable factor in electricity trading. For instance, in Germany about 30-50% of the produced electricity is traded by OTC contracts, even though this percentage tends to decline slowly. The remaining electricity is traded on electricity markets. Electricity markets offer several products that can be traded. Every product is a product with either physical or cash settlement (financial price settlement). Products with a physical settlement require the real delivery of electricity in a certain region at a specific time resp. over a specific period time. Mostly, products where the time from trading to the start of the delivery of electricity is short have a physical delivery. Thus, they are often referred as spot products. On the other hand, products (usually derivatives) where the electricity delivery is further in the future have a cash settlement. This cash settlement is set based on the spot market results. This issue will be clearer if we have an exemplary look at an electricity market for Germany. However, in this lecture we will focus on the modelling of spot markets, especially day-ahead markets as they play an important role in electricity trading. In the German case, day-ahead spot price results are used to settle major future products.

As mentioned, electricity markets are directly based on the delivery of electricity into a certain region. These regions are known as bidding zones or market zones. These regions are often, but not always, countries or states. Next to the bidding zones, there is another division of Europe into so called control areas. Figure 1 illustrates the corresponding regions in Europe. These control areas are regions that are responsible for ensuring the physical properties of the electricity system/network (esp. the balance of electricity supply and demand and net stability). In many cases in Europe bidding zones are made of only one control area that is a single country. But there are notable exceptions where either the countries are divided into multiple control areas or the bidding zones contain multiple parts of more than one country. All this information can be easily accessed under `https://transparency.entsoe.eu/` or `https://newtransparency.entsoe.eu/`.

For instance, electricity market neighbours of Germany countries like Netherlands, France, Switzerland, Austria, Czech Republic, Poland, Sweden, Denmark or Norway the have only one single control area. Germany contains 4 control areas, namely the area of the four former state-owned electricity suppliers, E.ON, RWE, Vattenfall and EnBW. Also the United Kingdom is split into two control areas. The largest one contains the main island, so the countries: England, Scotland and Wales. Northern Ireland is a separate control area. However, from the market perspective the division into control areas is not as relevant for us as the bidding zones, because prices are settled for each bidding zone separately.

For the day-ahead markets there are several bidding zones in Europe. Countries like Netherlands, France, Switzerland, Austria, the Czech Republic and Poland have single electricity markets where the bidding zones match the corresponding control areas. An important bidding zone is the zone that contains Germany and Luxembourg[1]. Till October 2018 Germany, Luxembourg and Austria shared a bidding zone. It contained five control areas: all four German control areas and the Austrian control area. With beginning of October 2018 this bidding zone got split into

[1]Even though Luxembourg is part of the bidding zone, we will refer it sometimes only to the German bidding zone

Figure 1: Maps of bidding zones (top) and control areas (bottom) in Europe (March 2022, source: *newtransparency.entsoe.eu*)

the German and Austrian one. Thus, if we refer to German electricity price data prior October 2018 it refers to the German, Austrian and Luxembourg bidding zone. The British Isles contain

two bidding zones, one that contains the control areas of the Republic of Ireland and Northern Ireland and a larger one that contains the remaining United Kingdom (England, Scotland and Wales), even though slightly incorrect, this is usually referred as UK zone (officially National Grid). Moreover, Italy and the Scandinavian countries, Denmark, Sweden and Norway contain multiple bidding zones in a single control area. For example, Denmark is divided into two price zones, Denmark West which mainly contains Jutland and Denmark East which contains the Island in the east of Denmark.

In the bidding zones, the power market is organized by electricity market operators. In many bidding zones there is only one day-ahead electricity market operator. These electricity markets may be run by the same power exchange.

For example, the EPEX SPOT (European Power Exchange) organizes day-ahead electricity price auctions for France, Switzerland, and Germany/Luxembourg, Austria, Netherlands, Belgium and UK. However, for the German and UK zones there are two day-ahead electricity markets. For the German price area there is, next to the EPEX day-ahead market, the EXAA (Energy Exchange Austria) that organizes day-ahead electricity auctions for the same delivery period. They slightly differ in terms of market rules what leads to different electricity prices for essentially the same physical product. However, the EPEX day-ahead market is the most important one in terms of volume. Also, from the policy point of view the EPEX day-ahead market is more important. For instance, several regulative calculations are based on the day-ahead EPEX price, like the renewable energy feed-in fees. For the UK zone there is the mentioned APX day-ahead electricity auction, but also Nordpool spot runs day-ahead auctions (via Powernext).

1.1 Electricity price auctions

For all European day-ahead electricity price markets there is one auction each auction day for the electricity prices for the next day(s). Most of the European electricity markets have an electricity auction every day for the electricity prices of the next day. However, the EXAA is an exception. Here the auction takes place only on every working day. So in a usual week (without public holidays) there is an auction on Monday for Tuesday, on Tuesday for Wednesday, on Wednesday for Thursday, on Thursday for Friday, and on Friday for Saturday, Sunday and Monday.

In the European countries contracts are based on quarter-hourly, half-hourly and hourly products. This means, there will be an electricity price for a certain amount of electricity for a period of each hour. Denote S the number of products on a day with 24 hours. So for quarter-hourly products, we have $S = 96$, for half-hourly products $S = 48$, and for hourly products $S = 24$. The majority of day-ahead electricity markets offers hourly products. For example, the German and French EPEX day-ahead markets have hourly products. But, the UK and Ireland markets use half-hourly products. The EXAA uses since 3rd September 2014 quarter-hourly products (before also hourly products).[2]

For all European electricity markets the auctions take place in the morning or at noon. For the vast majority the submission gate closes at 11:00 UTC+1/2[3], which is 12:00 CE(S)T. For instance the German day-ahead auction closes at 11:00 UTC+1 in winter time and 10:00 UTC+2 in summer time, which is always the local 12:00 time. The EXAA closes every auction day at 10:12 CE(S)T. After the closing of the auction the market clearing algorithm applies and the electricity prices for the next day are calculated. These electricity prices are published after a short time by the power exchange. For instance, the EXAA which closes at 10:12 CE(S)T publishes the results at 10:20 CE(S)T.

Given a day d, there is an auction for the day $d+1$ (or even $d+2$, $d+3$, etc. for EXAA). On a normal trading day d we get $V(d+1) = S$ electricity prices. This applies to all days of the year, except two days when the clock-change happens. Due to the day-light saving time we have the last Sunday in March with only 23 hours and the last Sunday in October with 25 hours. Thus, if d is the day before the last Sunday in March then we receive in total $V(d+1) = S(1-1/24)$ ($= 23$ for $S = 24$, $= 46$ for $S = 48$, $= 92$ for $S = 96$) values and if d is the day before the last Sunday in October, then we receive in total $V(d+1) = S(1+1/24)$ ($= 25$ for $S = 24$, $= 50$

[2]The EPEX spot offers a 3 hours delayed quarter-hourly day-ahead auction as well.

[3]UTC = universal time code, corresponds to winter time in Greenwich. In this lecture this is regarded as equivalent to the GMT = general mean time.

for $S = 48$, $= 100$ for $S = 96$) values. Obviously, the clock change issue causes problems when modelling electricity markets, as they disturb the daily periodicity. Moreover, people adjust quickly towards their behaviour in local time. To avoid structural breaks in electricity market data, the data is often clock-change adjusted. In this lecture clock-change adjustment means that the missing hour in March (last Sunday at 2 o'clock CET) is linearly interpolated with respect to the neighboring products and the double hour in October (last Sunday at 2 o'clock CET) is averaged. This gives us automatically S observations on every day that are easier to deal with. We discuss the day-light saving time issue more detailed later on.

On every auction there are market participants on the supply and on the demand side. The market participants submit bids under the market rules. For example, an electricity producer could offer for the hour with delivery from 15:00 to 16:00 a volume of 20MWh electricity for a price of 30EUR/MWh. For each electricity market strict market rules are applied. For instance, for the German/Austrian EPEX market there is a lower bidding bound of -500EUR/MWh and upper bound of 3000EUR/MWh. Additionally the minimum bid volume is restricted to 0.1MWh and the minimal price difference is 0.1EUR/MWh (i.e. a bid of 20MWh for 30.1EUR/MWh is possible, but not 20MWh for 30.01EUR/MWh). Furthermore, there are more complex orders, such as block orders possible.

The majority of European countries participate in the Single Day-ahead Coupling (SDAC) objective of the European Union which aims for a single pan European cross zonal day-ahead electricity market. Here, many Transmission System Operators (TSOs) and Nominated Electricity Market Operators (NEMOs) participate in this programme.[4] The participating parties/countries are visualised in Figure 2

The SDAC objective is currently partially satisfied by the the TSOs and NEMOs which participate in the so called price coupling of the regions (PCR) programme. For participating electricity markets the day-ahead electricity price of the main noon auction is determined by the EUPHEMIA algorithm.

EUPHEMIA is a very complex algorithm that takes into account a different bid types and possible transmission flows in the grid between countries to determine the electricity price that maximises the overall welfare (area between supply and demand curve until intersection of all market zones aggregated). The resulting electricity prices are called market clearing prices. The EUPHEMIA algorithm works as a constrained matching of the supply and demand in the bidding zones. Basically the bids for the supply side and demand side get sorted with respect to their price and aggregated in their volume. This builds a supply and demand curve that is widely known in microeconomic theory. The resulting interaction between both curves gives directly the market clearing price. This procedure is illustrated for the EPEX day-ahead electricity price in Figure 3[5]. There we see that the electricity price is the intersection of the supply and demand curves which directly yields the actual electricity price. There we can observe also a couple of characteristics of properties. For instance, we see that the electricity price realises positive and negative values, and exhibits specific seasonal pattern. So the electricity price tends to be lower during night than during morning or evening hours, but also lower on weekends than on working days. Obviously, there are more relevant features that can be observed. We want to explore them in more detail in the next sections.

Furthermore we want to point out that the market coupling can result situations where multiple market zones share exactly the same price. Figure 4 illustrates this behaviour for France, Germany and Poland. There we observe that there are periods, where two countries (France & Germany, Germany &Poland) are coupled but also moments where all three countries share the same price. Note that bidding zones can only share the same price due to market coupling if there are interconnected. However, this does not have to be a direct interconnection. For instance, theoretically could happen that France and Poland share the same price while Germany has a different one, because there are other possible connections. One possibility would be e.g. France $\rightarrow$ Italy North $\rightarrow$ Austria $\rightarrow$ Czech Republic $\rightarrow$ Poland. Still, in such

[4]In March 2022, we have the parcitipation of the TSOs: 50Hertz Transmission, ADMIE, Amprion, APG, AST, CEPS, Creos, EirGrid, Elering, ELES, ELIA, Energinet, ESO, Fingrid, HOPS, Litgrid, MAVIR, PSE, REE, REN, RTE, SEPS, SONI, Statnett, Svenska Kraftnät, TenneT DE, TenneT NL, Terna, Transelectrica, and TransnetBW, and the NEMOs: BSP, CROPEX, SEMOpx (EirGrid and SONI), EPEX, EXAA, GME, HEnEx, HUPX, IBEX, Nasdaq, Nord Pool, OMIE, OKTE, OPCOM, OTE, and TGE.

[5]To show the animation, Adobe Reader or Okular is required.

Figure 2: SDAC parties/countries involved and their status (March 2022), source: www.nemo-committee.eu/sdac

a situation all those countries would have to share the same price, but Germany would have a different one - which is certainly possible but quite unlikely.

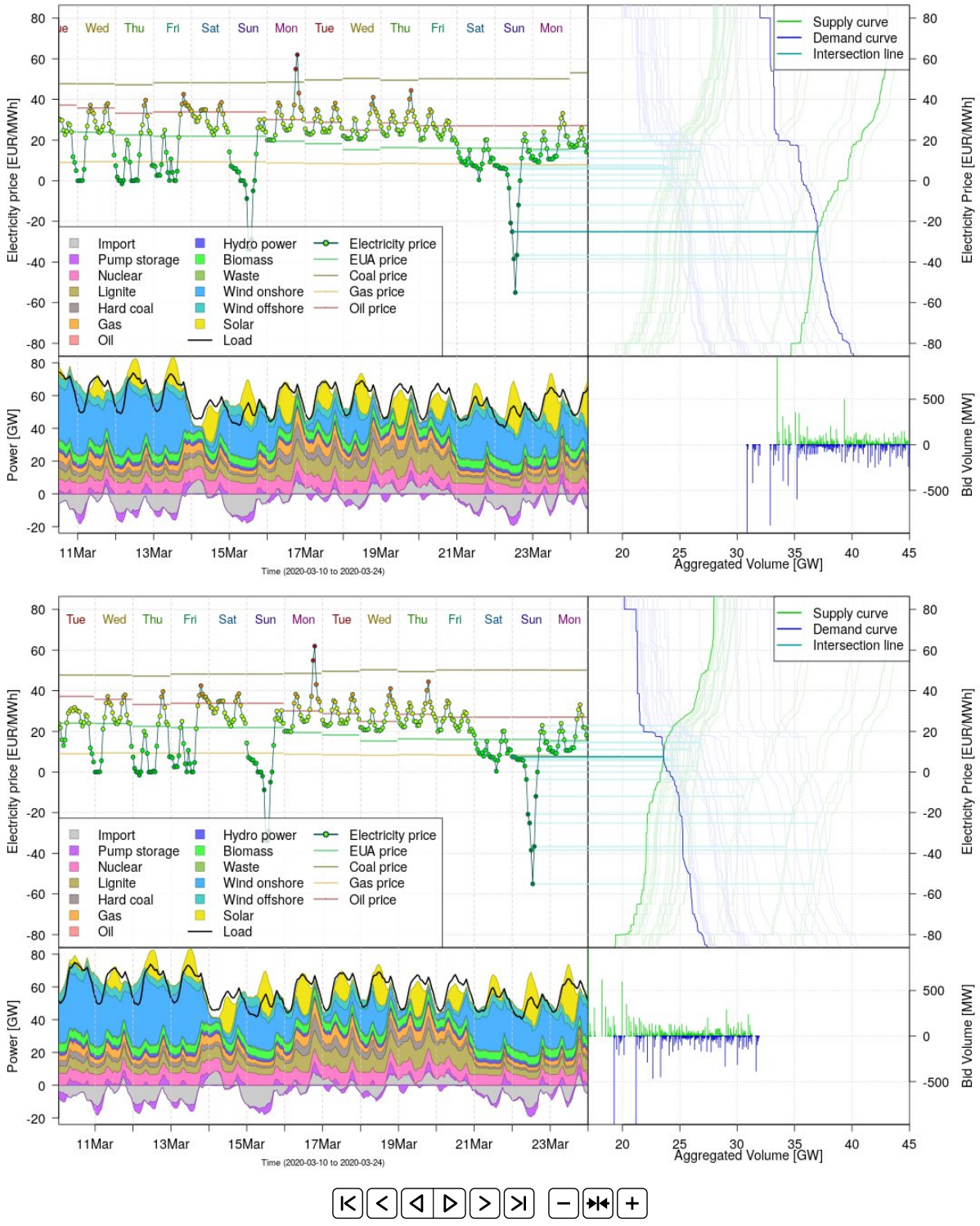


Figure 3: German day-ahead electricity prices for a selected time range with market clearing price and the corresponding supply and demand curves.

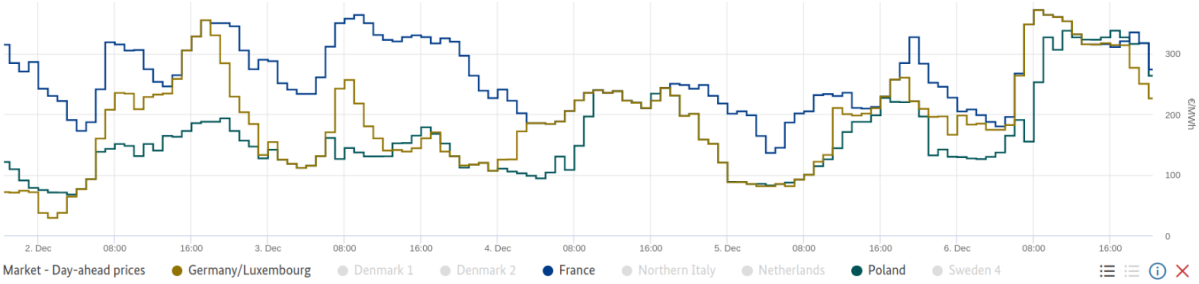


Figure 4: Day-ahead electricity prices for a selected time range in France, Germany and Poland. (source: www.smard.de)

2 Overview of Modelling Approaches.

There are several ways to classify electricity price modelling and forecasting approaches. There is a vast range of different modelling approaches that is suited for different purpose. In [Wer14] electricity price forecasting methods are classified by their different forecasting methods. We use a slightly different classification into two main classes structural/fundamental approaches and data driven approaches.

2.1 Structural/fundamental approaches

The first model class is economically motivated. It is based on the coupling of supply and demand. There are plenty of sophisticated models that explain the market behaviour, including market coupling. However, here we want to discuss only briefly the supply stack model (also merit order model).

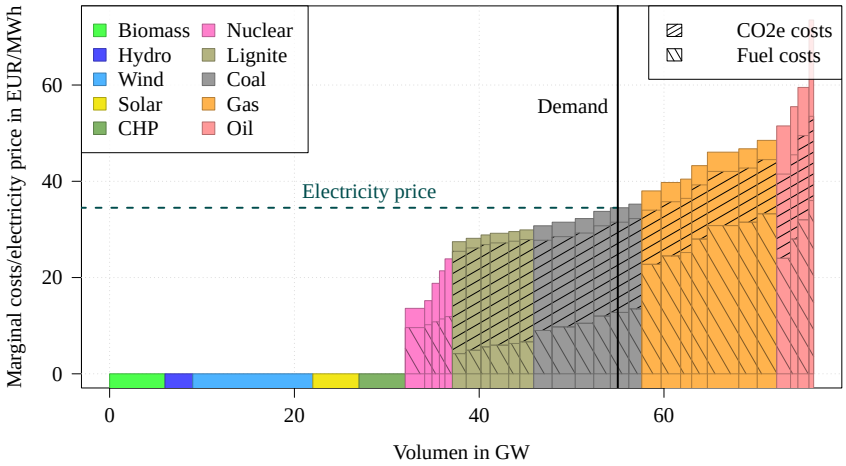
In the supply stack model a core assumption is that there are marginal costs for different power plant types in a certain region. Therefore, all available power plants get sorted by their marginal costs. The corresponding available capacity gets aggregated and results in the merit order curve (also *supply stack*). Based on econometric theory, the equilibrium price for electricity should be the one where all power plants are running with the lowest marginal costs given a certain electricity demand. This is where the merit order curve intersects the demand.

In Figure 5 this principle is visualized in three settings. The assumed power plant park matches roughly the situation in Germany 2020. For conventional power plants the marginal costs are usually the lowest for nuclear power plants, followed by lignite, coal, natural gas and oil. The marginal costs for renewable energy is usually assumed to be zero. Note that especially wind and solar power provides a weather dependent net-feed in. The first graph (see Figure 5(a)) is a moderate setting, that illustrates a kind of standard situation of electricity markets. The second graph (see Figure 5(b)) shows a situation where a lot of wind and solar energy is in the system. Additionally, there is relatively low demand. As there is a lot of cheap renewable energy available, the price is pulled down towards the marginal costs of the renewables which is zero. The price reducing effect of renewables is called merit order effect. The third graph shows the opposite situation where almost all conventional power plants run to satisfy a high electricity demand. Moreover, we can observe from Figure 5 that the electricity demand (which is in the system balance equal to the electricity load) influences the electricity price as well. As the human electricity demand has specific seasonal pattern, this hands down to the electricity price as well. This effect we have already observed in the left hand graph of Figure 3. There the electricity price tends to be lower during the night as less electricity is demanded than during the day time.

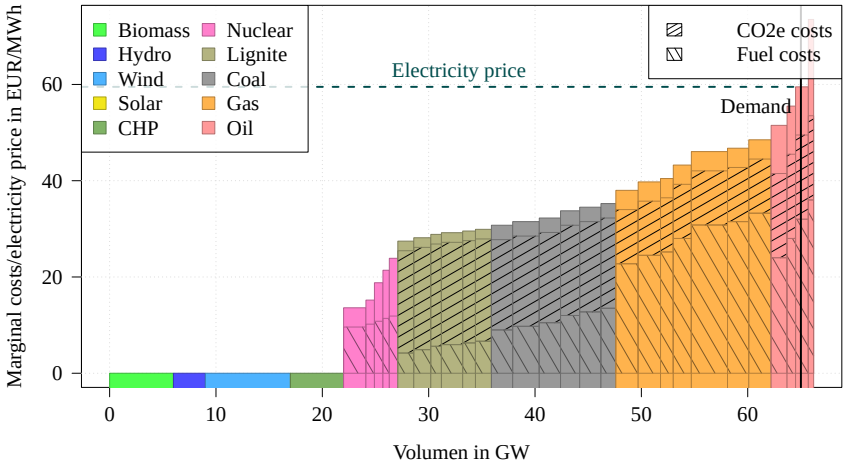
In academic literature the supply stack model is sometimes modified by introducing the residual demand/residual load, see Figure 6 in comparison to Subfigure 5(a). This is the demand minus all generated power at marginal costs of zeros (mainly the renewable energy production). Idea is that from the price perspective it does not matter, if we increase the renewable energy production by 1 GW or decrease the demand by 1 GW; the price effect is the same. Therefore, the residual load is the key driver of the price if the (conventional) merit order is known.

The supply stack modelling approach makes relatively easy to interpret several regulatory interactions causally (e.g. decommissioning of specific power plant, emissions/CO_2 tax, tax for specific fuels). Model extensions can include several features such as import and export, or even simple storage models. Another direction of extensions is the incorporation of game theoretic components in the corresponding electricity price auctions. There, the buyers and sellers are seen as agents that submit their bids in a strategic way. Here the aspects that are in the focus of strategical bidding due to an information asymmetry or market power and agent based modelling is usually the focus of research.

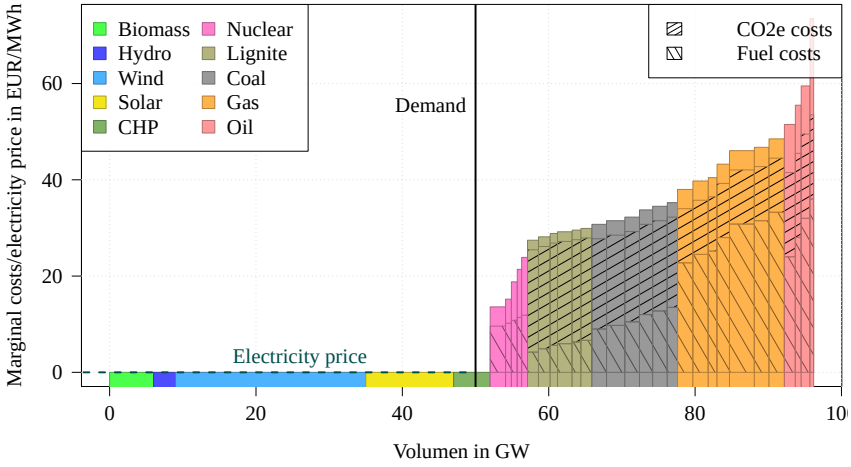
Structural modelling methods are especially suitable for longer-term forecasts. For short-term forecast the forecasting error is relatively high – even though these model approaches improved a lot in the last decade. They require a relatively good power plant data base to give suitable results. In this lecture, we focus on the next class of electricity price models which mainly evaluated historic price data.



(a) Merit order curve with net demand and resulting electricity price in a standard situation.



(b) Merit order curve with net demand and resulting electricity price in a low renewable and high demand situation.



(c) Merit order curve with net demand and resulting electricity price in a high renewable and low demand situation.

Figure 5: Supply stack model for three situations that roughly correspond to the market situation in Germany 2020.

2.2 Data driven approaches

The class of data-driven models contains all models which primarily use historical price data (and potentially other external inputs) to explain electricity prices. These are usually using all types of data science models ranging from econometric, to statistical and machine learning type models. In those model approaches it is an important assumption that we can learn about the

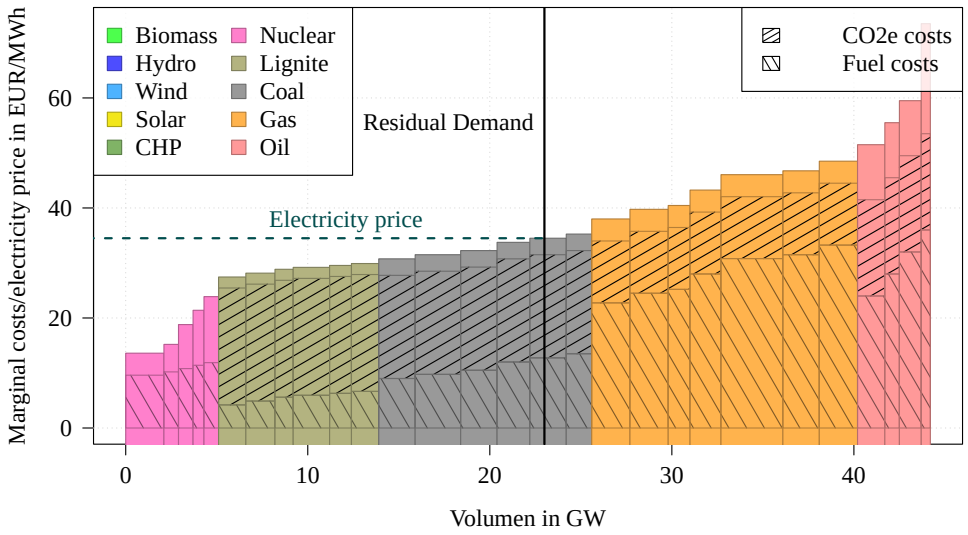


Figure 6: Supply stack model with residual load for the same market situations as in Subfigure 5(a).

electricity market behavior based on historical observations of the past electricity prices. This might be problematic if dramatic structural changes occur, e.g., due to regulatory changes.

Often linear regression based modelling techniques are seen as a standard model type. Here, the dependency between actual electricity prices and their past values. However, regression type techniques can also be used to describe seasonal pattern or influence of several other features such as the electricity load or the production of wind and solar power. In addition we want to remark, that data science models are often used for two purposes. First for understanding relationships, especially causal relationships that drive electricity prices, second for making predictions/forecasts. In the next sections we will cover and discuss both issues. However, the primary focus will be on forecasting.

In section 3, we start with a very simple regression based modeling approach for electricity prices. Then we will cover the basics on forecasting study design and evaluation. In the next sections we will study in more detail characteristics of electricity price data and improve our regression models. In the latter sections we cover more advances forecasting techniques, like forecasting combination and non-linear models like neural networks.

3 Simple electricity price models

Before we can start, we have to introduce some formal notations. We denote by $Y_{d,s}$ the electricity price after(!) the clock change adjustment at day d and time period s. If $S = 24$, there are 24 electricity prices s corresponding directly to the hour of each day. For convenience, we denote $\mathbb{S} = \{0, 1, \ldots, S-1\}$ the set of all delivery periods of a day, counting from 0. Thus, the beginning of the delivery hour is indexed. So for $S = 24$ we have $\mathbb{S} = \{0, 1, \ldots, 23\}$ which maps again the 24 hours of the day. In general, we observe the vector $\boldsymbol{Y}_d = (Y_{d,s})_{s\in\mathbb{S}} = (Y_{d,0}, Y_{d,1}, \ldots, Y_{d,S-1})'$ of electricity prices each day. Now, let us assume that the past D days of the electricity prices are observable. Thus, we have the $D \times S$ matrix of $\boldsymbol{Y} = (\boldsymbol{Y}_1, \boldsymbol{Y}_2, \ldots, \boldsymbol{Y}_D)'$ of electricity prices available to create an electricity price model and generate forecasts. The overall target is now to provide an accurate forecast for the electricity prices of the next day, so make a forecast for $\boldsymbol{Y}_{D+1} = (Y_{D+1,s})_{s\in\mathbb{S}} = (Y_{D+1,0}, Y_{D+1,1}, \ldots, Y_{D+1,S-1})'$.

In the remaining part of the script, we will always consider for illustration purpose German electricity price data of the standard 12:00 o'clock day-ahead auction. Thus, for all empirical results we have $S = 24$. The considered data range covers in total multiple years starting in 2015, as illustrated in Figure 7.

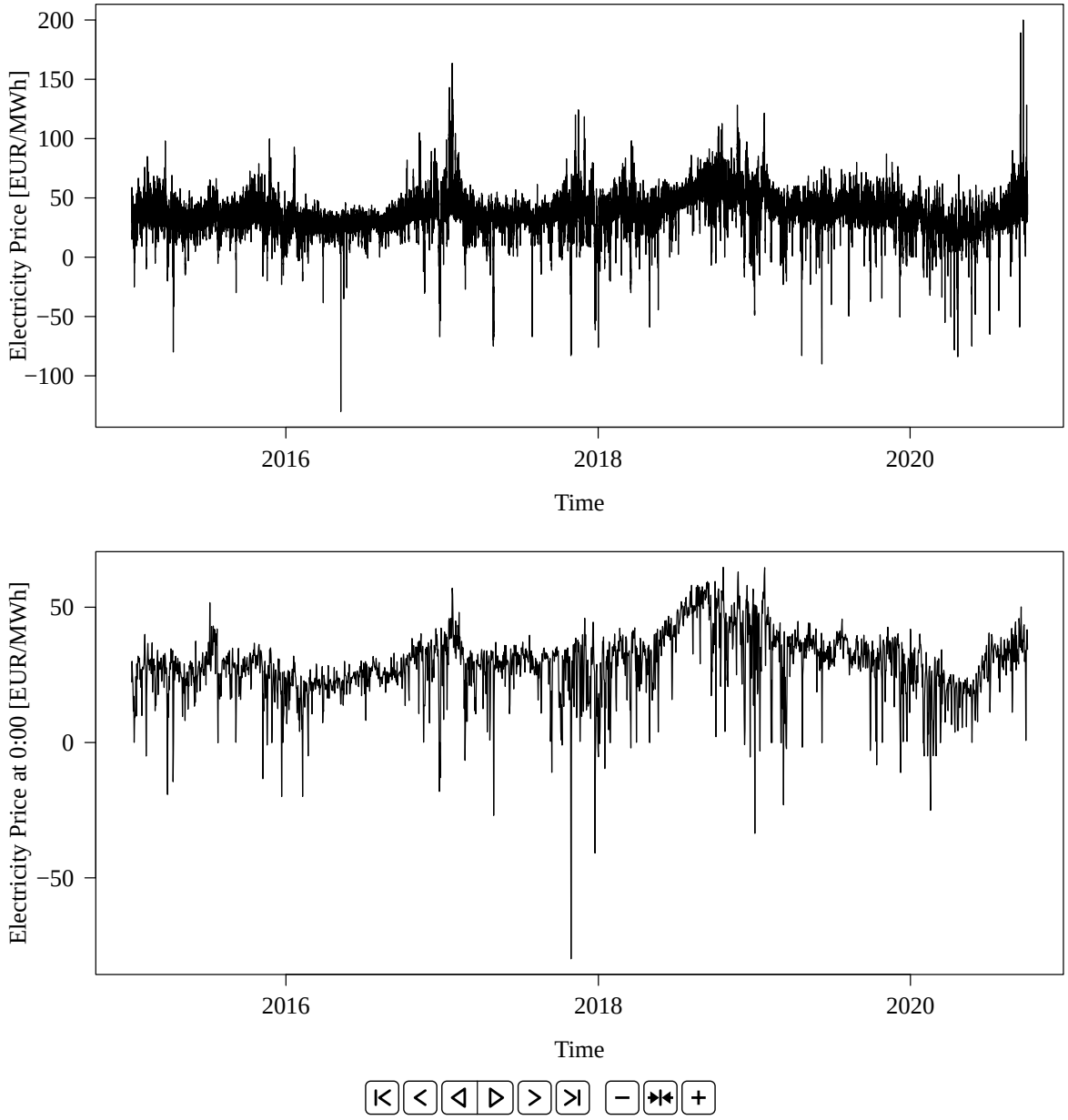


Figure 7: EPEX day-ahead electricity prices for a considered time range. The top figure shows all prices $Y_{d,s}$ for all $s \in \mathbb{S}$ as a single time series. The bottom graphs show the corresponding prices $Y_{d,s}$ for a selected delivery product $s \in \mathbb{S}$.

Now, we want to start to discuss data driven electricity price models. A simple class of models are linear models. The theory of linear models is well-established and many theoretical and computation properties of this modelling technique are known. Moreover, linear models are special cases of many more sophisticated modelling techniques that are also popular in electricity price forecasting, such as artificial neural networks (ANN) or gradient boosting machines (GBMs). Therefore, linear models are a suitable starting point for getting into the topic of electricity price modelling and forecasting methods. Moreover, we will see and understand that linear models can be very competitive.

In literature, it is relatively common to assume a single model for each time period s of the day. So there is a set of S single models that describe the electricity price $Y_{d,s}$ for $s \in \mathbb{S}$. Thus, this way of modelling the electricity prices is sometimes referred as multivariate approach. Now, we will introduce a few simple models.

3.1 Naive models

A very simple model is given by the following model equation for each $s \in \mathbb{S}$

$$Y_{d,s} = Y_{d-1,s} + \varepsilon_{d,s}. \tag{1}$$

Here the electricity price on day d and time period s is the electricity price on the previous day $Y_{d-1,s}$ and the same time period s plus an error term $\varepsilon_{d,s}$ that has zero mean, so $\mathbb{E}(\varepsilon_{d,s}) = 0$. This is a very special type of linear model, as there are no parameters and thus no parameter estimation is required. However, the fact that it requires no estimation at all, makes it popular for serving as a simple benchmark in forecasting evaluation. We call the model given by equation (1) **naive-S**.

An alternative version of the naive model considers not the electricity price of the previous day, but of the previous week. It is given by

$$Y_{d,s} = Y_{d-7,s} + \varepsilon_{d,s}. \tag{2}$$

This model should be more appropriate to capture weekly effects (esp. weekend effects). Similarly to (1), no parameter estimation is required. We call the model given by equation (2) **naive-7S**.

The last more advanced naive model that we introduce here combines the **naive-S** and **naive-7S** model. It is given by

$$Y_{d,s} = \begin{cases} Y_{d-7,s} + \varepsilon_{d,s} & , d \text{ is Monday, Saturday or Sunday} \\ Y_{d-1,s} + \varepsilon_{d,s} & , \text{otherwise} \end{cases}. \tag{3}$$

and we call it **naive** model.

3.2 Autoregressive and expert type models

Now, we consider slightly more advanced models where some parameter estimation is required. These models generalize some of the naive models covered before. Similarly as before, we assume that the electricity price depends on its past values. First, we assume that the electricity price at period s is a weighted sum of a constant and its past value in period s. The most simple model is an autoregressive model of order 1, called **AR(1)**. It is given by

$$Y_{d,s} = \phi_{s,0} + \phi_{s,1} Y_{d-1,s} + \varepsilon_{d,s}, \tag{4}$$

where $\phi_{s,0}$ and $\phi_{s,1}$ are parameters and $\varepsilon_{d,s}$ is the error term. If we fix $\phi_{s,0} = 0$ and $\phi_{s,1} = 1$ we receive the **naive-S** model in equation (1). So (4) can be regarded as extension of the **naive-S** model.

From the statistical point of view, the residual term $\varepsilon_{d,s}$ has to satisfy some assumptions. In general, it holds: the nicer the error structure $\varepsilon_{d,s}$, the easier our life in statistics, in the sense that more (simple) tests or significance statements like confidence intervals can be used or computed. However, during the full lecture we consider only very weak assumption to the

residuals. These are that they have zero mean ($\mathbb{E}(\varepsilon_{d,s}) = 0$), finite variance ($\mathbb{V}\text{ar}(\varepsilon_{d,s}) < \infty$) and zero covariance ($\mathbb{C}\text{ov}(\varepsilon_{d_1,s_1}, \varepsilon_{d_2,s_2}) = 0$ for $d_1 \neq d_2$)[6].

In the naive models we have seen that the previous week values also seems to be important, therefore another more general model option to (4) would be that $Y_{d,s}$ depends additionally on $Y_{d-7,s}$. The corresponding model for each period s is given by

$$Y_{d,s} = \phi_{s,0} + \phi_{s,1} Y_{d-1,s} + \phi_{s,7} Y_{d-7,s} + \varepsilon_{d,s} \tag{5}$$

and has three parameters ($\phi_{s,0}$ and $\phi_{s,1}$ and $\phi_{s,7}$). Even more general would be the **AR(7)** model, where $Y_{d,s}$ depends on the past seven days $Y_{d-k,s}$ for $k = 1, \ldots, 7$. It is given by

$$Y_{d,s} = \phi_{s,0} + \sum_{k=1}^{7} \phi_{s,k} Y_{d-k,s} + \varepsilon_{d,s} \tag{6}$$

and has in total 8 parameters. We can see that model (5) is a special case of the (6) where $\phi_{2,s} = \phi_{3,s} = \ldots = \phi_{6,s} = 0$.

In literature, these type of sparse models as (5) got improved and form the class of so-called expert models. They usually contain some (but not many) parameters, describing mainly autoregressive and seasonal effects. We now introduce a kind of standard expert model that we denote by **expert**. It is given by:

$$Y_{d,s} = \beta_{s,0} + \beta_{s,1} Y_{d-1,s} + \beta_{s,2} Y_{d-2,s} + \beta_{s,3} Y_{d-7,s} + \beta_{s,4} \text{DoW}_d^1 + \beta_{s,5} \text{DoW}_d^6 + \beta_{s,6} \text{DoW}_d^7 + \varepsilon_{d,s} \tag{7}$$

As in (5), we have autoregressive effects at lag 1 and 7, but also at lag 2. Moreover, there are three parameters $\beta_{s,4}$, $\beta_{s,5}$ and $\beta_{s,6}$ that describe the so-called day-of-the-week effects. The day-of-the-week dummies DoW_d^k specify this impact. For example, the Monday dummy is given by DoW_d^1, it is always zero expect for those days d that are Monday. Similarly, DoW_d^2 is the Tuesday dummy, DoW_d^3 for Wednesday up to DoW_d^7 for the Sunday dummy. Note that it is more convenient in literature to use β's for the parameters instead of ϕ's.

As introduced in academic literature the model (7) and similar models are usually referred as expert models. The reason is that they include some expert guesses on the relevant regressors. Ad hoc it is not clear why the autoregressive terms include the lags 1, 2 and 7 are chosen and not other lags. Similarly, there is no clear reason why the weekday dummies for Monday, Saturday and Sunday are considered, but no other weekdays. The motivation for the choices will get clearer during the lecture.

All the autoregressive models ((4), (5), (6))and the expert model (7) can be regarded as linear models. In contrast to the naive models that did not require any parameter estimation, we have to estimate the parameters of the linear model. The estimated parameters and the data will be used to compute forecasts. Hence, we briefly recall the linear model estimation theory via ordinary least squares (OLS). We rewrite any linear model for each $s \in \mathbb{S}$ in a vector representation:

$$Y_{d,s} = \boldsymbol{X}_{d,s}' \boldsymbol{\beta}_s + \varepsilon_{d,s} \tag{8}$$

where $\boldsymbol{\beta}_s = (\beta_{s,0}, \beta_{s,1}, \ldots, \beta_{s,p})'$ and $\boldsymbol{X}_{d,s} = (X_{d,s,0}, X_{d,s,1}, \ldots, X_{d,s,p})'$ are $(p_s + 1)$-dimensional vectors. p_s is the number of parameters, excluding the intercept. For the expert model (7) we receive $\boldsymbol{X}_{d,s} = (1, Y_{d-1,s}, Y_{d-2,s}, Y_{d-7,s}, \text{DoW}_d^1, \text{DoW}_d^6, \text{DoW}_d^7)'$ which is a $p_s + 1 = 7$-dimensional vectors with $p_s = 6$. Remember, that we define (8) for each $s \in \mathbb{S}$. Therefore, such a linear electricity price forecasting model has in total $\sum_{s \in \mathbb{S}} (p_s + 1) = S + \sum_{s \in \mathbb{S}} p_s$ parameters.

When we do the parameter estimation of (8), we always have a certain amount of historical data available. Above we assumed that we have always the past D days of data available and want to forecast day $D+1$. Thus, we denote $\mathcal{Y}_s = (Y_{1,s}, \ldots, Y_{D,s})'$ the vector of the D electricity prices from day 1 to D of period s. Similarly we introduce the regression matrix (also known as design matrix) $\mathcal{X}_s$ of dimension $D \times (p_s + 1)$, containing all the information of the regressors from day 1 to D. So we have

$$\mathcal{X}_s = (\mathcal{X}_{s,0}, \mathcal{X}_{s,1}, \ldots, \mathcal{X}_{s,p_s}) = \begin{pmatrix} \boldsymbol{X}_{1,s}' \\ \boldsymbol{X}_{2,s}' \\ \vdots \\ \boldsymbol{X}_{D,s}' \end{pmatrix} = \begin{pmatrix} X_{1,s,0} & X_{1,s,1} & \ldots & X_{1,s,p_s} \\ X_{2,s,0} & X_{2,s,1} & \ldots & X_{2,s,p_s} \\ \vdots & \vdots & \ddots & \vdots \\ X_{D,s,0} & X_{D,s,1} & \ldots & X_{D,s,p_s} \end{pmatrix} \tag{9}$$

[6]The latter one could be relaxed further, e.g. to covariance stationarity which allows for non-zero correlation.

and in the expert model setting

$$\mathcal{X}_s = \begin{pmatrix} \boldsymbol{X}'_{1,s} \\ \boldsymbol{X}'_{2,s} \\ \vdots \\ \boldsymbol{X}'_{D,s} \end{pmatrix} = \begin{pmatrix} 1 & Y_{0,s} & Y_{-1,s} & Y_{-6,s} & \mathrm{DoW}_1^1 & \mathrm{DoW}_1^6 & \mathrm{DoW}_1^7 \\ 1 & Y_{1,s} & Y_{0,s} & Y_{-5,s} & \mathrm{DoW}_2^1 & \mathrm{DoW}_2^6 & \mathrm{DoW}_2^7 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & Y_{D-1,s} & Y_{D-2,s} & Y_{D-7,s} & \mathrm{DoW}_D^1 & \mathrm{DoW}_D^6 & \mathrm{DoW}_D^7 \end{pmatrix}. \tag{10}$$

We will use the regression matrix (10) to estimate $\boldsymbol{\beta}_s$. The standard way to estimate $\boldsymbol{\beta}_s$ is the least squares approach, also known as ordinary least squares (OLS). Therefore, we do a small general recap of OLS in the next paragraph.

Consider the general linear model

$$Y_d = \boldsymbol{X}'_d\boldsymbol{\beta} + \varepsilon_d \tag{11}$$

with $\boldsymbol{X}_d$ and $\boldsymbol{\beta}$ p-dimensional regression and parameter vectors. For any D-dimensional response vector $\mathcal{Y}$ and $D \times p$-dimensional regression matrix the least squares estimator is defined by

$$\begin{aligned} \widehat{\boldsymbol{\beta}}^{\mathrm{ls}}(\mathcal{X}, \mathcal{Y}) &= \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\arg\min} \parallel \underbrace{\mathcal{Y} - \mathcal{X}\boldsymbol{\beta}}_{\text{error vector}} \parallel_2^2 = \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\arg\min} (\mathcal{Y} - \mathcal{X}\boldsymbol{\beta})'(\mathcal{Y} - \mathcal{X}\boldsymbol{\beta}) \\ &= \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\arg\min} \sum_{d=1}^{D} (\underbrace{Y_d - \boldsymbol{X}'_d\boldsymbol{\beta}}_{\text{error at day } d})^2. \end{aligned} \tag{12}$$

The minimization problem (12) has an explicit solution that is given by

$$\widehat{\boldsymbol{\beta}}^{\mathrm{ls}}(\mathcal{X}, \mathcal{Y}) = (\mathcal{X}'\mathcal{X})^{-1}\mathcal{X}'\mathcal{Y}. \tag{13}$$

It can be derived by noting that for

$$g(\boldsymbol{\beta}) = (\mathcal{Y} - \mathcal{X}\boldsymbol{\beta})'(\mathcal{Y} - \mathcal{X}\boldsymbol{\beta}) = \mathcal{Y}'\mathcal{Y} - 2\boldsymbol{\beta}\mathcal{X}'\mathcal{Y} + \boldsymbol{\beta}'\mathcal{X}'\mathcal{X}\boldsymbol{\beta}$$

it holds

$$\frac{\partial g}{\partial \boldsymbol{\beta}}(\boldsymbol{\beta}) = -2\mathcal{X}'\mathcal{Y} + 2\mathcal{X}'\mathcal{X}\boldsymbol{\beta}.$$

Solving $\frac{\partial g}{\partial \boldsymbol{\beta}}(\widehat{\boldsymbol{\beta}}^{\mathrm{ls}}) = 0$ yields immediately (13). Note that the OLS estimator $\widehat{\boldsymbol{\beta}}^{\mathrm{ls}}$ always depends on the considered regression matrix $\mathcal{X}$ and response vector $\mathcal{Y}$. However, it the setting is clear and the dependence of $\mathcal{X}$ and $\mathcal{Y}$ is not explicitly noted, or gets loosely hidden into subindexes. So, we will do for our setting with regression matrix $\mathcal{X}_s$ and response vector $\mathcal{Y}_s$ for all $s \in \mathbb{S}$. Here we define $\widehat{\boldsymbol{\beta}}_s^{\mathrm{ls}}$ by

$$\widehat{\boldsymbol{\beta}}_s^{\mathrm{ls}} = \widehat{\boldsymbol{\beta}}^{\mathrm{ls}}(\mathcal{X}_s, \mathcal{Y}_s) \tag{14}$$

to simplify notations.

When applying software (e.g `R` or `python`) we usually will not compute parameter estimators using the explicit formula (13), but using a built-in function as it is easier and faster. The expert model (7) can be visualized in a network (which embeds it into neural network theory). This visualisation of the dependency structure or information flow is given in Figure 8. There Lag1, Lag2, Lag7, Mon, Sat and Sun represent the three autoregressive and the three weekday dummy regressors with parameters $\beta_1, \ldots, \beta_6$. The intercept β_0 is represented by 1.

Before we deal with the interpretation of estimated expert models, we introduce a slightly simpler version of (7). This contains only the day-of-the-week effects, but not the autoregressive effects. We denote is as day-of-the-week model (**DoW**) model is given by

$$Y_{d,s} = \beta_{s,0} + \beta_{s,1}\mathrm{DoW}_d^1 + \beta_{s,2}\mathrm{DoW}_d^6 + \beta_{s,3}\mathrm{DoW}_d^7 + \varepsilon_{d,s} = \boldsymbol{X}'_{d,s}\boldsymbol{\beta}_s + \varepsilon_{d,s} \tag{15}$$

with $\boldsymbol{\beta}_s = (\beta_{s,0}, \beta_{s,1}, \ldots, \beta_{s,3})'$ and $\boldsymbol{X}_{d,s} = (1, \mathrm{DoW}_d^1, \mathrm{DoW}_d^6, \mathrm{DoW}_d^7)'$. Here, the OLS estimator $\widehat{\boldsymbol{\beta}}_s^{\mathrm{ls}} = \widehat{\boldsymbol{\beta}}^{\mathrm{ls}}(\mathcal{X}_s, \mathcal{Y}_s)$ is defined through

$$\mathcal{Y}_s = \begin{pmatrix} Y_{1,s} \\ Y_{2,s} \\ \vdots \\ Y_{D,s} \end{pmatrix} \quad \text{and} \quad \mathcal{X}_s = \begin{pmatrix} \boldsymbol{X}'_{1,s} \\ \boldsymbol{X}'_{2,s} \\ \vdots \\ \boldsymbol{X}'_{D,s} \end{pmatrix} = \begin{pmatrix} 1 & \mathrm{DoW}_1^1 & \mathrm{DoW}_1^6 & \mathrm{DoW}_1^7 \\ 1 & \mathrm{DoW}_2^1 & \mathrm{DoW}_2^6 & \mathrm{DoW}_2^7 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \mathrm{DoW}_D^1 & \mathrm{DoW}_D^6 & \mathrm{DoW}_D^7 \end{pmatrix}. \tag{16}$$

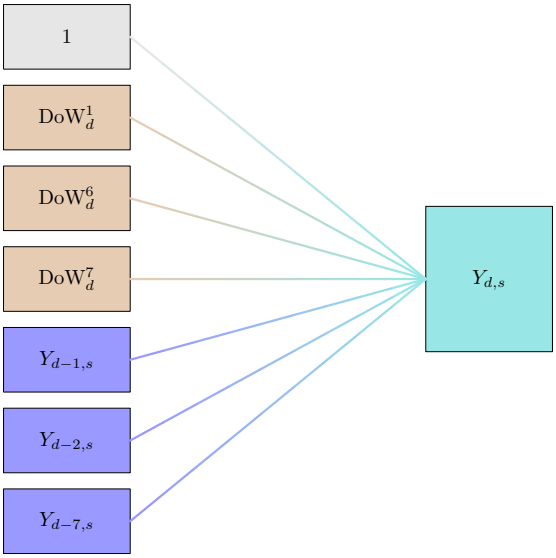
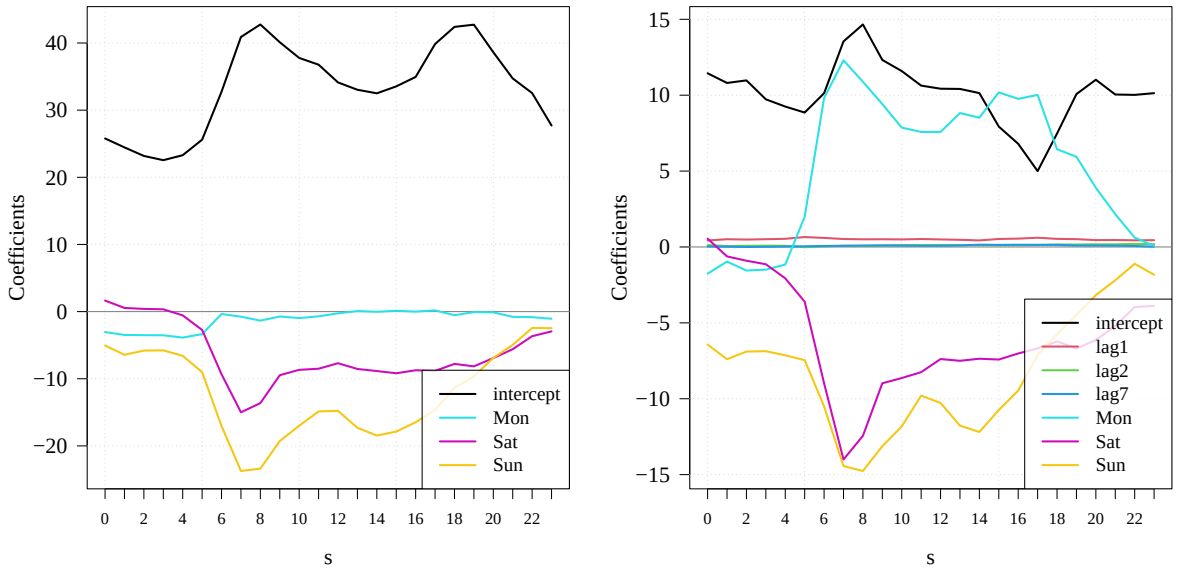


Figure 8: Network visualisation of the expert model (7).

Note that even though notationally $\mathcal{X}_s$ as defined in (16) contains the dependence on the period of the day s it actually does not depend on s in contrast to $\mathcal{X}_s$ of equation (10) for the expert model.

3.3 Interpretation of the estimated models

Now, we want to interpret the estimation results of the **expert** model (7). As we will notice that interpretation is sometimes not that obvious, we will also study the **DoW** model (15). Figure (9) shows the OLS-estimated parameters of both models of the full data set.



(a) Parameters of **DoW** model (15)

(b) Parameters of **expert** model (7)

Figure 9: Estimated parameter vector $\widehat{\boldsymbol{\beta}}_s^{\text{ls}}$ of selected models for full time range.

We first study the **DoW** model results, see (9(a)). The intercept ($= \beta_{s,0}$) provides the expected price level on all days, expect Mondays, Saturdays and Sundays. It has a well known, clearly positive, pattern. The prices are lower during night than during day hours which can be mainly explained by the lower demand during night hours. Further, we see indications of lower prices during the noon hours compared to the morning and evening hours. A plausible explanation is the increased solar power production during noon hours which reduces the price

due to the merit order effect. The other regressors for Monday, Saturday and Sunday explain the expected deviation from the default behaviour. Thus, we see that especially on Saturdays and Sundays the electricity prices are clearly reduced, especially during hours with high prices. The effect of the Monday parameter is relatively small. Except for the night hours which exhibit some negative effect it is relatively close to zero.

For the **expert** model the results are somewhat different, see (9(b)) (and remember the different scale of the y-axis compared to (9(a))). First, we observe that the intercept is clearly positive, and the Saturday and Sunday parameter are clearly negative, similarly to (9(a)). However, Monday parameter is mainly clearly positive, which is somewhat puzzling. Additionally, we see the autoregressive parameters (lag1, lag2 and lag7) are relatively close to zero, which is puzzling as well. Those aspects might lead to misleading interpretation because of two reasons: First, there is the complicated interaction between the autoregressive terms and weekday dummies, which we will study in more detail in Section 7. At this stage we want to remind, that we can not interpret the parameter estimates as means (or expected values) as we could do it for the DoW model (15). Second, we can not compare the absolute values of the parameters to deduce parameter importance as they are all at a different scale. To make them interpretable and compare the relevance we have to scale the regression setting.

Usually all regressors in $\boldsymbol{X}_{d,s}$ and $Y_{d,s}$ are scaled so that they have zero mean and variance of 1. Therefore, $\boldsymbol{\mu}_{\boldsymbol{X},s}$, $\mu_{Y,s}$ and $\boldsymbol{\sigma}_{\boldsymbol{X},s}$, $\sigma_{Y,s}$ denote the mean and standard deviations of $\boldsymbol{X}_{d,s}$ and $Y_{d,s}$. We introduce

$$\widetilde{Y}_{d,s} = \frac{Y_{d,s} - \mu_Y}{\sigma_Y} \text{ and } \widetilde{\boldsymbol{X}}_{d,s} = \text{Diag}(\boldsymbol{\sigma}_{\boldsymbol{X}})^{-1}(\boldsymbol{X}_{d,s}^{\text{r0}} - \boldsymbol{\mu}_{\boldsymbol{X}}) = \left(\frac{X_{d,s,1} - \mu_{\boldsymbol{X},1}}{\sigma_{\boldsymbol{X},1}}, \ldots, \frac{X_{d,s,p} - \mu_{\boldsymbol{X},p}}{\sigma_{\boldsymbol{X},p}}\right). \tag{17}$$

with $\boldsymbol{X}_{d,s}^{\text{r0}} = (X_{d,s,1}, \ldots, X_{d,s,p})$ as parameter vector without the zero. The corresponding scaled linear model of (8) is

$$\widetilde{Y}_{d,s} = \widetilde{\boldsymbol{X}}_{d,s}'\widetilde{\boldsymbol{\beta}}_s + \widetilde{\varepsilon}_{d,s} \tag{18}$$

with $\widetilde{\boldsymbol{\beta}}_s$ as scaled parameter vectors. By definition (17) this is equivalent to

$$\begin{aligned} Y_{d,s} &= \mu_Y + \sigma_Y \left(\left(\text{Diag}(\boldsymbol{\sigma}_{\boldsymbol{X}})^{-1}(\boldsymbol{X}_{d,s}^{\text{r0}} - \boldsymbol{\mu}_{\boldsymbol{X}})\right)' \widetilde{\boldsymbol{\beta}}_s + \widetilde{\varepsilon}_{d,s} \right) && (19) \\ &= \underbrace{\mu_Y - \sigma_Y \boldsymbol{\mu}_{\boldsymbol{X}}' \text{Diag}(\boldsymbol{\sigma}_{\boldsymbol{X}})^{-1}\widetilde{\boldsymbol{\beta}}_s}_{=\beta_{s,0}} + (\boldsymbol{X}_{d,s}^{\text{r0}})' \underbrace{\sigma_Y \text{Diag}(\boldsymbol{\sigma}_{\boldsymbol{X}})^{-1}\widetilde{\boldsymbol{\beta}}_s}_{=(\beta_{s,1},\ldots,\beta_{s,p})} + \sigma_Y \widetilde{\varepsilon}_{d,s} && (20) \\ &= \boldsymbol{X}_{d,s}' \underbrace{\left(\mu_Y - \sigma_Y \boldsymbol{\mu}_{\boldsymbol{X}}' \text{Diag}(\boldsymbol{\sigma}_{\boldsymbol{X}})^{-1}\widetilde{\boldsymbol{\beta}}_s, \quad \sigma_Y \text{Diag}(\boldsymbol{\sigma}_{\boldsymbol{X}})^{-1}\widetilde{\boldsymbol{\beta}}_s\right)}_{=\boldsymbol{\beta}_s} + \underbrace{\sigma_Y \widetilde{\varepsilon}_{d,s}}_{=\varepsilon_{d,s}}. && (21) \end{aligned}$$

This can be used for rescaling, i.e. computing $\boldsymbol{\beta}_s$ given $\widetilde{\boldsymbol{\beta}}_s$.

The scaled linear model is estimated in the same way. Denote $\widetilde{\mathcal{X}}_s$ and $\widetilde{\mathcal{Y}}_s$ the scaled regression matrix without the intercept and response vector, such that for each column the sample mean is zero and the sample standard deviation is one. Formally, this is

$$\widetilde{\mathcal{Y}}_s = \frac{\mathcal{Y}_s - m_{\mathcal{Y}}}{s_{\mathcal{Y}}} \text{ and } \widetilde{\mathcal{X}}_s = \left(\frac{\mathcal{X}_{s,1} - \boldsymbol{m}_{\mathcal{X},1}}{\boldsymbol{s}_{\mathcal{X},1}}, \ldots, \frac{\mathcal{X}_{s,p} - \boldsymbol{m}_{\mathcal{X},p}}{\boldsymbol{s}_{\mathcal{X},p}}\right)$$

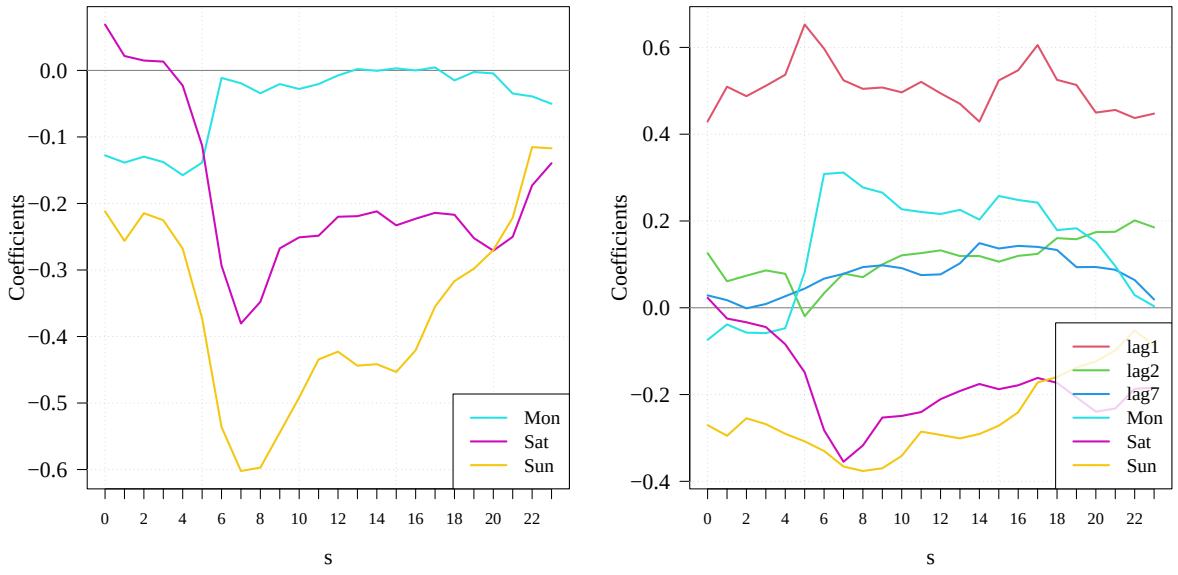
where $\boldsymbol{m}_{\mathcal{X}}$, $m_{\mathcal{Y}}$ and $\boldsymbol{s}_{\mathcal{X}}$, $s_{\mathcal{Y}}$ denote the sample mean and sample standard deviations (which are suitable estimators for $\boldsymbol{\mu}_{\boldsymbol{X},s}$, $\mu_{Y,s}$ and $\boldsymbol{\sigma}_{\boldsymbol{X},s}$, $\sigma_{Y,s}$).

Then the OLS estimator $\widehat{\widetilde{\boldsymbol{\beta}}}_s^{\text{ls}}$ of the scaled $\widetilde{\boldsymbol{\beta}}_s$ parameter vector is defined by

$$\widehat{\widetilde{\boldsymbol{\beta}}}_s^{\text{ls}} = \widehat{\boldsymbol{\beta}}^{\text{ls}}(\widetilde{\mathcal{X}}_s, \widetilde{\mathcal{Y}}_s) = \underset{\boldsymbol{\beta} \in \mathbb{R}^{ps}}{\arg\min} \|\widetilde{\mathcal{Y}}_s - \widetilde{\mathcal{X}}_s\boldsymbol{\beta}\|_2^2. \tag{22}$$

We easily receive $\widehat{\widetilde{\boldsymbol{\beta}}}_s^{\text{ls}}$ from $\widehat{\boldsymbol{\beta}}_s^{\text{ls}}$ using rescaling as for $\widetilde{\boldsymbol{\beta}}_s$. However, $\widehat{\widetilde{\boldsymbol{\beta}}}_s^{\text{ls}}$ as estimator for $\widetilde{\boldsymbol{\beta}}_s$ allows for a relative interpretation across the regressors.

In Figure (10), we see the estimated parameters of the scaled DoW and expert model. From (10(b)) we see that in absolute terms the lag1 parameter (β_1) is the largest. Thus, the most relevant one from this point of view. Furthermore, we observe that all the other parameter are of relatively similar magnitude.



(a) Parameters of scaled **DoW** model (15)

(b) Parameters of scaled **expert** model (7)

Figure 10: Estimated parameter vector $\widehat{\widehat{\boldsymbol{\beta}}}_s^{\text{ls}}$ of selected scaled models for full time range.

3.4 On multivariate models

The expert model (7) looks like a relatively simple model with just 7 parameters. However, we apply the same model for all S products (usually 24 hours). Thus, we have in total $7S$ parameters, which is 168 for $S = 24$. In general mathematical terms: model (8) has $p_s + 1$ parameters for each $s \in \mathbb{S}$ and our full model has $S + \sum_{s\in\mathbb{S}} p_s = S + p$ parameters.

However, it might be plausible to assume that there is no significant difference between the $p_s + 1$ parameters across the S hours. Thus, we might assume a model with the structure and size within p-dimensional regressor vector $\boldsymbol{X}_{d,s}$ (esp. $p_0 = \ldots = p_{S-1}$)

$$Y_{d,s} = \boldsymbol{X}_{d,s}'\boldsymbol{\beta} + \varepsilon_{d,s} \tag{23}$$

where the p-dimensional parameter vector $\boldsymbol{\beta}$ does not depend on s. This model has only p parameters. The corresponding version of the expert model (7) is

$$Y_{d,s} = \beta_0 + \beta_1 Y_{d-1,s} + \beta_2 Y_{d-2,s} + \beta_3 Y_{d-7,s} + \beta_4 \text{DoW}_d^1 + \beta_5 \text{DoW}_d^6 + \beta_6 \text{DoW}_d^7 + \varepsilon_{d,s} \tag{24}$$

with $p = 7$ total parameters.

It allows for a simple solution of the model by OLS. Therefore, we define the DS-dimensional response vector and $DS \times p$-dimensional regression matrix by

$$\boldsymbol{\mathcal{Y}} = \begin{pmatrix} \mathcal{Y}_0 \\ \vdots \\ \mathcal{Y}_{S-1} \end{pmatrix} \text{ and } \boldsymbol{\mathcal{X}} = \begin{pmatrix} \mathcal{X}_0 \\ \vdots \\ \mathcal{X}_{S-1} \end{pmatrix} \tag{25}$$

to get the OLS solution

$$\widehat{\boldsymbol{\beta}}^{\text{ls}}(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}) = (\boldsymbol{\mathcal{X}}'\boldsymbol{\mathcal{X}})^{-1}\boldsymbol{\mathcal{X}}'\boldsymbol{\mathcal{Y}}.$$

The estimator $\widehat{\boldsymbol{\beta}}^{\text{ls}}(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}})$ may be regarded a very special multivariate estimator. An important feature is that a change in the regression matrix $\mathcal{X}_s$ will influence the estimated parameters for other hours than s as well. In contrast, the standard estimator $\widehat{\boldsymbol{\beta}}_s^{\text{ls}}$ does not have this property.

Now, we want to consider a setting that is to some extent the opposite of model (23). We can consider a model where the parameter vector may vary across the hours s but the regression vector $\boldsymbol{X}_d$ remains constant across $s \in \mathbb{S}$, i.e. it holds $\boldsymbol{X}_d = \boldsymbol{X}_{d,0} = \ldots = \boldsymbol{X}_{d,S-1}$. That is

$$Y_{d,s} = \boldsymbol{X}_d'\boldsymbol{\beta}_s + \varepsilon_{d,s}. \tag{26}$$

where $\boldsymbol{X}_d$ is a p-dimensional regression vector. Such a model is often referred as multivariate regression model. The DoW-model (15) with regression matrices (16) is an example in the electricity price forecasting context.

Obviously, we can estimate model (26) by (13), so i.e. $\widehat{\boldsymbol{\beta}}_s^{\text{ls}} = \widehat{\boldsymbol{\beta}}^{\text{ls}}(\mathcal{X}, \mathcal{Y}_s)$ where

$$\mathcal{X} = \begin{pmatrix} \boldsymbol{X}_1 \\ \vdots \\ \boldsymbol{X}_D \end{pmatrix}. \tag{27}$$

However, the setting (26) there exists the multivariate least square estimator that can be used as well. This is

$$\widehat{\boldsymbol{\beta}}^{\text{mls}} = \underset{\boldsymbol{\beta} \in \mathbb{R}^{p \times S}}{\arg\min} \sum_{d=1}^{D} \|\boldsymbol{Y}_d - \boldsymbol{X}_d' \boldsymbol{\beta}\|_F^2. \tag{28}$$

which minimizes the squared error for all $s \in \mathbb{S}$ jointly. Here $\|\cdot\|_F$ is the Frobenius norm (or Hilbert-Schmidt norm) of a matrix, i.e. $\|\boldsymbol{A}\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{i,j}^2}$. Moreover note, that $\widehat{\boldsymbol{\beta}}^{\text{mls}}$ is defined as a matrix, so

$$\widehat{\boldsymbol{\beta}}^{\text{mls}} = \left(\widehat{\boldsymbol{\beta}}_0^{\text{mls}}, \ldots, \widehat{\boldsymbol{\beta}}_{S-1}^{\text{mls}}\right).$$

Interestingly, the estimator itself is equivalent to the standard OLS estimator (13), in the sense that it holds

$$\widehat{\boldsymbol{\beta}}_s^{\text{ls}} = \widehat{\boldsymbol{\beta}}_s^{\text{mls}}$$

for all $s \in \mathbb{S}$. Therefore, as long es we stick to standard OLS the MLS approach is without significant relevance.

The visualization of (8) showed the standard expert model. Even though it is correct, it is important to highlight that S models are required. However, there is also a visualization which illustrates the full multivariate model using S outputs. This corresponds to the visualization illustrated in Figure (11). Note that those Figures usually only illustrate the model itself parameter restrictions and estimation methods can not be seen in those illustrations. An advantage of this illlustration is that we can easily see that we see directly that the same dummies DoW_d^k are used to explain $Y_{d,s}$ for all s whereas the price dependency varies with s.

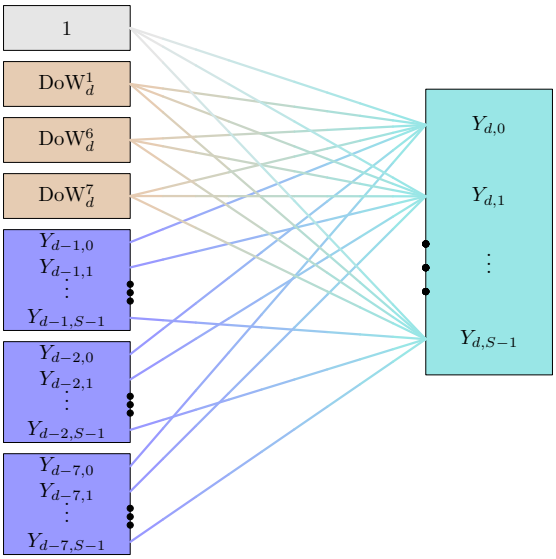


Figure 11: Network visualisation of the multivariate expert model which is equivalent (7) for all $s \in S$.

Now we have a look at another OLS estimator with multivariate characteristics. We can also

define

$$\boldsymbol{\mathcal{X}}_{\boldsymbol{O}} = \begin{pmatrix} \mathcal{X}_0 & \boldsymbol{O}_{D,p_1} & \cdots & \boldsymbol{O}_{D,p_{S-1}} \\ \boldsymbol{O}_{D,p_0} & \mathcal{X}_1 & \cdots & \boldsymbol{O}_{D,p_{S-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{O}_{D,p_0} & \boldsymbol{O}_{D,p_1} & \cdots & \mathcal{X}_{S-1} \end{pmatrix} \tag{29}$$

where $\boldsymbol{O}_{d_1,d_2}$ is a $d_1 \times d_2$-dimensional null matrix. Then

$$\widehat{\boldsymbol{\beta}}^{\boldsymbol{O}} = (\boldsymbol{\mathcal{X}}'_{\boldsymbol{O}}\boldsymbol{\mathcal{X}}_{\boldsymbol{O}})^{-1}\boldsymbol{\mathcal{X}}'_{\boldsymbol{O}}\boldsymbol{\mathcal{Y}} \tag{30}$$

corresponds to the standard OLS solution (13) for the S individual models (8), i.e. it holds

$$\widehat{\boldsymbol{\beta}}^{\boldsymbol{O}} = \begin{pmatrix} \widehat{\boldsymbol{\beta}}_0^{\text{ls}} \\ \vdots \\ \widehat{\boldsymbol{\beta}}_{S-1}^{\text{ls}} \end{pmatrix}.$$

Finally, we want to look at another multivariate model notation. Anyway, it is also possible to define a model where some parameters are the same for all $s \in \mathbb{S}$ as in (23), and some parameters are different for each $s \in \mathbb{S}$ as in (8). Such a model can be specified

$$Y_{d,s} = \breve{\boldsymbol{X}}'_{d,s}\boldsymbol{\beta} + \boldsymbol{X}'_{d,s}\boldsymbol{\beta}_s + \varepsilon_d = \begin{pmatrix} \breve{\boldsymbol{X}}_{d,s} \\ \boldsymbol{X}_{d,s} \end{pmatrix}' \begin{pmatrix} \boldsymbol{\beta} \\ \boldsymbol{\beta}_s \end{pmatrix} + \varepsilon_d. \tag{31}$$

with $\breve{\boldsymbol{X}}_{d,s}$ as $\breve{p}$-dimensional regressor vector for $\breve{p}$ parameters that are constant for all $s \in \mathbb{S}$ (still $\breve{\boldsymbol{X}}_{d,s}$ may depend on s). Further, $\boldsymbol{X}_{d,s}$ is a p_s-dimensional regressor vector where the corresponding parameters that vary across $\mathbb{S}$. In (31), the $\breve{\boldsymbol{X}}_{d,s}$ corresponds to $\boldsymbol{X}_{d,s}$ in the multivariate model (23) (with constant parameters) and $\boldsymbol{X}_{d,s}$ corresponds to the regression matrix $\boldsymbol{X}_{d,s}$ in (8). Hence, the model (31) nests both (8) and (23). It has in total $\breve{p} + \sum_{s\in\mathbb{S}} p_s$ parameters.

An example could the expert model where the autoregressive parameters are constant:

$$Y_{d,s} = \beta_{s,0} + \beta_1 Y_{d-1,s} + \beta_2 Y_{d-2,s} + \beta_3 Y_{d-7,s} + \beta_{s,4}\text{DoW}_d^1 + \beta_{s,5}\text{DoW}_d^6 + \beta_{s,6}\text{DoW}_d^7 + \varepsilon_{d,s} \tag{32}$$

Here, we have for (32) in the notation of (31) that $\breve{\boldsymbol{X}}_{d,s} = (Y_{d-1,s}, Y_{d-2,s}, Y_{d-7,s})'$ and $\boldsymbol{X}_{d,s} = (1, \text{DoW}_d^1, \text{DoW}_d^2, \text{DoW}_d^7)'$. Model (32) has now $3 + 4S$ parameters, which is 99 for $S = 24$.

For estimating (31) we can define

$$\mathbb{X} = \begin{pmatrix} \breve{\mathcal{X}}_0 & \mathcal{X}_0 & \boldsymbol{O}_{D,p_1} & \cdots & \boldsymbol{O}_{D,p_{S-1}} \\ \breve{\mathcal{X}}_1 & \boldsymbol{O}_{D,p_0} & \mathcal{X}_1 & \cdots & \boldsymbol{O}_{D,p_{S-1}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \breve{\mathcal{X}}_{S-1} & \boldsymbol{O}_{D,p_0} & \boldsymbol{O}_{D,p_1} & \cdots & \mathcal{X}_{S-1} \end{pmatrix} \tag{33}$$

where $\breve{\mathcal{X}}_s = (\boldsymbol{X}'_{1,s}, \ldots, \boldsymbol{X}'_{D,s})'$ and $\mathcal{X}_s = (\boldsymbol{X}'_{1,s}, \ldots, \boldsymbol{X}'_{D,s})'$. The OLS estimator is given by

$$\widehat{\boldsymbol{\beta}}^{\mathbb{X}} = \widehat{\boldsymbol{\beta}}^{\text{ls}}(\mathbb{X}, \boldsymbol{\mathcal{Y}}).$$

Note that (33) is a matrix which contains many zeros. A matrix which contains mainly zeros is usually called sparse matrix. For sparse matrices there are often fast computation alternatives available. We will come back to this later on.

Finally we want to mention a specific special case of (31) which is without clear use at the moment, but will be use in Section 10. Particularly, we mention the special case when $\breve{\mathcal{X}}_s = \mathcal{X}_s$ with $p_0 = \ldots = p_{S-1}$. Then the corresponding model is singular with regression matrix:

$$\mathbb{X} = \begin{pmatrix} \mathcal{X}_0 & \mathcal{X}_0 & \boldsymbol{O}_{D,p_1} & \cdots & \boldsymbol{O}_{D,p_{S-1}} \\ \mathcal{X}_1 & \boldsymbol{O}_{D,p_0} & \mathcal{X}_1 & \cdots & \boldsymbol{O}_{D,p_{S-1}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathcal{X}_{S-1} & \boldsymbol{O}_{D,p_0} & \boldsymbol{O}_{D,p_1} & \cdots & \mathcal{X}_{S-1} \end{pmatrix} \tag{34}$$

3.5 Tasks

Hints and the corresponding programming part are provided in Section R.1 in `R` and in Section py.1 in `python`.

1. Read the provided EXAA data and apply the clock-change adjustment. Create a time series plot of all prices and a graph for the latest product that is traded.

2. For which parameters model (7) is equal to (1) or (2)?

3. Write a function that models

$$Y_{d,s} = \begin{cases} Y_{d-7,s} + \varepsilon_{d,s} & , d \text{ is Saturday or Sunday} \\ \frac{1}{2}(Y_{d-1,s} + Y_{d-7,s}) + \varepsilon_{d,s} & , d \text{ is Friday} \\ \frac{1}{2}(Y_{d-3,s} + Y_{d-7,s}) + \varepsilon_{d,s} & , d \text{ is Monday} \\ Y_{d-1,s} + \varepsilon_{d,s} & , \text{otherwise} \end{cases} . \tag{35}$$

4. Write a function that generates day-ahead forecasts for the model

$$Y_{d,s} = \beta_{s,0} + \sum_{k \in \mathbb{L}} \beta_{s,\mathbb{L},k} Y_{d-k,s} + \sum_{k \in \mathbb{W}} \beta_{s,\mathbb{W},k} \mathrm{DoW}_d^k + \varepsilon_{d,s} \tag{36}$$

   with lag index set $\mathbb{L}$ and day-of-the-week index set $\mathbb{W}$. The function shall take the two (additional) input vectors, `expert.lag` and `expert_wd` which specify the index sets $\mathbb{L}$ and $\mathbb{W}$ of model (36). Give both the default value of the expert model (7) which we have considered so far.

5. Consider the special case

$$Y_{d,s} = \beta_0 + \sum_{k \in \mathbb{L}} \beta_{s,\mathbb{L},k} Y_{d-k,s} + \sum_{k \in \mathbb{W}} \beta_{\mathbb{W},k} \mathrm{DoW}_d^k + \varepsilon_{d,s} \tag{37}$$

   of model (36). Write a function that forecasts model (37). How many total parameters has model (37) for $S = 96$, $\mathbb{L} = \{1, 2, 6, 7, 8, 14\}$ and $\mathbb{W} = \{1, 2, 5, 6, 7\}$?

6. Estimate the model (37) with $\mathbb{L} = \{1, 2, 6, 7\}$ and $\mathbb{W} = \{1, 2, 5, 6, 7\}$ for the German EPEX data by considering the corresponding scaled model version. Visualize the parameters, similarly as in Figure 10.

4 Evaluation

In this section we want to study in relevant depth issues around the evaluation of electricity price forecasting models. As mentioned electricity price models are usually used for two purposes: i) identifying causal relationships in the data, and ii) providing accurate forecasts.

For the former it makes sense causal econometric tools will be suitable to indentify models. However, the standard statistical identification problem between causality and dependency (e.g. correlation as linear dependence) plays an important rule here as well. But as we have a couple of well established fundamental electricity price models (like the supply stack model) the need for identifying causality is usually less relevant than in many other econometric models. Also here we have some relevant questions that might be answered where statistical tools for model identification and significance analyses help. However, we do not go into details on some relevant questions, on model diagnostics, significance test and multiple comparison.

For the ii) purpose of electricity price forecasting. We only bother about the accuracy of the forecasts but not about fundamental relationships. Therefore we have to consider the design of forecasting studies and have to discuss evaluation metrics to compare the accuaracy of forecasts of different models.

In the next subsection we cover relevant aspects of model selection and parameter identification in the information criterion framework. In the next section we discuss the forecasting study design and relevant evaluation metrics.

4.1 Information criteria and cross-validation

Information criteria (IC) are tools for decision support to decide between a preferable models by evaluating the goodness-of-fit compared to its complexity given the data. In statistics information criteria are usually defined with negative orientation, thus the smaller an IC the better corresponding model with respect to the considered information criterion.

In model selection literature, there are two popular information criteria which were introduced more than 50 years ago. These are the Akaike information criterion (AIC) and the Bayesian information criterion (BIC) (also known as Schwarz information criterion, SBC). Usually, information criteria like AIC or BIC are evaluated on the likelihood of a model given the data while penalising the usage of parameter. Thus, if two models yield have the same explanatory power we will always prefer the smaller model as it faces lower estimation risk. However, instead of using likelihood based information criteria we can also use loss-based criteria, esp. least square based alternatives.

The AIC and BIC are all special cases of the so called generalised information criterion GIC[7] which is defined as

$$\mathrm{GIC} = \log(\mathrm{RSS}) + K\kappa(K, D)/D \tag{38}$$

where D is the sample size, K the number of non-zero parameters (or active parameters, or degrees of freedom) and $\kappa(K, D)$ is an information penalty function which determines a specific information criterion. The larger $\kappa(K, D)$, the more conservative the information criterion, so the less chance to overfit a certain model, but also the higher the chance to miss relevant effects. For $\kappa(K, D)$ we have some popular choices that were used in electricity price modelling

- $\kappa(K, D) = 2$ for Akaike information criterion (AIC),
- $\kappa(K, D) = 2 + \frac{2(K+1)}{D-K-1}$ for corrected Akaike information criterion (AICc),
- $\kappa(K, D) = 2\log(\log(D))$ for Hannan-Quinn information criterion (HQC),
- $\kappa(K, D) = \log(D)$ for Bayesian information criterion (BIC) or Schwarz criterion (SBC).

Out of the four, the BIC is the most conservative criterion, and AIC is the most aggressive one. Sometimes HQC is seen as a useful compromise.

It is proven that in a linear model Markov-framework the BIC performs consistent model selection if the sample size goes to infinity and some linear model regularity assumptions are

[7] Strictly, this definition contains some additional constants which are not relevant for the model selection purpose.

satisfied. Thus, the true model can be identified among a pool of different models which include the true model. In contrast, AIC tends to overestimate the true amount of model parameters, and performs inconsisten model selection. Still, the correct identification of relevant parameters is only important if we are interested in causal relationships in the electricity price model, e.g. for parameter interpretation. If we are only interested in forecasting, this is primarily of less interest. Even though the AIC tend to overestimate it has certain guarantees concerning the prediction error which is minimized if certain model assumptions are satisfied. Thus, in simple words: theoretically BIC is preferable for model identification and AIC for forecasting. However, all this theory is only valid if the model and the data is well-behaved. However, as this is often not satisfied alternatives like the HQC are used as well.

We observe that the penalty of the AIC does not depend on amount of observations D while the other ones does. For the HQC and BIC the penalty term growth to infinity in D, even log growth rates of log or even log(log) are very slow. It is interesting to note that the HQC is just at the edge of performing consistent model selection. So we require at least a log(log) growth rate in D to achieve consistent model selection, which is also a suitable argument for the application of HQC in applications.

The AICc is a corrected AIC version, it is easy to see that for $D \to \infty$ it coincides with the AIC. However, the AICc performs some 2nd order correction of the AIC which preserves theoretically better properties for small sample sizes. Note, that in standard regression settings both the AIC and AICc are equaivalent to leave-one-out cross-validation. Leave-one-out cross-validation is a special case of k-fold cross-validation. It is popular in statistics and machine learning, but applying it to correlated data settings, esp. time series settings, can be dangerous.

In k-fold cross-validation, we split the sample into k random parts, the so-called folds. Then, we estimate the model on all parts except the one which we use for performance evaluation. This procedure is repeated across all selected parts. The model which minimises the loss function across all parts is the optimal model. We will take about the specification of the loss in the next section. Anyway, we want to mention that cross-validation involves sampling. So the results are not reproducible unless a (sampling) seed is set.

If the folds are equally sized, we call the procedure k-fold cross-validation. Figure (12) illustrates the procedure for a 4-fold cross-validation.

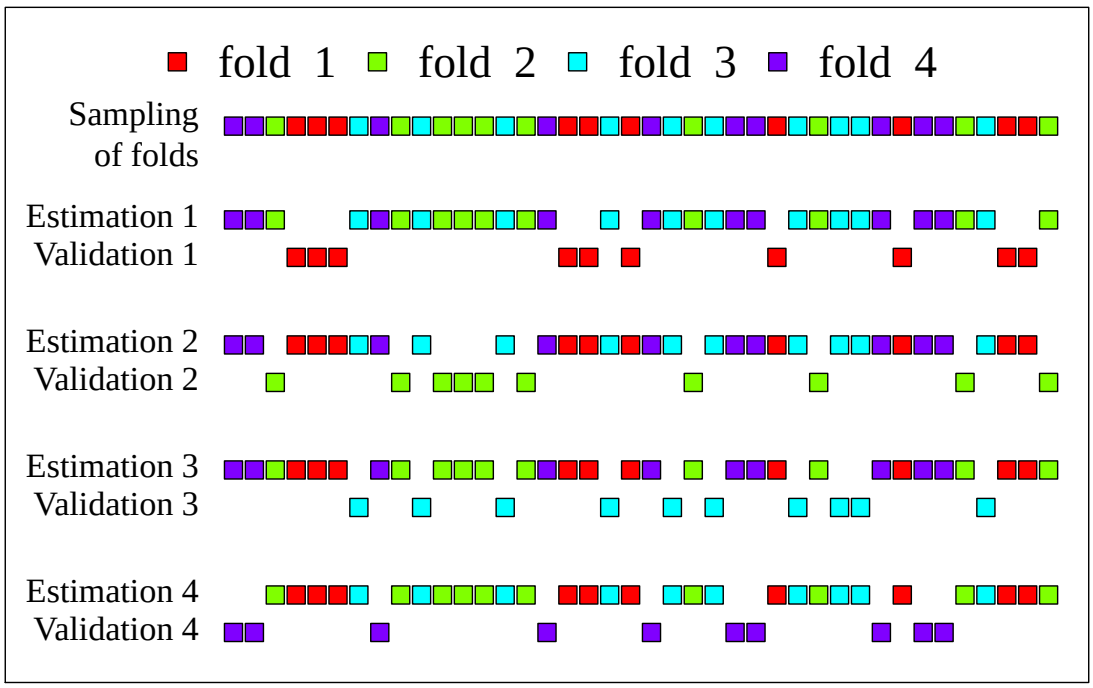


Figure 12: Illustration of k-fold cross-validation for $k = 4$ folds and in-sample size of $D = 40$

k-fold cross-validation mimics to some extent some out-of-sample behaviour of the data. However, the computational costs usually increase substantially. A k-fold cross-validation typically requires a $(k-1)$-times larger computational effort. For the special case $k = D$ we have always exactly one data point in the fold, which is the so called leave-one-out cross-validation.

Remember that leave-one-out cross-validation is asymptotically equivalent to AIC parameter selection which theoretically is preferable in forecasting settings. Thus, we expect k-fold cross-validation based techniques to behave rather similar to AIC than to BIC model selection, especially for large k.

In practice, 10-fold and 5-fold cross-validation is applied most often. Sometimes, practitioners also repeat the cross-validation multiple times to gain more stability concerning the initial sampling process. Then, the optimal tuning parameter is selected via the minimal loss function across all windows. Further, due to correlated and potentially autoregressive terms in the regression matrix $\boldsymbol{X}$, these samples will be correlated as well. Here, block-wise cross-validation usually helps to reduce this bias. A block-wise 4-fold cross-validation procedure with a block size of 7 is visualised in Figure (13).

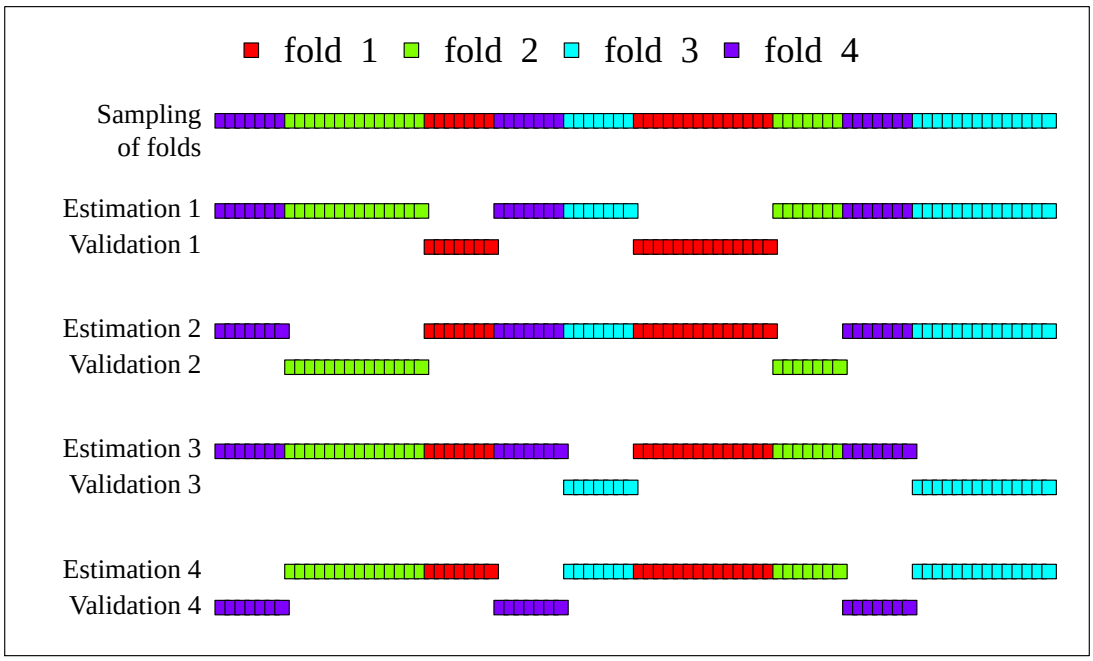


Figure 13: Illustration of k-fold block-wise cross-validation for $k = 4$ folds, block size $b = 7$ and in-sample size of $D = 84$

However, next to k-fold cross-validation and blockwise cross-validation there is a so called time series cross-validation which we explain in the next section as it corresponds to the design of a forecasting study.

4.2 Basic design of forecasting studies

Forecasting studies and the corresponding design is an important element for the proper evaluation of forecasting models. Unfortunately, some forecasters in industry but also in academia do not consider proper evaluation design. This might lead to wrong and potentially dangerous decisions concerning the favorability of forecasting models under consideration. Therefore, we motivate and study the design of forecasting studies in detail.

Suppose have two competing forecasting approaches, such as the naive model (3) and the expert model (7). Now, we want to decide which model is preferable for forecasting. Obviously, we could utilize information criteria do decide for the best model. However, this requires that we are able to compute comparable information criteria for the models. Unfortunately, this requires a nested structure of the models.[8] We also mentioned that plain cross-validation is also not preferable, as the time series structure is not taken into account adequately in the sampling design of cross-validation.

Taking into account the time series structure and information flow in the evaluation setting is the crucial part in a forecasting study. In classic statistics (e.g. statistics for physical experiments) we can easily repeat statistical experiments to gather more data and increase the sample

[8]For example we can not compare information criteria of exponential time series smoothing (ETS) models with AR(p) models.

size to allow for better inference. For time series this is impossible, we can run every forecasting experiment only once, e.g. forecasting the electricity prices of tomorrow. Further, we will apply forecasting models always for predicting the future. Thus, we will be able to evaluate the model only after the realization of the experiment. Due to mentioned problems we have to construct experiments which mimic as close as possible a realistic setting, we will refer those experiment as forecasting study.

The design of a forecasting study may depend on the application. However, a typical design is a window forecasting study. In finance this is often known as back-test study or back-testing study, in machine learning communities this is sometimes referred as time series crossvalidation. It considers N experiments which predict the days $D+i$ for $i = 1, \dots, N$. Precisely, we estimate a model based on days 1 to D and do a forecast for $D+1$. After that, we take the data from days 2 to $D+1$ and compute the forecast for $D+2$, and we proceed until we start the estimation on day N. Then, we estimate based on days N to $D+N-1$ and forecast $D+N$. Thus, we are left with N forecasts, say $\widehat{\boldsymbol{Y}}_{D+1}, \dots, \widehat{\boldsymbol{Y}}_{D+N}$ which gives us directly the forecasting errors $\widehat{\boldsymbol{\varepsilon}}_{D+1}, \dots, \widehat{\boldsymbol{\varepsilon}}_{D+N}$. The data from day $D+1$ to $D+N$ is usually referred as test set/data or out-of-sample data.

Ideally, we run a real forecasting study. Here, we do real time forecasts, and can proceed with the evaluation after realization of the data. The problem is that this real time evaluation takes N days. As N should be reasonable large to allow for suitable inference this usually takes a long time. In contrast, (pseudo) forecasting studies can be conducted immediately and are only restricted by the computation time. Here, we consider only historic data for both forecasting and evaluation. Thus, all the data up to day $D+N$ is known. In the i's experiment we assume that all data till day $D+i-1$ is known and can be used to predict $D+i$.

In literature, there are two main types of (pseudo) window forecasting studies, the expanding window study and the rolling window study (also known as sliding window study). In expanding window studies we utilize always all data (i.e. $1, \dots, D+i-1$) to predict the target day $D+i$. In rolling window studies we consider always only a fixed amount of historic information, so the last D days (i.e. $i, \dots, D+i-1$) to create the forecast.

As explained in [Die15] rolling window studies allow for easier evaluation situations as expanding window studies. The main reason is that in typical situations the accuracy of our estimator increases with increasing window/sample size. Thus, at the beginning of an expanding forecasting study we expect larger forecasting error than at the end. This makes the forecasting experiment in the forecasting study unnecessarily different from each other. In contrast rolling window forecasting studies do not have this disadvantage, for iid data expect similar variance of our estimators. Hence, significance testing is slightly easier.

Also in this script, we consider primarily rolling window forecasting studies. We often fix an in-sample data set size (= calibration window size) to $D = 2 \times 365 = 730$ days. For the application in practice, number of forecasting experiments N should be sufficiently large. From the statistical point of view, N should be chosen as large as possible. However, due to structural changes in the electricity price series, N that contains multiple decades of data is clearly questionable. Still, it is common sense N should be chosen large enough to capture all annual effects, so $N = 365$ could be seen as suitable lower bound. For the in-sample data set there is no strict rule neither, but $D = 365$ or $D = 2 \times 365$ are common choices.

We might observe that conducting a forecasting study might be computationally very costly if N is large and providing forecasts is costly as well. Most sophisticated forecasting models involve the estimation/training of parameters, say $\boldsymbol{\Theta}$ (e.g. the parameter vectors $\boldsymbol{\beta}_s$ for $s \in \mathbb{S}$, so $\boldsymbol{\Theta} = (\boldsymbol{\beta}_0, \dots, \boldsymbol{\beta}_{S-1})$). Ideally, we improve the estimation of $\boldsymbol{\Theta}$ once we receive new data. This is a so called refit or recalibration of the model in a sequential order. Thus, we have a sequence of parameter vectors $\boldsymbol{\Theta}_1, \dots, \boldsymbol{\Theta}_N$ which is used to predict the out-of-sample prices $\boldsymbol{Y}_{D+1}, \dots, \boldsymbol{Y}_{D+N}$. However, if computational costs for receiving a parameter vector $\boldsymbol{\Theta}$ is too high only a reduced number of updates of the parameter vectors $\boldsymbol{\Theta}_i$ are considered. In the extreme case we estimate the model only once using the initial calibration window $\boldsymbol{\Theta}_1$ and use this parameter vector for the prediction of all prices $\boldsymbol{Y}_{D+1}, \dots, \boldsymbol{Y}_{D+N}$.

Furthermore, according to Diebold (see [Die15]), a forecasting study can be used for this purpose to compare forecasts. However, here we remark that we are not comparing models but forecasts of the models. Still, this might give indications for the suitability of models as well.

4.3 On training-validation-test

It seems, that we have covered the relevant aspects for the design of a forecasting study. Indeed, if we only want to compare the forecast accuracy of two predefined models (e.g. naive vs expert) than we can proceed as described above. However, in practice we are looking for accurate forecasting models, and usually try to improve models. But, once we use the evaluation results of a forecasting study to construct a better forecasting model we start cheating. This, is clear as we use information that would not have been available at time of forecasting to create the forecasting model. To be precise: Any influence on the model of data which we want to predict in the forecasting study is not allowed. Strictly, even looking at the data (e.g. a graph or a summary output) is forbidden. Of course, it is difficult to exclude any impact as we have to read the data to evaluate it, but we should try to approximate a realistic setting as good as possible.

To allow for improvements of models, based on evaluation results, typically the training-validation-test framework is used. This framework is popular in machine learning and a kind of standard toolkit for proper model building. The general key idea is that the data used for evaluation contains a validation part and a test part. In our time series context, we may regard this as a split or an extension of the evaluation windows. To be in line with the notation above, we consider an extension here. Thus, we take the initial calibration window from $1, \dots, D$ and create N_{eval} windows within the initial calibration window. So we evaluate predictions for $D - N_{\text{eval}} + j$ with $j \in \{1, \dots, N_{\text{eval}}\}$ given all data up to $D - N_{\text{eval}} + j - 1$ and refer this as validation set. Now, we simply hope that a model that provides accurate forecast for days $D - N_{\text{eval}} + 1, \dots, D$ also provides accurate forecasts for the test set or out-of-sample set $D + 1, \dots, D + N$. This procedure time series based training/validation/test procedure is sometimes referred as forward chaining, and is visualized in Figure 14 for the rolling window and expanding window study approach.

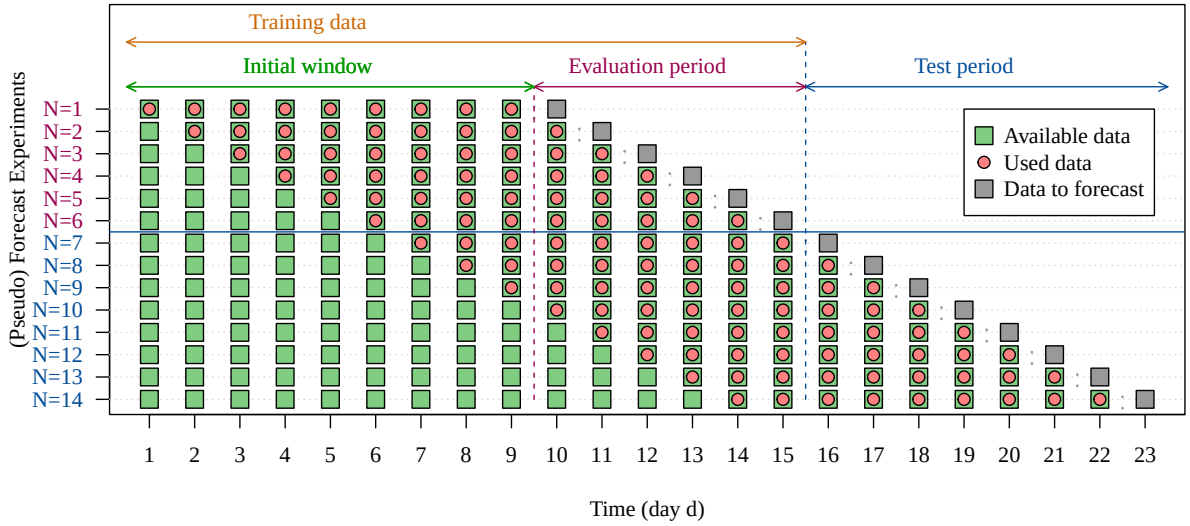
The clear advantage the usage of a validation set is that we can improve our models based on the validation loss without going into risk of cheating.

We mentioned that we ideally refit the model in each time step of the test set. Clearly this also holds for the validation set. We learned that the may interpret the forecasting model in a general way. Most importantly the parameter vector may contain model specifications. For instance we can interpret the calibration window size D_{calib} as a parameter of a model. Then we would define $(\boldsymbol{\beta}_0, \dots, \boldsymbol{\beta}_{S-1}, D_{\text{calib}})$ as parameter vector. However, the estimation of an optimal value for D_{calib} is quite complicated. One way would be to test several values for D_{calib}, e.g. $D_{\text{calib}} \in \{56, 365, 730\}$ using k-fold cross-validation and proceed with the cross-validation optimal choice for D_{calib} to do the prediction. Repeating this procedure for all N test experiments can be computationally very demanding.

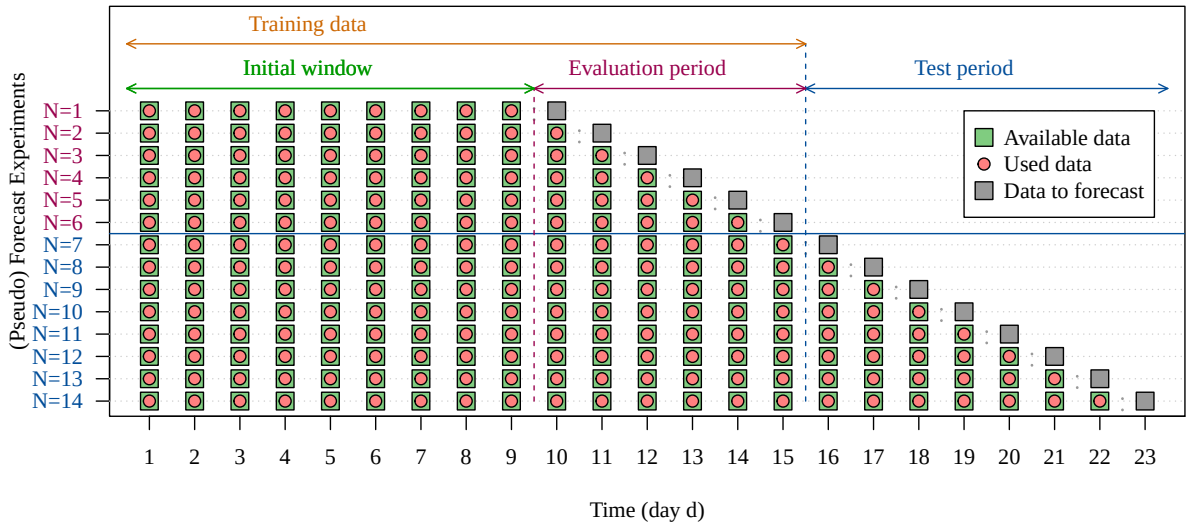
To deal with this problem all model parameters are disentabgled into hyper-parameters $\boldsymbol{\Psi}$ and (standard) parameters $\boldsymbol{\Theta}$. Typically, hyper-parameters $\boldsymbol{\Psi}$ are only optimized once on the validation set and then kept fix through all the test data. In contrast, the parameter vector $\boldsymbol{\Theta}$ is usually recalibrated for each of the N forecast experiments in the test set. Note, that strictly it is the choice of the forecaster to assign if a parameter is a hyper-parameter or not. However, usually the computational demand is forcing the forecaster to treat structural model parameters as hyper-parameters.

Now, we want to mention that the decision for a forecasting model in the evaluation and test set, can also be interpreted quite general. It does not mean that we always have stick to a specific model (e.g. the expert model) in the validation or test set. It is also possible to have more sophisticated schemas that involve multiple models (in machine learning this might be regarded as base learners). For instance, we could have a forecasting algorithm with deterministic selection rules, e.g. that predicts on Mondays and Tuesdays with the naive model and on other days with the expert model. However, we also have dynamic selection rules based on information criteria or past performance and also forecast combinations are feasible. We study those in more depth in Section 12.

We mentioned that the single experiments have to be evaluated with respect to a certain loss, we will study possible choices in the next section.



(a) Rolling window study



(b) Expanding window study

Figure 14: Illustration of window studies with evaluation and test sets with $D = 9$, $N_{\text{eval}} = 6$ and $N = 14$.

4.4 Evaluation criteria

To evaluate forecasts we consider mainly two popular error measures, the mean absolute error (MAE) and the root mean square error (RMSE). Both are proper and absolute error measures, i.e. they have properties that we expect from a forecasting evaluation criterion. The MAE is a more robust criterion and is very widely used. It is strictly proper for absolute deviation designs and thus strictly proper for median forecasts. So if there are multiple forecasts available, including the perfect median forecast model, then the MAE will filter out the perfect median forecast model. The RMSE is the optimal criterion for least square designs and thus strictly proper for mean forecasts, but it is more outlier depended. The MAE for an error vector $\boldsymbol{e} = (e_1, \ldots, e_N)$ is generally defined by

$$\text{MAE}(\boldsymbol{e}) = \frac{1}{N} \sum_{i=1}^{N} |e_i| \tag{39}$$

and the RMSE by

$$\text{RMSE}(\boldsymbol{e}) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} |e_i|^2}. \tag{40}$$

In engineering, but also in energy economics, the error measure MAPE (mean absolute

percentage error) is also popular. It is defined by

$$\text{MAPE}(\boldsymbol{e}) = \frac{1}{N}\sum_{i=1}^{N}\frac{|e_i|}{z_i} \tag{41}$$

where z_i denotes the target value to forecast that will be observed. However, it is a relative measure and has some properties that we do not expect for a proper evaluation measure (e.g. it cannot be evaluated if the price is exactly 0). It should not be used in electricity price forecasting studies.

Now, we will study the electricity price forecasting setting. We assume that the vector $\boldsymbol{\mathcal{E}} = (\widehat{\boldsymbol{\varepsilon}}_{D+1}, \dots, \widehat{\boldsymbol{\varepsilon}}_{D+N})$ of errors of the out-of-sample electricity price forecasting study is given. But the forecasting errors $\widehat{\boldsymbol{\varepsilon}}_{D+1}, \dots, \widehat{\boldsymbol{\varepsilon}}_{D+N}$ are vectors with S errors each day, as $\widehat{\boldsymbol{\varepsilon}}_d = (\widehat{\varepsilon}_{d,s})_{s\in\mathbb{S}} = (\widehat{\varepsilon}_{d,0}, \dots, \widehat{\varepsilon}_{d,S-1})'$. This makes the direct application of the MAE and RMSE not straightforward or, in other words, it opens multiple ways to compute MAE and RMSE measures.

One option is to consider each period s separately in the errors $\widehat{\varepsilon}_{D+1,s}, \dots, \widehat{\varepsilon}_{D+N,s}$. This leads to the measures

$$\text{MAE}_s(\boldsymbol{\mathcal{E}}) = \frac{1}{N}\sum_{i=1}^{N}|\widehat{\varepsilon}_{D+i,s}|, \tag{42}$$

$$\text{RMSE}_s(\boldsymbol{\mathcal{E}}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}|\widehat{\varepsilon}_{D+i,s}|^2} \tag{43}$$

If $S = 24$ then MAE_s and RMSE_s is also known as hourly MAE and hourly RMSE. The MAE_s and RMSE_s are popular measures in electricity price forecasting. However, the report of S measures through a comparison of each individual period s is sometimes not practical. Thus, we can also do a global evaluation where we evaluate all errors at once. We can evaluate the errors by treating all elements of $\boldsymbol{\mathcal{E}}$ as one vector[9]:

$$\text{MAE}(\boldsymbol{\mathcal{E}}) = \frac{1}{SN}\sum_{i=1}^{N}\sum_{s=1}^{S}|\widehat{\varepsilon}_{D+i,s}|, \tag{44}$$

$$\text{RMSE}(\boldsymbol{\mathcal{E}}) = \sqrt{\frac{1}{SN}\sum_{i=1}^{N}\sum_{s=1}^{S}|\widehat{\varepsilon}_{D+i,s}|^2} \tag{45}$$

Moreover, we could define it in a proper multivariate way by:

$$\text{MAE}_{\text{vec}}(\boldsymbol{\mathcal{E}}) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{S}\|\widehat{\boldsymbol{\varepsilon}}_{D+i}\|_1 = \frac{1}{NS}\sum_{i=1}^{N}\sum_{s=1}^{S}|\widehat{\varepsilon}_{D+i,s}|, \tag{46}$$

$$\text{RMSE}_{\text{vec}}(\boldsymbol{\mathcal{E}}) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{S}\|\widehat{\boldsymbol{\varepsilon}}_{D+i}\|_2 = \frac{1}{NS}\sum_{i=1}^{N}\sqrt{\sum_{s=1}^{S}|\widehat{\varepsilon}_{D+i,s}|^2}. \tag{47}$$

We observe that $\text{MAE}_{\text{vec}}(\boldsymbol{\mathcal{E}})$ is the same as $\text{MAE}(\boldsymbol{\mathcal{E}})$, but $\text{RMSE}(\boldsymbol{\mathcal{E}})$ differs from $\text{RMSE}_{\text{vec}}(\boldsymbol{\mathcal{E}})$. From the statistical point of view RMSE_{vec} would be preferable to RMSE, still the latter one is popular as well.

4.5 The DM-Test

A comparison of a forecast A with a forecast B using the MAE and RMSE is possible. However, even if we compute confidence intervals to make significance statements (e.g. 'forecast A is significantly better than forecast B') we lose the testing power, as the dependency structure between the forecast's resp. errors is ignored. The Diebold-Mariano test (DM-test) overcomes this problem. Historically, in the electricity price forecasting context the DM-test is often only considered for the comparison of the errors for each hour individually. However, it is also possible to consider a DM-test that compares the test at once.

[9] e.g. $(\widehat{\varepsilon}_{D+1,1}, \dots, \widehat{\varepsilon}_{D+N,1}, \widehat{\varepsilon}_{D+1,2}, \dots, \widehat{\varepsilon}_{D+N,2}, \widehat{\varepsilon}_{D+1,3}, \dots, \widehat{\varepsilon}_{D+N,S-1}, \widehat{\varepsilon}_{D+1,S}, \dots, \widehat{\varepsilon}_{D+N,S})$

The key element is to compare a loss differential of two losses $L_{A,t}$ and $L_{B,t}$ at a certain time t. $L_{A,t}$ measures the loss of model A at time t and $L_{B,t}$ the loss of model B at the same time t. The series $\Delta_{A,B,t} = L_{A,t} - L_{B,t}$ measures the difference between both losses. If this difference is not significantly different from zero, then from the statistical point of view no forecast is better than the other one. If $\Delta_{A,B,t}$ is significantly smaller than 0, then forecast A is significantly better than forecast B, with respect to the chosen significance level. And vice versa: if $\Delta_{A,B,t}$ is significantly greater than zero, then B is significantly better than A. The test works in analogy to a t-test or z-test, where we can compare the differences between two means. Here, an essential difference is that the assumptions on the test are weaker, especially correlated forecasting errors are allowed.

Similarly, as we have different plausible measures (e.g. MAE, RMSE), we have different options for applying the DM-test. Here a major focus in the interpretation of the DM-test is put on the definition of the loss functions $L_{A,t}$ and $L_{B,t}$. Moreover, we can create DM-tests that test for a difference in forecasting errors at a period s or we can construct tests that test globally for a difference in the full forecasting error vector.

Now, assume that the forecasting errors $\boldsymbol{\mathcal{E}}_A = (\widehat{\boldsymbol{\varepsilon}}_{A,D+1}, \dots, \widehat{\boldsymbol{\varepsilon}}_{A,D+N})$ and $\boldsymbol{\mathcal{E}}_B = (\widehat{\boldsymbol{\varepsilon}}_{B,D+1}, \dots, \widehat{\boldsymbol{\varepsilon}}_{B,D+N})$ for forecasts A and B with $\widehat{\boldsymbol{\varepsilon}}_{A,d} = (\widehat{\varepsilon}_{A,d,1}, \dots, \widehat{\varepsilon}_{A,d,S})'$ and $\widehat{\boldsymbol{\varepsilon}}_{B,d} = (\widehat{\varepsilon}_{B,d,1}, \dots, \widehat{\varepsilon}_{B,d,S})'$ are given. We first consider the case of S single tests for each delivery period s. Then the $\|\cdot\|_1$-loss (also L^1-loss, L_1-loss, ℓ_1-loss) DM-test for a difference in the forecasting performance at period s is given by

$$L_{A,i,s} = |\widehat{\varepsilon}_{A,D+i,s}| \quad \text{and} \quad L_{B,i,s} = |\widehat{\varepsilon}_{B,D+i,s}| \tag{48}$$

for forecasts A and B. Similarly, the $\|\cdot\|_2$-loss (also L^2-loss, L_2-loss, ℓ_2-loss) DM-test for a difference in the forecasting performance at period s is defined through

$$L_{A,i,s} = |\widehat{\varepsilon}_{A,D+i,s}|^2 \quad \text{and} \quad L_{B,i,s} = |\widehat{\varepsilon}_{B,D+i,s}|^2. \tag{49}$$

The first equation is easier to interpret and corresponds directly to the definition of the error measure MAE_s, the second equation to the RMSE_s.

With $L_{A,i,s}$ and $L_{B,i,s}$ we can define their differences and the corresponding mean:

$$\Delta_{A,B,i,s} = L_{A,i,s} - L_{B,i,s}, \tag{50}$$

$$\overline{\Delta}_{A,B,s} = \frac{1}{N} \sum_{i=1}^{N} \Delta_{A,B,i,s} \tag{51}$$

The DM test statistic is now defined by

$$t_{\text{DM}} = \frac{\overline{\Delta}_{A,B,s}}{\sigma(\overline{\Delta}_{A,B,s})},$$

where $\sigma(\overline{\Delta}_{A,B,s})$ is the standard deviation of $\overline{\Delta}_{A,B,s}$. The DM-test relies on the property that the test statistic t_{DM} converges with increasing N to the standard normal distribution. Note that this convergence holds independent of the choice of the loss function ($\|\cdot\|_1$ or $\|\cdot\|_2$ loss), independent of the considered forecasts A and B and for all periods s.

The multivariate (or vectorized, or global) DM-test is based on a single loss function for the full error vectors. So the $\|\cdot\|_1$-loss (also L^1-loss) DM-test for a difference in the forecasting performance is given by

$$L_{A,i} = \|\widehat{\boldsymbol{\varepsilon}}_{A,D+i}\|_1 = \sum_{s=1}^{S} |\widehat{\varepsilon}_{A,D+i,s}| \quad \text{and} \quad L_{B,i} = \|\widehat{\boldsymbol{\varepsilon}}_{B,D+i}\|_1 = \sum_{s=1}^{S} |\widehat{\varepsilon}_{B,D+i,s}| \tag{52}$$

for forecasts A and B. The $\|\cdot\|_2$-loss (also L^2-loss) DM-test for a difference in the forecasting performance is given by

$$L_{A,i} = \|\widehat{\boldsymbol{\varepsilon}}_{A,D+i}\|_2 = \sqrt{\sum_{s=1}^{S} |\widehat{\varepsilon}_{A,D+i,s}|^2} \quad \text{and} \quad L_{B,i} = \|\widehat{\boldsymbol{\varepsilon}}_{B,D+i}\|_2 = \sqrt{\sum_{s=1}^{S} |\widehat{\varepsilon}_{B,D+i,s}|^2} \tag{53}$$

for forecasts A and B. For the multivariate DM-test the loss differences and the test statistic are defined as above. Moreover, the test statistic converges to the asymptotic normal distribution as well.

4.6 Tasks

1. Perform a block-wise 10-fold cross validation for the expert model (7).

2. Analyse how the estimated parameter vectors $\boldsymbol{\beta}_s$ of the expert model (7) for a selected s evolve over time. How does this change if you consider $D = 365$ and $D = 56$?

3. We considered forecasting studies with reestimation every day as it should be beneficial to update the model if new data arrives. To illustrate the benefits perform a forecasting study where i) the expert model (7) is reestimated in every time step, and ii) The expert model is only estimate once at the beginning of the study. Compare your findings by evaluating the MAE and MAE_s.

4. Compute the RMSE_s and create a plot for RMSE_s. Compare your findings with the ones of MAE_s.

5. Do the Diebold-Mariano test based on the $\|\cdot\|_2$-norm (the L^2-loss) to compare the expert model (7) estimated with $D = 730$, $D = 365$ and $D = 56$.

5 Computation Time Issues

We have learned about the basics on a forecasting study design and evaluation. The crucial is obviously about finding an accurate forecasting model. We have seen already that a price forecasting model can become quite complex. Finding the optimal forecasting model, specially the optimal (hyper)parameters will be often a crucial question. Of course, computational costs limits us in application. Therefore, it is important to think carefully about computation time issues in this section. Moreover, we should have in mind, that a forecasting model which has considerably less computational demand for parameter estimation/training has the advantage that within the same time more models of the same complexity could be tested and evaluated. In forecasting literature, models are often compared with respect to forecasting accuracy, as covered in the previous section. However, sometimes this is done with respect to time constraints, to ensure practicability in practice and in the model building phase. Therefore, sometimes pareto fronts are considered, which consider on one axis the forecasting accuracy and on the other axis the computation time. Sometimes, it might be preferable to choose a less accurate forecasting algorithm which is very fast because it allows more time for decision process. If this decision process involves any complicated optimization tasks, we have more time find a (better local) optimum.

In this section, we first we discuss online learning tools, an efficient way to create predictions in a time series setting. Here, we will get a feeling about the computational edge we can reach when doing model building. Second, we discuss the hyper-parameter tuning issue in more detail, because it is often a very time consuming task the design should be taken carefully.

5.1 Online learning

To get valuable results, that can show significance we need usually many forecasts experiments. So N should be sufficiently large. Estimating N models might be a problem for model with high computational cost (e.g. neural networks). Here, online learning approaches may help to reduce the computational burdens. Evaluating all the past data again, and again makes the evaluation in a rolling window forecasting study troublesome. The key idea in online learning approaches is that avoid the evaluation of past data, but only store a few relevant characteristics of the model to predict our new outcome using the new observation data set. We want to illustrate the online learning approach the regression problem. However, this will be based on a expanding window approach first. This method is also known recursive least squares (RLS), [GXT+18].

Therefore let n be the n-th window that we just evaluated for a fixed $s \in \mathbb{S}$. The corresponding regression vector is $\boldsymbol{\beta}_n$ is

$$\widehat{\boldsymbol{\beta}}_n^{\text{ls}} = (\mathcal{X}_n'\mathcal{X}_n)^{-1}\mathcal{X}_n'\mathcal{Y}_n.$$

given e.g. $\mathcal{Y}_n = (Y_1, \ldots, Y_n)'$ and $\mathcal{X}_n = (\boldsymbol{X}_1', \ldots, \boldsymbol{X}_n')$ analog. In the next step we are interested in estimating $\widehat{\boldsymbol{\beta}}_{n+1}^{\text{ls}}$ given the new chunk of data Y_{n+1} and $\boldsymbol{X}_{n+1}$. The new data defines $\mathcal{Y}_{n+1} = (\mathcal{Y}_n', Y_{n+1})' = (Y_1, \ldots, Y_n, Y_{n+1})'$ and $\mathcal{X}_{n+1} = (\mathcal{X}_n', \boldsymbol{X}_{n+1}') = (\boldsymbol{X}_1', \ldots, \boldsymbol{X}_n', \boldsymbol{X}_{n+1}')'$. Obviously, it holds

$$\widehat{\boldsymbol{\beta}}_{n+1}^{\text{ls}} = (\mathcal{X}_{n+1}'\mathcal{X}_{n+1})^{-1}\mathcal{X}_{n+1}'\mathcal{Y}_{n+1}.$$

However, now our target is to derive a formula for $\widehat{\boldsymbol{\beta}}_{n+1}^{\text{ls}}$ that only requires $\widehat{\boldsymbol{\beta}}_n^{\text{ls}}$ (and maybe a few other recursively defined objects), Y_{n+1} and $\boldsymbol{X}_{n+1}$ but not knowledge about $\mathcal{Y}_n$ and $\mathcal{X}_n$.

Therefore we require the Sherman-Morrison formula which can evaluate rank-1 updates of inverse matrices in an elegant way, this is: For invertible matrix $\boldsymbol{A}$ and vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ it holds

$$(\boldsymbol{A} + \boldsymbol{u}\boldsymbol{v}')^{-1} = \boldsymbol{A}^{-1} - \frac{\boldsymbol{A}^{-1}\boldsymbol{u}\boldsymbol{v}'\boldsymbol{A}^{-1}}{1 + \boldsymbol{v}'\boldsymbol{A}^{-1}\boldsymbol{u}}.$$

As it holds $\mathcal{X}_{n+1}'\mathcal{X}_{n+1} = \mathcal{X}_n'\mathcal{X}_n + \boldsymbol{X}_{n+1}'\boldsymbol{X}_{n+1}$ this leads to

$$\begin{aligned}(\mathcal{X}_{n+1}'\mathcal{X}_{n+1})^{-1} &= (\mathcal{X}_n'\mathcal{X}_n + \boldsymbol{X}_{n+1}'\boldsymbol{X}_{n+1})^{-1} \\ &= (\mathcal{X}_n'\mathcal{X}_n)^{-1} - \frac{(\mathcal{X}_n'\mathcal{X}_n)^{-1}\boldsymbol{X}_{n+1}'\boldsymbol{X}_{n+1}(\mathcal{X}_n'\mathcal{X}_n)^{-1}}{1 + \boldsymbol{X}_{n+1}(\mathcal{X}_n'\mathcal{X}_n)^{-1}\boldsymbol{X}_{n+1}'}. \end{aligned} \tag{54}$$

Thus, we can compute $(\mathcal{X}_{n+1}'\mathcal{X}_{n+1})^{-1}$ given $(\mathcal{X}_n'\mathcal{X}_n)^{-1}$ and $\boldsymbol{X}_{n+1}$ without much effort.

Furthermore, we receive $\widehat{\boldsymbol{\beta}}^{\text{ls}}_{n+1}$ by noting that $\mathcal{X}'_n\mathcal{X}_n\widehat{\boldsymbol{\beta}}^{\text{ls}}_n = \mathcal{X}'_n\mathcal{Y}_n$:

$$\begin{aligned}
\widehat{\boldsymbol{\beta}}^{\text{ls}}_{n+1} &= (\mathcal{X}'_{n+1}\mathcal{X}_{n+1})^{-1}\mathcal{X}'_{n+1}\mathcal{Y}_{n+1} \\
&= (\mathcal{X}'_{n+1}\mathcal{X}_{n+1})^{-1}(\mathcal{X}'_n\mathcal{Y}_n + \boldsymbol{X}'_{n+1}Y_{n+1}) \\
&= (\mathcal{X}'_{n+1}\mathcal{X}_{n+1})^{-1}(\mathcal{X}'_n\mathcal{X}_n\widehat{\boldsymbol{\beta}}^{\text{ls}}_n + \boldsymbol{X}'_{n+1}Y_{n+1}) \\
&= (\mathcal{X}'_{n+1}\mathcal{X}_{n+1})^{-1}\left((\mathcal{X}'_{n+1}\mathcal{X}_{n+1} - \boldsymbol{X}'_{n+1}\boldsymbol{X}_{n+1})\widehat{\boldsymbol{\beta}}^{\text{ls}}_n + \boldsymbol{X}'_{n+1}Y_{n+1}\right) \\
&= \widehat{\boldsymbol{\beta}}^{\text{ls}}_n + (\mathcal{X}'_{n+1}\mathcal{X}_{n+1})^{-1}\left(-\boldsymbol{X}'_{n+1}\boldsymbol{X}_{n+1}\widehat{\boldsymbol{\beta}}^{\text{ls}}_n + \boldsymbol{X}'_{n+1}Y_{n+1}\right) \\
&= \widehat{\boldsymbol{\beta}}^{\text{ls}}_n + (\mathcal{X}'_{n+1}\mathcal{X}_{n+1})^{-1}\boldsymbol{X}'_{n+1}\underbrace{\left(Y_{n+1} - \boldsymbol{X}_{n+1}\widehat{\boldsymbol{\beta}}^{\text{ls}}_n\right)}_{\text{forecast error in } n+1}
\end{aligned} \tag{55}$$

Hence, with (55) and (54) we can compute efficiently expanding window updates.

It is not much more complicated to introduce recursive least squares with exponential forgetting. Therefor let $\varrho \in [0,1]$ a forgetting factor. $\varrho = 1$ corresponds to no forgetting and the closer ϱ to zero the less valued is information from the past. The corresponding weighted OLS estimator is

$$\widehat{\boldsymbol{\beta}}^{\text{ls}}_s(\varrho) = \underset{\boldsymbol{\beta}\in\mathbb{R}^{ps+1}}{\arg\min} \sum_{d=1}^{D} \varrho^{D-d}(Y_{d,s} - \boldsymbol{X}'_{d,s}\boldsymbol{\beta})^2. \tag{56}$$

where $\varrho = 1$ corresponds to the standard OLS estimator with no forgetting. Note that $n_{\text{eff}}(\varrho) = \frac{1}{1-\varrho}$ is often referred as effective sample size. E.g. if $\varrho = 0.9$ we get $n_{\text{eff}} = 10$. In practice you see usually ϱ close to 1.

For this problem the corresponding recursive least squares solution is given by the two formulas

$$\mathbb{G}_{n+1}(\varrho) = \frac{1}{\varrho}\left(\mathbb{G}_n(\varrho) + \frac{\mathbb{G}_n(\varrho)\boldsymbol{X}_{n+1}\boldsymbol{X}'_{n+1}\mathbb{G}_n(\varrho)}{\varrho + \boldsymbol{X}'_{n+1}\mathbb{G}_n(\varrho)\boldsymbol{X}_{n+1}}\right) \tag{57}$$

$$\widehat{\boldsymbol{\beta}}^{\text{ls}}_{n+1}(\varrho) = \widehat{\boldsymbol{\beta}}^{\text{ls}}_n(\varrho) + \mathbb{G}_{n+1}(\varrho)\boldsymbol{X}'_{n+1}\underbrace{\left(Y_{n+1} - \boldsymbol{X}'_{n+1}\widehat{\boldsymbol{\beta}}^{\text{ls}}_n\right)}_{\text{forecast error in } n+1} \tag{58}$$

for $\mathbb{G}_n(\varrho) = (\mathcal{X}_n(\varrho)'\mathcal{X}_n(\varrho))^{-1}$ and $\mathcal{X}_n(\varrho)$ as exponentially discounted regression matrix as in (56).

We have seen that the RLS approach gives exactly the same results as OLS, but it is computationally much more efficient than OLS. However, there is no free lunch. The RLS requires matrix multiplications (involving $\mathbb{G}_n$) which is relatively costly for a large parameter space. However, it is not the only online learning approach. There are faster ones available. However, they are not as accurate as RLS, as they approximate only the OLS solution. A popular procedure is online gradient descent (ODG). Here, the parameter vector is updated by a gradient step. It is given by

$$\widehat{\boldsymbol{\beta}}_{n+1} = \widehat{\boldsymbol{\beta}}_n + \eta\boldsymbol{X}'_{n+1}\underbrace{(Y_{n+1} - \boldsymbol{X}'_{n+1}\boldsymbol{\beta}_n)}_{\text{forecast error in } n+1}. \tag{59}$$

where the gradient step size is controlled by a learning rate $\eta > 0$. We see that (59) has essentially the same update structure as (58). In simple words the $\mathbb{G}_n$ matrix which is used to update the parameter vector is replaced by the learning rate η, formally ODG is a special case of RLS if $\mathbb{G}_n = \eta\boldsymbol{I}$. For ODG learning rate η is a hyperparameter that has to be tuned, we discuss this in the next subsection. If η is too small, then the parameter vector is updated too slowly and the model is not able to adapt to new data. If η is too large, then the parameter vector is updated too fast and the model is not able to learn the underlying structure of the data. Still, OGD is not really applied in electricity price forecasting practice, as it is not as accurate as RLS. However, it helps us to better understand the idea of online learning. Later on when we discuss neural networks, we will see that OGD is a relevant part of the training process. Also in the context of forecast combination it is useful for better understanding.

5.2 On Hyperparameter tuning.

We briefly mentioned the problem that the modeller often disentangles the model parameter into hyper-parameters $\boldsymbol{\Psi}$ and (standard) parameters $\boldsymbol{\Theta}$ where the optimal values for $\boldsymbol{\Psi}$ are less frequently updated than $\boldsymbol{\Theta}$.

Typically $\boldsymbol{\Psi}$ contains more structural model parameters, that specify the model and optimization structure, e.g. of the design structure of the regression matrices $\boldsymbol{X}$ or $\boldsymbol{X}_s$, and the least squares optimization. Examples that we know so far, are the calibration window size D, a forgetting rate ρ or just dummies which indicate if a certain parameter (e.g. a specifically lagged price or a weekday dummy) is active in the model or not.

A common difficulty in hyper-parameter tuning is that there are often tuning parameters of different popular types:

- logical/boolean: a parameter that takes values only in $\{0, 1\}$ usually `TRUE` or `FALSE`.
- categoricals: a parameter that takes values in $\{0, 1, \dots, n-1\}$ for n categories which have in general no additional distance or ordering relationship. This is a special case of logical/boolean.
- integers: a parameter that takes values in $\{l, 1, \dots, l+n-1\}$ for n numbers starting from $l \in \mathbb{Z}$ which have the natural ordering and distance measures. This is a special case of logical/boolean.
- real interval (represented by numerics/real/floats on machines) a value from the interval $[a, b] \subseteq \mathbb{R}$ equipped with standard ordering and metric.

Thus, of we require optimization algorithms that can deal with some, or even all of the mentioned types. In practice we see of

Grid search. only possible for a few logicals, categorical and small seach space integers. The clear advantage is to be guaranteed to find the optimal value.

Random search. Usually, uniformly on a certain interval, however, sometimes parameters are drawn on an exp-scale, have to reparametrised so that de facto an exp-scale applies. This is often useful, if only a few hyperparameters (ca. < 10) have to be tuned.

Sophisticated search algorithms, usually consider some type of Bayesian search algorithms. Depending on the input structure. Tree-type methods are poular as they are capable to deal with intergers, numerics and categoricals efficiently. All algorithms require some initialization runs to have some data points to start with, here, we can also add manually parameter choices.

In general, all hyperparameter tuning theory can be applied to electricity price forecasting problems as well. However, a specific situation is that we always deal with an S-dimensional forecasting problem. Thus, optimizing all S equations separately, is computationally very costly, even when using simple linear models in an online learning framework.

We suggest to use the package `mlrMBO` in `R` and `opttuna` in `python` for professional hyper-parameter training.

We have seen that hyper-parameter tuning usually helps to find suitable models. However, when designing models it is also very important to contain stylized facts that we see in data into the model. We will learn about those in more detail in the next sections.

5.3 Tasks

Hints and the corresponding programming part are provided in Section R.3 in `R` and in Section py.2 in `python`.

1. Could an online learning approach be designed for a standard rolling window forecasting study? Hint: Remember that $\mathcal{X}_{n+1}$ can represented as $\mathcal{X}'_{n+1}\mathcal{X}_{n+1} = \mathcal{X}'_n\mathcal{X}_n + \boldsymbol{X}'_{n+1}\boldsymbol{X}_{n+1} - \boldsymbol{X}'_1\boldsymbol{X}_1$ holds.
2. Consider the general expert model (36) with $\mathbb{L} = \{1, \dots, 8\}$, $\mathbb{W} = \{1, \dots, 7\}$. Consider a linear model online learning setting and interprete the calibration window size D, the forgetting rate ρ as hyperparameter and the activity of $\mathbb{W}$ and $\mathbb{L}$ as Booleans. Do a hyperparameter optimization on a suitable validation set. Report the three best models, what do you observe?

6 Correlation structure of electricity prices.

We have seen that AR process or models with autoregressive features can be used to construct electricity price models. In the expert model (7), three autoregressive terms are used, at lag 1, 2 and 7. To some extent this choice looks arbitrary but makes more sense when we explore the correlation structure of the electricity prices in this section.

6.1 The ACF and PACF

However, first we recall or introduce a few objects that are widely known in time series analysis. For a time series $(Z_t)_{t\in\mathbb{Z}}$ the autocorrelation at lag $k \in \mathbb{N}$ and time t is defined by

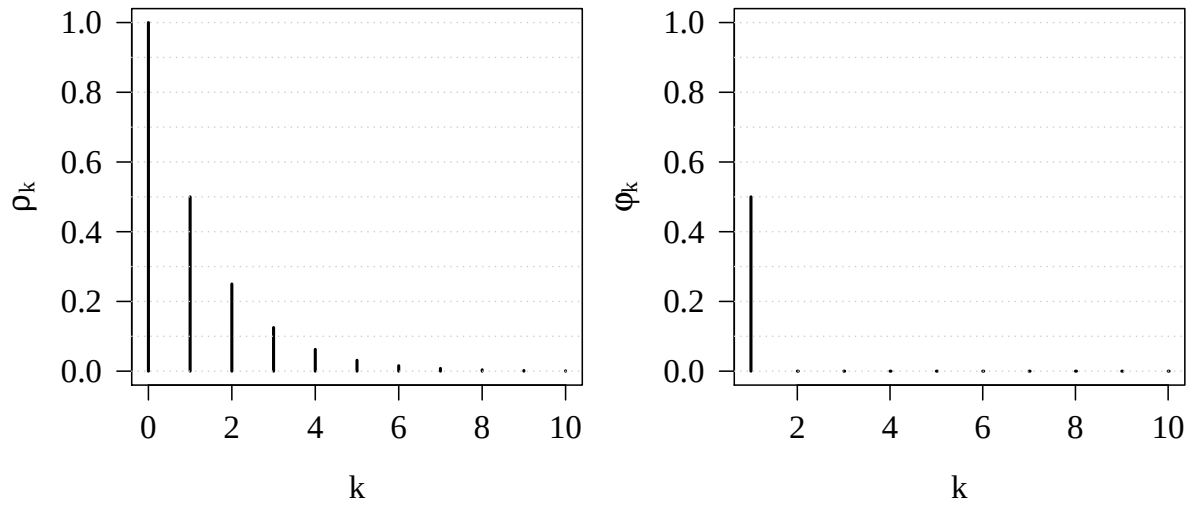
$$\rho_{t,k} = \mathbb{C}\mathrm{or}(Z_t, Z_{t-k}). \tag{60}$$

If for all $\rho_{t,k}$ with $k \in \mathbb{N}$ the sequences $(\rho_{t,k})_{t\in\mathbb{Z}}$ are constant over time then the process is called covariance stationary[10]. In such case we simply report ρ_k and drop the time dependence in the notation of the autocorrelation. The full sequence $(\rho_k)_{k\in\mathbb{Z}}$ is also known as autocorrelation function (ACF) and describes the full correlation structure of the process. Note that ρ_0 is always the variance of a process. The object $\rho_{t,k}$ resp. ρ_k is useful to describe theoretical properties of a process. For instance, an AR(1) process (4) is covariance stationary and has the autocorrelation function $\rho_k = \phi_1^k$. Still, for dealing with autoregressive processes there is a nicer interpretable property, the partial autocorrelation.

Therefore, we consider the partial autocorrelation at lag $k \in \mathbb{N}$ and time t that is given by

$$\varphi_{t,k} = \mathbb{C}\mathrm{or}(Z_t, Z_{t-k}|Z_{t-1}, \ldots, Z_{t-(k-1)}). \tag{61}$$

So $\varphi_{t,k}$ is the correlation between Z_t and Z_{t-k} conditioned on all the information in between. It is clear that for $k = 1$ there is nothing between Z_t and Z_{t-1}, so $\rho_{t,1} = \varphi_{t,1}$. Similarly as above, if a process is covariance stationary then $\varphi_{t,k}$ does not depend on time and the sequence $(\varphi_k)_{k\in\mathbb{Z}}$ is also known as partial autocorrelation function (PACF). AR(p) processes have the nice property that for all $k \geq p$ it holds $\varphi_k = 0$. In other words, only the partial autocorrelations φ_k up to lag k are (potentially) non-zero. An illustrative example for the ACF and PACF of an AR(1) is given in Figure 15.



(a) ACF (b) PACF

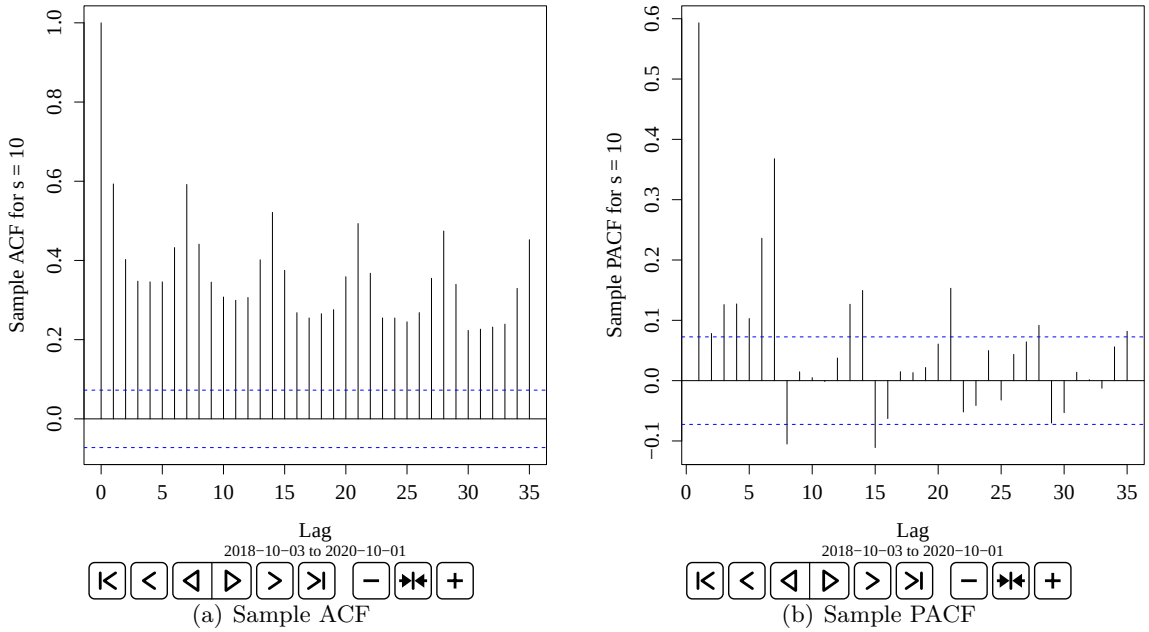
Figure 15: Autocorrelation function (ACF) and partial autocorrelation function (PACF) of an AR(1) with $\phi_1 = 0.5$.

Note that all statements on the ACF and PACF of autoregressive processes are theoretical properties that are unknown in practice. Still, we can use sample estimators to estimate the ACF and PACF of a process, and we denote these estimators by $\widehat{\rho}_k$ and $\widehat{\varphi}_k$[11].

[10] Note that we can introduce covariance stationarity via the autocovariance function $\gamma_{t,k} = \mathbb{C}\mathrm{ov}(Z_t, Z_{t-k})$ as usually done in time series literature.

[11] The explicit formulas for the estimation are not important at this stage, still we will compute them in R

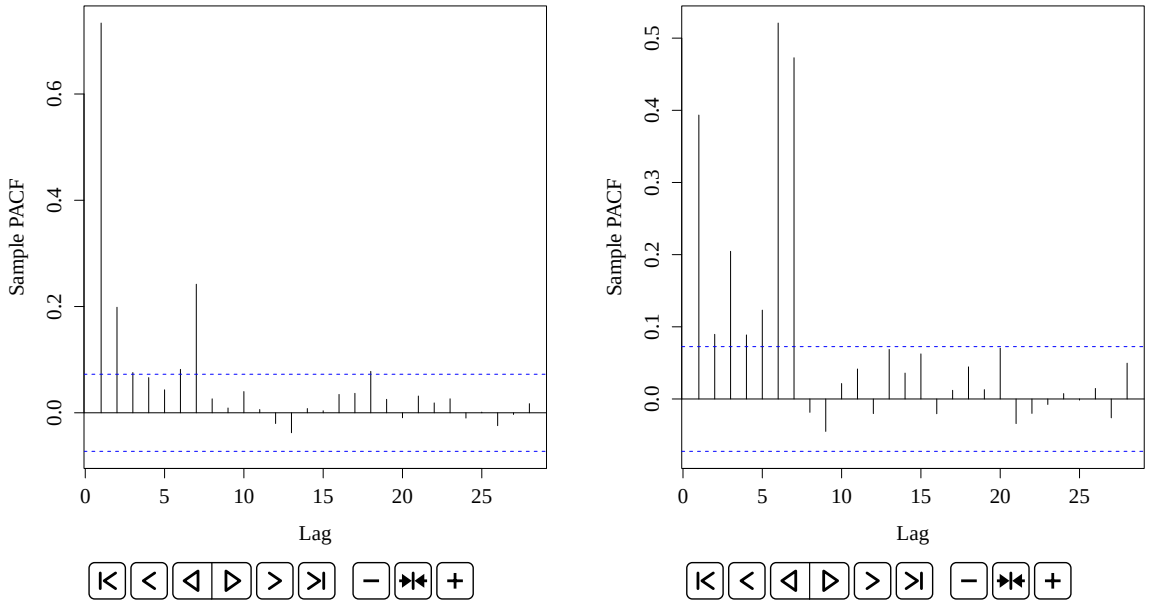
In Figure 16, we see the sample ACF and PACF plots for a considered selected electricity price series. We never observe there $\widehat{\varphi}_k = 0$, so the sample PACF equal exactly to zero. Instead,



(a) Sample ACF (b) Sample PACF

Figure 16: Sample ACF and PACF for EPEX electricity price data sets $(Y_{1,s}, \ldots, Y_{D,s})'$ with $D = 730$ with 95% confidence bands.

we always observe at least slightly positive or negative values - but this does not automatically mean that $\varphi \neq 0$ holds. We have to interpret these graphs carefully. Therefore, the provided 95% confidence bands help. If the original data is generated by an AR(p), then all sample partial autocorrelations $\widehat{\varphi}_k$, where the corresponding true partial autocorrelations φ_k are exactly 0, will lie to 95% of the given confidence band. In contrast, a $\widehat{\varphi}_k$ value that is clearly significantly different from zero indicates that $\varphi_k \neq 0$ might be plausible. But there is no guarantee that $\varphi_k \neq 0$, even if the underlying process is an AR(p) process. This problem is illustrated in Figure 17 for simulations of two AR(7) processes.



(a) $\phi_1 = 0.5$, $\phi_2 = 0.2$, $\phi_3 = \phi_4 = \phi_5 = \phi_6 = 0$, $\phi_7 = 0.2$ (b) $\phi_1 = \phi_2 = \phi_3 = \phi_4 = \phi_5 = 0$, $\phi_6 = 0.4$, $\phi_7 = 0.5$

Figure 17: Sample PACFs of two simulated AR(7) processes for two different parameter settings.

However, in Figure 16, we see that in the PACF the $\widehat{\varphi}_1$ and $\widehat{\varphi}_7$ are clearly significantly positive. Thus, this is a hint that $Y_{d-1,s}$ and $Y_{d-7,s}$ should be included into the model, (remember

but no guarantee (17)). However for $k > 7$, we still observe that some lags that are significantly different from zero. So we might consider even larger autoregressive lags in our model, especially $k = 14$ or $k = 21$ seem to be plausible candidates.

However, we have to interpret sample PACF plots even more carefully, as all the interpretation is only (properly) valid, if the underlying process is covariance stationary - so the price process should behave like an AR(p). But this is extremely unlikely to be true due to deterministic trends, especially seasonal (daily, weekly and annual) and public holiday impacts, potential structural breaks among other reasons for instationarity. In fact, even if the true price process would follow the expert model (7) all the analysis from above would not hold true due to the impact of the weekday dummies. At this stage we do not want go into details, but the presence of weekly seasonality might lead to the fact that the sample ACF and PACF spurious significant lags. This could be avoided by removing deterministic pattern (e.g. deseasonlising) the data prior autocorrelation analysis.

However, all large lags (in absolute values) in the sample PACF are worth trying to include into a model. A forecasting study can show, if it might be worth to include a specific lag or not.

6.2 Unit roots in electricity price data.

A critical question when dealing with the modeling of electricity prices, is whether electricity price time series exhibit a unit root or not. Therefore, we briefly recap the corresponding definition. First we remember the definition of an zero-mean AR(p) process, this is:

$$Y_t = \sum_{k=1}^{p} \phi_k Y_{t-k} + \varepsilon_t = \sum_{k=1}^{p} \phi_k B^k Y_t + \varepsilon_t \quad \Leftrightarrow$$

$$\underbrace{\left(1 - \sum_{k=1}^{p} \phi_k B^k\right)}_{=\Phi(B)} Y_t = \varepsilon_t \tag{62}$$

where we use the Backshift operator (also Lag operator) B notation. It holds $Bx_t = x_{t-1}$ and $BBx_t = B^2 x_t = x_{t-2}$. If the polynomial $\Phi(B)$ can be decomposed into the two polynomials $\Phi(B) = \Phi(B)(1-B)^k$, then Y_t has a unit root of order k. An important property is that a process with a unit is not stationary. Note that $(1-B)Y_t = Y_t - Y_{t-1}$ is the price difference, a process with a single unit root of order 1 is called difference stationary, if the time series after differencing is stationary.

In economics and finance there are plenty of time series which exhibit a unit root and are (close to) difference stationary. Thus, a valid question is, if this holds for electricity prices as well or not. Especially for forecasting tasks of longer horizons the unit root question is of high importance in (energy) risk managment. This is mainly because of the implications on the variance. If a time series is stationary then the (predicted) variance of process far is in the future is constant. However, if the process has a unit root, then the long-term (predicted) variance will be unbounded (=infinite).

A common modeling framework for univariate time series with unit roots are ARIMAX models. They are a generalization of (62). A general ARIMAX(p,d,q) and defined by:

$$\underbrace{\left(1 - \sum_{k=1}^{p} \phi_k B^k\right)}_{=\Phi(B)} (1-B)^d (Y_t - \boldsymbol{\beta X}_t) = \underbrace{\left(1 - \sum_{k=1}^{q} \theta_k B^k\right)}_{=\Theta(B)} \varepsilon_t \tag{63}$$

with Lag-polynomials Φ and Θ in $(1-B)^d(Y_t - \boldsymbol{\beta X}_t)$ and ϵ_t, and $\boldsymbol{\beta X}_t$ describe some external regressor inputs. The external regressors could contain for instance the weekday dummy information which is also included in the expert model (7). Indeed, we can define a model that covers basically the same characteristics as the expert model, but contains a unit root. This is

$$(1-\phi_{s,1}B)(1-B)(Y_{d,s} - \beta_{s,1}\mathrm{DoW}_d^1 - \beta_{s,6}\mathrm{DoW}_d^6 - \beta_{s,7}\mathrm{DoW}_d^7) = (1-\theta_{s,1}B)(1-\theta_{s,2}B^7)\varepsilon_{d,s} \tag{64}$$

with 6 parameters similar to (7) except for the intercept. Model (64) also contains 3 weekday parameters, and 3 autoregressive parameters. Interestingly, the predictive performance of (64)

is better than that of the expert model (7). Also the ARMAX model that corresponds to the ARIMAX model (64) defined by

$$(1-\phi_{s,1}B)(Y_{d,s}-\beta_{s,0}-\beta_{s,1}\mathrm{DoW}_d^1-\beta_{s,6}\mathrm{DoW}_d^6-\beta_{s,7}\mathrm{DoW}_d^7)=(1-\theta_{s,1}B)(1-\theta_{s,2}B^7)\varepsilon_{d,s} \quad (65)$$

performs usually worse. Thus, this is a clear indication that the electricity price time series exhibits a unit root. And a corresponding consideration can improve the predictive accuracy, even in the short run.

Finally, we want to mention the fact that we know from fundamental models (see Fig. 5) that fuel and emission price series influence the electricity price. It might me plausible that the electricity price data gets at least some unit root behaviour from those external regressor inputs. Indeed, we will come back to this question, when considering corresponding external regressors in Section 8.2.

6.3 Cross-period dependencies

We have learned some facts about the correlation structure of electricity price data. But we have to remember that our electricity price time series is in fact an S-dimensional time series, where we always observe S prices at once. So far we described expert models where the price $Y_{d,s}$ depends on its past prices $Y_{d-k,s}$. This choice can be backed up by evaluating the sample partial autocorrelation function of $Y_{d,s}$ or more precisely the given vector $(Y_{1,s},\dots,Y_{D,s})'$. But it might be that not only the yesterday's electricity price $Y_{d-1,s}$ carries some information that explains $Y_{d,s}$, but also the yesterday's electricity price $Y_{d-1,l}$ at a different period $l\neq s$. Moreover, it is clear that this is not only restricted to the price of yesterday, but could also affect larger lags.

To describe this dependency, we introduce another object, the cross-period autocorrelation at lag k for period s and l, defined as

$$\xi_{s,l,k}=\mathbb{C}\mathrm{or}(Y_{d,s},Y_{d-k,l}). \quad (66)$$

In Figure 18, we see estimated cross-period autocorrelations (sample cross-period autocorrelations) for representative EPEX data for $k=1$. We observe that the values on the diagonal

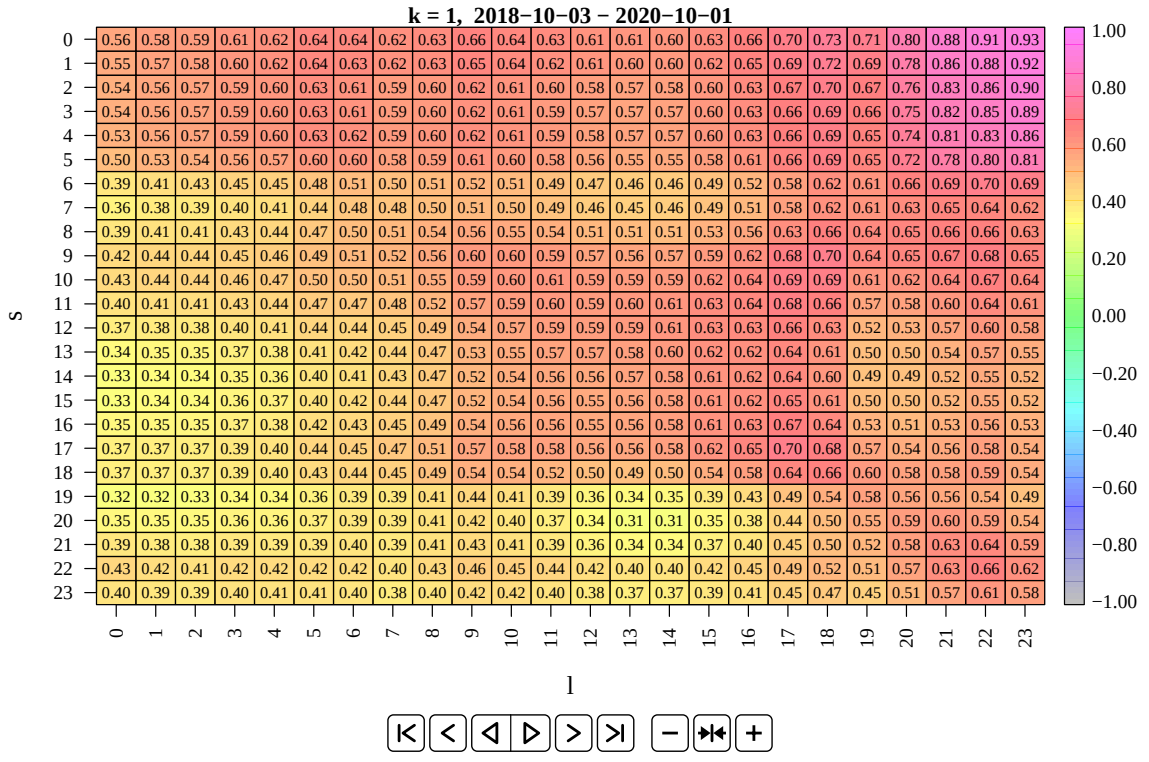


Figure 18: Sample cross-period autocorrelations $\widehat{\mathbb{C}\mathrm{or}}(Y_{d,s},Y_{d-k,l})=\widehat{\xi}_{s,l,k}$ for $s,l\in\mathbb{S}$ (s y-axis, l x-axis) and selected k for day-ahead EPEX data with $S=24$.

$\xi_{s,s,1}$ tend to have relatively high values. This supports the modeling approach that $Y_{d,s}$ depends on $Y_{d-1,s}$. However, we also see that the very last column ($l=23$) has relatively large values.

For the periods in the night and morning, they are even larger than the diagonal values. For example $\widehat{\xi}_{0,0,1}$ is smaller than $\widehat{\xi}_{0,23,1}$. So the correlation between the electricity price at 0:00-1:00 ($Y_{d,0}$) correlates stronger with the yesterday's price at 23:00-0:00 ($Y_{d-1,S-1}$) as with the yesterday's price at 0:00-1:00 ($Y_{d-1,0}$). But this makes perfectly sense, if we remember that this is a price for the physical product electricity. It is plausible that the physical market situation (e.g. generation of renewable energy, electricity consumption, etc.) is more similar between today 0:00 and yesterday 23:00 than yesterday 0:00, just because the time difference is shorter. Thus, we get an indication that we should also include $Y_{d-1,S-1}$ into a properly designed expert model.

So we extend the **expert** model (7) by the $Y_{d-1,S-1}$ term:

$$\begin{aligned} Y_{d,s} = \beta_{s,0} + \beta_{s,1}Y_{d-1,s} + \beta_{s,2}Y_{d-2,s} + \beta_{s,3}Y_{d-7,s} + \beta_{s,4}Y_{d-1,S-1} \\ + \beta_{s,5}\mathrm{DoW}_d^1 + \beta_{s,6}\mathrm{DoW}_d^6 + \beta_{s,7}\mathrm{DoW}_d^7 + \varepsilon_{d,s} \end{aligned} \tag{67}$$

and get a new expert model, called **expert.last**. Note that the parameter $\beta_{s,4}$ models the 'new' effect. We see that model (67) has 8 parameters. However, if $s = S-1$ it holds $Y_{d-1,s} = Y_{d-1,S-1}$. Thus $\beta_{s,1}$ and $\beta_{s,4}$ act on the same object $Y_{d-1,S-1}$. This introduces singularities in the model. Hence the effective number of parameters is reduced by 1 in this case.[12] Model (67) can be visualized in a network representation as well. The multivariate representation is given in Figure (19), which looks quite similar to (11) as only connections from $Y_{d-1,S-1}$ to $Y_{d,s}$ are added. When counting the connections to $Y_{d,s}$ we also see that every delivery product s is modelled by 8 parameters except for $s = S-1$ where only 7 are required.

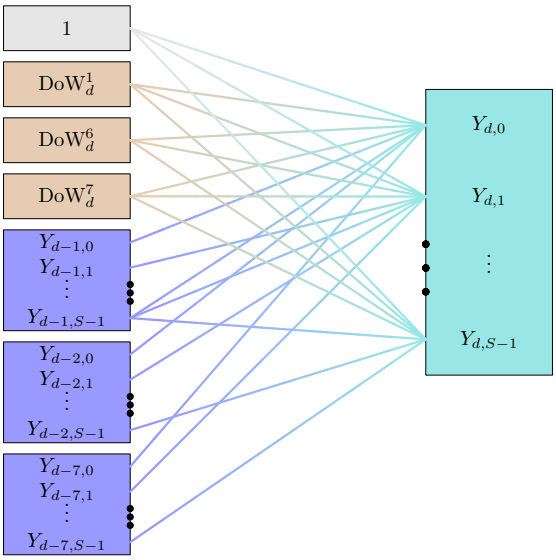


Figure 19: Network visualisation of model (67).

6.4 More on the partial correlations.

For deriving the model (67) above, we are using the fact that $\widehat{\mathbb{C}\mathrm{or}}(Y_{d,s}, Y_{d-k,S-1})$ is always relatively large (sometimes larger than $\widehat{\mathbb{C}\mathrm{or}}(Y_{d,s}, Y_{d-k,s})$) to motivate the **expert.last** model. However, we just evaluated (normal) autocorrelations, instead of looking at the partial autocorrelations, more detailed the partial cross-period autocorrelations. The theoretical background is similar with the ACF and PACF for AR(p) processes, where the PACF gives us 'better' information than the ACF.

Indeed, it is possible to define something like partial cross-period autocorrelation. Still, it turns out to become relatively complex. The main difficulty is to describe properly what means 'in between' for $Y_{d,s}$ and $Y_{d-k,l}$ as motivated for the standard partial autocorrelation. In fact 'in

[12] From the statistical point of view the Gramian $\mathcal{X}_s'\mathcal{X}_s$ of the OLS estimator in (13) has no full rank, thus $(\mathcal{X}_s'\mathcal{X}_s)^{-1}$ is not determined. In the $s = S-1$ situation, the rank of $\mathcal{X}_s'\mathcal{X}_s$ is only 7 instead of the full rank 8. Thus, we are left with 7 effective parameters.

between' should be read as 'conditioned on' because by definition, a partial autocorrelation is a correlation that is conditioned on something.

To make this clearer, we first define a partial correlation between X and Y conditioned on $\boldsymbol{Z} = (Z_1, \ldots, Z_M)'$:

$$\rho_{X,Y|\boldsymbol{Z}} = \mathbb{C}\text{or}(X, Y|\boldsymbol{Z}). \tag{68}$$

If we suppose for a moment that $X = Z_t$, $Y = Z_{t-3}$ and $\boldsymbol{Z} = (Z_{t-1}, Z_{t-2})'$ then we are exactly in the definition of the partial autocorrelation as in equation (61) for $k = 3$ (or $\varphi_{t,3} = \mathbb{C}\text{or}(Z_t, Z_{t-3}|Z_{t-1}, Z_{t-2})$).

Now assume that our actual model is the simple AR(1) (see (4), $Y_{d,s} = \phi_{s,0} + \phi_{s,1} Y_{d-1,s} + \varepsilon_{d,s}$) which depends linearly on $Y_{d-1,s}$. If the question is, should we include $Y_{d-1,S-1}$ as a new term into the model, then the object we should evaluate is $\rho_{Y_{d,s},Y_{d-1,S-1}|Y_{d-1,s}} = \mathbb{C}\text{or}(Y_{d,s}, Y_{d-1,S-1}|Y_{d-1,s})$. If the corresponding sample estimate $\widehat{\rho}_{Y_{d,s},Y_{d-1,S-1}|Y_{d-1,s}}$ is significantly different from zero, this is a clear indication that including $Y_{d-1,S-1}$ improves our modelling performance (and thus the forecasting performance).

In general, the computation of confidence intervals for $\widehat{\rho}_{Y_{d,s},Y_{d-1,l}|Y_{d-1,s}}$ is relatively complicated. Still, if $\widehat{\rho}_{Y_{d,s},Y_{d-1,l}|Y_{d-1,s}}$ is large (in absolute terms), we have good hint that it is worth to include $Y_{d-1,S-1}$ as regressor in the electricity price model. Consequently, a new model could be compared in a statistical sense with another model (esp. the one) or we can simply perform a forecasting study with the new model and compare the model performance.

In Figure 20, we see sample cross-period autocorrelations for the day-ahead EPEX data. In 20(a), we observe $\widehat{\rho}_{Y_{d,s},Y_{d-1,l}|Y_{d-1,s}}$ which is the case that we just discussed, i.e. we condition on $Y_{d-1,s}$. We see that if we condition on $Y_{d-1,s}$, the correlations as seen in Figure 18 tend to become smaller. Still, we observe that for the later periods of l (e.g. $l = S - 1 = 23$ or also $l = 22$), they are relatively large. So the corresponding elements tend to carry some relevant information. Moreover, the diagonal elements ($s = l$), the ones that we condition on, are essentially zero. This matches well statistical theory.

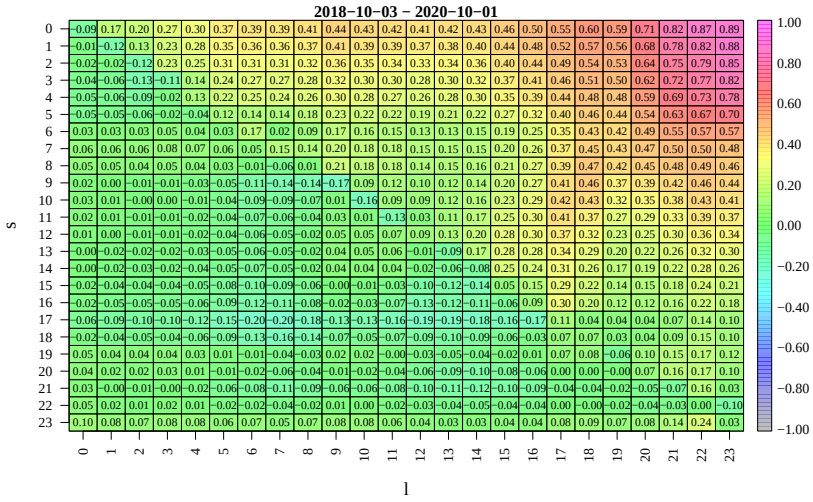
In Figure 20(b), we condition on $Y_{d-1,S-1}$ and the situation is somehow opposite. Again, the overall correlations tend to reduce. The last column has correlations that are close to zero, but close to the diagonal higher values remain. Finally, in Figure 20(c) where we condition on both cases, most of the correlation structure seems to be captured.

However, this is not the end of the story concerning model building. The partial (68) is a suitable building block in general when analysing new potential regressors in a linear model framework. In Figure (20) we illustrated that it is important to condition on a regressor to take into account for information that is already considered. Thus, if we take the position that e.g. the **expert** model (7) or the **expert.last** (67) model is a suitable model then we have to condition on all information that is already included into the model, for proper evaluation. Thus, for model (67) we should condition on $\boldsymbol{Z}_t = (Y_{d-1,s}, Y_{d-2,s}, Y_{d-7,s}, Y_{d-1,S-1}, \text{DoW}_d^1, \text{DoW}_d^6, \text{DoW}_d^7)'$ when doing the correlation analysis with other regressors. This corresponding exercise is illustrated in Figure (21). There we see no clear remaining information is left. However, we can observe same cross-period lags that are worth to try out in a forecasting model.

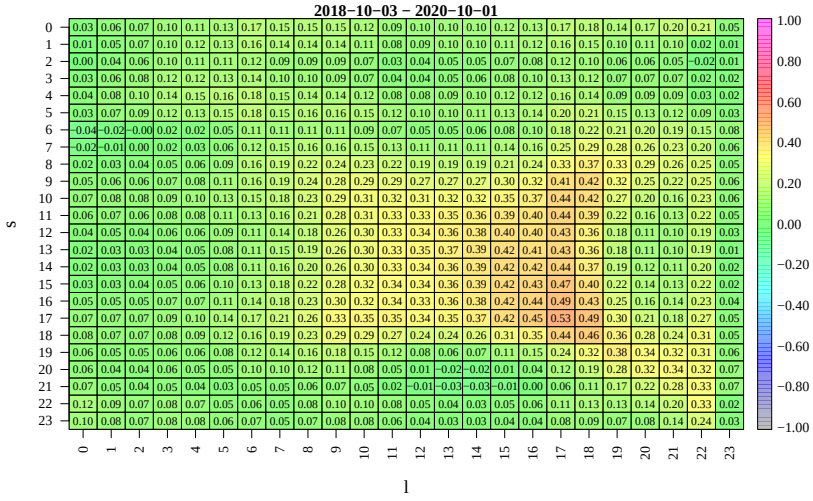
6.5 Tasks

Hints and the corresponding programming part are provide in Section R.5 in `R` and in Section py.3 in `python`.

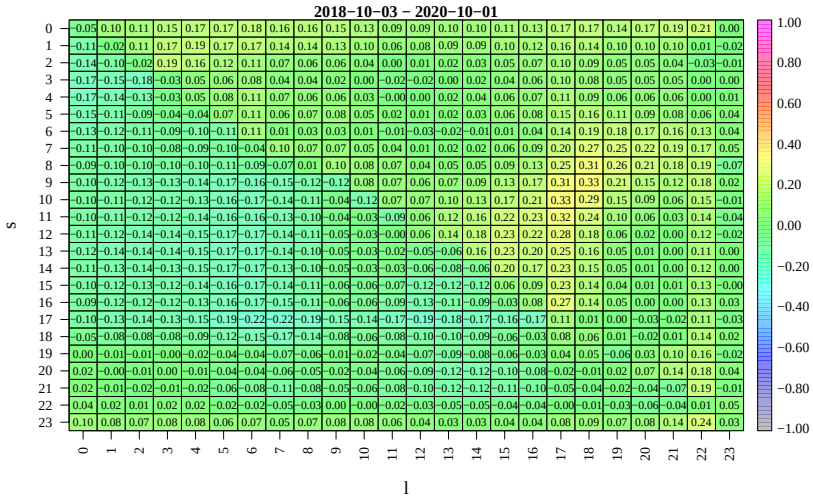
1. Perform a forecasting study for the expert type models (36) with $\mathbb{W} = \{1, 6, 7\}$ and
   a) $\mathbb{L} = \{1, 7\}$
   b) $\mathbb{L} = \{1, 2, 7\}$
   c) $\mathbb{L} = \{1, 2, 6, 7\}$
   d) $\mathbb{L} = \{1, 2, 3, 5, 6, 7\}$
2. Write down the matrix representation of $\mathcal{X}_s$ as in (10) for the new **expert.last** model (67). Which columns do not depend on s?



(a) Sample cross-period partial autocorrelations $\widehat{\rho}_{Y_{d,s},Y_{d-1,l}|Y_{d-1,s}}$.



(b) Sample cross-period partial autocorrelations $\widehat{\rho}_{Y_{d,s},Y_{d-1,l}|Y_{d-1,S-1}}$.



(c) Sample cross-period partial autocorrelations $\widehat{\rho}_{Y_{d,s},Y_{d-1,l}|Y_{d-1,s},Y_{d-1,S-1}}$.

Figure 20: Several sample cross-period partial autocorrelations for day-ahead EPEX data with $S = 24$.

3. Code a `forecast.expert.last` model function that performs the forecast of model (67). You might use the function `forecast.expert` as a basis. Using a forecasting study, compare the model's performance with the **expert** model (7).

4. Consider Figure 22. It shows a network visualisation of a linear model and the same inputs as model 19. However, the corresponding graph is fully connected. Define the

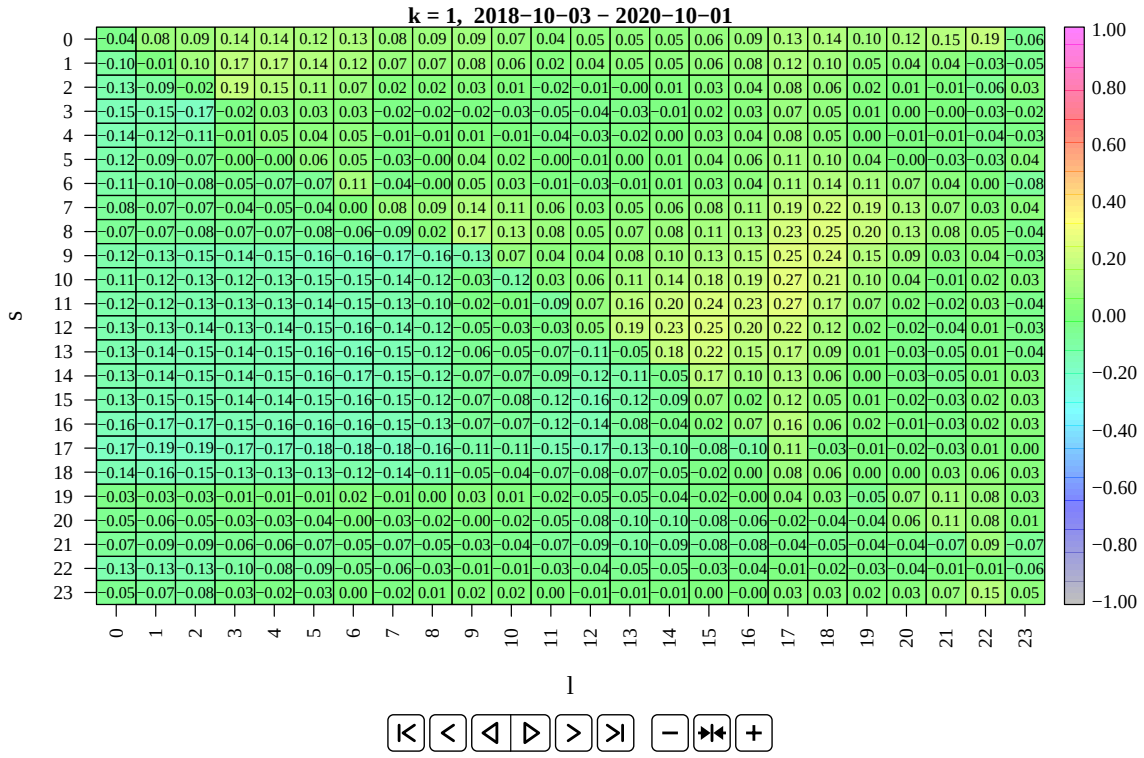


Figure 21: Sample conditional correlations $\widehat{\mathbb{C}\text{or}}(Y_{d,s}, Y_{d-k,l}|\boldsymbol{Z}_t)$ with $\boldsymbol{Z}_t = (Y_{d-1,s}, Y_{d-2,s}, Y_{d-7,s}, Y_{d-1,S-1}, \text{DoW}_d^1, \text{DoW}_d^6, \text{DoW}_d^7)'$ as the regressors of the model (67).

corresponding model (i.e. write down the model equation) and determine the number of parameters.

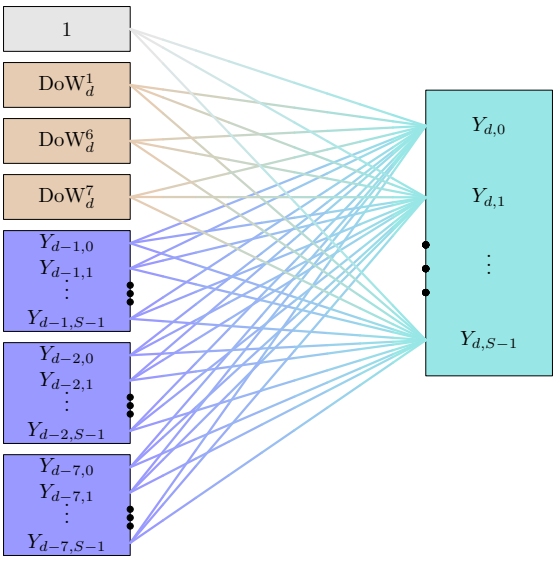


Figure 22: Network visualisation of model an extension of the expert model.

5. Create partial correlation plots as in 20(a), 20(b) and 20(c).

7 Seasonal/periodic structures

We have seen that electricity prices exhibit seasonal pattern. Mainly there are three different seasonal cycles that are relevant for us, the daily, the weekly and the annual ones. The same seasonal cycles occur for other energy related time series, like electricity load or wind and solar power production. However, purely meteorological influenced components like wind and solar power production have no weekly seasonality but daily and annual ones. In general, it can help in the modelling procedure to distinguish between meteorological and human-impact seasonality. The energy modelling and forecasting objects that are only influenced by meteorological impacts have relatively smooth regular cycles. As mentioned, meteorological components cycles have usually a periodicity of a day and a year (esp. wind and solar), rarely we have moon-based natural cycles as in tidal power plants. Note that seasonal effects of shorter frequencies (like daily seasonality with frequency of 1 day) can vary over the time of a longer period (like annual seasonality with frequency of approx. 365.24 days). An obvious example is the solar power generation. It is clear that solar power has daily cycles. During the night the sun does not shine, whereas during the day the sun shines and produces photovoltaic energy. However, we also observe annual cycles. So on the northern sphere we have more sun during summer than during winter. This goes hand in hand with the movement of the sunrise and sunset. So it is clear that the daily impact varies over the year. This we will call seasonal interactions, so the daily seasonality is varying over the year. In solar power, modelling this effect is widely known and captured using so-called 'clear sky models'.

We do not only have supply side impacts from wind and solar, but also demand side impacts, especially due to the temperature. Depending on the temperature, the usage of energy for heating and cooling varies and influences the electricity demand.

All these meteorological impacts influence the electricity supply and demand balance, and thus the electricity price. On top, we have human made seasonal effects. These are mainly the weekly effects, especially the effect of the weekend. Moreover, the annual electricity demand cycle shows, next to meteorological impacts (e.g. due to temperature), clear human activity impacts. For instance, we observe in many western countries a clear reduction in energy demand within the Christmas and New Year's Day holiday period. Note that in some countries other calendar cycle effects can occur as well, so for instance in Muslim countries we may observe Islamic calendar effects which do not match the standard year cycle. Some public holiday effects could be regarded as seasonal, but within this lecture we ignore the impact of public holiday effects. We now turn to the explicit modelling of seasonal effects. First, we restrict ourself to weekly effects.

7.1 Weekly seasonal structure and the expert model

As we considered so far only period-wise price models for the electricity prices $(Y_{d,s})_{d\in\mathbb{Z}}$, $Y_{d,s}$ exhibits no daily cycles, but potentially weekly and annual cycles. Note that this does not mean that there are no daily seasonal effects. They are only not explicitly modelled, as the general modelling, estimation, and forecasting design implies that the price characteristics of every period of the day are different.

In Figure 23, we see a plot of the sample mean prices of every hour of the week for the German EPEX price. So we see the hourly electricity price for each day of the week. We clearly observe that overall the electricity price on the weekend is smaller than during the week. This effect is usually referred as weekend effect. Still, there are some differences between the Saturday and the Sunday. Therefore, it is often suitable to include so called day-of-the-week effects to capture these differences. Furthermore, we observe that during the night the difference between the working days and the weekends is not as large as during the day. We even see that during some weekend hours the prices are higher than during the night. Furthermore, we can see the so called transition effects. During the night/morning hours of Monday the electricity price seems to be clearly smaller than during the other working days. Similarly, on Friday afternoon/evening the electricity prices differ from the working week as well. These transition effects can be well explained by the human activity behaviour. Still, we see that the day-of-the-week effects might differ depending on the hour of the day. The same pattern is known for the electricity demand/consumption. This is where the pattern comes from.

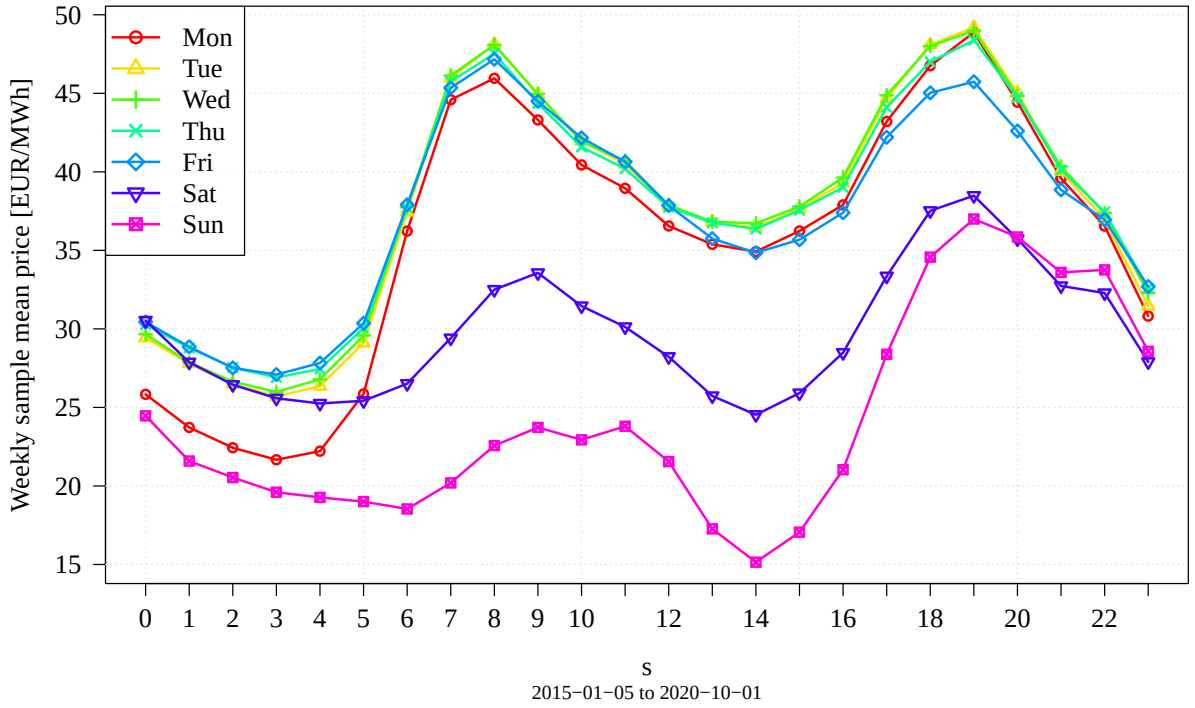


Figure 23: Weekly period-wise sample mean for EPEX price data.

In the expert type models that we considered so far we already included dummies, which capture the most relevant weekly effects. Now, we have a closer look at the short-term seasonal effects, so daily and weekly structure in the electricity prices. Therefore, we recap the expert model (7).

$$Y_{d,s} = \beta_{s,0} + \beta_{s,1} Y_{d-1,s} + \beta_{s,2} Y_{d-2,s} + \beta_{s,3} Y_{d-7,s} + \beta_{s,4} \mathrm{DoW}_d^1 + \beta_{s,5} \mathrm{DoW}_d^6 + \beta_{s,6} \mathrm{DoW}_d^7 + \varepsilon_{d,s} \quad (69)$$

There are Saturday, Sunday and Monday dummies. The Saturday dummy is negative, and describes the decrease from the Friday price level to the Saturday price level. The Sunday dummy, again shifts the price level from the Saturday level to the Sunday level. And finally the Monday dummy then shifts the the price level from the Sunday level to the Monday level. The expert model (69) is a periodically stationary model[13] with periodicity 7. This means that all statistical marginal properties (e.g. mean, standard deviation, density) repeat every 7 observations. In other words, the mean on all Sundays is the same, but can be different from the mean of all Mondays.

As illustrative example, we compute the mean of model (69). Therefore, we evaluate the 7 days of the week.

$$\begin{aligned}\mathbb{E}(Y_{d,s}) &= \mathbb{E}(\beta_{s,0} + \beta_{s,1} Y_{d-1,s} + \beta_{s,2} Y_{d-2,s} + \beta_{s,3} Y_{d-7,s} + \beta_{s,4} \mathrm{DoW}_d^1 + \beta_{s,5} \mathrm{DoW}_d^6 + \beta_{s,6} \mathrm{DoW}_d^7 + \varepsilon_{d,s}) \\ &= \beta_{s,0} + \beta_{s,1} \mathbb{E}(Y_{d-1,s}) + \beta_{s,2} \mathbb{E}(Y_{d-2,s}) + \beta_{s,3} \underbrace{\mathbb{E}(Y_{d-7,s})}_{=\mathbb{E}(Y_{d,s})} + \beta_{s,4} \mathrm{DoW}_d^1 + \beta_{s,5} \mathrm{DoW}_d^6 + \beta_{s,6} \mathrm{DoW}_d^7\end{aligned}$$

If d is a Monday, then we define $m_{s,1} = \mathbb{E}(Y_{d,s})$, and thus $m_{s,2} = \mathbb{E}(Y_{d+1,s}), \ldots, m_{s,7} = \mathbb{E}(Y_{d+6,s})$. This turns to

$$m_{s,1} = \beta_{s,0} + \beta_{s,1} m_{s,7} + \beta_{s,2} m_{s,6} + \beta_{s,3} m_{s,1} + \beta_{s,4}.$$

Note that for d being a Monday only the Monday dummy with parameter $\beta_{s,4}$ is active. We can

[13] Strictly the model is only periodically stationary under specific parameter combinations. But electricity price models satisfy almost always these conditions.

do now the same for d from Tuesday up to Sunday. This gives the system of 7 equations:

$$\begin{aligned}
m_{s,1} &= \beta_{s,0} + \beta_{s,1} m_{s,7} + \beta_{s,2} m_{s,6} + \beta_{s,3} m_{s,1} + \beta_{s,4} \\
m_{s,2} &= \beta_{s,0} + \beta_{s,1} m_{s,1} + \beta_{s,2} m_{s,7} + \beta_{s,3} m_{s,2} \\
m_{s,3} &= \beta_{s,0} + \beta_{s,1} m_{s,2} + \beta_{s,2} m_{s,1} + \beta_{s,3} m_{s,3} \\
m_{s,4} &= \beta_{s,0} + \beta_{s,1} m_{s,3} + \beta_{s,2} m_{s,2} + \beta_{s,3} m_{s,4} \\
m_{s,5} &= \beta_{s,0} + \beta_{s,1} m_{s,4} + \beta_{s,2} m_{s,3} + \beta_{s,3} m_{s,5} \\
m_{s,6} &= \beta_{s,0} + \beta_{s,1} m_{s,5} + \beta_{s,2} m_{s,4} + \beta_{s,3} m_{s,6} + \beta_{s,5} \\
m_{s,7} &= \beta_{s,0} + \beta_{s,1} m_{s,6} + \beta_{s,2} m_{s,5} + \beta_{s,3} m_{s,7} + \beta_{s,6}
\end{aligned}$$

This system can be rewritten to

$$\begin{pmatrix}
1-\beta_{s,3} & 0 & 0 & 0 & 0 & -\beta_{s,2} & -\beta_{s,1} \\
-\beta_{s,1} & 1-\beta_{s,3} & 0 & 0 & 0 & 0 & -\beta_{s,2} \\
-\beta_{s,2} & -\beta_{s,1} & 1-\beta_{s,3} & 0 & 0 & 0 & 0 \\
0 & -\beta_{s,2} & -\beta_{s,1} & 1-\beta_{s,3} & 0 & 0 & 0 \\
0 & 0 & -\beta_{s,2} & -\beta_{s,1} & 1-\beta_{s,3} & 0 & 0 \\
0 & 0 & 0 & -\beta_{s,2} & -\beta_{s,1} & 1-\beta_{s,3} & 0 \\
0 & 0 & 0 & 0 & -\beta_{s,2} & -\beta_{s,1} & 1-\beta_{s,3}
\end{pmatrix}
\begin{pmatrix} m_{s,1} \\ m_{s,2} \\ m_{s,3} \\ m_{s,4} \\ m_{s,5} \\ m_{s,6} \\ m_{s,7} \end{pmatrix}
=
\begin{pmatrix} \beta_{s,0} + \beta_{s,4} \\ \beta_{s,0} \\ \beta_{s,0} \\ \beta_{s,0} \\ \beta_{s,0} \\ \beta_{s,0} + \beta_{s,5} \\ \beta_{s,0} + \beta_{s,6} \end{pmatrix} \tag{70}$$

which can be written as

$$\boldsymbol{A}_s \boldsymbol{m}_s = \boldsymbol{c}_s.$$

So we have an explicit solution for the mean of the expert model given by $\boldsymbol{m}_s = \boldsymbol{A}_s^{-1} \boldsymbol{c}_s$. This we can easily compute for all periods s of the day.

In Figure 24, the mean of the estimated expert model is given and compared with the sample mean using German EPEX data. We see that the overall structure is well captured, so we have a smaller electricity price for the weekend than during the week. Some effects like the Friday afternoon/evening transition effect are not captured well. To include such an effect we can use a model like

$$\begin{aligned}
Y_{d,s} =& \beta_{s,0} + \beta_{s,1} Y_{d-1,s} + \beta_{s,2} Y_{d-2,s} + \beta_{s,3} Y_{d-7,s} \\
&+ \begin{cases}
\beta_{s,4} \mathrm{DoW}_d^1 + \beta_{s,6} \mathrm{DoW}_d^6 + \beta_{s,7} \mathrm{DoW}_d^7 + \varepsilon_{d,s} & , s \in \{0, \dots, S/2-1\} \\
\beta_{s,4} \mathrm{DoW}_d^1 + \beta_{s,5} \mathrm{DoW}_d^5 + \beta_{s,6} \mathrm{DoW}_d^6 + \beta_{s,7} \mathrm{DoW}_d^7 + \varepsilon_{d,s} & , s \in \{S/2, \dots, S-1\}
\end{cases}.
\end{aligned} \tag{71}$$

Model (71) has the same structure as the model before, but for the second half of the day we include an additional Friday dummy effect. Thus, for these hours, the Friday effect should be covered in a better way. In contrast, the first hours are the same. Obviously, other models can be designed as well. We can evaluate the suitability by comparing sample characteristics with the model characteristics or alternatively use out-of-sample forecasting studies with statistical evaluation criteria to justify the model choice.

Note that for more complex model characterization like expert.last model (67), the explicit computation of the mean is more complicated. However, using Monte-Carlo simulation techniques we can compute an approximation of mean (and other distributional properties) of any model with arbitrary small accuracy.

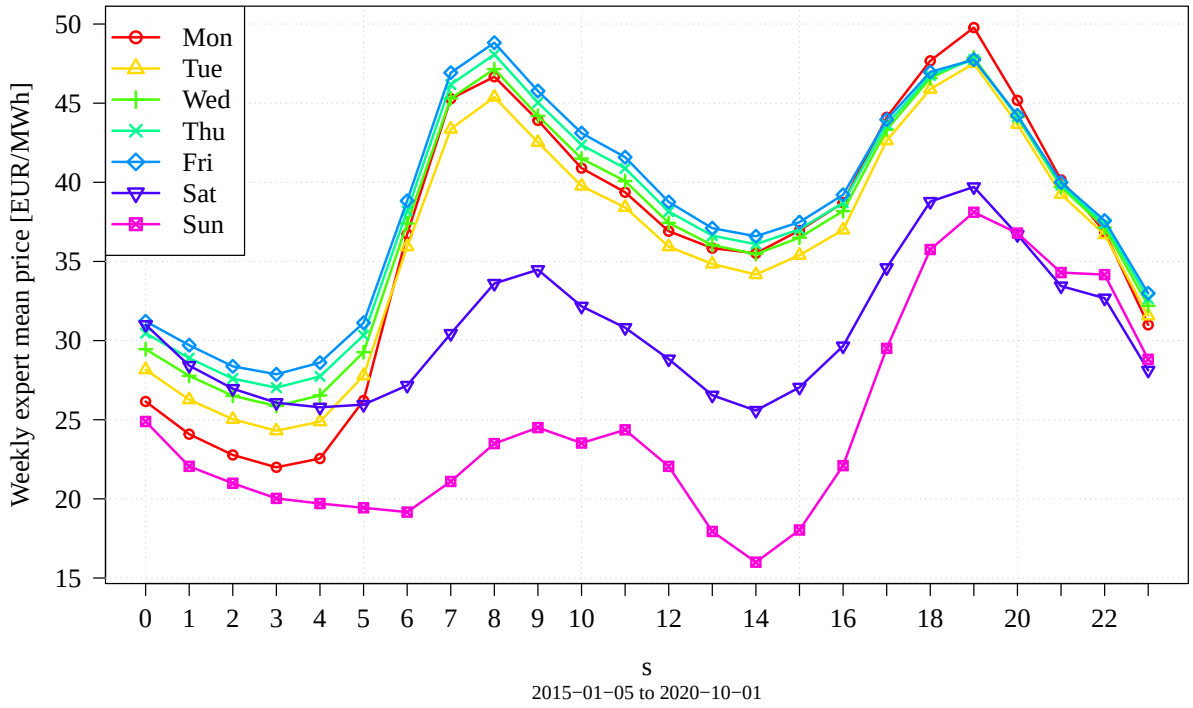
As mentioned, expert model (69) is a periodically stationary time series model. Such a model can be rewritten in periodic parameter representation. The parameter impact is due to the week-day dummies which act in the intercept. So the intercept (or mean component or trend component) is the only periodic parameter in this model. Hence (69) can be rewritten as

$$Y_{d,s} = \beta_{s,0}(d) + \beta_{s,1} Y_{d-1,s} + \beta_{s,2} Y_{d-2,s} + \beta_{s,3} Y_{d-7,s} + \varepsilon_{d,s} \tag{72}$$

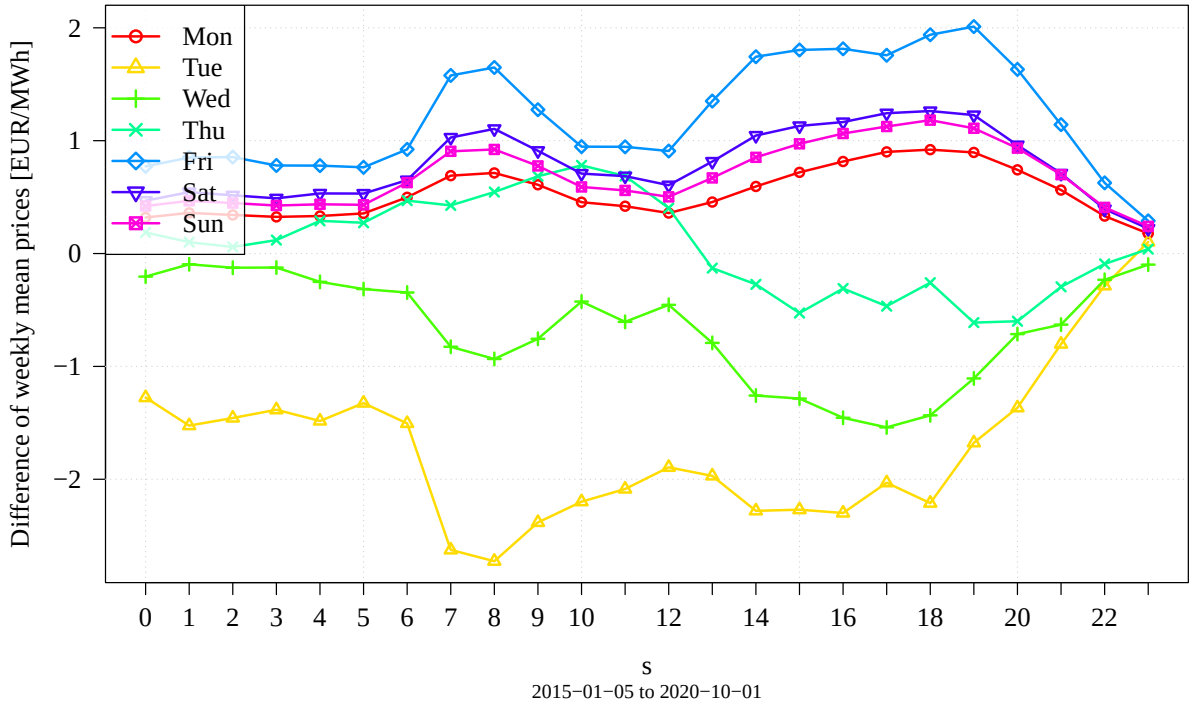
where

$$\beta_{s,0}(d) = \beta_{s,0,1} + \beta_{s,0,2} \mathrm{DoW}_d^1 + \beta_{s,0,3} \mathrm{DoW}_d^6 + \beta_{s,0,4} \mathrm{DoW}_d^7.$$

So $\beta_{s,0}(d)$ is a periodic parameter that has periodicity 7. Its values vary over time, still on Tuesday, Wednesday, Thursday and Friday they are the same. But remember, this does not



(a) Weekly mean expert



(b) Difference of weekly expert model mean and sample mean.

Figure 24: Weekly mean of expert model (69) and difference to sample mean

mean that the mean is the the same for these days. It is easy to see that expert type models with a different weekday specification (as e.g. (71)) have only one periodic parameter as well, namely the intercept.

There are several other ways to generalize the periodic parameter feature. [UNW16] or [ZSH15] propose periodic effects on the autoregressive parameters. So not only the intercept is periodically varying over time, but also selected autoregressive parameters.

A very simple model version is given by

$$Y_{d,s} = \beta_{s,0}(d) + \beta_{s,1}(d)Y_{d-1,s} + \beta_{s,2}Y_{d-2,s} + \beta_{s,3}Y_{d-7,s} + \varepsilon_{d,s} \tag{73}$$

where

$$\beta_{s,0}(d) = \beta_{s,0,1} + \beta_{s,0,2}\mathrm{DoW}_d^1 + \beta_{s,0,3}\mathrm{DoW}_d^6 + \beta_{s,0,4}\mathrm{DoW}_d^7 \tag{74}$$

$$\beta_{s,1}(d) = \beta_{s,1,1} + \beta_{s,1,2}\mathrm{DoW}_d^1 + \beta_{s,1,3}\mathrm{DoW}_d^6 + \beta_{s,1,4}\mathrm{DoW}_d^7. \tag{75}$$

Here, additionally to the intercept $\beta_{s,0}$, the autoregressive parameter of lag 1 $\beta_{s,1}$ is time varying in a periodic manner. Model (73) is very similarly used in [UNW16].

Using some mathematics, we can show that the corresponding model is periodically stationary as well. The mean of model (73) is given by the solution of

$$\begin{pmatrix} 1-\beta_{s,3} & 0 & 0 & 0 & 0 & -\beta_{s,2} & -\beta_{s,1,1}-\beta_{s,1,2} \\ -\beta_{s,1,1} & 1-\beta_{s,3} & 0 & 0 & 0 & 0 & -\beta_{s,2} \\ -\beta_{s,2} & -\beta_{s,1,1} & 1-\beta_{s,3} & 0 & 0 & 0 & 0 \\ 0 & -\beta_{s,2} & -\beta_{s,1,1} & 1-\beta_{s,3} & 0 & 0 & 0 \\ 0 & 0 & -\beta_{s,2} & -\beta_{s,1,1} & 1-\beta_{s,3} & 0 & 0 \\ 0 & 0 & 0 & -\beta_{s,2} & -\beta_{s,1,1}-\beta_{s,1,3} & 1-\beta_{s,3} & 0 \\ 0 & 0 & 0 & 0 & -\beta_{s,2} & -\beta_{s,1,1}-\beta_{s,1,4} & 1-\beta_{s,3} \end{pmatrix} \begin{pmatrix} m_{s,1} \\ m_{s,2} \\ m_{s,3} \\ m_{s,4} \\ m_{s,5} \\ m_{s,6} \\ m_{s,7} \end{pmatrix} = \begin{pmatrix} \beta_{s,0,1}+\beta_{s,0,2} \\ \beta_{s,0,1} \\ \beta_{s,0,1} \\ \beta_{s,0,1} \\ \beta_{s,0,1} \\ \beta_{s,0,1}+\beta_{s,0,3} \\ \beta_{s,0,1}+\beta_{s,0,4} \end{pmatrix} \tag{76}$$

For the extended expert models where we include cross-hour dependencies (as e.g. in (67)), we have a more complex structure. The corresponding means cannot be evaluated by solving S systems of 7 equations. However, it is still possible to compute the mean explicitly. Therefore, we have to solve either a large system of $7S$ equations or alternatively $S-1$ systems of 14 equations and a single one with 7 equations. We illustrate the latter way. For instance, for the model (67) we have the mean

$$\begin{aligned} \mathbb{E}(Y_{d,s}) = \beta_{s,0} + \beta_{s,1}\mathbb{E}(Y_{d-1,s}) + \beta_{s,2}\mathbb{E}(Y_{d-2,s}) + \beta_{s,3}\underbrace{\mathbb{E}(Y_{d-7,s})}_{=\mathbb{E}(Y_{d,s})} + \beta_{s,S,1}\mathbb{E}(Y_{d-1,S-1}) \\ + \beta_{s,5}\mathrm{DoW}_d^1 + \beta_{s,6}\mathrm{DoW}_d^6 + \beta_{s,7}\mathrm{DoW}_d^7 + \varepsilon_{d,s}. \end{aligned} \tag{77}$$

This yields us the system of equations

$$\begin{aligned} m_{s,1} &= \beta_{s,0} + \beta_{s,1}m_{s,7} + \beta_{s,2}m_{s,6} + \beta_{s,3}m_{s,1} + \beta_{s,S-1,1}m_{S-1,7} + \beta_{s,5} \\ m_{s,2} &= \beta_{s,0} + \beta_{s,1}m_{s,1} + \beta_{s,2}m_{s,7} + \beta_{s,3}m_{s,2} + \beta_{s,S-1,1}m_{S-1,1} \\ m_{s,3} &= \beta_{s,0} + \beta_{s,1}m_{s,2} + \beta_{s,2}m_{s,1} + \beta_{s,3}m_{s,3} + \beta_{s,S-1,1}m_{S-1,2} \\ m_{s,4} &= \beta_{s,0} + \beta_{s,1}m_{s,3} + \beta_{s,2}m_{s,2} + \beta_{s,3}m_{s,4} + \beta_{s,S-1,1}m_{S-1,3} \\ m_{s,5} &= \beta_{s,0} + \beta_{s,1}m_{s,4} + \beta_{s,2}m_{s,3} + \beta_{s,3}m_{s,5} + \beta_{s,S-1,1}m_{S-1,4} \\ m_{s,6} &= \beta_{s,0} + \beta_{s,1}m_{s,5} + \beta_{s,2}m_{s,4} + \beta_{s,3}m_{s,6} + \beta_{s,S-1,1}m_{S-1,5} + \beta_{s,6} \\ m_{s,7} &= \beta_{s,0} + \beta_{s,1}m_{s,6} + \beta_{s,2}m_{s,5} + \beta_{s,3}m_{s,7} + \beta_{s,S-1,1}m_{S-1,6} + \beta_{s,7} \end{aligned}$$

for all $s \in \{0, \ldots, S-1\}$. Thus, we get systems for $s < S-1$

$$\begin{pmatrix} 1-\beta_{s,3} & 0 & 0 & 0 & 0 & -\beta_{s,2} & -\beta_{s,1} & 0 & 0 & 0 & 0 & 0 & 0 & -\beta_{s,S-1,1} \\ -\beta_{s,1} & 1-\beta_{s,3} & 0 & 0 & 0 & 0 & -\beta_{s,2} & -\beta_{s,S-1,1} & 0 & 0 & 0 & 0 & 0 & 0 \\ -\beta_{s,2} & -\beta_{s,1} & 1-\beta_{s,3} & 0 & 0 & 0 & 0 & 0 & -\beta_{s,S-1,1} & 0 & 0 & 0 & 0 & 0 \\ 0 & -\beta_{s,2} & -\beta_{s,1} & 1-\beta_{s,3} & 0 & 0 & 0 & 0 & 0 & -\beta_{s,S-1,1} & 0 & 0 & 0 & 0 \\ 0 & 0 & -\beta_{s,2} & -\beta_{s,1} & 1-\beta_{s,3} & 0 & 0 & 0 & 0 & 0 & -\beta_{s,S-1,1} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\beta_{s,2} & -\beta_{s,1} & 1-\beta_{s,3} & 0 & 0 & 0 & 0 & 0 & -\beta_{s,S-1,1} & 0 & 0 \\ 0 & 0 & 0 & 0 & -\beta_{s,2} & -\beta_{s,1} & 1-\beta_{s,3} & 0 & 0 & 0 & 0 & 0 & -\beta_{s,S-1,1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1-\beta_{S-1,3} & 0 & 0 & 0 & 0 & -\beta_{S-1,2} & -\beta_{S-1,1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\beta_{S-1,1} & 1-\beta_{S-1,3} & 0 & 0 & 0 & 0 & -\beta_{S-1,2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\beta_{S-1,2} & -\beta_{S-1,1} & 1-\beta_{S-1,3} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\beta_{S-1,2} & -\beta_{S-1,1} & 1-\beta_{S-1,3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\beta_{S-1,2} & -\beta_{S-1,1} & 1-\beta_{S-1,3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\beta_{S-1,2} & -\beta_{S-1,1} & 1-\beta_{S-1,3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\beta_{S-1,2} & -\beta_{S-1,1} & 1-\beta_{S-1,3} \end{pmatrix} \begin{pmatrix} m_{s,1} \\ m_{s,2} \\ m_{s,3} \\ m_{s,4} \\ m_{s,5} \\ m_{s,6} \\ m_{s,7} \\ m_{S-1,1} \\ m_{S-1,2} \\ m_{S-1,3} \\ m_{S-1,4} \\ m_{S-1,5} \\ m_{S-1,6} \\ m_{S-1,7} \end{pmatrix} = \begin{pmatrix} \beta_{s,0}+\beta_{s,5} \\ \beta_{s,0} \\ \beta_{s,0} \\ \beta_{s,0} \\ \beta_{s,0} \\ \beta_{s,0}+\beta_{s,6} \\ \beta_{s,0}+\beta_{s,7} \\ \beta_{S-1,0}+\beta_{S-1} \\ \beta_{S-1,0} \\ \beta_{S-1,0} \\ \beta_{S-1,0} \\ \beta_{S-1,0} \\ \beta_{S-1,0}+\beta_{S-1} \\ \beta_{S-1,0}+\beta_{S-1} \end{pmatrix} \tag{78}$$

which have to be solved.

These calculations above give us the feeling that we can calculate the expected mean of much more complicated expert models. Indeed, if we consider the general model

$$Y_{d,s} = \sum_{j=1}^{7} \beta_{s,j}\mathrm{DoW}_d^j + \sum_{k=1}^{p}\sum_{l\in\mathbb{S}} \beta_{s,k,j,l}\mathrm{DoW}_d^j Y_{d-k,l} + \varepsilon_{d,s} \tag{79}$$

we can solve the corresponding problem. To present the solution we define the index sets $I_7^{k,p} = \{n \in \{1, \ldots, p\} | (n-k)/7 \in \mathbb{Z}\}$ which contain the k-shifted multiples of 7 up to p,

e.g. we have $I_7^{5,21} = \{5, 12, 19\}$. Then we define the matrices $\boldsymbol{B}_{s,l} = (b_{s,l,i,j})_{(i,j)} \in \mathbb{R}^{7\times 7}$ by $b_{s,l,i,j} = \sum_{k\in I_7^{7+i-j,p}} \beta_{s,k,i,l}$ and $\boldsymbol{c}_s = (\beta_{s,1}, \ldots, \beta_{s,7})'$. However, as in (78) we can not solve the S systems independently as in (76). Instead we have to solve a large system with $7S$ equations. This is

$$\begin{pmatrix} \boldsymbol{I}_7 - \boldsymbol{B}_{0,0} & -\boldsymbol{B}_{0,1} & \cdots & -\boldsymbol{B}_{0,S-1} \\ -\boldsymbol{B}_{1,0} & \boldsymbol{I}_7 - \boldsymbol{B}_{1,1} & \cdots & -\boldsymbol{B}_{1,S-1} \\ \vdots & \vdots & \ddots & \vdots \\ -\boldsymbol{B}_{S-1,0} & -\boldsymbol{B}_{S-1,1} & \cdots & \boldsymbol{I}_7 - \boldsymbol{B}_{S-1,S-1} \end{pmatrix} \begin{pmatrix} \boldsymbol{m}_0 \\ \boldsymbol{m}_1 \\ \vdots \\ \boldsymbol{m}_{S-1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{c}_0 \\ \boldsymbol{c}_1 \\ \vdots \\ \boldsymbol{c}_{S-1} \end{pmatrix}$$

Note that this model with $7(pS + 1)$ total parameters and nests all expert type models covered so far.

Finally, we want to apply something, we have learned in the previous section. The seen Friday effect can also be evaluated using the conditional (or partial) correlation approach from the previous section. Suppose that the 'actual model' is the (67) and we are interested in the potential additional impact of a certain weekday dummy. Then, we can compute the corresponding sample conditional correlation of the 7 weekday dummies conditioned on the model regressors.

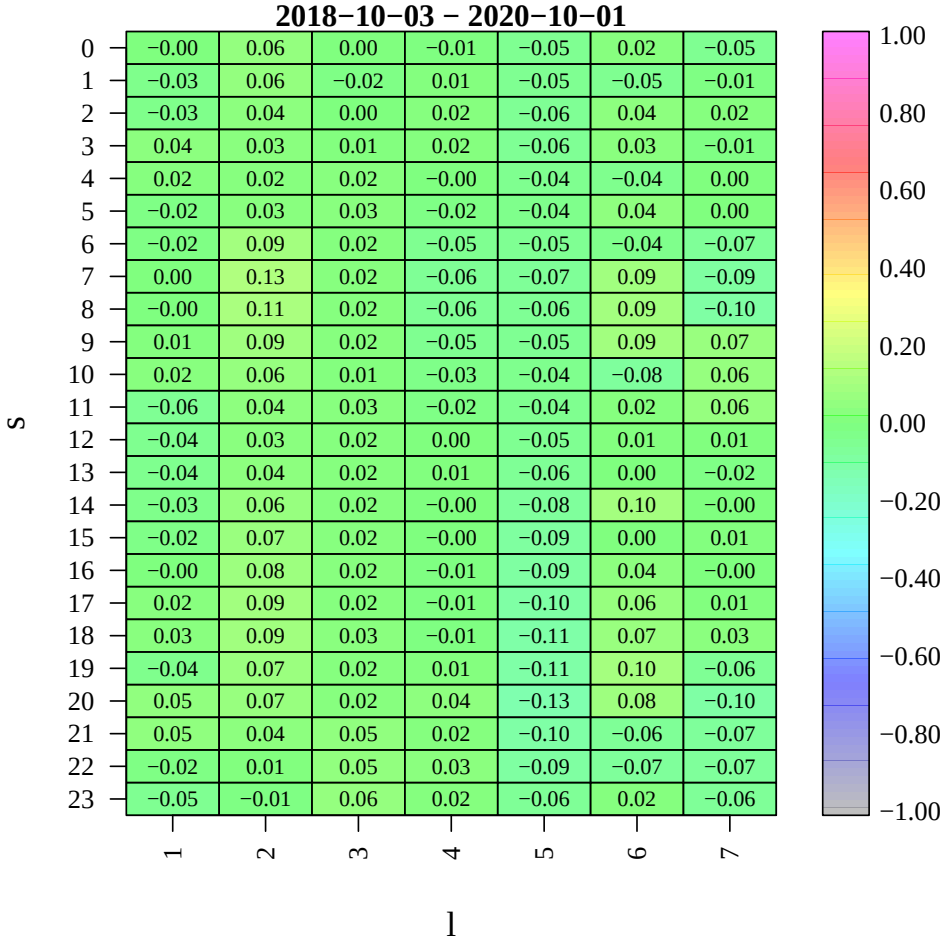


Figure 25: Sample conditional correlations $\widehat{\mathbb{C}\mathrm{or}}(Y_{d,s}, \mathrm{DoW}_d^l | \boldsymbol{X}_d)$ with $\boldsymbol{X}_d$ as the regressors of the model (67). (1=Mon, 2=Tue, ...)

These sample correlations are given in Figure 25. We clearly see that for Monday, Saturday and Sunday there is no effect at all, as we conditioned already on these dummies. Furthermore, we also see that there is a small Friday effect for the afternoon and evening hours. Additionally, we see that there might be a small Tuesday effect as well.

7.2 Annual seasonality

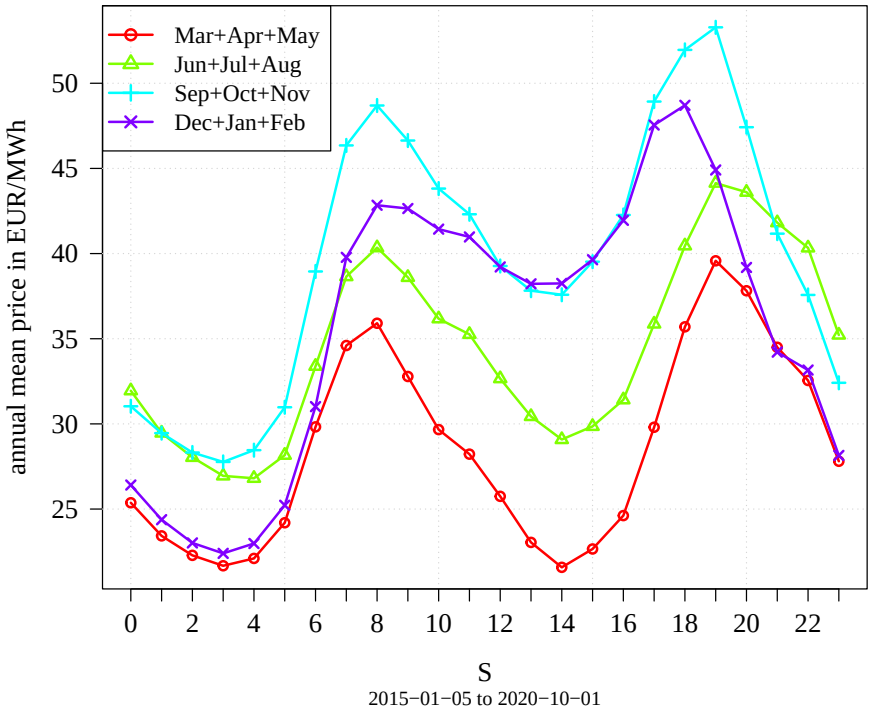
We have seen various methods to deal with the weekly seasonality. In principle, we can deal with annual seasonality in the same way. Still, there are important differences. Firstly, the period

of a year is quite large. Thus, we usually observe only a few cycles which makes it relatively difficult to learn about annual seasonal behaviour. Secondly, the strict meteorological year has a periodicity of about $A = 365.24$ days which is not an integer. Instead, a calendar year has always 365 or 366 days, depending on the fact if it is a leap year with the 29th of February or not. Thus, we have to deal with the meteorological annual effects and the calendar/date-based annual effects as mentioned in the introduction. Additionally, we have to think about seasonal interactions, especially between the daily and annual components. Finally, we have to think about spurious effects, a topic that is closely related to long term trend behaviour analysis.

The standard expert model that we considered so far has no annual seasonal component. Applications have shown that usually the annual cycle is present but the magnitude of the impact is not clear [NW16].

In electricity price modelling and forecasting, annual cycles are often ignored. Another popular alternative is a two-step estimation approach. There, the long-term trend and annual seasonal cycle is removed in a first step, and the residuals are modelled afterwards (e.g. with an expert type model). Here we do not consider these two-step approaches, even though they can be a suitable alternative. On the other hand, they tend to struggle stronger with the spurious regression problem. Moreover, the estimated parameters in both estimation steps are usually not correlated which induces a suboptimal model estimation.

To learn more about annual seasonality, we start in a simple way. Remember the periodic parameter representation (72) of the expert model. There, the periodic parameter $\beta_{s,0}(d)$ carries the weekly periodically varying information based on the corresponding weekdays. For annual periodicity modelling, we can proceed similarly and introduce calendar-based dummies. A very simple option is to introduce dummies based on the quarters (or 4 seasons). The four seasons are usually defined as spring (Mar, Apr, May), summer (Jun, Jul, Aug), fall (Sep, Oct, Nov) and winter (Dec, Jan, Feb). We call the corresponding dummies simply SoY_d^1 for spring, SoY_d^2 for summer, SoY_d^3 for fall and SoY_d^4 for winter. In Figure 26, we see the sample mean of a large data sample. There, we observe that during fall and partially during winter the electricity price



(a) Annual sample mean: four seasons

Figure 26: Annual sample mean for four seasons.

seems to be higher than during the rest of the year. So by looking at this plot it looks like taking annual effects into account makes sense.

An expert model that is generalized by such a four seasons dummy component is given by

$$Y_{d,s} = \beta_{s,0}(d) + \beta_{s,1} Y_{d-1,s} + \beta_{s,2} Y_{d-2,s} + \beta_{s,3} Y_{d-7,s} + \varepsilon_{d,s} \tag{80}$$

where

$$\beta_{s,0}(d) = \beta_{s,0,1} + \beta_{s,0,2}\mathrm{DoW}_d^1 + \beta_{s,0,3}\mathrm{DoW}_d^6 + \beta_{s,0,4}\mathrm{DoW}_d^7 + \beta_{s,0,5}\mathrm{SoY}_d^1 + \beta_{s,0,6}\mathrm{SoY}_d^2 + \beta_{s,0,7}\mathrm{SoY}_d^3. \quad (81)$$

Due to singularities, a winter dummy SoY_d^4 is not required, as $\beta_{s,0,1}$ takes automatically over the winter parameter. Therefore, compared to the expert model we require 3 additional parameters.

Note that the chosen 4-seasonal annual modelling approach is not necessarily a very good one. The selected partitioning could vary, by either reducing the number of months, increasing the number of months or grouping different months together. Furthermore, we do not have to restrict ourself to partitions based on monthly dummies even though this methodology is relatively popular in energy forecasting ([KGMF12] and [LNHW15]). As Figure 26 suggests, for some hours of the day (like the night hours) the difference between the seasons is very small. So it also might be suitable to adjust the seasonal specification dependent on the period of the day that is going to be modelled.

As mentioned, the modelling approach based on calendar dummies (esp. month-based) is relatively popular in energy forecasting. Still, strictly speaking the resulting model is usually not a periodically stationary model, as the February has sometimes 28 days and sometimes 29, so the periodicity gets disturbed due to the leap years. Moreover, month-based partitions of the year have the property that the seasons do not have the same length (e.g. summer has 30+31+31=92 days and fall has 30+31+30=91 days). This might be wanted, but might be impractical as well.

Moreover, there are modelling techniques that are not calendar-based but based on the meteorological annual periodicity of $A = 365.24$ days. These modelling approaches use a periodic basis to describe the annual periodicity of the parameters. So a time-varying parameter in a model, say $\xi(d)$ (like $\beta_{s,0}(d)$ in model (80)), is assumed to be periodic with periodicity A. Thus, $\xi(d) = \xi(d + kA)$ for any integer k. The function $\xi(d)$ is now parametrized using periodic basis function, in general given by

$$\xi(d) = \sum_{i=1}^{K} \xi_i B_i(d)$$

where B_i is an A-periodic basis function and K is the number of basis functions.

In general, there are two classes of periodic basis functions, global basis functions and local basis functions. We will shortly cover both approaches. An approach is called global if the support of a function is the complete function space. Otherwise, it is a local approach. So, e.g. an effect of a change of the structure in a single time point in winter influences directly the model effect in summer. In contrast, a local basis function approach would only influence the neighbouring winter time points.

Anyway, the global approaches seem to be more popular in recent energy modelling even though the local methods seem to be superior in terms of theoretical properties. Still, the most popular technique is the Fourier approximation which is a global basis function method. The reason for the popularity is likely the simplicity.

The Fourier approximation is based on the approximation of the true function by sine and cosine functions. In general, the basis functions are given by

$$\xi(d) = \xi_0 + \sum_{i=1}^{M} \xi_{i,1} \sin(2\pi i d/A) + \xi_{i,2} \cos(2\pi i d/A), \quad (82)$$

where M is the order of the Fourier approximation. The number of parameters is $K = 2M + 1$. In practice, usually a small order like $M = 1$ or $M = 2$ is often applied. The Fourier approach is a global basis function approach as the sine and cosine is almost everywhere non-zero. In Figure 27, the basis functions (without the constant term) are visualized.

A relatively popular local basis function approach is based on periodic B-splines. The explicit definition of the periodic B-splines is relatively complicated. The construction is described in [ZCA16] and is based on a partition of the periodic space (often called a set of knots).

In Figure **??**, B-spline basis functions are visualized using an equally spaced grid. There we see linear, quadratic and cubic B-splines with degrees 1, 2 and 3. The degree gives order

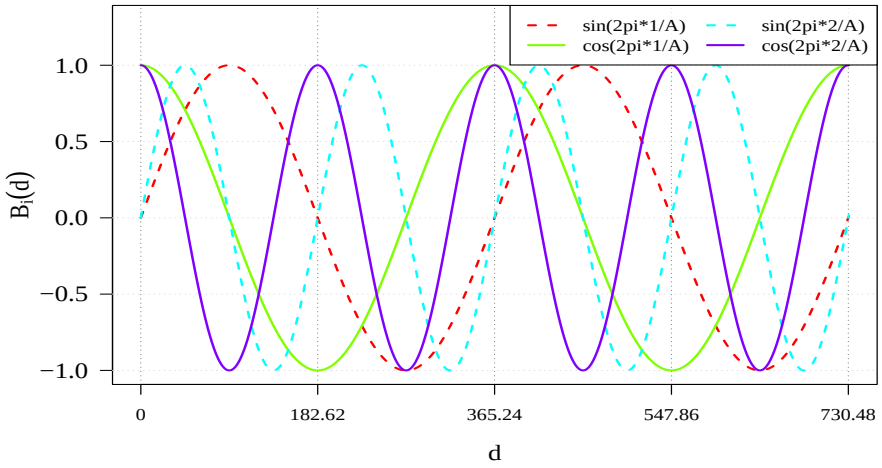


Figure 27: Fourier basis functions with periodicity A of order 3.

of the polynomials that are used to construct the B-spline. Moreover, the degree characterizes the smoothness of the basis. So a B-spline of degree p is $p-1$ times continuously differentiable. So the linear B-splines (degree = 1) are not differentiable, the quadratic B-splines are one time differentiable, etc. In Figure 28, we also observe that the B-spline basis functions sum up to 1. Thus, if we include a complete periodic B-spline basis as a regressor, we have to take this issue into account. This can be done by either deleting the constant term (the intercept) in the regression formulation or alternatively by deleting an arbitrary basis function of the periodic basis (which is usually the first or last basis function in the specification).

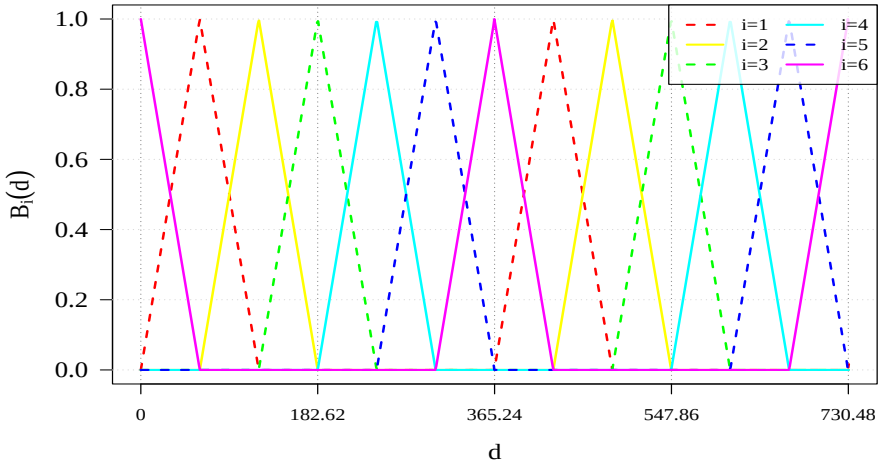
In Figure 28, equally spaced grids are used to construct the periodic B-splines. Thus, we use the same amount of parameters within every part of the seasonality. However, this can be relaxed. We could put more basis functions into a certain region where we expect more distinct changes within the period. In energy forecasting this might be very useful for the Christmas and New Year period around December and January, because we usually see a clear drop in the electricity demand/consumption/load during this holiday period which lasts only for a few days.

In the application of these annual effects, usually we mix them with the weekly periodic effects (as used in the expert model). Thus, the resulting model contains effects from the weekly periodicity (7) and the annual periodicity ($A = 365.24$). As the larger period $A = 365.24$ is not divisible by the smaller period 7, the resulting model is not periodically stationary.

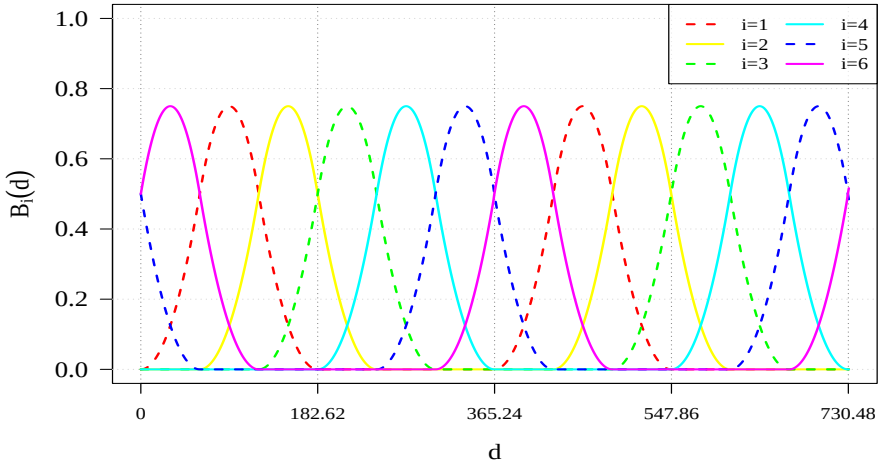
7.3 Tasks

Hints and the corresponding programming part are provide in Section R.6 in `R` and in Section py.4 in `python`.

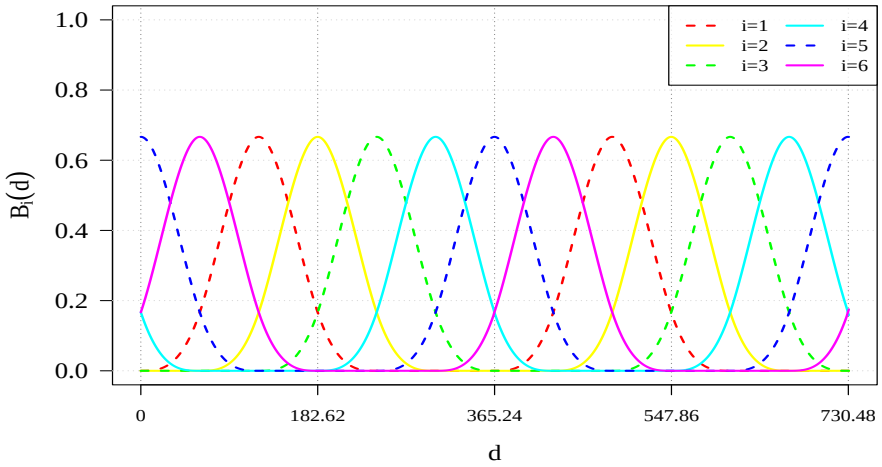
1. Implement model (71) and create plots, that correspond to those in Figure 24.
2. Implement model (73) and compute its corresponding mean.
3. Write down and implement an expert model generalization with an intercept that models the annual effect by monthly dummies, instead of 4-season dummies. Create a plot that is similar to this one in Figure 26(a) but used twelve months.
4. Implement an expert model that uses additionally periodic cubic B-splines with annual periodicity with 6 basis functions.
5. Using a forecasting study, decide whether it is better to use the 4-season dummies, monthly dummies, periodic cubic B-splines or none of them.



(a) Linear B-spline (degree = 1)



(b) Quadratic B-spline (degree = 2)



(c) Cubic B-spline (degree = 3)

Figure 28: Periodic B-spline basis functions with periodicity A on an equal distant grid with 6 basis functions within a period.

8 Models with external regressors

In this section, we are discussing the issue of external regressors. From the supply stack model (5) we know that several other external impacts such as electricity demand/load, wind and solar power generation can be used to explain the electricity price. The most popular external regressors in day-ahead electricity price forecasting is the electricity demand/load. In some countries (such as Germany) the renewable energy production is very important as well. The renewable energy production consists mainly of wind and solar power production. Those regressors are sometimes referred as fundamentals, the key characteristic is that they all depend on meteorologic conditions and potentially behavioural characteristics.

Moreover, we know from the supply stack model (5), that also fuel prices (esp. uranium, coal, natural gas, oil) and emission allowance price (CO_{2e}-price, carbon price) impact the merit order, and thus the electricity price. In literature is is relatively convenient to call this group of external regressors fuel prices which includes the CO_{2e}-price.

There are other external regressors such import and export flows, such as the available import and export capacity, planned unavailability of power plants (e.g. due to maintenance or recent outages), electricity prices of neighbouring countries such as their fundamental characteristics. However, here we are not considering these inputs. Also meteorologic data, like temperature may influence the electricity price. A smaller temperature will likely increase the must-run capacity of CHP power plants (see (5)) and this impact the electricity price.

In this section, we will consider for empirical applications the impacts of the fundamentals load, wind, and solar power by considering corresponding day-ahead forecast. We will split the wind power forecast in on-shore and off-shore wind. Further, we consider the fuel prices from natural gas, oil and (hard) coal[14], such as European Emission Allowances (EUA).

8.1 On fundamentals and their day-ahead forecasts.

From the statistical point of view we can deal with external regressors as with other inputs. However, next to problems concerning data reliability and missing data, the data availability in real time application is an important issue, especially for fundamentals like load, wind and solar. To get a first overview of the considered fundamental data, the series and their weekly sample means are visualised along with the electricity price data in Figures 29, 30 and 31, by noting that we separated the wind power production into onshore and offshore Wind.

We have the clear weekly pattern in the electricity consumption with lower consumption over the weekend. We also have annual seasonality in the electricity load. The consumption is higher in the colder winter season than in summer, potentially caused by higher illumination and electric heating. However, during the Christmas and New Years holiday period the load level is reduced significantly. For the renewables, we observe that the penetration seems partially to increase (slowly) over time. We also observe seasonal pattern for the wind and solar data. In winter we have more wind than in summer (but also more volatile during winter). For solar the situation is vice versa. Here, we even have a situation where we have small solar power production during winter, but a substantial production during summer. This is due to the changing sunrise and sunset time over the year. Moreover, there is a clear structural change in the solar data due to the clock change adjustment for the morning and evening hours in March and October. For the wind and solar data, there is no clear weekly pattern. Indeed, based on the meteorology there is no indication that the weather is impacted by the weekday. However, this is the actually generated wind and solar power production. Due to curtailment measures of renewable energy production units some weekly seasonalities might be plausible. For the temperature we see a clear annual cycle. By eyeballing this periodic behaviour looks close to sine- or cosine wave. In the considered climatologically short time range no clear effect of global warming is visible here.

Remember that we are modeling day-ahead markets where the auction takes place at noon for the next day values. So the electricity delivery is about 12 to 36 hours in the future. Thus, we cannot have fundamental data such as wind power production or electricity load available for this period. We can only work with day-ahead forecasts. In Europe, it is quite common to have forecasts of the most important features like electricity load, wind and solar power production

[14] specifically TTF Gas, Amsterdam Coal, Brent Oil

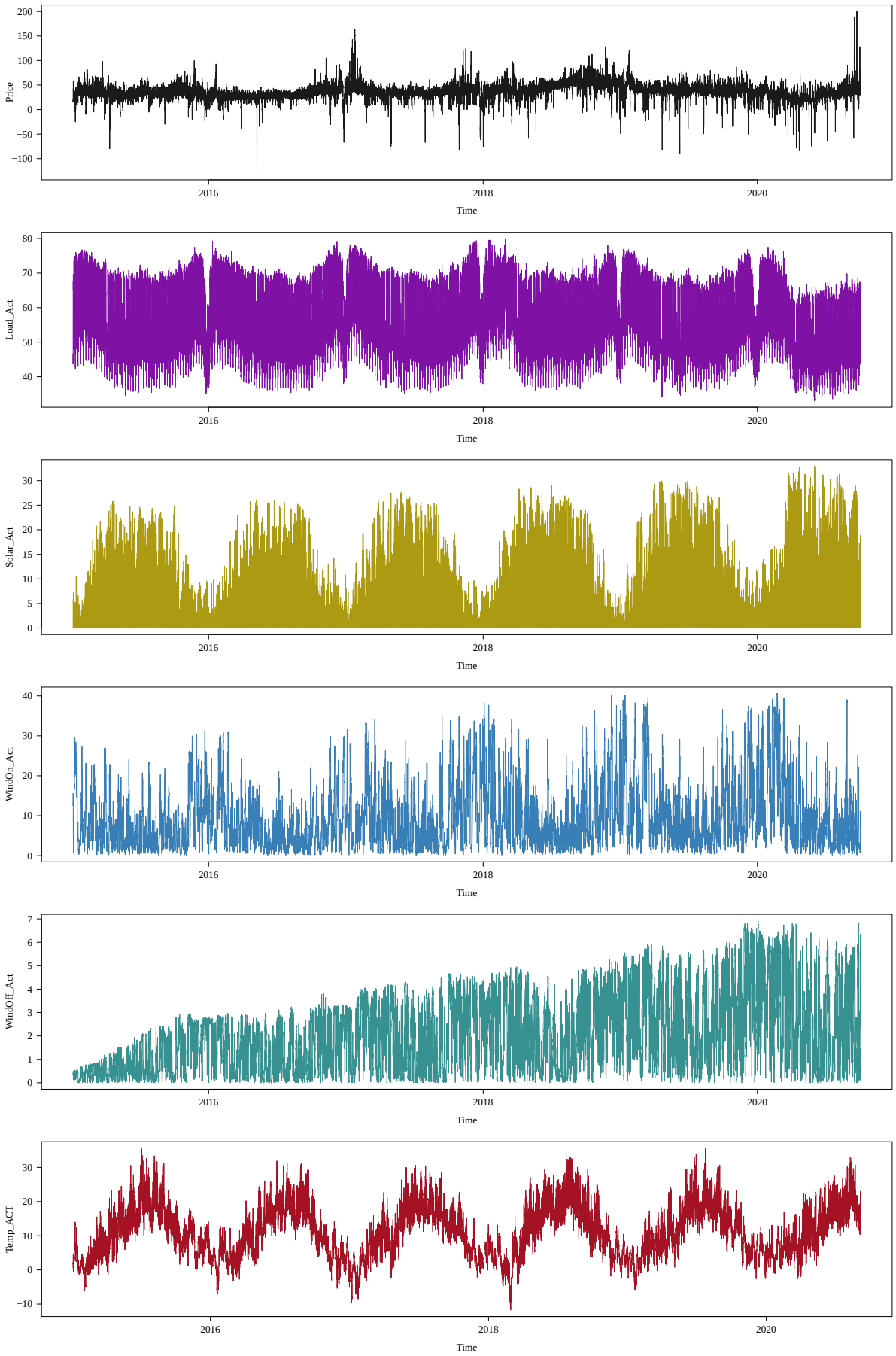


Figure 29: Time series plot of electricity price and considered fundamentals (load, solar, wind onshore and offshore, temperature).

available before the main auction at 12:00. Thus, the information can be used for forecasting. However, usually these forecasts are not freely available. The market participants usually have

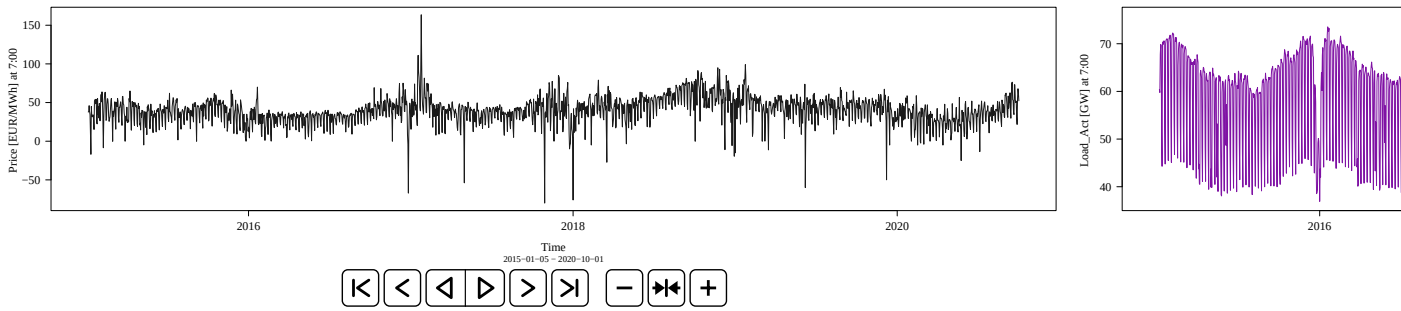


Figure 30: Time series plot of electricity price and considered fundamentals (load, solar, wind onshore and offshore, temperature) for each $s \in \mathbb{S}$.

to buy (high-quality) forecasts of load, wind and solar power production, similarly for weather forecasts.

In this subsection we want to analyse in more detail how we can deal with mentioned fundamentals. From the merit order model (5) we have already expectations on the impact of these drivers. Increasing demand/load should increase the electricity prices, and larger renewable energy net feed-in should decrease the electricity prices. We even know that if the supply stack is roughly linear (as well as the demand curve) then the resulting price effect should be linear as well. For the renewable energies this price reducing effect is also known as merit order effect.

However, as mentioned we do not have the actual fundamental data available, we have to deal with day-ahead forecasts. Figure (32) shows the correlation of the considered fundamentals with the electricity price data. There we observe that the sign of the correlation with the price matches our expectations. So high prices value appear together with high prices and low price appear together with high renewable energy productions. This holds for the day-ahead forecasts in the same way as for the actual observations. We also see extremely high correlations between the day-ahead forecast and the corresponding actuals. Still, we sometimes observe large deviations, as illustrated in Figure (33). For instance, it happens that the load and wind power forecast deviates more than 10 GW.

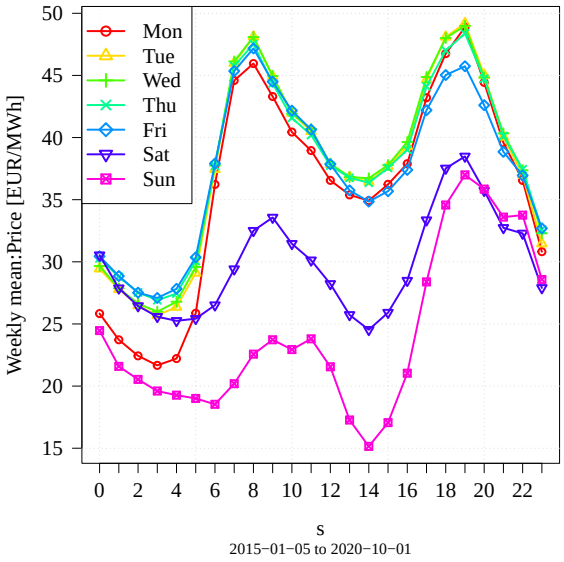
We will discuss the impact of those fundamental external inputs by considering exemplary the external regressor $\text{WindOn}_{d,s}$. It is clear that the quality of the forecasts of the external regressor is crucial for the impact of the electricity price forecasting accuracy. If e.g. the wind power $\text{WindOn}_{d,s}$ could be forecasted perfectly by $\text{DAWindOn}_{d,s}$ (i.e. $\mathbb{E}|\text{WindOn}_{d,s} - \text{DAWindOn}_{d,s}| = 0$ or $\text{MAE} = 0$), then it could be regarded as deterministic external regressor that is very valuable. If the external regressors cannot be forecasted at all (e.g. it is a pure noise process), then the external regressor has no potential to improve the electricity price forecast. So if we cannot forecast the external regressor well, it is of less potential value. In practice, most of the regressors like load, wind and solar are somewhere in between of the deterministic and non-predictable regressor case. Still, the expected absolute error (e.g. $\mathbb{E}|\text{WindOn}_{D+h,s} - \text{DAWindOn}_{D+h,s}|$) increases usually with forecasting horizon. So it is easier to predict the wind power for tomorrow than for the day after tomorrow. Obviously, the same holds within the day, the wind power forecast 12 hours ahead is usually more precise than 36 hours ahead.

As mentioned, we require day-ahead forecasts for our regressor of interest (e.g. $\text{DAWindOn}_{d,s}$ for $\text{WindOn}_{d,s}$). If we want to include a linear impact of the wind power feed-in to the electricity price model, we have usually two options to do so. First, we have the situation where we estimate the price model based on the observed wind $\text{WindOn}_{d,s}$ and replace it in the forecast by $\text{DAWindOn}_{D+1,s}$:

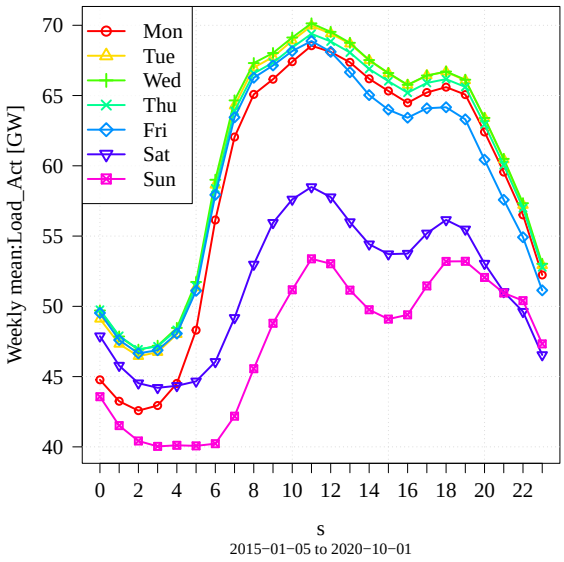
$$\text{estimation: } Y_{d,s} = \beta_{s,0} + \beta_{s,1}\text{WindOn}_{d,s} + \varepsilon_{d,s}, \text{ forecast: } \widehat{Y}_{D+1,s} = \widehat{\beta}_{s,0} + \widehat{\beta}_{s,1}\text{DAWindOn}_{D+1,s}. \tag{83}$$

In the second situation, we estimate the price model based on past forecasted wind $\text{DAWindOn}_{d,s}$ and use if for the forecast:

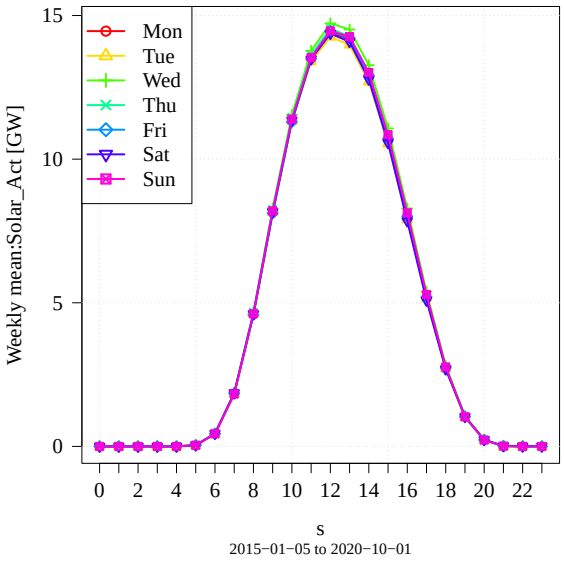
$$\text{estimation: } Y_{d,s} = \beta_{s,0} + \beta_{s,1}\text{DAWindOn}_{d,s} + \varepsilon_{d,s}, \text{ forecast: } \widehat{Y}_{D+1,s} = \widehat{\beta}_{s,0} + \widehat{\beta}_{s,1}\text{DAWindOn}_{D+1,s}. \tag{84}$$



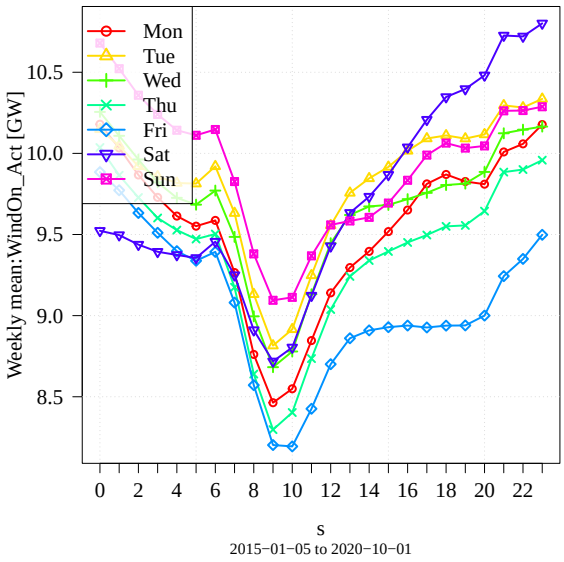
(a) Weekly mean price



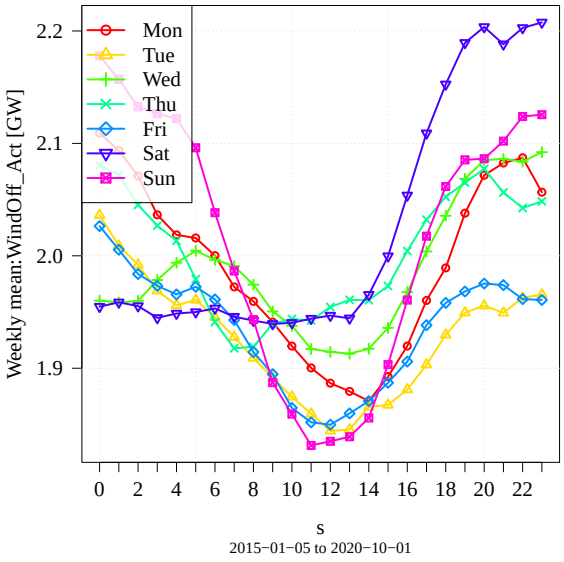
(b) Weekly mean load



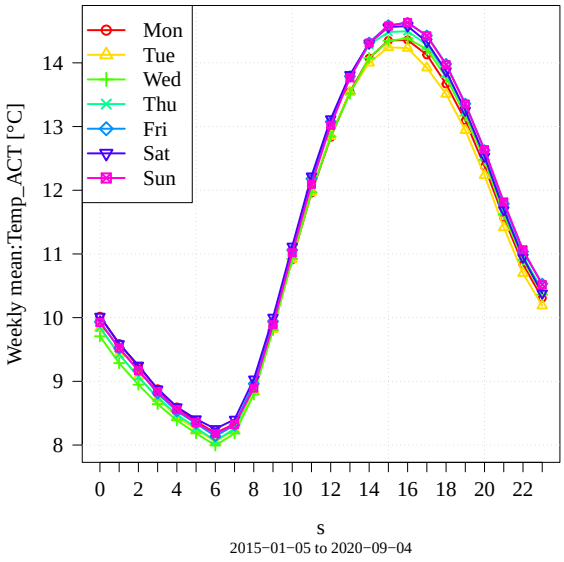
(c) Weekly mean solar



(d) Weekly mean windoff



(e) Weekly mean windon



(f) Weekly mean temp

Figure 31: Weekly sample means of electricity prices and fundamentals.

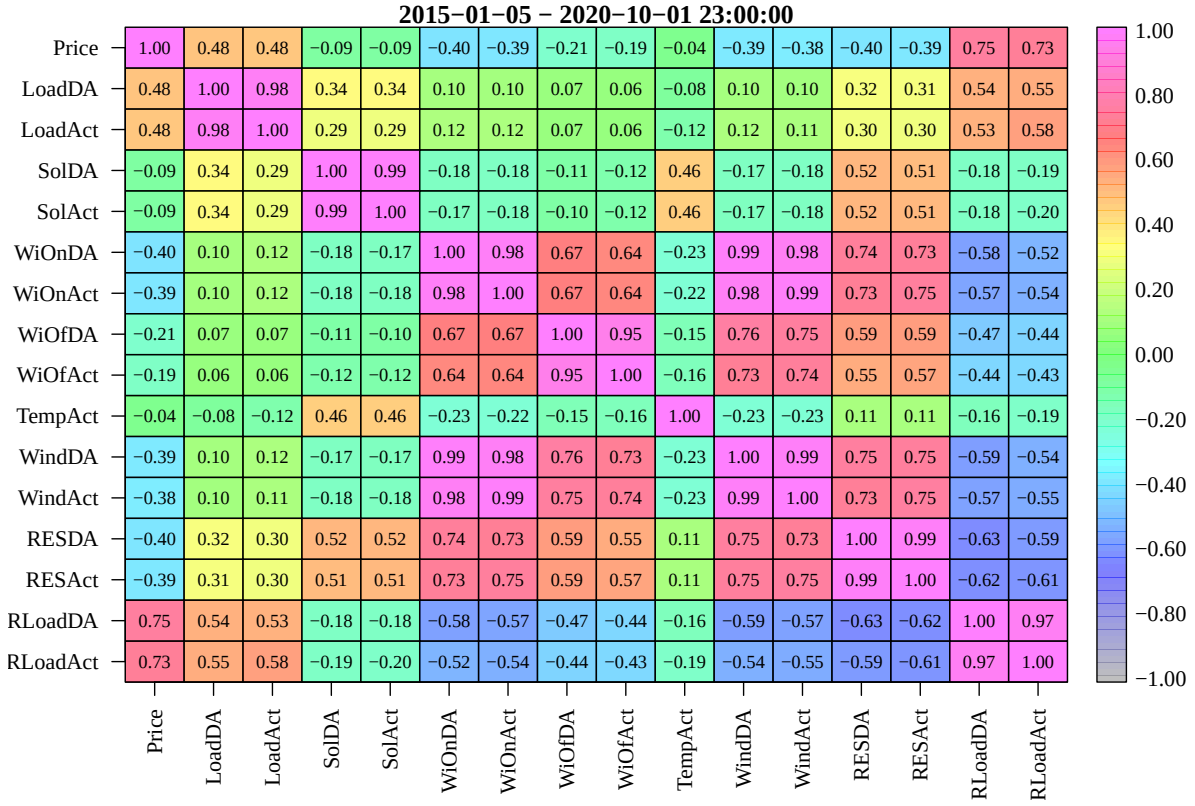


Figure 32: Correlation of day-ahead forecast and actual fundamentals and the electricity price.

The second option trains and forecasts the model on the same data set, so it can potentially adjust for any systematic bias within the forecasts (e.g. $\text{DAWindOn}_{d,s}$). It is usually seen as being favourable but it is only available if we have past forecasts available. In the data set that we consider, we have day-ahead forecasts of certain regressors available (where we assume that market participants have these (or similar) forecasts available). Therefore, we are only looking at these forecasts.

Sometimes, in literature the fundamental data is used to compute new objects that may be relevant for electricity price modelling and forecasting. For instance, we may aggregate wind on-shore and off-shore to receive a single wind power production time series:

$$\text{DAWind}_{d,s} = \text{DAWindOn}_{d,s} + \text{DAWindOff}_{d,s}. \tag{85}$$

Other typical examples are the (fluctuating) renewable energy production (RES) $\text{DARES}_{d,s}$ and the residual load (also residual demand) $\text{DAResLoad}_{d,s}$ which are defined by

$$\text{DARES}_{d,s} = \text{DAWind}_{d,s} + \text{DASolar}_{d,s} \tag{86}$$

$$\text{DAResLoad}_{d,s} = \text{DALoad}_{d,s} - \text{DAWindOn}_{d,s} - \text{DAWindOff}_{d,s} - \text{DASolar}_{d,s}. \tag{87}$$

The residual demand is the load minus the renewable energy feed-in[15]. It has a lot of explanatory power in the classical supply stack/merit-order models, see Fig. 5. The reason is that here reduction of 1GW demand has the same price effect as the increase of 1GW of zero-marginal cost capacity (e.g. renewable energy).

However, it is not clear if the residual load $\text{DAResLoad}_{d,s}$ should be included into the model or not. As the model

$$\begin{aligned} Y_{d,s} =& \beta_{s,0} + \beta_{s,1}\text{DAResLoad}_{d,s} + \varepsilon_{d,s} \\ =& \beta_{s,0} + \beta_{s,1}\text{DALoad}_{d,s} - \beta_{s,1}\text{DAWindOn}_{d,s} - \beta_{s,1}\text{DAWindOff}_{d,s} - \beta_{s,1}\text{DASolar}_{d,s} + \varepsilon_{d,s} \end{aligned} \tag{88}$$

is nested in

$$Y_{d,s} = \beta_{s,0} + \beta_{s,1}\text{DALoad}_{d,s} + \beta_{s,2}\text{DAWindOn}_{d,s} + \beta_{s,3}\text{DAWindOff}_{d,s} + \beta_{s,4}\text{DASolar}_{d,s} + \varepsilon_{d,s} \tag{89}$$

[15]Here usually only the fluctuating renewables wind and solar are subtracted.

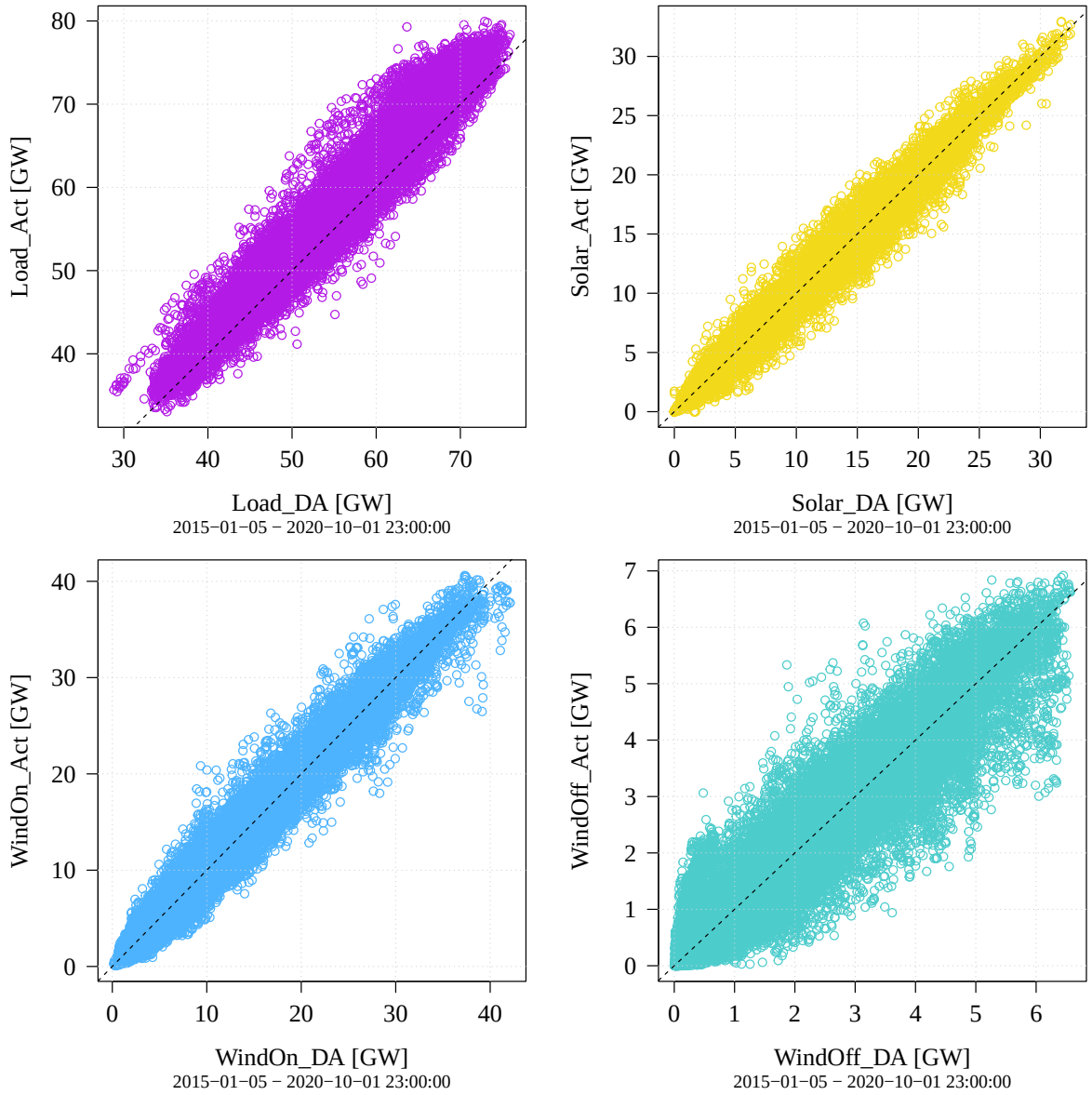


Figure 33: Scatterplot of day-ahead forecasts of fundamentals with its actual values.

it might be worth considering the more general model. Still, if the $-\beta_{s,1}$ is very similar to $\beta_{s,2}$, $\beta_{s,3}$ and $\beta_{s,4}$ it is a strong sign that the residual load carries all relevant information.

8.2 On fuel and emission allowance prices

Now, we want to consider fuel and emission allowance prices as external regressors. Note that for fuel and emission allowance prices as external information the situation is different than for the fundamentals. Those commodities are traded on continuous financial markets[16] and are usually available in real time for market participants. Still, relevant commodities are usually traded only on working days during trading hours. This means, that we have to work with the missing data when we have no trading (esp. on weekends). However, it is important to remember that there is not THE price for coal, natural gas, oil or electricity. Similarly as for electricity are different products with traded on the derivative markets, most importantly they differ in delivery period and for some also in location. Concerning the delivery period of popular future products they are typically daily, monthly, quartly and yearly delivery periods. Many electricity price modelers use front period futures to explain the underlying cost structure of the conventional power plants. In the context of day-ahead electricity price forecasting, it seems plausible to look at day-futures for the corresponding fuel price or the EUA price. However, day-futures are commonly only traded for a few commodities. Natural gas and electricity are two of the few

[16]For European Emission Allowances (EUA) there are in addition aution results available which we ignore here. However, due to market efficiency we do not expect substantial differences the derivative market.

cases. For coal, oil and EUA we would have to use month future. However, even for natural gas is all but clear that the price the day-future really represents the marginal costs for natural gas of the power plants well. One reason is that because of standard diversification or portfolio arguments natural gas is bought in a portfolio of products with different delivery periods, which may vary from supplier to supplier. Thus, the day-future price is not necessarily the price that is paid for the natural gas that is used for the power plant. It rather remains unclear which natural gas future price or combination represents the cost structure well. Obviously, this holds not only for natural gas, but also for coal, oil and emission allowances. Thus, it is up to the modellers choice to deal with the problem accordingly. In academic literature, researchers are often not mentioning explicitely which specific product they consider. But the consideration front month or front year futures seem to be considered most of the times. If not mentioned otherwise we consider always front month futures.

Figure 34 shows the fuel and emission allowance data together with the average electricity price across all S products for selected front period. We observe the mentioned fact that all the external prices vary more slowly than the electricity price. We see that the fuels prices (coal, gas, oil) cover more or less the long term trend behaviour of the electricity prices. To make this relationship clearer, we also show the standardized time series in Figure 34. Here, we observe that the fuel processes of coal and natural gas are roughly aligned with the electricity prices. In contrast, the relationship the EUA price seems not obvious. The same fact can be seen in Figure 35, where we show the correlations among the considered price series. We see moderate correlations between the fuel prices and the electricity price, the correlations with EUA are closer to zero.

As mentioned, fuel prices might be useful to describe the long term trend behaviour of electricity prices, at least to some extent. Behind this observation, there is the important question of the stationarity behaviour of our electricity price time series which we want to study a bit more detailed.

We know already that the electricity price data does not follow a stationary process, like an AR(p). The most obvious reason is the presence of seasonal effects. However, the assumption of periodically stationary seems to be more plausible. The full weekly periodic expert model (79) would an example of such a process. We also briefly discussed that the considered process of electricity prices could have two seasonalities, a weekly (7 days) and an annual one (365.24 days). But again, we can reject the hypothesis that the process is a multi-periodic stationary process due to the presence of holidays. However, the next weaker assumption could be so called trend-stationarity. A process is trend stationary if after removing the trend, the resulting process is trend stationary. Here, the trend refers to a deterministic relationship. Seasonal pattern would be an example of trend stationarity, but also holiday pattern falls into this category. The important fact is that trend behaviour is deterministic and can be predicted even far in the future. Holidays satisfy this criterion, we know that this year, next year, etc. that Christmas is on 25th December, we know that the electricity demand will be reduced and can predict this price reducing effect. Therefore, a valid question is: Are electricity price time series trend stationary? This question can be answered using the fuel and EUA prices.

We discussed that the fuel and EUA prices **??** may impact the prices. This also makes sense with respect to the supply stack model 5. Hence we should analyse the structure of fuel prices in more detail. They are typical financial products. On efficient financial markets, price data exhibit usually martingal structure, or in simple words a Brownian motion reps. random walk type behaviour. This is that the best predictor for price tomorrow is the price today, esp. it holds

$$\mathbb{E}[X_{d+1}^{\text{fuel}} | X_d^{\text{fuel}}, X_{d-1}^{\text{fuel}}, \ldots] = X_d^{\text{fuel}} \tag{90}$$

From the statistical perspective the processes exhibit a unit root and are not stationary. However, those processes are usually difference-stationary. This means that the differenced time series $\Delta X_d^{\text{fuel}} = X_d^{\text{fuel}} - X_{d-1}^{\text{fuel}}$ is a stationary time series. As, mentioned usually all price process of efficient financial markets exhibit this property, so this holds usually for our EUA and fuel price data as well. If we would have a look at electricity price future data for a full year of delivery (to eliminate seasonal effects) we would observe the same unit root type structure.

If we consider the linear model

$$Y_{d,s} = \beta_{s,0} + \beta_{s,1}\text{EUA}_d + \beta_{s,2}\text{Coal}_d + \beta_{s,3}\text{NGas}_d + \beta_{s,4}\text{Oil}_d + \varepsilon_{d,s} \tag{91}$$

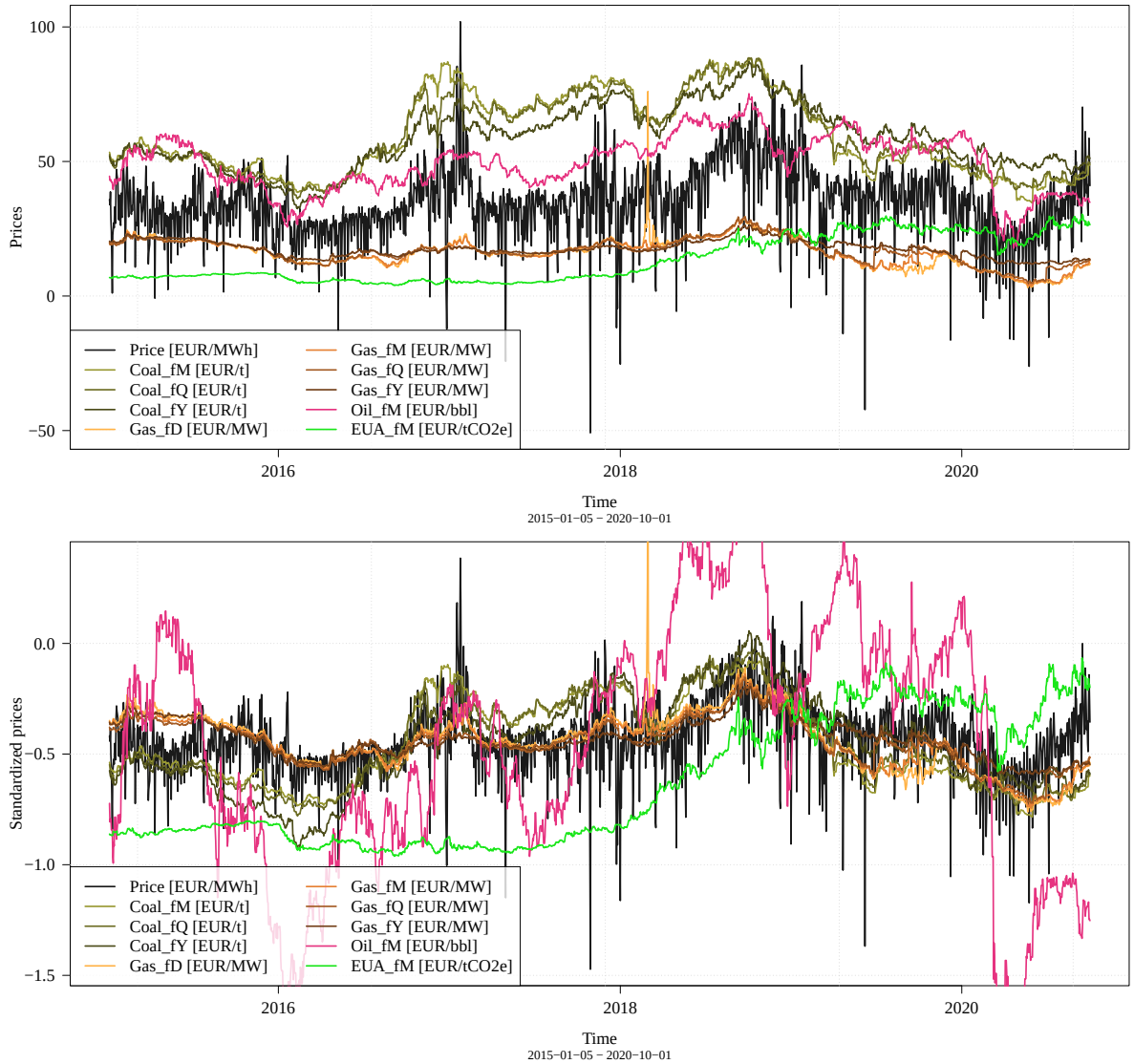


Figure 34: Time series plot [top] and standardised versions [bottom] of considered coal, natural gas, oil and emission allowance front futures, the front period is indicated by D/M/Q/Y (D=day, M=month, Q=quarter, Y=year).

then in the model our electricity price $Y_{d,s}$ inherits the unit root behaviour of the external inputs if $\beta_{s,i} \neq 0$ for $i > 0$.

Still, in practice we can not directly estimate (91) for similar reasons as for the fundamental inputs. The considered external inputs (e.g. EUA_d) are not available at the time of the day-ahead electricity auction. Thus, we have to predict it. But in contrast to the fundamentals the situation is better concerning the availability of forecasts due to (90). So if (90) holds, a suitable forecast for the fuel price is the last available price. Typically, the fuel and EUA price time series are end-of-day data from future contracts. As we have the day-ahead auction at $d-1$ at 12:00 the last available fuel price has usually the index $d-2$.[17] Therefore, we may consider $\widehat{X}_d^{\text{fuel}} = X_{d-2}^{\text{fuel}}$. Hence, a model that we can estimate would be

$$Y_{d,s} = \beta_{s,0} + \beta_{s,1}\text{EUA}_{d-2} + \beta_{s,2}\text{Coal}_{d-2} + \beta_{s,3}\text{NGas}_{d-2} + \beta_{s,4}\text{Oil}_{d-2} + \varepsilon_{d,s} \tag{92}$$

Finally, we want to mention and extended expert model that takes into account the major

[17] Note that financial markets are often trading only on working days, but not on weekends and holidays. Therefore, the price of a non-trading day is usually considered as the last available one, motivated by (90).

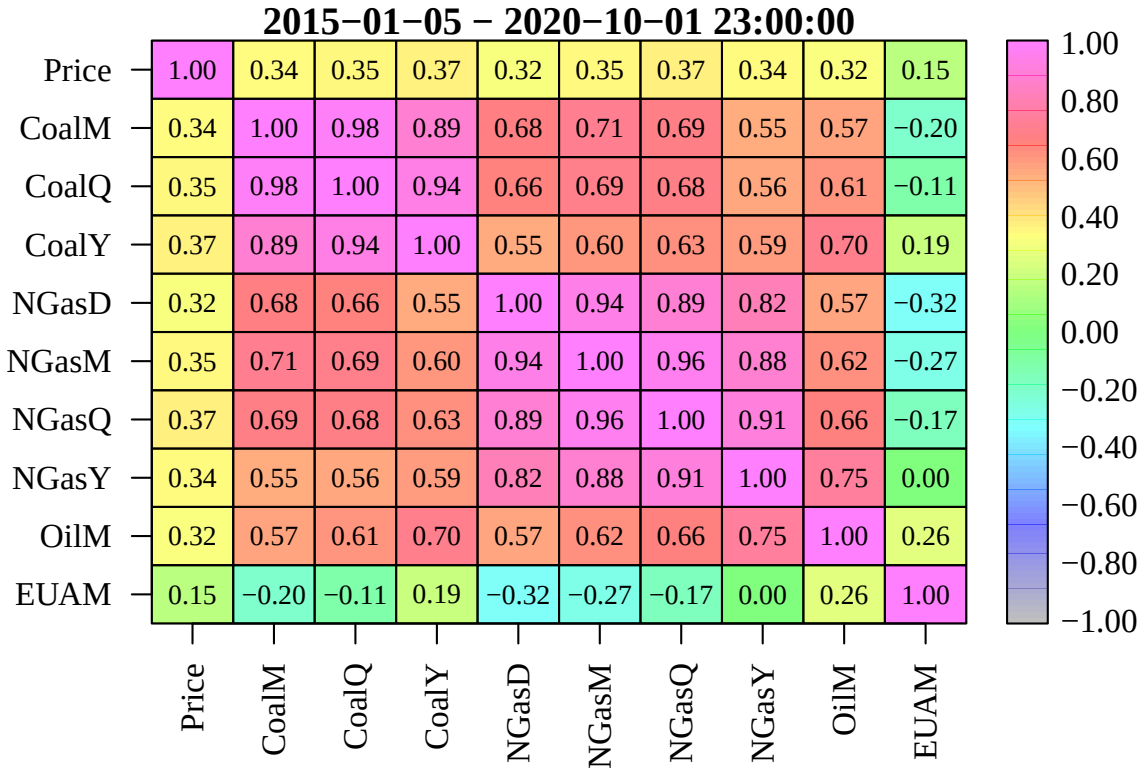


Figure 35: Correlation of all fuels (coal, natural gas, oil, EUA) futures and the electricity price.

relevant external inputs.

$$\begin{aligned} Y_{d,s} = {} & \beta_{s,0} + \beta_{s,1}\text{DoW}_d^1 + \beta_{s,2}\text{DoW}_d^6 + \beta_{s,3}\text{DoW}_d^7 \\ & + \beta_{s,4}Y_{d-1,s} + \beta_{s,5}Y_{d-2,s} + \beta_{s,6}Y_{d-7,s} + \beta_{s,7}Y_{d-1,S-1} \\ & + \beta_{s,8}\text{DALoad}_{d,s} + \beta_{s,9}\text{DASolar}_{d,s} + \beta_{s,10}\text{DAWindOn}_{d,s} + \beta_{s,11}\text{DAWindOff}_{d,s} \\ & + \beta_{s,12}\text{EUA}_{d-2} + \beta_{s,13}\text{Coal}_{d-2} + \beta_{s,14}\text{NGas}_{d-2} + \beta_{s,15}\text{Oil}_{d-2} + \varepsilon_{d,s} \end{aligned} \tag{93}$$

So we extend the **expert.last** model (67) by the day-ahead forecasts of discussed fundamentals and fuel prices and refer this new model as **expert.ext**. Note that the fuel considers here a lag of 2 days, as we use end-of-day data from future markets which is a day prior the price auction, thus two days before delivery. Figure (36) shows the network illustration of the expert model with considered external inputs. We see that for the day-ahead forecasts only the relevant products s are evaluated, in the same as e.g. for $Y_{d-7,s}$ whereas for the fuel prices the same input is explaining $Y_{d,s}$.

Figure (37) shows the estimated coefficients of model (93). We see that the external regressors contribute to some extend to the electricity price. The interpretation of the specific coefficients starts to get a bit confusing. However, some clear pattern maintain, for instance that the last known price $Y_{d,S-1}$ is very important for the night hours.

8.3 Relationships to supply stack model

We have discussed that the supply stack model is a suitable fundamental model for electricity price relationships. In the (inelastic) residual demand version, the electricity price is simply the merit order curve MO at the residual demand. The merit order is mainly shaped by the available conventional power plants with their marginal cost structure. The latter is mainly influenced by fuel and EUA prices, but also by the available generation capacities.

A general supply-stack electricity model based on the residual load is defined by

$$Y_{d,s} = \text{MO}(\text{ResLoad}_{d,s}) + \varepsilon_{d,s}. \tag{94}$$

However, this model can only be used for historic evaluation. But in practice on day-ahead markets, and forecasting purposes we know that we have to replace $\text{ResLoad}_{d,s}$ by the corresponding

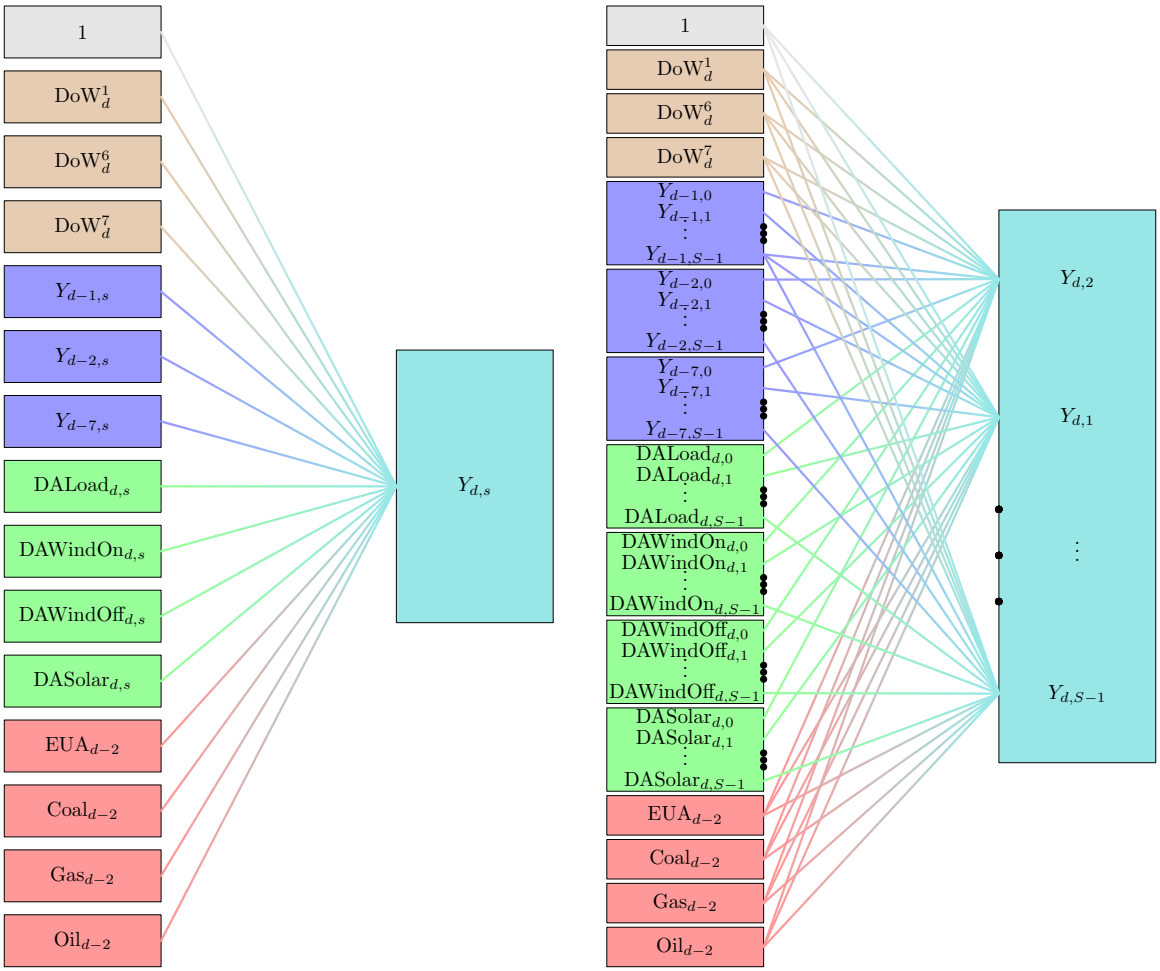


Figure 36: Network visualisation of the model (93), in the simple illustration for all $s \in \mathbb{S}$ (right) and the full illustration (right).

day-ahead forecast $\text{DAResLoad}_{d,s}$:

$$Y_{d,s} = \text{MO}(\text{DAResLoad}_{d,s}) + \varepsilon_{d,s}. \tag{95}$$

We learned that the supply-stack model is a suitable fundamental approach. Now, keep in mind that the merit order curve MO is a function that maps volume to prices. If we assume that MO is linear in the residual load, i.e. $\text{MO}(x) = \beta_0 + \beta_1 x$, then we receive for (95) that

$$Y_{d,s} = \text{MO}(\text{DAResLoad}_{d,s}) + \varepsilon_{d,s} = \beta_0 + \beta_1 \text{DAResLoad}_{d,s} + \varepsilon_{d,s}. \tag{96}$$

for all $s \in \mathbb{S}$. Note that here we assume that the merit order is the same for all hours $s \in \mathbb{S}$. Thus, we have only 2 parameters (β_0 and β_1).

However, nothing stops us from assuming that the merit order curve is different for each delivery period $s \in \mathbb{S}$ which gives $\text{MO}_s(x) = \beta_{s,0} + \beta_{s,1} x$. Then the resulting electricity price model (96) changes to

$$Y_{d,s} = \text{MO}_s(\text{DAResLoad}_{d,s}) = \beta_{s,0} + \beta_{s,1} \text{DAResLoad}_{d,s} + \varepsilon_{d,s}. \tag{97}$$

Thus, we have $2 \times S$ ($=2 \times 24 = 48$ in hourly markets) parameters in total. At least from the fundamental perspective there are no clear arguments in favour for merit order curve that varies from hour to hour. Therefore, we will consider here only MO which does not depend on s.

However, as mentioned the merit order is driven substantially by fuel and emission prices. Assuming that we have only natural gas power plants and zero emission costs. Then a merit order model with linear fuel cost relationships is linear. Thus, for the intercept in (96) we may assume $\beta_0 + \beta_2 \text{NGas}_{d-2,s}$ and for the slope $\beta_1 + \beta_3 \text{NGas}_{d-2}$ while remembering that NGas_{d-2}

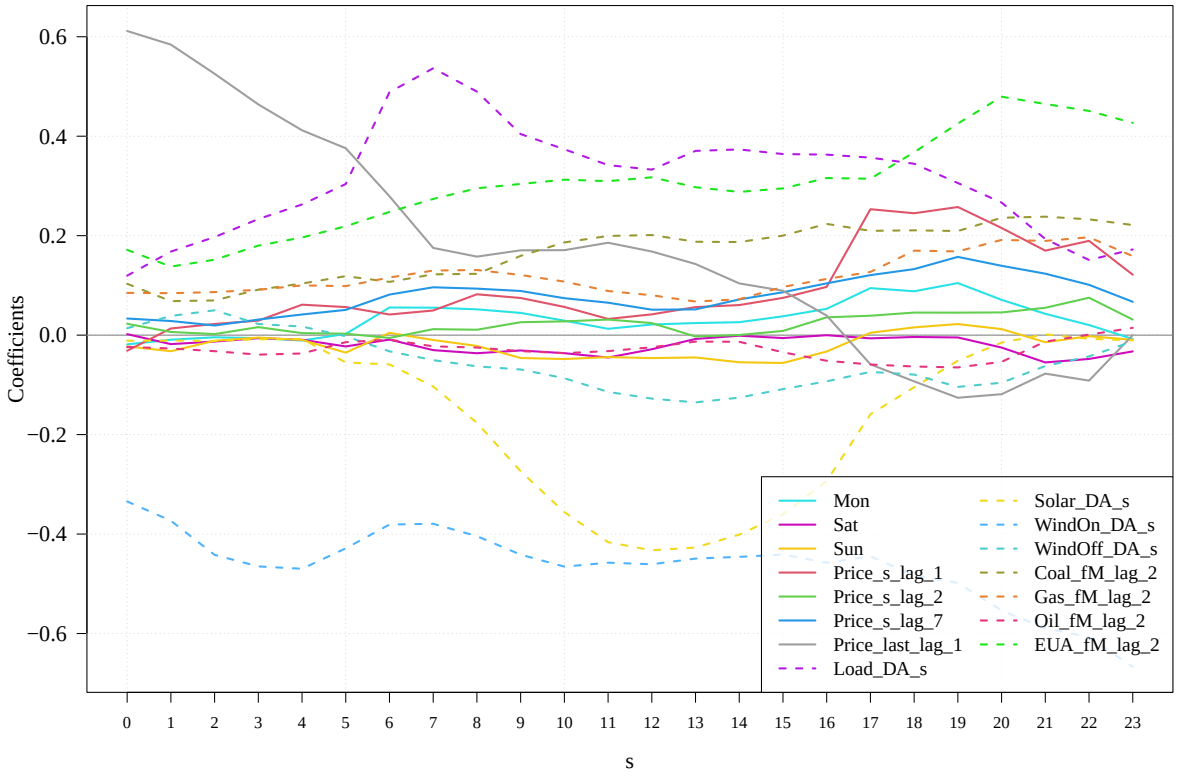


Figure 37: Estimated scaled parameter vector $\widehat{\widetilde{\boldsymbol{\beta}}}_s^{\text{ls}}$ of model (93) for the full time range.

is the most recent available natural gas price and a suitable estimator for NGas_d. Then, overall we receive

$$\begin{aligned} Y_{d,s} = & \underbrace{\beta_0 + \beta_2 \text{NGas}_{d-2}}_{\text{NGas price dependent intercept}} + \underbrace{(\beta_1 + \beta_3 \text{NGas}_{d-2})}_{\text{NGas price dependent slope}} \text{DAResLoad}_{d,s} + \varepsilon_{d,s} \\ = & \beta_0 + \beta_1 \text{DAResLoad}_{d,s} + \beta_2 \text{NGas}_{d-2} + \beta_3 \text{DAResLoad}_{d,s} \text{NGas}_{d-2} + \varepsilon_{d,s}. \end{aligned} \tag{98}$$

Obviously, EUA costs could be added in the same way to the linear model:

$$\begin{aligned} Y_{d,s} = & \beta_0 + \beta_1 \text{DAResLoad}_{d,s} + \beta_2 \text{NGas}_{d-2} + \beta_3 \text{DAResLoad}_{d,s} \text{NGas}_{d-2} \\ & + \beta_4 \text{EUA}_{d-2} + \beta_5 \text{DAResLoad}_{d,s} \text{EUA}_{d-2} + \varepsilon_{d,s} \\ = & \underbrace{\beta_0 + \beta_2 \text{NGas}_{d-2} + \beta_4 \text{EUA}_{d-2}}_{\text{price dependent intercept}} + \underbrace{(\beta_1 + \beta_3 \text{NGas}_{d-2} + \beta_5 \text{EUA}_{d-2})}_{\text{price dependent slope}} \text{DAResLoad}_{d,s} + \varepsilon_{d,s} \end{aligned} \tag{99}$$

The interesting part about model (99) is that it is a statistical model that has (simple) fundamental interpretations. It serves as a simple estimator for the merit order curve without any knowledge of the power plant park structure. The curve is only deduced from the price data.

However, it is a bit more tricky to add further fuel classes to (99) such that we receive a supply stack model that we expect. Note, that just adding further fuel prices (e.g. coal prices) to (99) would correspond to a power plant park that is fired by natural gas and the additional fuel (e.g. coal).

To understand the general solution it is more suitable to study first the two-fuel class solution, because even this special case is quite tricky, [CCS13]. Assume that we have two fuel classes (e.g. coal and natural gas) with trapezoid supply stack and for simplicity first with no emission costs. Then, we have two cases: Either the most expensive power plant from one fuel class is not more expensive than the cheapest power plant from the other class, or the alternative holds. Both options are visualized in Figure 38

In the former case we have a simple structure of the supply stack (or merit order), the two power plant classes just appear next to each other in the supply stack. So the supply stack has

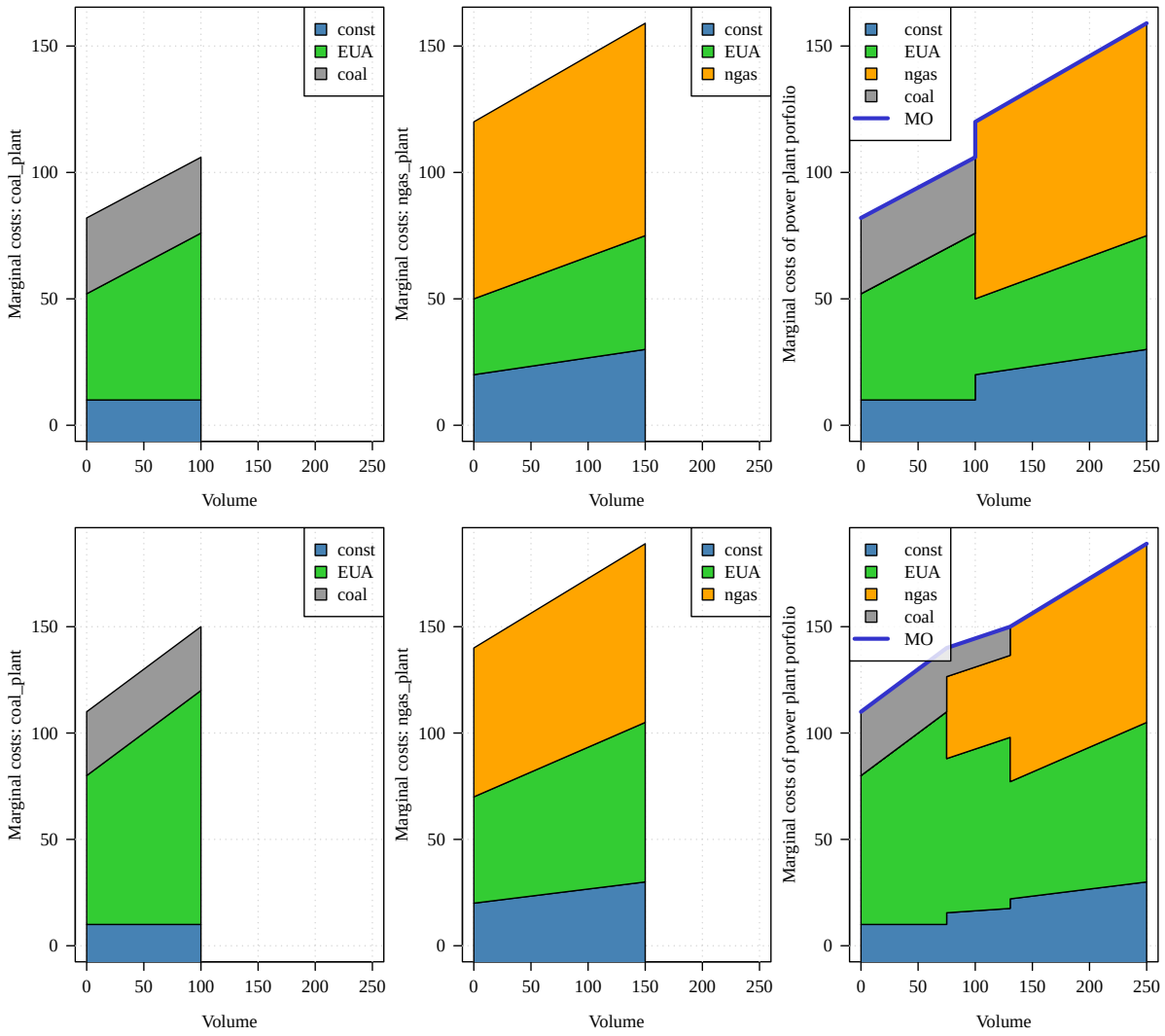


Figure 38: Illustrative merit order (MO) curve (right) of a supply stack model for a coal and natural gas power plant class (left and center) for two situations: Non-overlapping marginal costs (top) and overlapping marginal costs (bottom).

just 2 potentially different slopes. In this case we receive formally,

$$Y_{d,s} = \begin{cases} \beta_0 + \beta_1 \text{DAResLoad}_{d,s} + (\beta_2 + \beta_3 \text{DAResLoad}_{d,s}) \text{Fuel}_{1,d-2} & \text{, if DAResLoad}_{d,s} \leq \text{VolStack}_1 \\ \beta_4 + \beta_5 \text{DAResLoad}_{d,s} + (\beta_6 + \beta_7 \text{DAResLoad}_{d,s}) \text{Fuel}_{2,d-2} & \text{, if DAResLoad}_{d,s} > \text{VolStack}_1 \end{cases} + \varepsilon_{d,s} \tag{100}$$

where VolStack_1 is the volume of the first fuel class. In the second case we have overlapping marginal costs, we see that the fuel classes are mixing, see Fig. 38. Up to three different slopes are possible in the merit order curve. The part where marginal costs overlap create a fuel mixed class that has generation capacities from both fuel types. Therefore, we receive also that the marginal costs structure mixes. The slope in this mixed are smaller because more generation capacity is active in the corresponding marginal costs segment. However, writing down the corresponding model equation formally is feasible, but is also relatively complex. Therefore we study this only in the more general case in more detail.

8.4 The formal supply stack model

In general, we have to take the inverse functions of the supply stacks, aggregate them. This is feasible and illustrated in Figure 39. We want to understand a bit better how to construct those merit order curves. Figure 39 also illustrates that it is quite difficult to estimate the effect of a price change in a certain fuel, because it interacts with other fuel prices and input variables.

Just for simplification we skip here the time dependence, but of course this to be considered for empirical applications.

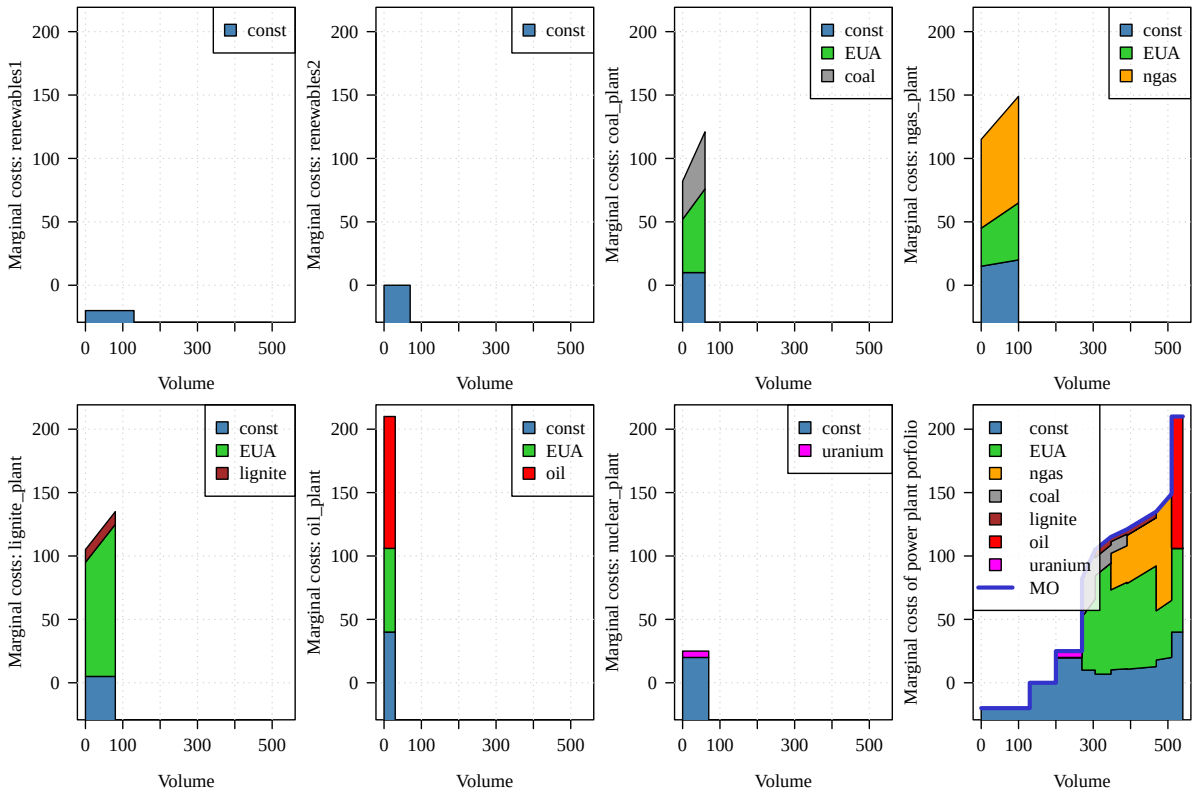


Figure 39: Illustrative merit order (MO) curve of a supply stack model for multiple power plants.

Let L the number of cost specific components. In the case of prices for EUA, oil, coal and gas we would have $L = 4$. This gives us an $(L+1)$-dimensional $\mathbf{price} = (1, \text{price}_1, \ldots, \text{price}_L)$ of the considered prices including the 1 in the first coordinate. In the mentioned situation with of prices for EUA, oil, coal and gas we have the 5-dimensional vector $\mathbf{price} = (1, \text{EUA}, \text{Oil}, \text{Coal}, \text{NGas})$.

Now, we consider K generation units, or K classes/groups of generation units. Next, we require $K \times (L+1)$-dimensional efficiency cost matrices $\boldsymbol{C}^{\text{lower}}$ and $\boldsymbol{C}^{\text{upper}}$. For $\text{bnd} \in \{\text{lower}, \text{upper}\}$.

$$\boldsymbol{C}^{\text{bnd}} = \begin{pmatrix} \boldsymbol{c}_1^{\text{bnd}} \\ \boldsymbol{c}_2^{\text{bnd}} \\ \vdots \\ \boldsymbol{c}_K^{\text{bnd}} \end{pmatrix} = \begin{pmatrix} c_{1,0}^{\text{bnd}} & c_{1,1}^{\text{bnd}} & \cdots & c_{1,L}^{\text{bnd}} \\ c_{1,0}^{\text{bnd}} & c_{2,1}^{\text{bnd}} & \cdots & c_{2,L}^{\text{bnd}} \\ \vdots & & & \\ c_{1,0}^{\text{bnd}} & c_{K,1}^{\text{bnd}} & \cdots & c_{K,L}^{\text{bnd}} \end{pmatrix}.$$

where $c_{k,0}^{\text{bnd}}$ represents the constant marginal costs of generation unit k. For $\boldsymbol{C} = \boldsymbol{C}^{\text{lower}} = \boldsymbol{C}^{\text{upper}}$ the efficiencies for unit k is constant. This situation is considered in the standard supply-stack models where each power plant is represented by its own class with fixed marginal costs, as we considered and illustrated it in Figure 6. Otherwise it will be linearly interpolated. This is suitable to model full power plant classes at once.

In addition to the cost efficiency matrices, we require capacity volume vector $\mathbf{cap} = (\text{cap}_1, \ldots, \text{cap}_K)$ which represent the volume of the k's power plant class. The k's unit have the marginal costs

$$\text{MC}_k(v) = \mathbf{price}' \boldsymbol{c}_k^{\text{lower}} + \mathbf{price}'(\boldsymbol{c}_k^{\text{upper}} - \boldsymbol{c}_k^{\text{lower}})v \tag{101}$$

for $v \in [0, \text{cap}_k]$. Now, keep in mind that MC_k maps a volume of the interval $[0, \text{cap}_k]$ to a price $[p_{k,\min}, p_{k,\max}] = [\mathbf{price}' \boldsymbol{c}_k^{\text{lower}}, \mathbf{price}' \boldsymbol{c}_k^{\text{upper}}]$. Thus, the inverse MC_k^{-1} maps a price from $[p_{k,\min}, p_{k,\max}]$ to the volume interval $[0, \text{cap}_k]$. This is

$$\text{MC}_k^{-1}(p) = \begin{cases} \frac{p - \mathbf{price}' \boldsymbol{c}_k^{\text{lower}}}{\mathbf{price}'(\boldsymbol{c}_k^{\text{upper}} - \boldsymbol{c}_k^{\text{lower}})} & , \text{ if } \mathbf{price}'(\boldsymbol{c}_k^{\text{upper}} - \boldsymbol{c}_k^{\text{lower}}) \neq 0 \\ \text{cap}_k 1_{\{p_{k,\min}\}}(p) & , \text{ if } \mathbf{price}'(\boldsymbol{c}_k^{\text{upper}} - \boldsymbol{c}_k^{\text{lower}}) = 0 \end{cases}. \tag{102}$$

for $p \in [p_{k,\min}, p_{k,\max}]$ where we treat the degenerate case with $\mathbf{price}'(\boldsymbol{c}_k^{\text{upper}} - \boldsymbol{c}_k^{\text{lower}}) = 0$ as a jump function (or delta function) in $p_{k,\min} = \mathbf{price}'\boldsymbol{c}_k^{\text{lower}}$ with size of the capacity cap_k. Now, on the inverse level of the marginal costs, we can aggregate to receive the invert joint merit order:

$$\text{MO}^{-1}(p) = \sum_{k=1}^{K} \text{MC}_k^{-1}(p) \tag{103}$$

which defines us the merit order curve $\text{MO} = (\text{MO}^{-1})^{-1}$ by taking the (generalized) inverse of MO^{-1}. Thus, the resulting electricity price model is given by

$$Y_{d,s} = \text{MO}(\text{ResLoad}_{d,s}) + \varepsilon_{d,s} = \left(\sum_{k=1}^{K} \text{MC}_k^{-1}\right)^{-1} (\text{ResLoad}_{d,s}) + \varepsilon_{d,s}. \tag{104}$$

This model (104) is proper supply-stack model which depends on the capacity vector **cap**, the price vector **price** the efficiency matrices $\boldsymbol{C}^{\text{lower}}$ and $\boldsymbol{C}^{\text{upper}}$. Note that in practice, not only the residual demand $\text{ResLoad}_{d,s}$ depends on time, but also the components may depend on time. This is obvious for the price vector **price**, but also the availabile capacities change over time due to maintenance, decommisioning and new installations.

8.5 Tasks

Hints and the corresponding programming part are provide in Section R.7 in `R` and in Section py.5 in `python`.

1. [FM20] report that the simple model (88) performs surprisingly well for the Belgium electricity market, compared for its simplicity. This holds especially for very short calibration window sizes. Can you confirm this results for the German market?

2. Estimate (89), and perform an test with $H_0 : -\beta_{s,1} = \beta_{s,2} = \beta_{s,3} = \beta_{s,4}$. Are these findings consistent across $\mathbb{S}$. If it is easier for you, try first the analogue problem for

$$Y_{d,s} = \beta_{s,0} + \beta_{s,1}\text{Load}_{d,s} + \beta_{s,2}\text{DARES}_{d,s} + \varepsilon_{d,s}$$

3. Compute the partial correlations between $\boldsymbol{Y}_{d,s}$ and the considered external regressors conditioned on all information of the **expert.last** last model (DA-Load, DA-Solar, DA-WindOn, DA-WindOff, DA-Wind, DA-RES, DA-ResLoad, EUA, Coal, NGas, Oil). Is it a suitable approach to include the fundamental DA-forecasts only for the hour s?

4. Compare the forecasting performance of **expert.ext** with **expert.last** in a suitable forecasting study.

5. Write a function that extends model **expert.last** by the residual load and the natural gas prices, compare the forecasting performance with **expert.ext** and **expert.last**.

6. Estimate model (99). Then draw the (linear) merit order derived from the model, similarly as seen in the centre of Figure 38.

7. The supply stack model introduced in this section is only valid for non-connected (island or copper-plate systems). In coupled electricity markets import and export flows are relevant for system stability and influence the price mechanism. Discuss which components of the supply-stack-model have to be adjusted to adequately address interconnected markets.

9 Non-linear effects

Non-linear effects are a very complex area in econometric modelling and forecasting. The main reason for this is that there are many ways in which the assumptions of the linear model can be violated or generalised. In a general regression framework, we can divide non-linear effects into two groups. First, non-linear effects that are linear in the parameter but non-linear in the regressors. Second, those effects that are non-linear in the parameter, these corresponding models are called non-linear model. In this manuscript, we first only consider effects that are linear in the parameter but non-linear in the regressors, and turn later on to (pure) non-linear models, see 14. So we use linear models to describe linear effects (but not non-linear models). The main reason for this is that we can estimate within the framework of linear regression, for which much statistical and computational theory is known. Moreover, any (sufficiently smooth) model that is non-linear in at least one parameter can be well approximated by a linear model using Taylor expansion. We will return to this later in this section.

9.1 The square and absolute value effect

The most standard non-linear transformations of any variable x are usually squares ($f(x) = x^2$) and absolute values ($f(x) = |x|$). We have seen both transformations already when we discussed the evaluation of forecasts, or forecast errors, leading to e.g. the definition of the MAE and RMSE.

For exemplary purpose we consider a simple linear model in the regressor X_t. This is given by

$$Y_t = \beta_0 + \beta_1 X_t + \varepsilon_t. \tag{105}$$

A natural extension could be

$$Y_t = \beta_0 + \beta_1 X_t + \beta_2 X_t^2 + \varepsilon_t \tag{106}$$

for an additional squared effect in X_t or alternatively

$$Y_t = \beta_0 + \beta_1 X_t + \beta_2 |X_t| + \varepsilon_t \tag{107}$$

for the absolute value effect. In both equations β_2 describes a non-linear effect. Still, they exhibit different characteristics:

i) The square function $f(x) = x^2$ is a smooth continuous function, which is continuously differentiable and has many nice calculus properties e.g. it is strictly convex, and its first derivative is a linear function. However, it is not uniformly continuous on the real axis.

ii) Instead, the absolute value $f(x) = |x|$ is a continuous function, which is not differentiable in zero (but the subderivative exists). Still it is a convex function and Lipschitz continuous on the real axis.

These listed properties are relevant for modelling of random variables like electricity prices. Still, before we discuss this further, we want to introduce an important characteristic from statistical theory.

A key property or characteristic in the statistical theory when analyzing non-linear transformations is the so called tail index α. The tail index $\alpha \geq 0$ of X_t is the least upper bound α such that $\mathbb{E}(|X_t|^\alpha)$ just exists (or equivalently is finite). Mathematically more precisely, it can be defined by

$$\alpha = \sup\{a > 0|\ |\mathbb{E}(|X_t|^a)| < \infty\}. \tag{108}$$

It is important remember that the expected value $\mathbb{E}$ is an integral operator. The existence of those integrals in (108) is determined by the behaviour in the tails of the distribution of X_t, so in simple words what happens for limits at $\pm\infty$. If X_t follows a normal distribution, then α is ∞. If X_t is drawn from a t-distribution with k degrees of freedom, then we have $\alpha = k$. So α is measuring the heavy tails of a distribution. For the t-distribution it holds the smaller α, the smaller k. Or in general the smaller α the heavier tailed is the distribution. Furthermore, we know for the first moment that if $\alpha > 1$ then X_t has a mean, so $\mathbb{E}(X_t) < \infty$, if $\alpha > 2$ then X_t

has a variance, so $\mathbb{V}\text{ar}(X_t) < \infty$. If $\alpha > 3$ then X_t has a skewness and if $\alpha > 4$ then X_t has a kurtosis. There are also methods to estimate α from data, like the Hill-estimator. However, in practice the estimation of a tail-index is very tricky due to slow convergence, as the tails describe theire behaviour and by nature we have only few observations in the tails.

But why do those tail indexes matter at all? From the statistical modelling perspective it is important to know that specific convergence theorems require different assumptions to the tail index α. The minimal requirement for the strong law of large number (SLL) is that $\alpha > 1$. This SLL holds if X_t is iid. However, most practical convergence theorems (other SLLs which allow for dependence and the central limit theorem) require that $\alpha > 2$, so they need X_t to have a finite variance. If this does not hold we run usually into troubles with many estimation methods, like with the standard OLS estimator. However, even if $\alpha > 2$ holds we have no guarantee for fast convergence (if the sample size increases) of the estimators in absolute terms. We just know that the stander statiscial $\sqrt{t}$-convergence rate hold, which we remember means that if we want to double the precision of our estimation procedure we have square the amount of input data resp. the sample size. However, to control the speed of the $\sqrt{t}$ convergence (so the factor in front of the $\sqrt{t}$) we usually need Berry-Esseen type theorems which require $\alpha > 3$.

For us it is important to know that a non-linear transformation that is applied to X_t can change the tail index of X_t. For the square function ($f(x) = x^2$) it holds that if X_t has a tail index α then X_t^2 has a tail index of $\alpha/2$, so the tail index decreases. This makes intuitively sense as squaring numbers will increase large numbers a lot and makes the underlying distribution more heavy tailed. From the modelling perspective this is an unwanted feature, as we might lose nice convergence properties in our model. This is especially critical if X_t has a tail-index that is already smaller than 4, because then X_t^2 will have a tail index below 2 and many convergence theorems can not be applied anymore. So we have to be careful when applying squared effects. In contrast, the absolute value function ($f(x) = |x|$) does not change the tail index. $|X_t|$ has the tail index α if X_t has the tail index α. We remain on the same convergence level. But we have the often unwanted disadvantage of heaving a non-differentiable function which leads to non-smooth effects. Still, we want to remark that there are smooth non-linear transformations that preserve the tail index and combine both positive features.

The absolute value $|x|$ is often not directly applied but shifted before by a constant value a, so $f(x-a) = |x-a|$. Such a shift on squares yields $f(x-a) = (x-a)^2 = x^2 - 2ax + a^2$ which is a polynomial of order 2. If the corresponding linear effect and a constant is already included in the model as $x^2 - 2ax + a^2$ contains next to the quadratic effect only a linear and a constant term. This invariance against linear translation is a positive property of the square.

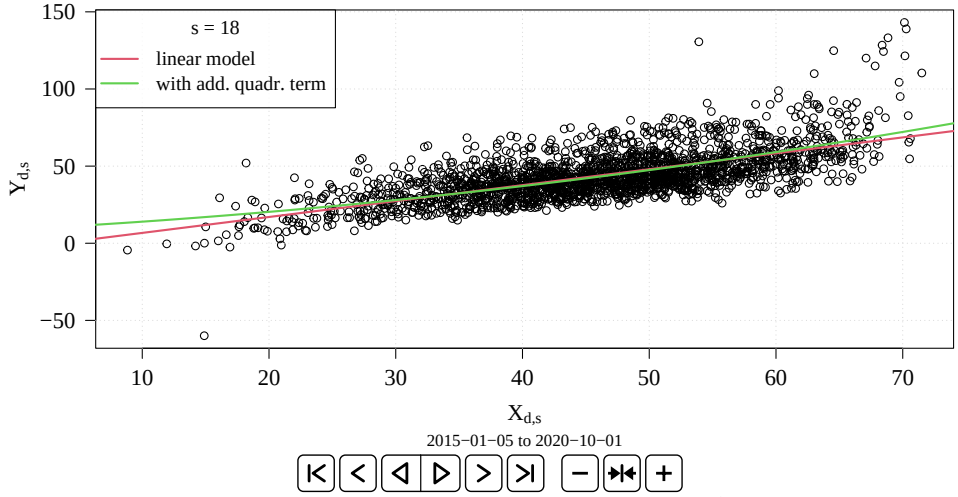
The shifted absolute value function $|x-a|$ depends a lot on the choice of a. If a is too small or too large (in the empirical part smaller or larger than any observation) then it basically acts as a linear function. In general a should be chosen at such a value, where we expect a change in the behaviour. The shift a is often chosen ad hoc by expert knowledge or as a mean or median of the data. It describes a certain regime change around a.

The two seen transformations (square and absolute value) can be applied to any regressor X. However, all non-linear transformations will be invariant to dummy regressors which take only the value 0 or 1. So it does not make sense to apply them on dummies. If we include external regressors, it definitively can make sense to apply them to these.

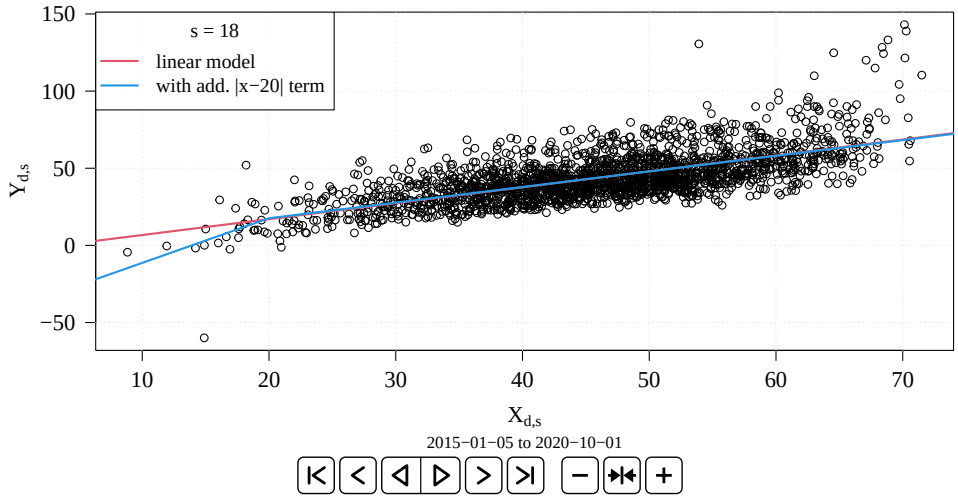
For illustration purpose we will focus on the residual load, as we know that as a single explanatory variable this should have relatively high predictive power. In Figure 40, we see three simple fitted electricity price models for the German EPEX data with corresponding residual load.

There we use a the very simple model $Y_{d,s} = \beta_{s,0} + \beta_{s,1}\text{ResLoad}_{d,s} + \varepsilon_{d,s}$ and extend it by applying non-linear transformations on $\text{ResLoad}_{d,s}$. In detail, the square transformation (see Figure 40(a)), the absolute value transformation (see Figure 40(b) and 40(c)) with a shift of $a = 20$ and $a = 50$ (measures in Gigawatt). In general, it seems that the non-linear effects on the autoregressive coefficient do not seem to be strong. The non-linear model curve lies for all 3 models close to the corresponding linear model fit. This gives us the idea that non-linear transformations on the residual load have a limited contribution to the forecasting accuracy. However, by eyeballing it seems that especially, in the extreme situations some non-linear effects might be present. The standard shape of our merit order (see 5) supports this intuition.

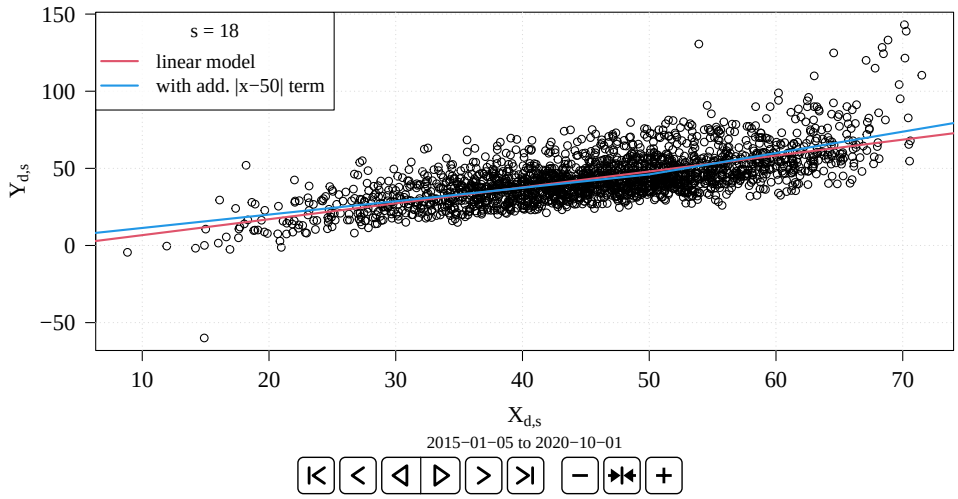
However, a specific class of simple non-linear effects models that has been applied in elec-



(a) non-linear model with additional $\beta_{s,2}(X_{d,s}^{\text{ResLoad}})^2$ term



(b) non-linear model with additional $\beta_{s,2}|X_{d,s}^{\text{ResLoad}} - a|$ with $a = 20$ term



(c) non-linear model with additional $\beta_{s,2}|X_{d,s}^{\text{ResLoad}} - a|$ with $a = 50$ term

Figure 40: Scatter plot of $X_{d,s}^{\text{ResLoad}}$ and $Y_{d,s}$ for $s \in \mathbb{S}$ with fitted linear model $Y_{d,s} = \beta_{s,0} + \beta_{s,1} X_{d,s}^{\text{ResLoad}} + \varepsilon_{d,s}$ and simple non-linear effect extensions.

tricity price forecasting is the class of threshold autoregressive processes (see e.g. [MTW06]). If we consider a standard AR(1) model (4) and add the nonlinear effect $|Y_{d-1,s} - a|$ with $a = \mu_s$ as the process mean $\mu_s = \mathbb{E}(Y_{d,s})$, then we receive a TAR(1) model. This is given by

$$Y_{d,s} = \phi_0 + \phi_{1,0} Y_{d-1,s} + \phi_{1,1} |Y_{d-1,s} - \mu_s| + \varepsilon_{d,s}$$

with $\mu_s = \mathbb{E}(Y_{d,s})$. In empirical application, μ is usually estimated in advance using the sample

mean as an appropriate estimator. The TAR(1) model is a very simple example of a regime switching model, where we have 2 regimes. In literature TAR(1) model is usually augmented using expert model features like, higher order lags or weekday dummies.

9.2 On the minimum and maximum effects

Another non-linear class of effects are minimum and maximum effects. They are also quite popular in electricity price modelling and forecasting. Here, we want to mention that both the minimum and the maximum effects are special cases of general absolute value effects. As it holds

$$\min(x,y) = \frac{1}{2}(x+y) - \frac{1}{2}|-x+y| \tag{109}$$

and

$$\max(x,y) = \frac{1}{2}(x+y) + \frac{1}{2}|x-y| \tag{110}$$

every minimum and maximum function is a non-linear function, namely a linear combination of linear and absolute value functions. Moreover every maximum can be written as minimum and vice versa, at it holds $\min(x,y) = -\max(-x,-y)$. Note that the minimum and maximum of more than 2 number (e.g. $\max(x,y,z)$ or in general $\max_{i\in\{1,\dots,K\}}(x_i)$) can be represented with multiple linear and absolute functions. In machine learning the $\max(x,0)$ is also known as *rectifier*. In the context of neural networks is also known as rectified linear unit (ReLU).

Some models assume that the electricity price $Y_{d,s}$ depends not only on the previous day maximum or minimum price across all S products, so $Y_{d,\min} = \min_{s\in\mathbb{S}}(Y_{d,s})$ or $Y_{d,\max} = \max_{s\in\mathbb{S}}(Y_{d,s})$.

Both $Y_{d-k,\min}$ and $Y_{d-k,\max}$ are sometimes used in electricity price forecasting models as regressors, mainly using a lag of $k=1$. Note that the minimum and maximum have the same tail index properties as the absolute value function. Especially, the tail index of $Y_{d,\min}$ is the same as $Y_{d,\min}$. It is given by $\min_{s\in\{1,\dots,S\}}(\alpha_s)$ is α_s denotes the tail index of $Y_{d,s}$. So the tail index is the smallest one of all $Y_{d,s}$ that are involved in the computation of the minimum or maximum.

So, we can write down the model

$$\begin{aligned}
Y_{d,s} = \beta_{s,0} &+ \underbrace{\beta_{s,1}Y_{d-1,s} + \beta_{s,2}Y_{d-2,s} + \beta_{s,3}Y_{d-7,s}}_{\text{autoregressive effects on same } s} + \underbrace{\beta_{s,4}Y_{d-1,S-1}}_{\text{last product effect}} + \underbrace{\beta_{s,5}Y_{d-1,\min} + \beta_{s,6}Y_{d-1,\max}}_{\text{non-linear effects}} \\
&+ \underbrace{\beta_{s,7}\mathrm{DoW}_d^1 + \beta_{s,8}\mathrm{DoW}_d^6 + \beta_{s,9}\mathrm{DoW}_d^7}_{\text{Day-of-the-week effects}} \\
&+ \underbrace{\beta_{s,10}X_{d,s}^{\mathrm{Load}} + \beta_{s,11}X_{d,s}^{\mathrm{DASolar}} + \beta_{s,12}X_{d,s}^{\mathrm{DAWindOn}} + \beta_{s,13}X_{d,s}^{\mathrm{DAWindOff}}}_{\text{Effects of day-ahead forecast of fundamentals on the same } s} \\
&+ \underbrace{\beta_{s,14}X_{d-2}^{\mathrm{EUA}} + \beta_{s,15}X_{d-2}^{\mathrm{Coal}} + \beta_{s,16}X_{d-2}^{\mathrm{NGas}} + \beta_{s,17}X_{d-2}^{\mathrm{Oil}}}_{\text{EUA and fuel price effects}} + \varepsilon_{d,s}
\end{aligned} \tag{111}$$

So we extend the **expert.ext** model (93) by the minimum and maximum effect and refer this new model as **expert.adv**, representing an advanced expert model. This model is usually a well working forecasting model for most European markets. Obviously, if we consider a country where a certain fundamental forecast is not available it is dropped from the corresponding equation.

9.3 Interaction effects

Another topic that often arises when talking about non-linear effects are the so called interactions. In general interaction effects can only occur when we have at least two different regressors say X_t and Z_t. A standard linear model then looks like

$$Y_t = a_0 + a_1X_t + a_2Z_t + \varepsilon_t.$$

This is also the result of a Taylor approximation of the model

$$Y_t = f(X_t, Z_t) + \varepsilon_t. \tag{112}$$

of order 1. Here f is a bivariate smooth function that acts on X_t and Z_t. Still, if we use the Taylor approximation of order 2 on the model (112) then we receive the corresponding linear model

$$Y_t = a_0 + a_1 X_t + a_2 Z_t + a_3 X_t^2 + a_4 X_t Z_t + a_5 Z_t^2 + \varepsilon_t.$$

where quadratic terms with coefficients a_3, a_4 and a_5 are added. The terms X_t^2 and Z_t^2 are standard quadratic effects that we already introduced. The term $X_t Z_t$ is also known as mixed quadratic effect. This term is describing an interaction effect between X_t and Z_t. To get a better understanding, consider X_t and Z_t as dummies (which take only values in 0 and 1). So now suppose that X_t is a Sunday dummy and Z_t is a January dummy. Then the corresponding mixed quadratic term $X_t Z_t$ would be always zero, expect for the Sundays in January. So $X_t Z_t$ is only active if both events are active (or interact). Therefore, it is called interaction effect.

In model (73), we have seen interaction effects. There the periodic parameter $\beta_{s,1}(d)$ as defined in (75) contains interactions between the weekday-dummies (DoW_d^1, DoW_d^6 and DoW_d^7) and the past price $Y_{d-1,s}$. Similarly, also in the linear models derived from the merit order curves have interaction effects. Model (**??**) has an interaction between the natural gas price X_t^{NGas} and the EUA price X_t^{EUA}.

9.4 General function approximations using linear models

The quadratic effect is a specific non-linear effect which belongs to the class of polynomial effects. Polynomial effects have high theoretical values, as every (sufficiently smooth) model that is non-linear in at least one parameter which can be well approximated by a linear model using Taylor expansion. This gives a suitable approximation of every function in terms of polynomials. For example the non-linear model $Y_t = a_0 + e^{a_1 X_t} + \varepsilon_t$ has Taylor expansion around 0 of order 2

$$Y_t = a_0 + a_1 X_t + \frac{1}{2} a_1^2 X_t^2 + R_{2,t} + \varepsilon_t \tag{113}$$

where $R_{2,t}$ is a remainder term. In the next Taylor approximation level of order 3, we would get

$$Y_t = a_0 + a_1 X_t + \frac{1}{2} a_1^2 X_t^2 + \frac{1}{6} a_1^3 X_t^3 + R_{3,t} + \varepsilon_t \tag{114}$$

which additionally contains a cubic effect, a polynomial effect of order 3. Based on this, we can theoretically include many polynomial orders say P into a polynomial regression model. This is

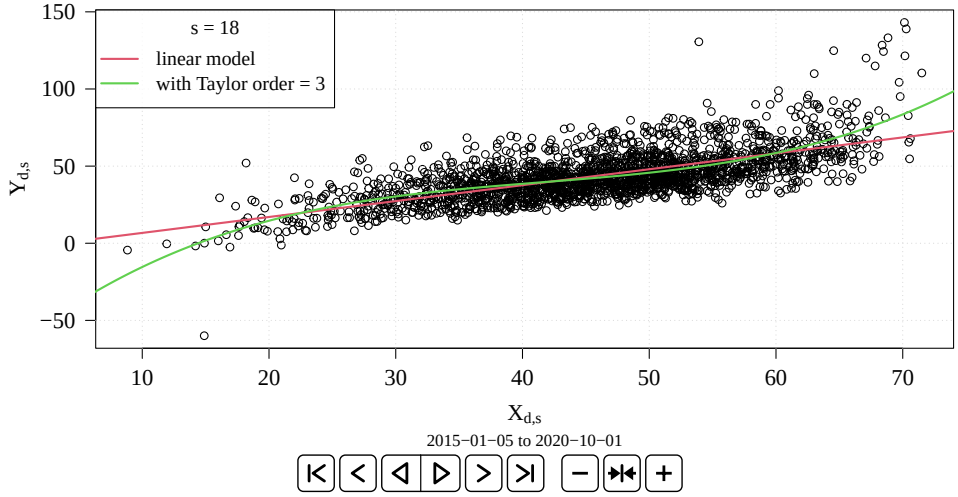
$$Y_t = \beta_0 + \sum_{p=1}^{P} \beta_p X_t^p + \varepsilon_t. \tag{115}$$

A problematic aspect about polynomial regression is the tail index issue. Similarly to the square, it holds that X_t^p has a tail index of α/p if X_t has a tail index of α. So high order polynomial approximations should be avoided if we have heavy tailed data. Unfortunately, this is often the case for electricity price data.

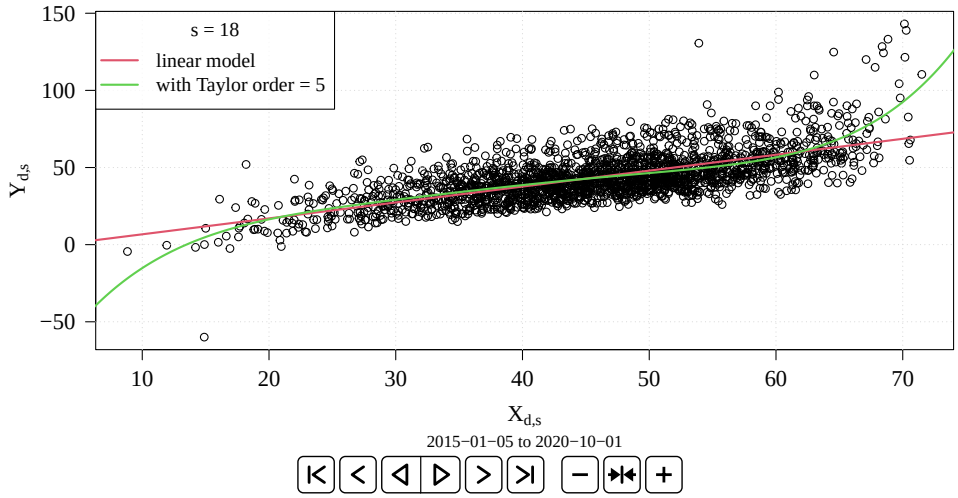
Moreover, high-order polynomial regression tend to be unstable in areas where we have only a few observations. This is usually, the case for very low and high values of our regressor X_t. This phenomenon is visualized in Figure 41 where we see the polynomial regression for selected orders. We see that low orders give reasonable results. But if P is too large we tend to get unstable results.

We have seen that every smooth function can be approximated using polynomials, but they have also the tail index problem. In the introduction to non-linear effects modelling, we introduced the absolute value as well. Here, we want to mention that the absolute value function can be used for the approximation of every smooth function as well. So the resulting technique is also known as linear spline approximation. The key element of the approximation is a grid $\mathbb{G}$ (also known as set of knots) on the support of X_t. The grid $\mathbb{G} = (c_1, c_2, \ldots, c_K)$ has K knots that are usually ordered increasingly, so $c_k < c_{k+1}$. The corresponding model using absolute values (linear splines) as non-linear approximation technique is given by

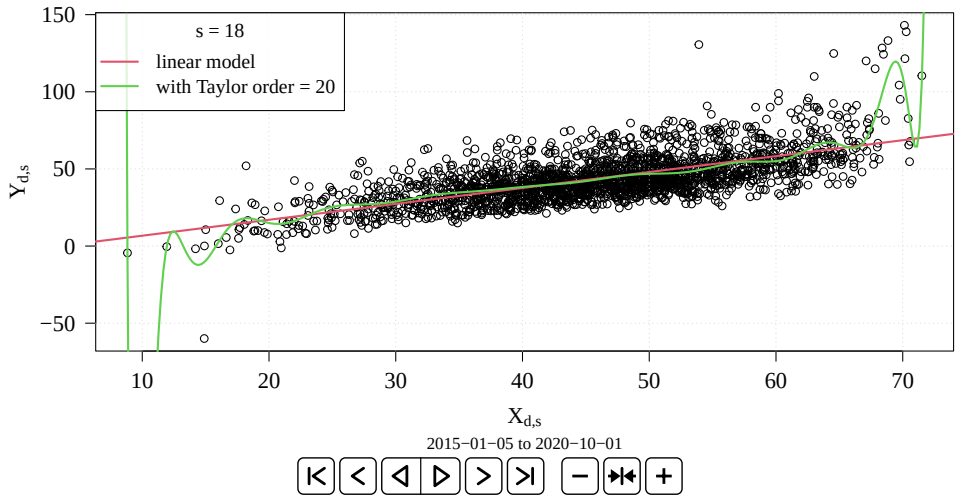
$$Y_t = a_{0,0} + a_{0,1} X_t + \sum_{k=1}^{K} a_k |X_t - c_k| + \varepsilon_t.$$



(a) Model with Taylor approximation of order $P = 3$



(b) Model with Taylor approximation of order $P = 5$



(c) Model with Taylor approximation of order $P = 20$

Figure 41: Scatter plot of $X_{d,s}^{\text{ResLoad}}$ and $Y_{d,s}$ for $s \in \mathbb{S}$ with fitted linear model $Y_{d,s} = \beta_{s,0} + \beta_{s,1} X_{d,s}^{\text{ResLoad}} + \varepsilon_{d,s}$ and polynomial regressions of selected orders P.

As we know that every absolute can be written as combination of maximum functions. Therefore we can rewrite this to

$$Y_t = \widetilde{a}_0 + \sum_{k=0}^{K} \widetilde{a}_k \max(X_t, c_k) + \varepsilon_t. \tag{116}$$

In this notation Often, $c_0 = -\infty$ is included in the grid as it gives automatically the linear part.

We can estimate the model using standard OLS. Still, we have to specify the grid $\mathbb{G}$ in advance. A suitable approach in practice is to use certain quantiles of the given data X_t. In detail we may take the grid $\mathbb{G} = \{q_p(X_t)|p \in \mathbb{P}\}$ with the grid probabilities $\mathbb{P} = \{\frac{1}{K}, \frac{2}{K}, \ldots, \frac{K-1}{K}\}$ and p-quantile q_p.

Figure illustrates the mentioned approach 42. As pointed out, this is usually a relatively robust approach compared to the polynomial regression. So for similar size of parameters the linear spline models behave more robust in the sense that we do not observe the wiggle behaviour for very low and high values of or regressors as seen in Figure 41. However, if the grid size K is too large we get unstable results as well. However, the unstable areas appear usually across the full support of our regressor X_t.

In general non-linear effects in electricity price modelling are mostly the time in the price spike areas which are usually very rarely observed. In the often observed areas, the non-linear effects seem to be weak. There is a clear difference between electricity price and load forecasting where we usually observe strong non-linear relationships between the temperature and the load.

Finally, note that there are several modern estimation techniques that use these kinds of non-linear function approximation techniques by default. These methods are sometimes regarded as semi-parametric or non-parametric approaches. E.g. in literature spline approximation methods are often regarded as non-parametric method. A modelling technique that is quite popular in energy forecasting in general and is also used in electricity price forecasting (see e.g. ISF2016) is the generalized additive model (GAM). Here the model structure is relatively general as well. An example, of such a model structure is given by

$$Y_t = f_1(X_t) + f_2(Z_t) + \varepsilon_t \tag{117}$$

where f_1 and f_2 are non-linear functions that characterize the non-linear impact of X_t and Z_t on Y_t. Both f_1 and f_2 have to be estimated/determined within the estimation procedure. Typically, those models face the problem overfitting and are often combined with corresponding solutions, esp. using shrinkage methods. Therefore, we first cover in the next section the overfitting problem and standard solutions first. Afterwards, we spend a full section just on GAMs.

9.5 Data transformations

In the previous subsection, we have seen that we can apply non-linear transformations on the regressors $\boldsymbol{X}_{d,s}$ to extend the feature space. Those transformation can also be used to reduce the tail index of the regressors. This is especially useful if we have heavy tailed data, hence those transformatons are often referred as variance stabilizing transformations (VST). In this subsection, we want to focus on the transformation of the response variable $Y_{d,s}$. In general, we can apply any function g which has an inverse g^{-1} on the response variable $Y_{d,s}$ and consider model for the transformed data, create the forecast $\widehat{g(Y)}_{d,s}$ and transform it back by $\widehat{Y}_{d,s} = g^{-1}(\widehat{g(Y)}_{d,s})$. However, when doing so there are multiple issues we have to be aware about. First, we have to think clearly about the consequences on the remaining regressors $\boldsymbol{X}_{d,s}$, they might have to be transformed as well. Consider e.g. an AR(1) model, here it is quite obvious, that we have to transform the response variable $Y_{d,s}$ and the regressor $Y_{d-1,s}$ in the same way:

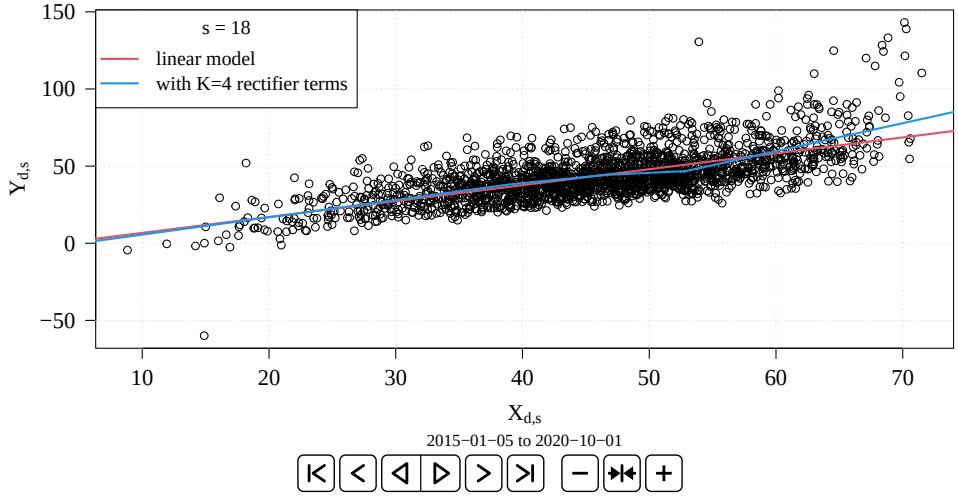
$$g(Y_{d,s}) = g(Y_{d,s}) + \epsilon_{d,s}$$

Otherwise, the model does not make sense anymore. In the next, we concentrate on the transformation of the response variable $Y_{d,s}$, but the same can and sometimes should be applied to some regressors $X_{d,s,i}$ as well.

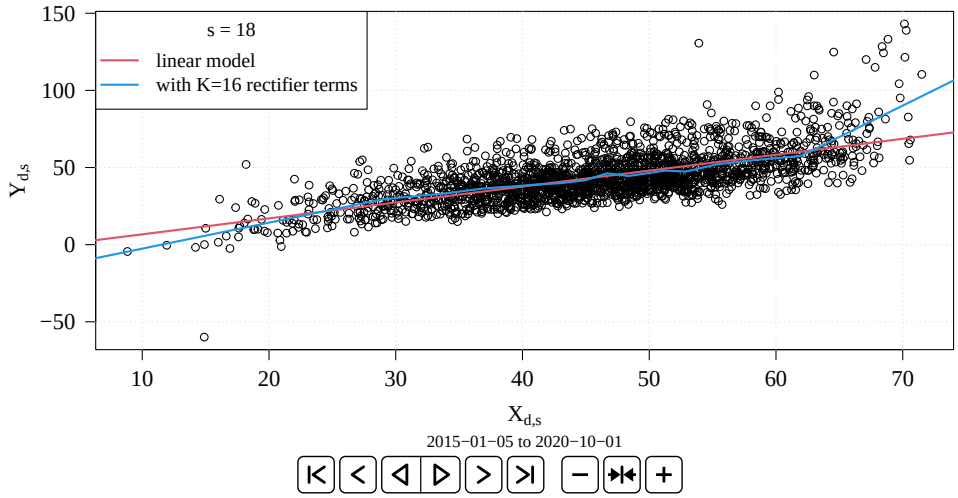
The function g will be a non-linear function (because for linear functions we would receive a linear model again). This means that the scale on which we apply the transformation is relevant. Thus, typically, we apply a normalization/standardization of $Y_{d,s}$ prior to application of g, this is that

$$g\left(\frac{Y_{d,s} - a}{b}\right)$$

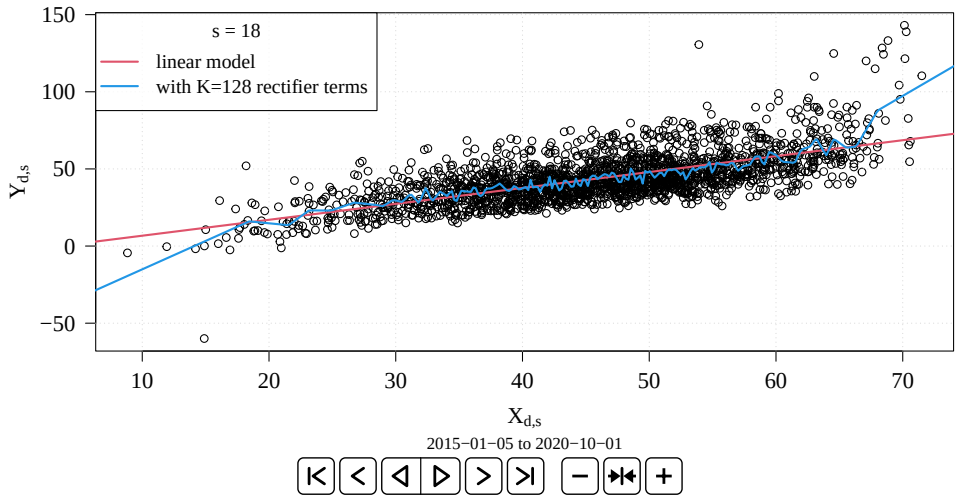
for some location parameter a and scale parameter b. The latter also takes the role of making the scaled result unit-free as b should have the same unit as $Y_{d,s}$. Usually, two optiones are considered. The first option is to standardize $Y_{d,s}$ using the sample mean and sample standard



(a) Linear spline model with $K = 4$ rectifiers



(b) Linear spline model with $K = 16$ rectifiers



(c) Linear spline model with $K = 128$ rectifiers

Figure 42: Scatter plot of $X_{d,s}^{\text{ResLoad}}$ and $Y_{d,s}$ for $s \in \mathbb{S}$ with fitted linear model $Y_{d,s} = \beta_{s,0} + \beta_{s,1} X_{d,s}^{\text{ResLoad}} + \varepsilon_{d,s}$ and linear spline regressions with K rectifiers.

deviation, as considered already in (17). The second option is to standardize $Y_{d,s}$ using the median for a and the median absolute deviation (MAD) for b. This is typically referred as robust standardization. Note, that for the MAD there are multiple definitions. The most common one is the median of the absolute deviations from the median. However, there are also other definitions, e.g. the mean of the absolute deviations from the median. In addition, there are different considerations of the scaling. Often, the plain MAD is multiplied with a factor

to make the MAD to a consistent estimator of the standard deviation under normality. The scaling is also differently treated by default in several software packages, for instance in base `R` the MAD is multiplied by 1.4826 to make it consistent with the standard deviation under normality. In `python` the MAD is not divided by any factor. In the following, we will use the MAD with factor.

Now, the question is on the choice of the transformation function g. In macroeconomics and finance, a typical transformation for prices is the logarithmic transformation. The log-transformation makes sense if there is some underlying economic growth in the data. However, electricity prices may be negative, therefore such transformations are not available.

In econometrics, the most popular alternative transformation to the logarithm is the Box-Cox transformation (which has also extensions to the negative values), but there are also other alternatives. In [UWZ17] several different options are tested in expert model frameworks for electricity price forecasting in various European markets. We can distinguish the typical transformations into two classes, those which can act with the normalization parameters a and b, and those which do not need a and b, but a suitable amount of historic data (usually the sample used for parameter estimation).

Plausible transformations for g which rely on location-scale-transformation with a and b are:

- identity:
$$\mathrm{id}_{a,b}(y) = \frac{y-a}{b} \tag{118}$$
with its inverse
$$\mathrm{id}^{-1}_{a,b}(z) = bz + a \tag{119}$$
which has no effect in linear regression models, but potentially in non-linear models.

- K-sigma clippling:
$$\mathrm{sigmaclip}_{a,b,K}(y) = \begin{cases} \mathrm{sgn}(\frac{y-a}{b})K & \text{for } |\frac{y-a}{b}| > K, \\ \frac{y-a}{b} & \text{for } |\frac{y-a}{b}| \leq K, \end{cases} \tag{120}$$
with its *inverse*
$$\mathrm{sigmaclip}^{-1}_{a,b,K}(z) = bz + a \tag{121}$$
In practice, especially at the beginning of the 2000's the 3-sigma clippling was quite popular in electricity price forecasting. However, it is not a strictly monotonic increasing function, therefore the inverse is not well defined. A suitable solution is the 3-sigma-log clippling which continues the 3-sigma clippling in the tails using the logarithm.

- K-sigma-log clippling
$$\mathrm{sigmalogclip}_{a,b,K}(y) = \begin{cases} \mathrm{sgn}(\frac{y-a}{b}) \left(\log(|\frac{y-a}{b}| - K - 1) + K\right) & \text{for } |\frac{y-a}{b}| > K, \\ \frac{z-a}{b} & \text{for } |\frac{z-a}{b}| \leq K, \end{cases} \tag{122}$$
with inverse:
$$\mathrm{sigmalogclip}^{-1}_{a,b,K}(z) = \begin{cases} b\,\mathrm{sgn}(z) \left(e^{|z|-K} + K - 1\right) + a & \text{for } |z| > K, \\ bz + a & \text{for } |z| \leq K. \end{cases} \tag{123}$$
Similarly as for sigmaclip the clipping at $K = 3$ seems to be a plausible option.

- logistic transformation
$$\mathrm{logistic}_{a,b}(y) = \frac{1}{1 + e^{-\frac{y-a}{b}}} \tag{124}$$
with inverse
$$\mathrm{logistic}^{-1}_{a,b}(z) = b \log\left(\frac{z}{1-z}\right) + a \tag{125}$$
This transformation is often used in binary classification problems, but it is also a suitable transformation for electricity prices. It is strictly monotonic increasing, bounded on $(0, 1)$ and point symmetric.

- generalized logistic transformation

$$\text{glogistic}_{a,b,\lambda}(y) = \left(1 + e^{-\frac{y-a}{b}}\right)^{-\lambda} \tag{126}$$

with its inverse

$$\text{glogistic}^{-1}_{a,b}(z) = b \log\left(\frac{\text{sgn}|z|^{1/\lambda}}{1 - \text{sgn}(z)|z|^{1/\lambda}}\right) + a \tag{127}$$

Obviously, for $\lambda = 1$ the glogistic yields the original logistic function. The function is strictly monotonic increasing, bounded on $(0, 1)$ and but not symmetric, expect for $\lambda = 1$.

- asinh (inverse hyperbolic sine, area sinus hyperbolicus)

$$\text{asinh}_{a,b}(y) = \text{asinh}\left(\frac{y-a}{b}\right) = \log\left(\frac{y-a}{b} + \sqrt{1 + \left(\frac{y-a}{b}\right)^2}\right) \tag{128}$$

with inverse

$$\text{asinh}^{-1}_{a,b}(z) = b \sinh(z) + a = b\left(\frac{e^z - e^{-z}}{2}\right) + a \tag{129}$$

This is a strictly monotonic which is point symmetric and takes values on the full real line. It has exponential dumping thus, a finite tail index will be transformed to ∞, thus it omits problems with heavy tails in the algorithms.

- tanh (hyperbolic tangent, tangens hyperbolicus)

$$\tanh_{a,b}(y) = \tanh\left(\frac{y-a}{b}\right) = \frac{e^{\frac{y-a}{b}} - e^{-\frac{y-a}{b}}}{e^{\frac{y-a}{b}} + e^{-\frac{y-a}{b}}} \tag{130}$$

with its inverse

$$\tanh^{-1}_{a,b}(y) = b\,\text{atanh}(y) + a = \frac{b}{2}\log\left(\frac{1+y}{1-y}\right) + a. \tag{131}$$

It has similar properties to the logistic function, it is also point symmetric and is bounded.

- atan (arctangent, arcus tangens)

$$\text{atan}_{a,b}(y) = \text{atan}\left(\frac{y-a}{b}\right) = \arctan\left(\frac{y-a}{b}\right) \tag{132}$$

and its inverse

$$\text{atan}^{-1}_{a,b}(z) = b \tan(z) + a \tag{133}$$

- (generalized) Box-Cox transformation

$$\text{BoxCox}_{a,b,\lambda}(y) = \text{sgn}\left(\frac{y-a}{b}\right)\begin{cases}\frac{\left(|\frac{y-a}{b}|+1\right)^{\lambda}-1}{\lambda} & \text{for } \lambda > 0,\\ \log(|\frac{y-a}{b}|+1) & \text{for } \lambda = 0\end{cases} \tag{134}$$

with its inverse

$$\text{BoxCox}^{-1}_{a,b,\lambda}(z) = a + b\,\text{sgn}(z)\begin{cases}(\lambda|z|+1)^{\frac{1}{\lambda}} - 1 & \text{for } \lambda > 0,\\ e^{|z|} - 1 & \text{for } \lambda = 0.\end{cases} \tag{135}$$

For $0 < \lambda < 1$ it exhibits a polynomial dumping effects, thus polynomial tails keep polinomial but the tail index increases.

- poly-transformation

$$\text{poly}_{a,b,c,\lambda}(y) = \text{sgn}\left(\frac{y-a}{b}\right)\left(\left|\left|\frac{y-a}{b}\right| + \left(\tfrac{c}{\lambda}\right)^{\frac{1}{\lambda-1}}\right|^{\lambda} - \left(\tfrac{c}{\lambda}\right)^{\frac{\lambda}{\lambda-1}}\right), \tag{136}$$

with inverse:

$$\text{poly}^{-1}_{a,b,c,\lambda}(z) = \text{sgn}(z)\left(\left(|z| + \left(\tfrac{c}{\lambda}\right)^{\frac{\lambda}{\lambda-1}}\right)^{\frac{1}{\lambda}} - \left(\tfrac{c}{\lambda}\right)^{\frac{1}{\lambda-1}}\right) b + a. \tag{137}$$

may be regarded as a generalization of the Box-Cox transformation. It has similar properties as the Box-Cox transformation, but it is more flexible du to the tuning parameter c. It is also a polynomial transformation, but the tail index increases for $0 < \lambda < 1$. It is designed so that the slope in the origin is c.

- mirror log-transformation

$$\text{mlog}_{a,b,c}(y) = \text{sgn}\left(\frac{y-a}{b}\right)\left[\log\left(\left|\frac{y-a}{b}\right| + \tfrac{1}{c}\right) + \log(c)\right], \tag{138}$$

with inverse

$$\text{mlog}^{-1}_{a,b,c}(z) = \text{sgn}(z)\left(e^{|z|-\log c} - \tfrac{1}{c}\right) b + a \tag{139}$$

In addition there are transformations which do not require the location-scale transformation, but require the full historic data. Considered examples in practice are propability integral transforms (PITs) for specific distributions continuous distribution G. This is defined by

$$G\text{PIT}_{\boldsymbol{Y}}(y) = G^{-1}\left(\text{PIT}_{\boldsymbol{Y}}(y)\right) = G^{-1}\left(\hat{F}_{\boldsymbol{Y}}(y)\right), \tag{140}$$

where G^{-1} is the inverse of G, $\boldsymbol{Y}$ the historic data vector and $\hat{F}_{\boldsymbol{Y}}$ the empirical distribution function. Thus, if $\text{GPIT}_{\boldsymbol{Y}}$ is applied on the full data $\boldsymbol{Y}$ it has essentially the same distribution as G, e.g. the corresponding histogram looks like a perfect fit. The backtransformation

$$G\text{PIT}^{-1}_{\boldsymbol{Y}}(z) = \hat{F}^{-1}_{\boldsymbol{Y}}\left(G(z)\right), \tag{141}$$

can be used to create forecasts given a forecast of the transformed data. When considering the normal distribution for G this results in the NPIT transformation. This removes all polynomial tails from the data. Other options are e.g. the t-distribution for a fixed degree of freedom.

However, empirical results in [UWZ17] show that the asinh (inverse hyperbolic sine, or area sinus hyperbolicus) shows robust results. We regard it as suitable as well, especially because no hyperparameter tuning is required.

Still, when applying data transformations on the response (but similarly also on selected regressors) we have to be aware of the consequences. When we want to predict the median of the electricity prices $Y_{d,s}$ (i.e. minimizing MAE) we do not have a problem in the application. Because the median of $Y_{d,s}$ is the same as the median of $g(Y_{d,s})$ backtransformed afterwards using g^{-1}. The reason is that quantiles are preserved und monotonic transformations, and the median is a quantile. Unfortunately, this does not hold for the mean. The reason is that the expected value operator is a linear operator which is not preserved under non-linear transformations. Thus, if we want to predict the mean of $Y_{d,s}$ (i.e. minimizing MSE) we have to apply the backtransformation g^{-1} on the prediction $\widehat{g(Y)}_{d,s}$ as well. This is also known as backtransformation bias. The bias is usually not very large if the data is close to symmetric, but it is not negligible as well. However, when considering a probabilistic forecasting setting (e.g. by specifying a distribution for $\varepsilon_{d,s}$) the mean resp. the expected value can be predicted without a bias, see e.g. [NZ20a] for more details.

Finally, we want to mention that online learning is still possible in all situations, but allows a more complex data handling for the PIT-approaches. For the a and b based approaches, the mean and standard deviation or the median and MAD have to be updated online as well, but corresponding formulas are available. Also the PIT-approaches can be updated online, but the corresponding formulas are more complex.

Another issue which we did not mention clearly how to estimate a and b - on full sample or for each hour seperately.

9.6 Tasks

Hints and the corresponding programming part are provide in Section R.8 in `R` and in Section py.6 in `python`.

1. Consider the model variant of (93) which we refer as **expert.redadv** and uses somewhat different parameters:

$$\begin{aligned} Y_{d,s} = \beta_{s,0} &+ \beta_{s,1} Y_{d-1,s} + \beta_{s,2} Y_{d-2,s} + \beta_{s,3} Y_{d-7,s} + \beta_{s,4} Y_{d-1,S-1} + \beta_{s,5} Y_{d-1,\min} + \beta_{s,6} Y_{d-1,\max} \\ &+ \beta_{s,7} \mathrm{DoW}_d^1 + \beta_{s,8} \mathrm{DoW}_d^6 + \beta_{s,9} \mathrm{DoW}_d^7 + \beta_{s,10} X_{d,s}^{\mathrm{Load}} + \beta_{s,11} X_{d,s}^{\mathrm{DARES}} \\ &+ \beta_{s,12} X_{d-2}^{\mathrm{EUA}} + \beta_{s,13} X_{d-2}^{\mathrm{Coal}} + \beta_{s,14} X_{d-2}^{\mathrm{NGas}} + \varepsilon_{d,s} \end{aligned} \tag{142}$$

   Implement a forecast function for **expert.redadv**.

2. Calculate sample conditional correlations between prices and $Y_{d-1,min}$ and $Y_{d-1,max}$ conditioned on the expert model **expert.last**. Is it worth to consider a model defined as

$$\begin{aligned} Y_{d,s} = \beta_{s,0} &+ \beta_{s,1} Y_{d-1,s} + \beta_{s,2} Y_{d-2,s} + \beta_{s,3} Y_{d-7,s} + \beta_{s,4} Y_{d-1,S-1} + \beta_{s,5} Y_{d-1,\min} + \beta_{s,6} Y_{d-1,\max} \\ &+ \beta_{s,7} \mathrm{DoW}_d^1 + \beta_{s,8} \mathrm{DoW}_d^6 + \beta_{s,9} \mathrm{DoW}_d^7 + \varepsilon_{d,s}? \end{aligned} \tag{143}$$

3. Implement an expert model modification that includes quadratic terms. To do it, use equation (112) with Taylor approximation of order 2.

4. Consider again the simple model (88). Compare the corresponding polynomial regression for $P \in \ldots 1, 2, \ldots, 7$ for various calibration window lengths D. Which combination gives you the best forecasting performance?

5. Estimate a model of the same structure as **expert.adv** but without any autoregressive effects. How many parameter has this model? Now, add all interaction terms to the model. How many parameters has this model with interactions? Does the forecasting performance improve? Use $D = 1095$ for the forecasting study.

6. Perform a forecasting study for the standard expert model, and compare different variance stabilizing transformations. Do you find something better performing than the identity?

10 Dealing with overfitting in linear models

We have seen many dependency effects that can be incorporated into electricity price forecasting models. These are for instance, autoregressive effects including cross-period autoregressive effects, different seasonal effects, esp. daily, weekly and annual patterns, non-linear effects including all variants of interaction effects. It is clear that a sophisticated electricity price model will include multiple or even all of these effects. However, it remains challenging to identify an appropriate model structure. This is because there is a huge number of possible combinations of all these effects. It is computationally infeasible to check all possible models. Therefore, it is suitable to consider more sophisticated estimation method that can deal with that problem.

Another problem with the huge amount of possible effects is that we can only include a certain amount of regressors into a model to guarantee the existence of an OLS solution. If the number of parameters (the length of $\boldsymbol{\beta}_s$ in (8)) in a linear regression is larger than the number of available observations (strictly the number of rows of $\boldsymbol{X}_s$ in (13)), then the matrix $\boldsymbol{X}_s'\boldsymbol{X}_s$ in the OLS estimator (13) has no inverse. So we cannot perform OLS estimation. Still, even if we can compute the OLS estimator, the estimator $\widehat{\boldsymbol{\beta}}_s^{\text{ls}}$ might not be a good one for the true $\boldsymbol{\beta}_s$. This happens if the number of parameters (p_s) is relatively close to the number of observations (D). This general problem is also known as overfitting or more general the curse of dimensionality. In particular, the fraction of parameters to observation p_s/D is important. The closer p_s/D to 1 the larger the problems in the estimation of $\boldsymbol{\beta}_s$. As a rule of thumb, we can say that if approximately $p_s/D < 1/100$ then the estimation error due to the dimensionality is not dominant. If approximately $p_s/D > 1/100$, but $p_s/D < 1/10$ then the dimensionality of the model causes a reasonable amount of error so that the consideration of sophisticated estimation methods should be taken into account. If $p_s/D > 1/10$, a sophisticated estimation method should be definitively taken into account, as the corresponding model is likely overfitted. If $p_s/D > 1/2$, the OLS estimation results become highly unreliable. Note that these are just rules of thumb, mainly tailored for electricity price forecasting applications.

In this section, we will learn linear model estimation methods that can deal with the combinatorial model selection and the curse of dimensionality problem.

First, we recap the OLS estimator. Given the sample $\mathcal{Y}_s = (Y_{1,s}, \ldots, Y_{D,s})'$, the OLS estimator is the solution of

$$\widehat{\boldsymbol{\beta}}_s^{\text{ls}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^{p_s+1}}{\arg\min} \|\mathcal{Y}_s - \mathcal{X}_s\boldsymbol{\beta}\|_2^2 = \underset{\boldsymbol{\beta}\in\mathbb{R}^{p_s+1}}{\arg\min} (\mathcal{Y}_s - \mathcal{X}_s'\boldsymbol{\beta})'(\mathcal{Y}_s - \mathcal{X}_s'\boldsymbol{\beta}) = \underset{\boldsymbol{\beta}\in\mathbb{R}^{p_s+1}}{\arg\min} \sum_{d=1}^{D} (Y_{d,s} - \boldsymbol{X}_{d,s}'\boldsymbol{\beta})^2. \quad (144)$$

So the estimator $\widehat{\boldsymbol{\beta}}_s^{\text{ls}}$ is the minimizer of the residual sum of squares as given in equation (144). The sophisticated linear model estimation methods rely usually on the evaluation of the variation of the different regression vectors in the regression matrix $\mathcal{X}_s$. Therefore, the estimation algorithms require scaled regressors. Hence, we remember the scaling procedure as discussed in Subsection 3.3, i.e. $\widetilde{\mathcal{Y}}_s$ and $\widetilde{\mathcal{X}}_s$ are scaled such that the columns have zero mean and $\|\cdot\|_2$-norm of 1 (which corresponds to variance of 1 for p_s zero mean variables).

10.1 Dimension reduction using SVD/PCA

One way to reduce the overfitting problem is to reduce the dimension of the regressor matrix $\mathcal{X}_s$ or its scaled version $\widetilde{\mathcal{X}}_s$. Remember, $\widetilde{\mathcal{X}}_s$ has p_s columns. The idea of dimension reduction techniques is to find a new matrix say $\mathcal{Z}_s$ that has $r_s < p_s$ columns but contains us much of the (hopefully relevant) information as possible. The likely most popular dimension reduction technique is principal component analysis (PCA). PCA can be regarded as a special case of singular value decomposition (SVD) which we will study in more detail. For any $D \times p_s$-dimensional matrix $\widetilde{\mathcal{X}}_s$ the SVD is a decomposition defined by

$$\widetilde{\mathcal{X}}_s = \mathcal{U}_s\mathcal{D}_s\mathcal{V}_s' \quad (145)$$

with $D \times p_s$-dimensional matrix $\mathcal{U}_s$, $p_s \times p_s$-dimensional diagonal matrix $\mathcal{D}_s$ of singular values and $p_s \times p_s$-dimensional matrix $\mathcal{V}_s$. The decomposition (145) is an SVD decomposition if $\mathcal{U}_s'\mathcal{U}_s = \boldsymbol{I}_D$ and $\mathcal{V}_s'\mathcal{V}_s = \boldsymbol{I}_{p_s}$ holds. Further, without loss of generality it is usually assumed that the singular

values on the diagonal of $\mathcal{D}_s$ have only non-negative values that are sorted decreasingly. Such a singular value decomposion is (usually) unique and can be computed easily using `R` (function `svd`) or `python` (`linalg.svd` in `numpy`) and computing this decomposition is usually quite fast. SVD is highly related to PCA. Most importantly, we can define the $D \times p_s$-dimensional matrix of principal components (or scores) $\mathcal{S}_s$ of $\widetilde{\mathcal{X}}_s$ by

$$\mathcal{S}_s = \widetilde{\mathcal{X}}_s \mathcal{V}_s = \mathcal{U}_s \mathcal{D}_s. \tag{146}$$

In the PCA terminologythe matrix $\mathcal{L}_s = \mathcal{V}_s'$ is usually referred to as the loadings matrix. The loadings can be interpreted easily in the PCA context. Every column in the scores $\mathcal{S}_s$ is decomposed as a linear composition of a row of the loadings (or a column of $\mathcal{V}_s$). $\mathcal{U}_s$ can be interpreted as a scaled version of $\mathcal{S}_s$ but having the same correlation structure. Due to $\mathcal{U}_s'\mathcal{U}_s = \boldsymbol{I}_D$ all columns in $\mathcal{U}_s$ or $\mathcal{S}_s$ are uncorrelated, this has important implications for OLS estimators. For applications it is important to know that the singular values (the diagonal elements of $\mathcal{D}_s$) decode the information density of the corresponding column in $\mathcal{U}_s$ resp. $\mathcal{S}_s$ with respect to $\mathcal{X}_s$. Thus, the first column in $\mathcal{U}_s$ or $\mathcal{S}_s$ contains most dominant information, the second column the 2nd most dominant information etc..

Next we want to utilize the SVD resp. PCA decomposition for estimating our linear model, see e.g. [MRNF21]. Therefore, assume we want to solve a regression with respect to $\mathcal{S}_s$ Due to (146) the linear regression term $\mathcal{S}_s\gamma_s$ is equivalent to $\widetilde{\mathcal{X}}_s\mathcal{V}_s\gamma_s$. Thus, comparing with standard OLS on $\widetilde{\mathcal{X}}_s$ we have $\boldsymbol{\beta}_s = \mathcal{V}_s\gamma_s$. Hence, the optimization

$$\widehat{\boldsymbol{\gamma}}_s^{\text{ls-PCA}} = \underset{\boldsymbol{\gamma} \in \mathbb{R}^{p_s}}{\arg\min} \, \|\mathcal{Y}_s - \mathcal{S}_s\boldsymbol{\gamma}\|_2^2 \tag{147}$$

gives an equivalent scaled solution of (144) which satisfies $\widehat{\widetilde{\boldsymbol{\beta}}}_s = \mathcal{V}_s\widehat{\boldsymbol{\gamma}}_s^{\text{ls-PCA}}$.

Now, we may use the idea of dimension reduction for $\mathcal{S}_s$ to reduce the number of columns in $\mathcal{S}_s$ and consider a regression like in (147) with the dimension reduced matrix. Let $\mathcal{S}_{s,\lambda}$ for $\lambda \in \{1, \ldots, p_s\}$ the number of selected first columns in $\mathcal{S}_s$. Thus, for $\lambda = p_s$, it holds $\mathcal{S}_{s,p_s} = \mathcal{S}_s$, and e.g. $\mathcal{S}_s(2)$ is a $D \times 2$-dimensional matrix of the two most dominant information columns of the scores. This allows us to define $\lambda \in \{1, \ldots, p_s\}$

$$\widehat{\boldsymbol{\gamma}}_{s,\lambda}^{\text{ls-PCA}} = \underset{\boldsymbol{\gamma} \in \mathbb{R}^{\lambda}}{\arg\min} \, \|\mathcal{Y}_s - \mathcal{S}_{s,\lambda}\boldsymbol{\gamma}\|_2^2 \tag{148}$$

Now, $\widehat{\widetilde{\boldsymbol{\beta}}}_{s,\lambda}^{\text{ls-PCA}} = \mathcal{V}_{s,\lambda}\widehat{\boldsymbol{\gamma}}_{s,\lambda}^{\text{ls-PCA}}$, where the $p_s \times \lambda$-dimensional matrix $\mathcal{V}_{s,\lambda}$ is $\mathcal{V}_s$ with only the first λ columns, will in general differ from $\widehat{\widetilde{\boldsymbol{\beta}}}_s$ but the closer λ to p_s the closer the solution should be to $\widehat{\widetilde{\boldsymbol{\beta}}}_s$. For $\lambda = p_s$ both coincide, i.e. we have $\widehat{\widetilde{\boldsymbol{\beta}}}_{s,\lambda}^{\text{ls-PCA}} = \widehat{\widetilde{\boldsymbol{\beta}}}_s$.

$\mathcal{S}_{s,\lambda}$ has only λ columns, the estimation risk reduced while hopefully retaining the relevant information in the remaining λ columns of $\mathcal{S}_s$. Figure (43) illustrates $\widehat{\widetilde{\boldsymbol{\beta}}}_{s,\lambda}^{\text{ls-PCA}}$ for $\lambda \in \{1, \ldots, p_s\}$. We see that with increasing λ, the estimated parameters tend to have a greater spread, around zero. This indicates a higher estimation risk. However, in this example the estimation risk increase is moderate as we have only relatively few parameters. If we would add more parameters to the model, especially highly correlated ones, we would see a greater effect.

Obviously, a critical question is on the optimal selection of λ. From the methodological point of view we can compute all solutions for $\lambda \in \{1, \ldots, p_s\}$ of (148), the so called solution path and choose based on suitable criterion the optimal λ. We postpone the answer on the optimal selection of λ to the next section. However, here we want to point out that it is easy to compute the solution path (148) for all $\lambda \in \{1, \ldots, p_s\}$. Remember, by design, all columns in $\mathcal{S}_s$ are uncorrelated. Thus, it holds that $\widehat{\boldsymbol{\gamma}}_{s,\lambda}^{\text{ls-PCA}}$ is the same as the first λ values of $\widehat{\boldsymbol{\gamma}}_s^{\text{ls-PCA}}$, i.e.

$$\widehat{\boldsymbol{\gamma}}_{s,\lambda}^{\text{ls-PCA}} = (\widehat{\gamma}_{s,1}^{\text{ls-PCA}}, \ldots, \widehat{\gamma}_{s,\lambda}^{\text{ls-PCA}})'.$$

Thus, we only have to solve (147) to receive the full solution path which speeds up computations substantially. This also allows us to choose an optimal λ-value without additional effort based on an in-sample criteria, like an information criterion.

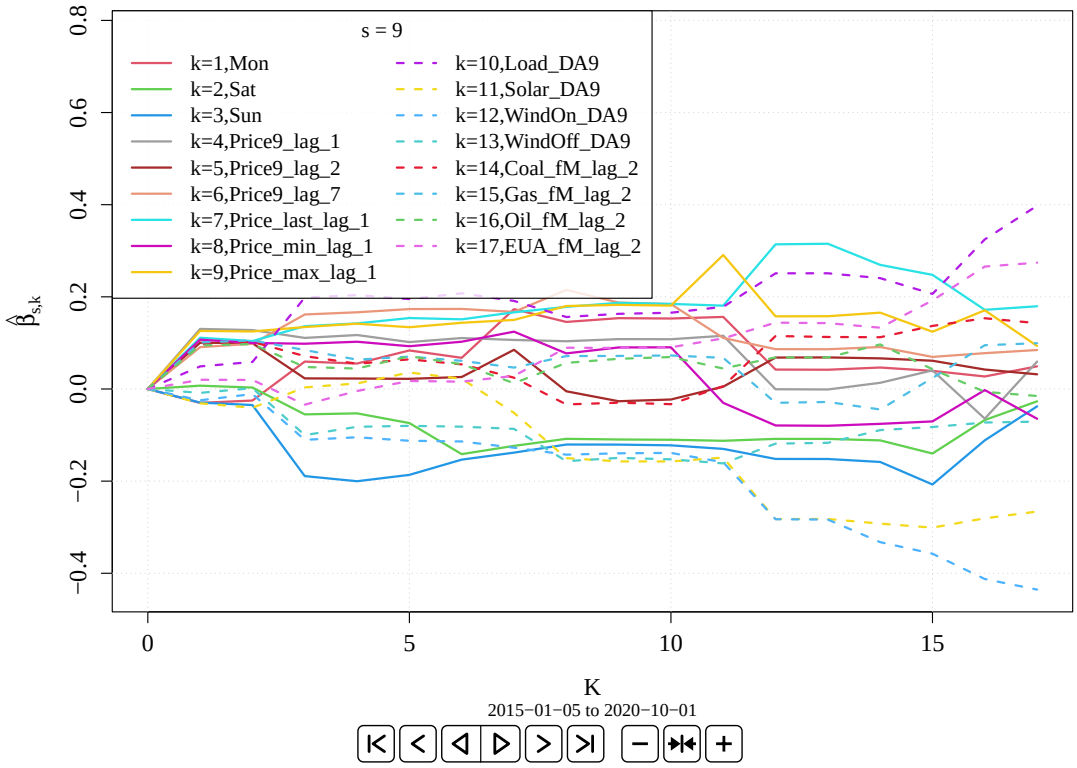


Figure 43: Estimated scaled parameter vectors $\widehat{\widetilde{\boldsymbol{\beta}}}_{s,\lambda}^{\text{ls-PCA}}$ for different $\lambda_s \in \{1, \ldots, p_s\}$ values and $s = 10$.

In equation (145) we considered a scaled regression matrix $\widetilde{\mathcal{X}}_s$ for the singular value decomposition. Clearly, we can replace it formally without any problems by the unscaled matrix $\mathcal{X}_s$. Doing so may have some pros but also has to be done carefully. The biggest problem is that SVD is scale dependent. For example if we scale one regressor by a huge margin, the corresponding error will dominate the decomposition. So the first score will be essentially the corresponding regressor. Thus, we essentially loose the wanted dimensions reduction properties that combines(!) redundant information in the scores. However, when the regressor matrix contains regressors in the same unit that carry similar information PCA is a very suitable choice. This might be the case when considering the autoregressive components e.g. $(Y_{d-1,s}, \ldots, Y_{d-k,s})$ or the set of all S residual demands $(X_{d,0}^{\text{ResLoad}}, \ldots, X_{d,S-1}^{\text{ResLoad}})$. Here, we want to remind that the external inputs, the day-ahead load, as well as the solar, wind onshore and offshore production $(X_{d,s}^{\text{Load}}$, $X_{d,s}^{\text{Solar}}$, $X_{d,s}^{\text{WindOn}}$, $X_{d,s}^{\text{WindOff}})$ have the same units and contribute with the same interpretation in the supply-stack model to the electricity price formation. When doing so, we really concentrate on the uncertainty driving components in the dimension reduction approach.

It is also feasible to consider the four mentioned external regressor parts $(X_{d,0}^{\text{ext}}, \ldots, X_{d,S-1}^{\text{ext}})$ with ext $\in \{\text{Load}, \text{Solar}, \text{WindOn}, \text{WindOff}\}$ and apply PCA separately to each component. Due to the same units, we can choose a single cut-off value $\mathrm{d}_{\min}$ as tuning parameter for selecting λ. $\mathrm{d}_{\min}$ specifies the minimal principal component level in a reduced target matrix. Note that the principal components carry the same scale as the inputs. Therefore, determining λ for several matrices based on a single threshold $\mathrm{d}_{\min}$ will lead usually to matrices of different size. Their size corresponds to the explainable variation with respect to the corresponding units. For instance, consider wind onshore and offshore in Germany. We substantially more installed onshore capacity than offshore, and therefore larger absolute variations in the corresponding produced power. When using described dimension reduction we will see usually more remaining components in the onshore matrix than for the offshore matrix.

Thus, it makes sense to consider PCA only for relevant parts of the data. For some parts, especially deterministic parts like the weekday dummies it is not recommended to use PCA.

We discussed the application of PCA resp. SVD based on a linear model with regression matrix $\boldsymbol{X}_s$ for every delivery product $s \in \mathbb{S}$ separately. However, we can also consider multivariate models like (33) which involve only one big estimation with a large regression matrix $\mathbb{X}$. Intu-

itively, for PCA should be a more power tool for matrices with many regressors/columns. Still, we have to be careful when applying it, because some multivariate models contain plenty zeros in the design matrix $\mathbb{X}$ and can benefit from sparsity in the parameter estimation. However, when applying PCA on $\mathbb{X}$ we will destroy all sparsity pattern. This may dramatically increase the computational costs.

We have seen that PCA can be used to reduce the dimension of the regressor matrix and thus reduces the overfitting bias. Even though applying PCA is relatively popular in data science and various fields of application it is only an orthogonalization strategy for the input data. The dimension reduction process involves the loss of information. And this is completely decoupled from the parameter estimation/training procedure. In very simple words, it might happen that we through away relevant information for our linear model while keeping irrelevant information in the remaining matrix. Therefore, it makes sense to consider methods that can deal with overfitting in such a way that they measure the relevance of the regressors in our input matrix in relationship to our prediction target, the electricity prices. We will study one solution in the next subsection 10.2.

Finally, note that if required there are also online updating algorithms for PCA and SVD available which might be useful to speedup a forecasting study.

10.2 The lasso and ridge estimators

Now, we introduce another method to avoid overfitting. When defining the lasso and ridge estimators, we do this by defining the scaled versions first. The scaled lasso and ridge estimators are similarly defined as $\widehat{\widetilde{\boldsymbol{\beta}}}_s^{\text{ls}}$. They are given by

$$\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda,s}^{\text{lasso}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^D}{\arg\min} \|\widetilde{\mathcal{Y}}_s - \widetilde{\mathcal{X}}_s\boldsymbol{\beta}\|_2^2 + \lambda_s\|\boldsymbol{\beta}\|_1 \tag{149}$$

and

$$\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda,s}^{\text{ridge}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^D}{\arg\min} \|\widetilde{\mathcal{Y}}_s - \widetilde{\mathcal{X}}_s\boldsymbol{\beta}\|_2^2 + \lambda_s\|\boldsymbol{\beta}\|_2^2. \tag{150}$$

with both dependent on a tuning parameter $\lambda_s \geq 0$. Both $\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda,s}^{\text{lasso}}$ and $\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda,s}^{\text{ridge}}$ are minimizers of a loss function. This loss function contains as $\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda,s}^{\text{ls}}$ the residual sum of squares term $\|\widetilde{\mathcal{Y}}_s - \boldsymbol{\beta}'\widetilde{\mathcal{X}}_s\|_2^2$, but additionally a penalty term. For the ridge estimator we have an explicit solution of equation (150) available. This is given by

$$\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda,s}^{\text{ridge}} = (\widetilde{\mathcal{X}}_s'\widetilde{\mathcal{X}}_s + \lambda_s\boldsymbol{I})^{-1}\widetilde{\mathcal{X}}_s\widetilde{\mathcal{Y}}_s \tag{151}$$

with similar structure as the scaled OLS estimator (22). In general, the lasso estimator has no explicit solution. Only, if all inputs are orthogonal we have an explicit solution using so-called soft thresholding. The soft-thresholding operator $\mathcal{S}_\lambda$ is defined by

$$\mathcal{S}_\lambda(\beta) = \begin{cases} \text{sign}(\beta)\max(|\beta-\lambda|, 0) & |\beta| > \lambda \\ 0 & \text{, otherwise} \end{cases}. \tag{152}$$

It provides the lasso solution in orthogonal designs by applying it elementwise on the OLS solution: $\widehat{\widetilde{\beta}}_{\lambda,s,i}^{\text{lasso}} = \mathcal{S}_{\lambda_s}(\widehat{\widetilde{\beta}}_{\lambda,s,i}^{\text{OLS}})$. However, the lasso (149) with general regression matrix can be estimated efficiently using a numerical approximation algorithms. In 2004 [EHJT04] provided a fast algorithm that provides the entire solution path. This algortihm is based on LARS (least angle regression) and it provides a solution with 1 non-zero parameter, a next one with 2 non-zero parameters, etc. up to the OLS solution. Still, this algorithm has a computation time that increases quadratically in the number of parameters p of the linear model, formally the complexity is $\mathcal{O}(Dp^2)$. Fortunately, there is a fast coordinate descent algorithm, first explained in detail in [FHHT07][18] and implemented in the `glmnet` package in `R`, [FHT10]. This algorithm

[18] Similar versions were also known a few years before and known as shooting algorithm.
A few years later, [FHHT07] were the first who provided an efficient implementation.

iterates the application of $\mathcal{S}_{\lambda_s}$ operators until convergence. In contrast to the LARS algorithm the coordinate descent algorithm does provide the full solution path, but only a solution path for a selected grid of λ_s values. In turns out that this algorithm is basically of computational complexity of $\mathcal{O}(Dp)$[19]. Thus, doubling the number of parameters only doubles the computation time. Thus, coordinate descent is preferable to LARS for (very) large parameter spaces with hundreds or thousands of parameters.

If we have $\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda,s}^{\text{lasso}}$ and $\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda,s}^{\text{ridge}}$, we can rescale the scaled parameter estimate and receive the lasso and ridge estimators $\widehat{\boldsymbol{\beta}}_{\lambda,s}^{\text{lasso}}$ and $\widehat{\boldsymbol{\beta}}_{\lambda,s}^{\text{ridge}}$. Of course, we could also compute an non-scaled lasso or ridge estimator by applying formula (149) and (150) directly to $\mathcal{X}_s$ without any scaling. However, those estimators should treated carefully. Most the time, those estimators make only sense if all inputs are measured in the same units (e.g. prices, volumes, dummies, etc). Note that many (but not all) software packages for the lasso or ridge estimation perform the scaling of $\mathcal{X}_s$ and $\mathcal{Y}_s$ (and the corresponding rescaling) automatically and directly provide $\widehat{\boldsymbol{\beta}}_{\lambda,s}^{\text{lasso}}$ and $\widehat{\boldsymbol{\beta}}_{\lambda,s}^{\text{ridge}}$. Still, we have to be careful and check the implementation when applying a software package. In `R`, we use the software package `glmnet` which performs efficiently the estimation of lasso and ridge type problems. From the mathematical point of view the scaling is necessary only for the regression matrix $\widetilde{\mathcal{X}}_s$ but not for $\widetilde{\mathcal{Y}}_s$. However, it is convenient to scale both objects and it does not effect the theoretical estimation result, but still it usually increases the numerical stability of the algorithms.

Both the lasso and ridge estimators $\widehat{\boldsymbol{\beta}}_{\lambda,s}^{\text{lasso}}$ and $\widehat{\boldsymbol{\beta}}_{\lambda,s}^{\text{ridge}}$ depend on a tuning parameter λ_s. The choice of λ_s will heavily influence the estimation results. If we choose $\lambda_s = 0$ then both estimators turn into the standard OLS estimator. In contrast, for $\lambda_s > 0$ the estimators will differ from the OLS solution and get biased. Still, note that in time series settings even the OLS solution is biased, but the bias increases. The OLS solution is biased as the assumptions of the Gauss-Markov theorem ('OLS is BLUE', best linear unbiased estimator) are not satisfied, due to the inclusion of autoregressive terms. So even if we want to estimate an AR(1) using OLS, the resulting estimator is biased.

In fact, the values get shrunk towards zero. This is why the lasso and the ridge estimators are also known as shrinkage estimators. However, both estimators are different as they shrink the parameters differently towards zero. The lasso estimator has another interesting property which is known as sparsity. This means that for some λ_s values some parameters are exactly 0. So they are effectively dropped from the model. So the lasso estimator can be directly used for model selection. Note that for very large λ_s values the lasso estimator is always zero, as all parameters are exactly zero. In contrast to the lasso, the ridge or the OLS estimator do not have parameter estimates that are exactly 0.

These interpretations can be derived as well by looking at an alternative definition of the lasso and ridge estimator. Those equivalent definitions are given by

$$\widehat{\widetilde{\boldsymbol{\beta}}}_{\tau,s}^{\text{lasso}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^D, \|\boldsymbol{\beta}\|_1 \leq \tau_s}{\arg\min} \|\widetilde{\mathcal{Y}}_s - \widetilde{\mathcal{X}}_s\boldsymbol{\beta}\|_2^2 \tag{153}$$

and

$$\widehat{\widetilde{\boldsymbol{\beta}}}_{\tau,s}^{\text{ridge}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^D, \|\boldsymbol{\beta}\|_2 \leq \tau_s}{\arg\min} \|\widetilde{\mathcal{Y}}_s - \widetilde{\mathcal{X}}_s\boldsymbol{\beta}\|_2^2. \tag{154}$$

Thus, the lasso and ridge estimators can be characterized as an OLS with restricted parameter space which is an L_1- or L_2-ball with radius τ_s. This τ_s is a tuning parameter. An important result is that every τ_s-characterised solution has an λ_s-solution counterpart from (149) and (150). However, the functional relationship between τ_s and λ_s is non-trivial.

If we want to apply the lasso or ridge regression, we have to choose the tuning parameter λ_s, or equivalently the tuning parameter τ_s in (153) and (154). Unfortunately, there is no known optimal way to choose the tuning parameter λ_s[20]. As mentioned, if the parameter space is small or medium sized (e.g. less than a few hundred parameters) we can use LARS to estimate the full solution path. Otherwise we should go for coordinate descent. As the latter works well in all

[19] We ignore additional log terms here.

[20] But there are bounds available, see [LTTT14]

situation in applications often only the latter is used, [Zie16, UW18, BFMV19]. Anyway, there are also notable applications of LARS, see e.g. [ZSH15, LMDSW21].

For coodinate descent, we estimate the lasso or ridge estimator on a given grid of λ_s values, say $\Lambda = \{\lambda_1, \ldots, \lambda_N\}$ with $\lambda_i < \lambda_{i+1}$. Then, the estimators are computed for all $\lambda_s \in \Lambda$ and then following a certain decision criterion, like an information criterion or cross-validation. Usually the grid Λ is chosen to be exponential, so e.g. $\Lambda = \{2^{-a} | a \in \mathbb{A}\}$ where $\mathbb{A}$ is an equidistant grid. Again, many software packages as `glmnet` perform a suitable grid construction by default, but allow for user defined λ_s-grids as well.

To illustrate the lasso and ridge, we consider the advanced expert model **expert.adv**, see (111). In Figures 44 and 45 the estimated parameters for the expert model are visualised for German data. In Figure 44 the dependency to the tuning parameter λ_s is given in a log scale. Figure 45 shows a similar graph illustrating the dependency to the $\|\cdot\|_1$-norm of the estimated vector. This is the more convenient plot for the interpretation of a lasso or ridge estimator solution path. From Figure 44, we can see that for λ_s close to zero the ridge and lasso solutions are similar, as they are close to the OLS solution. Moreover, we see that for very large λ_s values both estimators give essentially a zero vector as parameter estimate. For the lasso and ridge estimators, cross-validation base decision techniques (esp. k-fold cross-validation) are popular methods for the tuning parameter selection.

For the lasso estimator there are other popular alternatives available that rely on the parameter selection property. As some parameters are set to zero, information criterion based evaluation methods can be used.

As mentioned, we estimate the model using lasso. Moreover, we report the corresponding AIC, HQC and BIC solutions within the tuning parameter solution path. In Figure 46, we see the corresponding estimated lasso traces for the considered advanced expert model, by cumulating the parameters depending on their sign. There we see how the impact of all parameters evolves with changing λ_s value (but remember we plot $\|\boldsymbol{\beta}_s\|_1$ on the x-axis). We observe that for all hours never any external regressor seems to be the most important parameter, autoregressive parameters and weekday dummies seem to be dominating. Still, always even the conservative BIC solution contains at least one external regressor.

With many parameters Figure 46 tends to become quite difficult to read and interpret, still usually Figures like 46 are usually easier to interpret than Figure (45). Sometimes in literature the relative impact of a parameter $\beta_{\lambda,s,k}$ with respect to $\boldsymbol{\beta}_{\lambda,s}$ by

$$\mathcal{I}_{\lambda,s,k} = \frac{\beta_{\lambda,s,k}}{\|\boldsymbol{\beta}_{\lambda,s}\|_1}$$

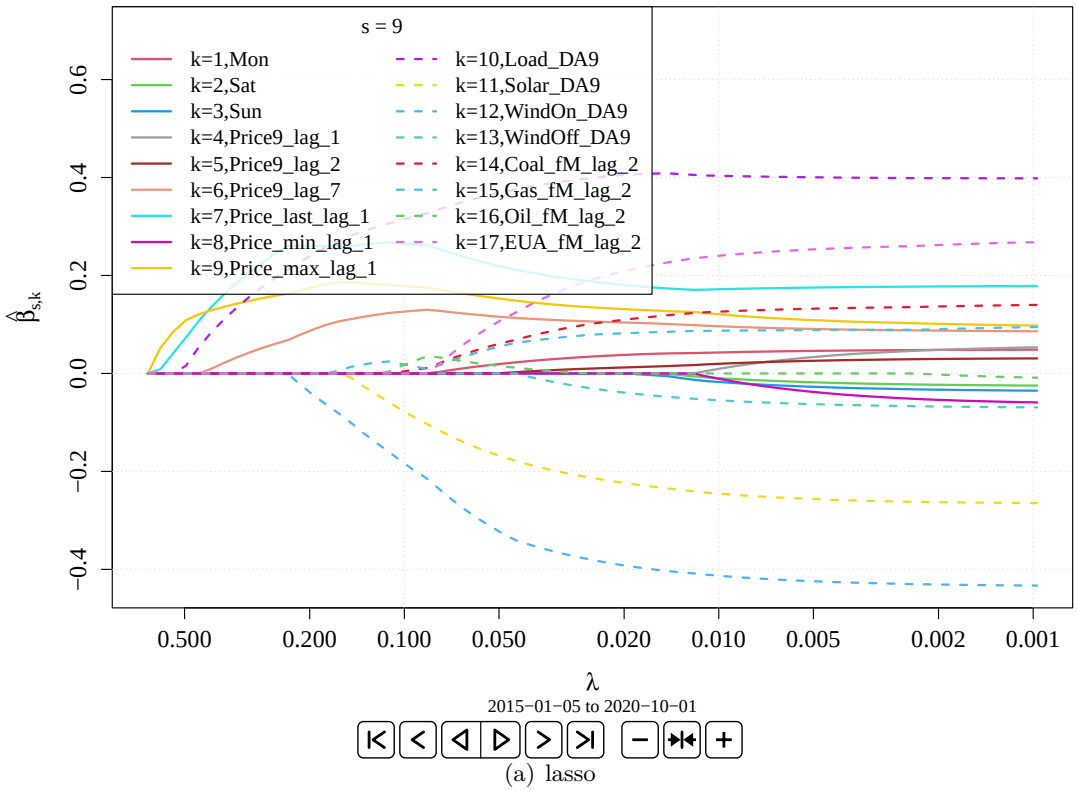
is reported. Positive impacts $\mathcal{I}_{\lambda,s,k}$ are added in a positive direction to the bar chart, and negative $\mathcal{I}_{\lambda,s,k}$ values are added in a negative direction, similarly as in Figure 46.

Finally, there is another method to choose the tuning parameter which is based on past performance. However, here the in-sample data set must be split into two subsequent parts, a so-called estimation/training and calibration part. If we have $1, \ldots, D$ in-sample observations, then we usually allocate $1, 2, \ldots, \lfloor \eta D \rfloor$ to the estimation/training part with $\eta = 0.9$ (other common values are $\eta = .5$, $\eta = .8$ or even $\eta = .99$). The model is estimated on the estimation/training part and evaluated on the remaining calibration part $\lfloor \eta D \rfloor + 1, \lfloor \eta D \rfloor + 2, \ldots, D$. The tuning parameter which minimises the loss in the calibration window is chosen as the optimal one and applied for estimation on the full in-sample data. This procedure shares similarities to the 2-fold cross-validation method, but the fold is not chosen randomly and for $\eta \neq 0.5$ the estimation/training and calibration sets are of different size.

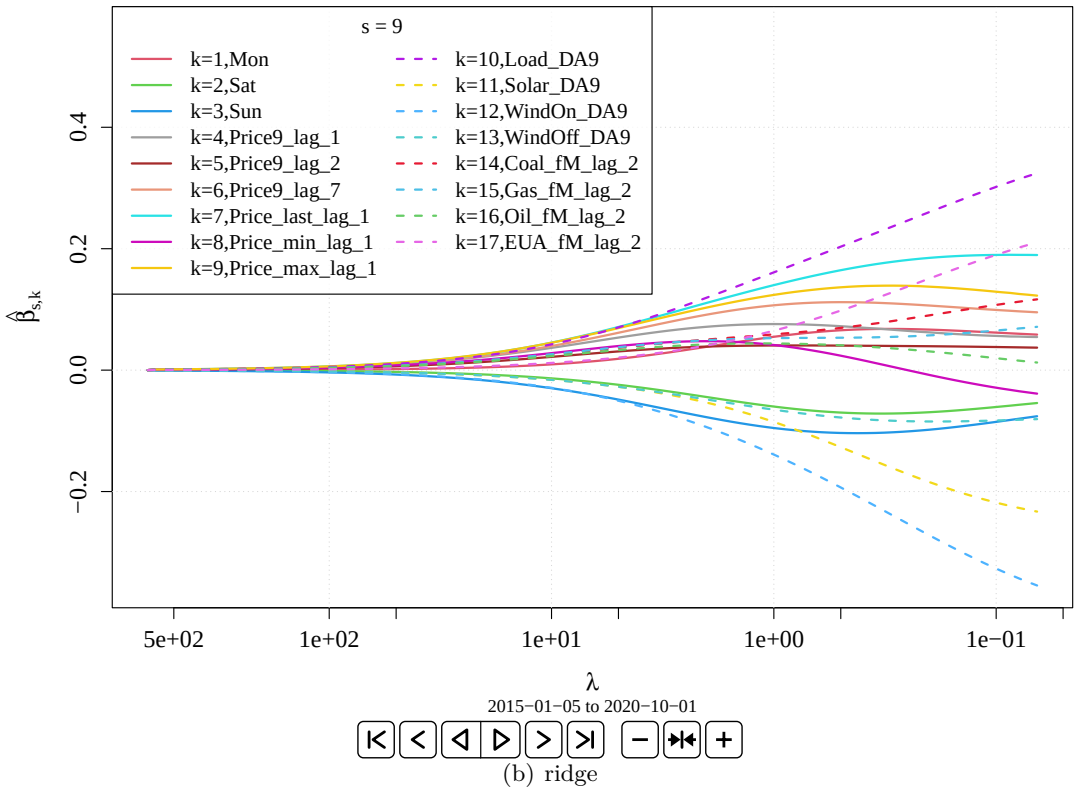
Finally, we want to remark that the mentioned (tuning) parameter selection procedures (cross-validation, information criterion, selection based on past performance) are applied to many models and parameter selection problems, not only in the context of lasso and ridge regression.

10.3 Penalized regression estimators beyond lasso and ridge

We have studied the lasso and the ridge estimators as representative shrinkage estimation methods for linear models. Both methods are special cases of so-called penalized regression methods.



(a) lasso



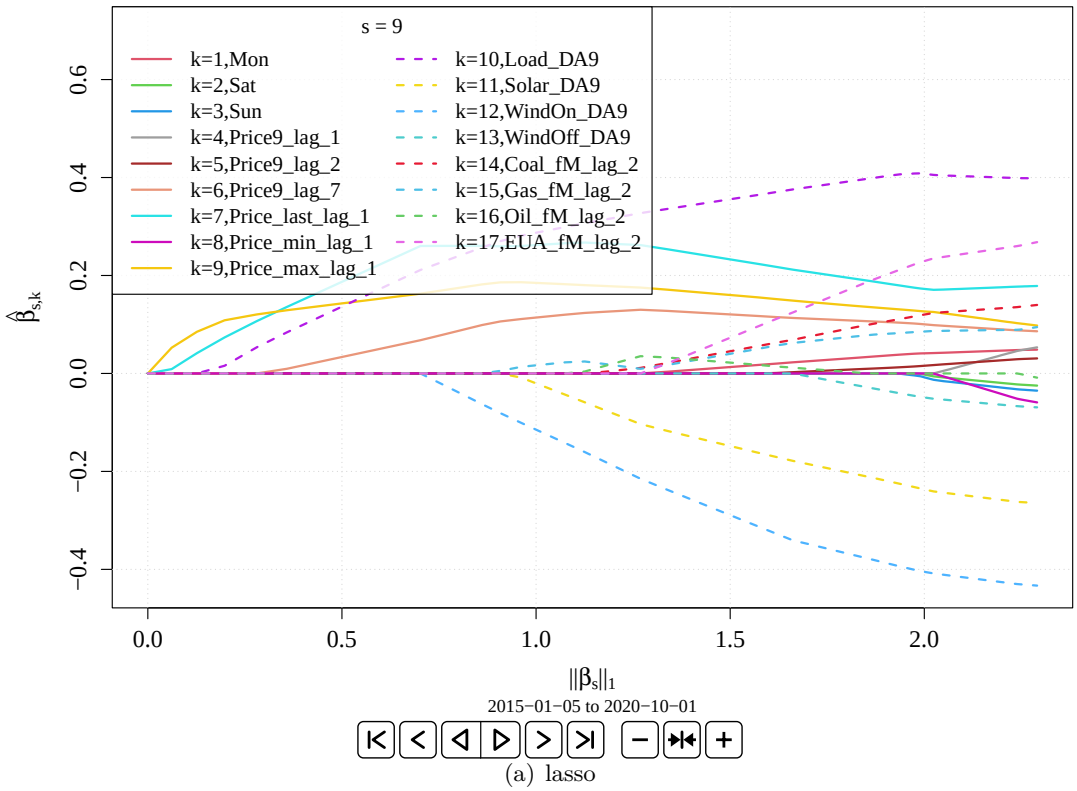
(b) ridge

Figure 44: Estimated scaled parameter vectors $\widehat{\widetilde{\boldsymbol{\beta}}}^{\text{lasso}}_{\lambda,s}$ and $\widehat{\widetilde{\boldsymbol{\beta}}}^{\text{ridge}}_{\lambda,s}$ for different λ_s values and $s = 10$.

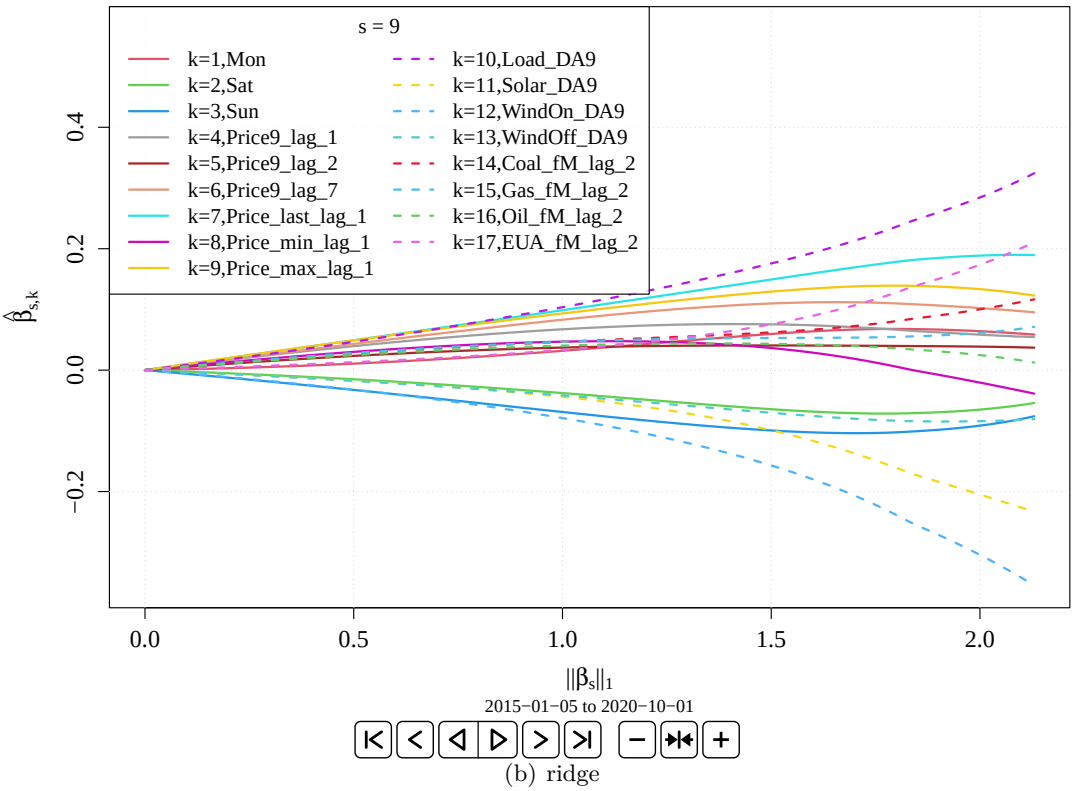
Similarly as the lasso and ridge estimators, they are usually applied to the scaled response $\widetilde{\mathcal{Y}}_s$ and scaled regression matrix $\widetilde{\mathcal{X}}_s$.

$$\widehat{\widetilde{\boldsymbol{\beta}}}^{\text{shrinkage}}_{\mathcal{P}_{\boldsymbol{\lambda}_s},s} = \underset{\boldsymbol{\beta}\in\mathbb{R}^D}{\arg\min} \|\widetilde{\mathcal{Y}}_s - \widetilde{\mathcal{X}}_s\boldsymbol{\beta}\|_2^2 + \mathcal{P}_{\boldsymbol{\lambda}_s}(\boldsymbol{\beta}) \tag{155}$$

where $\mathcal{P}_{\boldsymbol{\lambda}}(\boldsymbol{\beta})$ is a non-negative penalty function that is usually convex and that (potentially) depends on a tuning parameter (vector) $\boldsymbol{\lambda}$. If $\mathcal{P}_{\lambda_s}(\boldsymbol{\beta}) = \lambda_s\|\boldsymbol{\beta}\|_1$, then equation (155) leads to



(a) lasso



(b) ridge

Figure 45: Estimated scaled parameter vectors $\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda,s}^{\text{lasso}}$ and $\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda,s}^{\text{ridge}}$ dependent on $\|\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda,s}^{\text{lasso}}\|_1$ and $\|\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda,s}^{\text{ridge}}\|_1$ for $s = 10$.

the lasso estimator, if $\mathcal{P}_{\lambda_s}(\boldsymbol{\beta}) = \lambda_s \|\boldsymbol{\beta}\|_2^2$, then it gives the ridge estimator.

However, there are other penalized estimation methods that are useful in application as well. A straight forward generalization of the lasso and ridge is the so-called elastic net. It is given by

$$\widehat{\widetilde{\boldsymbol{\beta}}}_{\mathcal{P}_{\boldsymbol{\lambda}_s},s}^{\text{elnet}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^D}{\arg\min} \|\widetilde{\mathcal{Y}}_s - \widetilde{\mathcal{X}}_s\boldsymbol{\beta}\|_2^2 + \lambda_{s,1}\|\boldsymbol{\beta}\|_1 + \lambda_{s,2}\|\boldsymbol{\beta}\|_2^2 \tag{156}$$

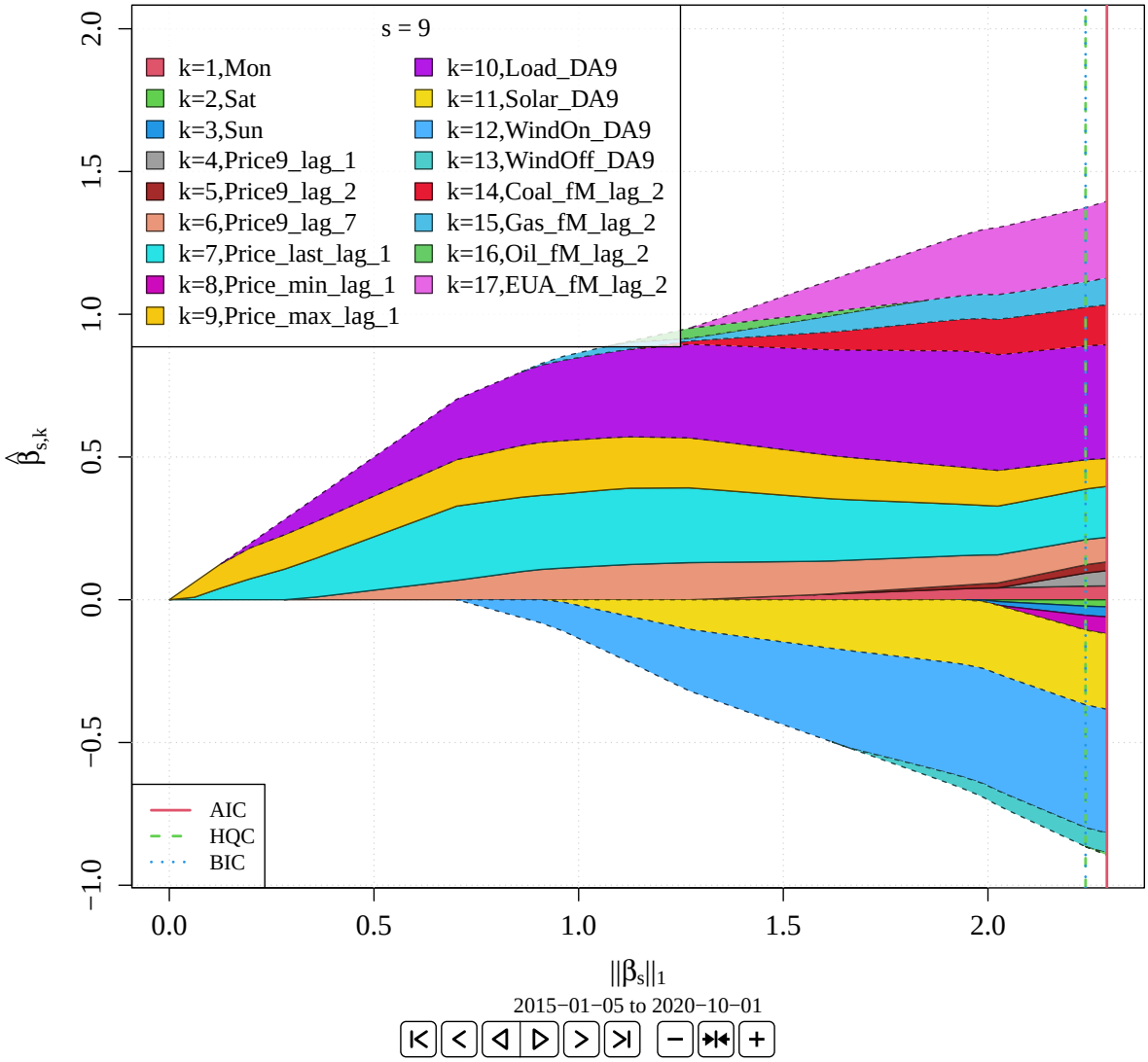


Figure 46: Estimated scaled parameter vectors $\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda,s}^{\text{lasso}}$ dependent on $\|\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda,s}^{\text{lasso}}\|_1$ for $s \in \mathbb{S}$ with selected information criteria.

with tuning parameters $\lambda_{s,1} \geq 0$ and $\lambda_{s,2} \geq 0$. The penalty function $\mathcal{P}_{\boldsymbol{\lambda}_s}(\boldsymbol{\beta}) = \mathcal{P}_{\lambda_{s,1},\lambda_{s,2}}(\boldsymbol{\beta}) = \lambda_{s,1}\|\boldsymbol{\beta}\|_1 + \lambda_{s,2}\|\boldsymbol{\beta}\|_2^2$ of the elastic net (156) has an $\|\cdot\|_1$-norm and an $\|\cdot\|_2$-norm penalty term as the ridge and the lasso. With $(\lambda_{s,1}, \lambda_{s,2}) = (\lambda_s, 0)$ we get the lasso and with $(\lambda_{s,1}, \lambda_{s,2}) = (0, \lambda_s)$ the ridge estimator. In general, the elastic net can also have better statistical properties than the lasso or the ridge. Especially, it holds that every elastic net with a non-zero ridge penalty ($\lambda_{s,2} > 0$) has the sparsity property, so some parameters will be exactly zero. This allows for the selection of the tuning parameter via information criterion. Moreover, the elastic net is often rewritten to

$$\widehat{\widetilde{\boldsymbol{\beta}}}_{\mathcal{P}_{\lambda_s,\alpha},s}^{\text{elnet}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^D}{\arg\min}\, \|\widetilde{\mathcal{Y}}_s - \widetilde{\mathcal{X}}_s\boldsymbol{\beta}\|_2^2 + \lambda_s \left(\alpha\|\boldsymbol{\beta}\|_1 + \frac{1-\alpha}{2}\|\boldsymbol{\beta}\|_2^2\right) \tag{157}$$

Here λ_s can be interpreted similarly as in the lasso or ridge case before. The parameter α puts the weight on the lasso and ridge parts. If $\alpha = 1$, we are in the lasso case and if $\alpha = 0$, we are in the ridge case. Any elastic net can be rewritten as an augmented lasso estimation problem. Therefore, it has the same computational complexity. However, the major drawback is that it requires a tuning of two parameters $\lambda_{s,1}$ and $\lambda_{s,2}$. This tends to increase the computational demand.

We pointed out that the lasso (and elastic net) has nice selection properties and can be efficiently computed. However, one disadvantage is the additional bias due to the penalty term. One problem is that lasso does not satisfies the so called oracle property, [FL01]. In simple words, lasso can not have selection consistency and asymptotic normality with optimal convergence rate at the same time. The reason is the fixed bias in the penalty term in the $\|\cdot\|_1$-norm. We observe

that even if there a regressor in (149) is extremely sure to be very important (and has therefore) clearly non-zero value it will still introduce a bias to the solution due to the impact of $\lambda|\beta_i|$. This problem is even more crucial because in a scaled setting important parameters come often with a large absolute value. In the expert model (7) we have seen that the $Y_{d-1,s}$ has significant explanatory power and takes often values clearly larger than 0.

To avoid the mention problem, we would require a method that is able to identify the very important regressors and reduce the penalty of them to some extent. Indeed, this is feasible. One possible solution is the so called adaptive lasso, discussed in detail in [Zou06]. It is defined by

$$\widehat{\widetilde{\boldsymbol{\beta}}}^{\text{alasso}}_{\lambda,s} = \underset{\boldsymbol{\beta}\in\mathbb{R}^D}{\arg\min} \|\widetilde{\mathcal{Y}}_s - \widetilde{\mathcal{X}}_s\boldsymbol{\beta}\|_2^2 + \sum_{i=1}^{p} \lambda_{i,s}|\beta_{i,s}| \tag{158}$$

with tuning parameters $\lambda_{i,s}$ for $i \in \{1,\ldots,p\}$ and some $s \in \mathbb{S}$. Model (158) is essentially the same model equation as for the lasso (158) but the penalty parameter $\lambda_{i,s}$ for the i'th parameter may vary. Of course choosing $\lambda_{i,s} = \lambda_s$ brings us back to the lasso case. However, tuning every $\lambda_{i,s}$ is usually impractical in applications. Hence, a smart choice for $\lambda_{i,s}$ is required. [Zou06] suggest to use $\lambda_{i,s} = \frac{1}{|\widehat{\beta}_i^{\text{init}}|^\tau}$ for some $\tau > 0$ and initial estimator $\widehat{\beta}_i^{\text{init}}$. The idea is that the initial estimator tracks that a certain variable is important (and therefore clearly non-zero) and reduces penalty contribution. If the initial estimator is a consistent estimator, the resulting adaptive lasso will satisfy the oracle properties. Obviously, now a crucial question is the choice of the initial estimator. Popular choices are usually the OLS, lasso and ridge estimators itself - if they provide suitable results depending on the application.

Moreover, there are other ways to modify the lasso model such that it preserves the nice selection properties but also exhibits the oracle property. Two popular and efficient shrinkage methods that satisty the oracle property are the SCAD (smoothly clipped absolute deviation) and the MC+ (MCplus) (see [Z$^+$10]). The SCAD was proposed by [Fan97, FL01]. The SCAD is defined by

$$\widehat{\widetilde{\boldsymbol{\beta}}}^{\text{SCAD}}_{\mathcal{P}_{\lambda_s,\gamma},s} = \underset{\boldsymbol{\beta}\in\mathbb{R}^D}{\arg\min} \|\widetilde{\mathcal{Y}}_s - \widetilde{\mathcal{X}}_s\boldsymbol{\beta}\|_2^2 + \lambda_s \sum_{i=1}^{p} \mathcal{P}_{\text{SCAD},\lambda,\gamma}(\beta_i) \tag{159}$$

$$\mathcal{P}_{\text{SCAD},\lambda,\gamma}(\beta_i) = \begin{cases} \lambda|x| & \text{if } |x| \le \lambda \\ \frac{2\gamma\lambda|x| - |x|^2 - \lambda^2}{2(\gamma-1)} & \text{if } \lambda < |x| < \gamma\lambda \\ \lambda^2(\gamma+1)/2 & \text{if } |x| \ge \gamma\lambda \end{cases} \tag{160}$$

with $\gamma > 2$. The MC+ is defined by

$$\widehat{\widetilde{\boldsymbol{\beta}}}^{\text{MC+}}_{\mathcal{P}_{\lambda_s,\gamma},s} = \underset{\boldsymbol{\beta}\in\mathbb{R}^D}{\arg\min} \|\widetilde{\mathcal{Y}}_s - \widetilde{\mathcal{X}}_s\boldsymbol{\beta}\|_2^2 + \lambda_s \sum_{i=1}^{p} \mathcal{P}_{\text{MC+},\lambda,\gamma}(\beta_i) \tag{161}$$

$$\mathcal{P}_{\text{MC+},\lambda,\gamma}(\beta_i) = \begin{cases} \lambda|x| - \frac{x^2}{2\gamma} & \text{if } |x| \le \gamma\lambda \\ \lambda^2\gamma/2 & \text{if } |x| \ge \gamma\lambda \end{cases} \tag{162}$$

with $\gamma > 1$. Similarly to the elastic net, the SCAD and MC+ require an additional tuning parameter. Still, in both cases γ can be relatively well interpreted.

The SCAD, behave for parameters of low importance like lasso, for parameter of high importance like OLS and 'in-between' it interpolates the penalty linearly. The tuning parameters define this 'in-between' range. Thus, SCAD is unbiased for parameters that have high impact and are unbiased in an OLS framework. In contrast, the MC+ is biased, but this bias is marginal and exponentially decaying towards zero. It is decaying very fast to zero with increasing importance of the corresponding regressors. Therefore, it is an almost unbiased estimator.

So from the statistical and methodological perspective the SCAD and MC algorithms are likely more appealing than the elastic net or lasso. However, the computational costs are usually about 10 times as high as for the lasso – even for a fixed γ. The reason is that the penalty term is non-convex. This makes the optimization problem serious more complicated. In turn, the numerical optimization algorithms require an additional (convergence) loop which increases the computation time substantially (by a rule of thumb by about 10).

We discussed several alternatives to the lasso that satisfy the oracle property. Still, all considered shrinkage estimators have in common that we have to choose at least one tuning parameter which can be computationally costly. A plausible question is if there are alternatives to lasso which avoid the selection of the tuning parameter on a grid, but rather provide a reasonable estimate for it or modify the design in say a way that no tuning parameter selection is required to perform variable selection. Indeed it is feasible, to construct such a lasso based estimators, see e.g. [LM15].

To understand tuning parameter free lasso based estimators a bit better, we have to study a bit deeper the asymptotics of the plain lasso (149), with particular focus on how the tuning parameter $\lambda_{D,s}$ should be decreased when the sample size D of the data is increasing. As pointed out in [LM15] it is known that the optimal tuning parameter $\lambda_{D,s}$ should growth as

$$\lambda_{D,s} \sim \frac{\sigma_s}{n} \|\widetilde{\mathcal{X}}_s' \widetilde{\boldsymbol{\varepsilon}}_s\|_\infty$$

where σ_s is the standard deviation of the error term $\varepsilon_{d,s}$ and $\|\cdot\|_\infty$ is the supremum norm. Thus, determining the optimal rate for $\lambda_{D,s}$ requires knowledge about σ_s and $\widetilde{\mathcal{X}}_s' \widetilde{\boldsymbol{\varepsilon}}_s$. As both are unknown they would have to be estimated.

The so called square root lasso modifies the lasso such that it eliminated the dependence on σ_s from the optimal rate behavior:

$$\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda,s}^{\sqrt{\text{lasso}}} = \underset{\boldsymbol{\beta} \in \mathbb{R}^D}{\arg\min} \frac{1}{\sqrt{n}} \|\widetilde{\mathcal{Y}}_s - \widetilde{\mathcal{X}}_s \boldsymbol{\beta}\|_2^2 + \lambda_s \|\boldsymbol{\beta}\|_1 \tag{163}$$

Still, this estimator requires the selection of λ.

Even further goes the so called TREX estimator which provides tuning parameter regression given by

$$\widehat{\widetilde{\boldsymbol{\beta}}}_{s}^{\text{TREX}} = \underset{\boldsymbol{\beta} \in \mathbb{R}^D}{\arg\min} \frac{\|\widetilde{\mathcal{Y}}_s - \widetilde{\mathcal{X}}_s \boldsymbol{\beta}\|_2^2}{\frac{1}{2}\|\widetilde{\mathcal{X}}_s'(\widetilde{\mathcal{Y}}_s - \widetilde{\mathcal{X}}_s \boldsymbol{\beta})\|_\infty} + \|\boldsymbol{\beta}\|_1 \tag{164}$$

while noting that $\sigma\|\widetilde{\mathcal{X}}_s'(\widetilde{\mathcal{Y}}_s - \widetilde{\mathcal{X}}_s\boldsymbol{\beta})\|_\infty$ is a consistent estimator of $\sigma\|\widetilde{\mathcal{X}}_s'\boldsymbol{\varepsilon}_s\|_\infty$. However, in practice a tuned TREX estimator

$$\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda,s}^{\text{TREX}} = \underset{\boldsymbol{\beta} \in \mathbb{R}^D}{\arg\min} \frac{\|\widetilde{\mathcal{Y}}_s - \widetilde{\mathcal{X}}_s \boldsymbol{\beta}\|_2^2}{\frac{1}{2}\|\widetilde{\mathcal{X}}_s'(\widetilde{\mathcal{Y}}_s - \widetilde{\mathcal{X}}_s \boldsymbol{\beta})\|_\infty} + \lambda\|\boldsymbol{\beta}\|_1 \tag{165}$$

can lead to better out of sample results, despite the theoretical results on the TREX.

Finally, there exist many other directions of generalization of the lasso, e.g. the Danzig selector (general $\|\cdot\|_p$ penalty) or group-lasso where a group of parameters is penalised jointly in the sense that all parameters in the group are jointly shrinked towards zero.

10.4 More on the multivariate penalized models

For the estimation equation of penalized regression problems defined by (155) (e.g. the lasso and ridge) we consider only estimation techniques where the tuning parameters in $\mathcal{P}_{\boldsymbol{\lambda}_s}$ are selected for each of the $s \in \mathbb{S}$ model equation separately.

However, we have studied that there are multivariate estimation techniques for least squares. First we remember that for the standard least squares estimator as defined in (13) coincides with the multivariate estimator (30) that considers all jointly on a diagonal. For penalized regression estimators this equivalence usually does not holds true. The reason is that we are in general not selecting $\mathcal{P}_{\boldsymbol{\lambda}_s}$ for each $s \in \mathbb{S}$ but we apply the same shrinkage parameter $\boldsymbol{\lambda}$ for each of the s equations. The corresponding general model equation is then

$$\widehat{\widetilde{\boldsymbol{\beta}}}_{\mathcal{P}_{\boldsymbol{\lambda}}}^{\text{shrink-mv}} = \underset{\boldsymbol{\beta} \in \mathbb{R}^p}{\arg\min} \|\widetilde{\boldsymbol{\mathcal{Y}}} - \widetilde{\boldsymbol{\mathcal{X}}}\boldsymbol{\beta}\|_2^2 + \mathcal{P}_{\boldsymbol{\lambda}}(\boldsymbol{\beta}) \tag{166}$$

with $\widetilde{\boldsymbol{\mathcal{X}}}$ and $\widetilde{\boldsymbol{\mathcal{Y}}}$ as scaled $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$ as defined in (25).

It can be estimated by using the same solution algorithms as before, but using it a multivariate way on the regression matrix $\boldsymbol{\mathcal{X}}^{\boldsymbol{O}}$ from equation(29). As we have no explicit solution

for most models, we illustrate the differences directly for most methods. However, for the ridge estimator we have an explicit solution available. With $\widetilde{\boldsymbol{\mathcal{X}}}^{\boldsymbol{O}}$ as scaled version of $\boldsymbol{\mathcal{X}}^{\boldsymbol{O}}$ we get

$$\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda}^{\text{ridge}} = \left((\widetilde{\boldsymbol{\mathcal{X}}}^{\boldsymbol{O}})'\widetilde{\boldsymbol{\mathcal{X}}}^{\boldsymbol{O}} + \lambda \boldsymbol{I}\right)^{-1} \widetilde{\boldsymbol{\mathcal{X}}}^{\boldsymbol{O}}\widetilde{\boldsymbol{\mathcal{Y}}} \tag{167}$$

Note that in academic literature it is barely distinguished between the two version.

Further, we may apply it on (34) with $\breve{\mathcal{X}}_s = \mathcal{X}_s$ for a specific model specification, e.g. the advanced expert model (111). Then the model has $(1+S)\times(p+1)$ with $p = 17$ parameters and is over-specified. Thus, lass may decide to add a parameter β_k which is the same for all s or parameters $\beta_{k,s}$ which may vary across s. The resulting (unscaled) estimate for the model is given in Figure (47).

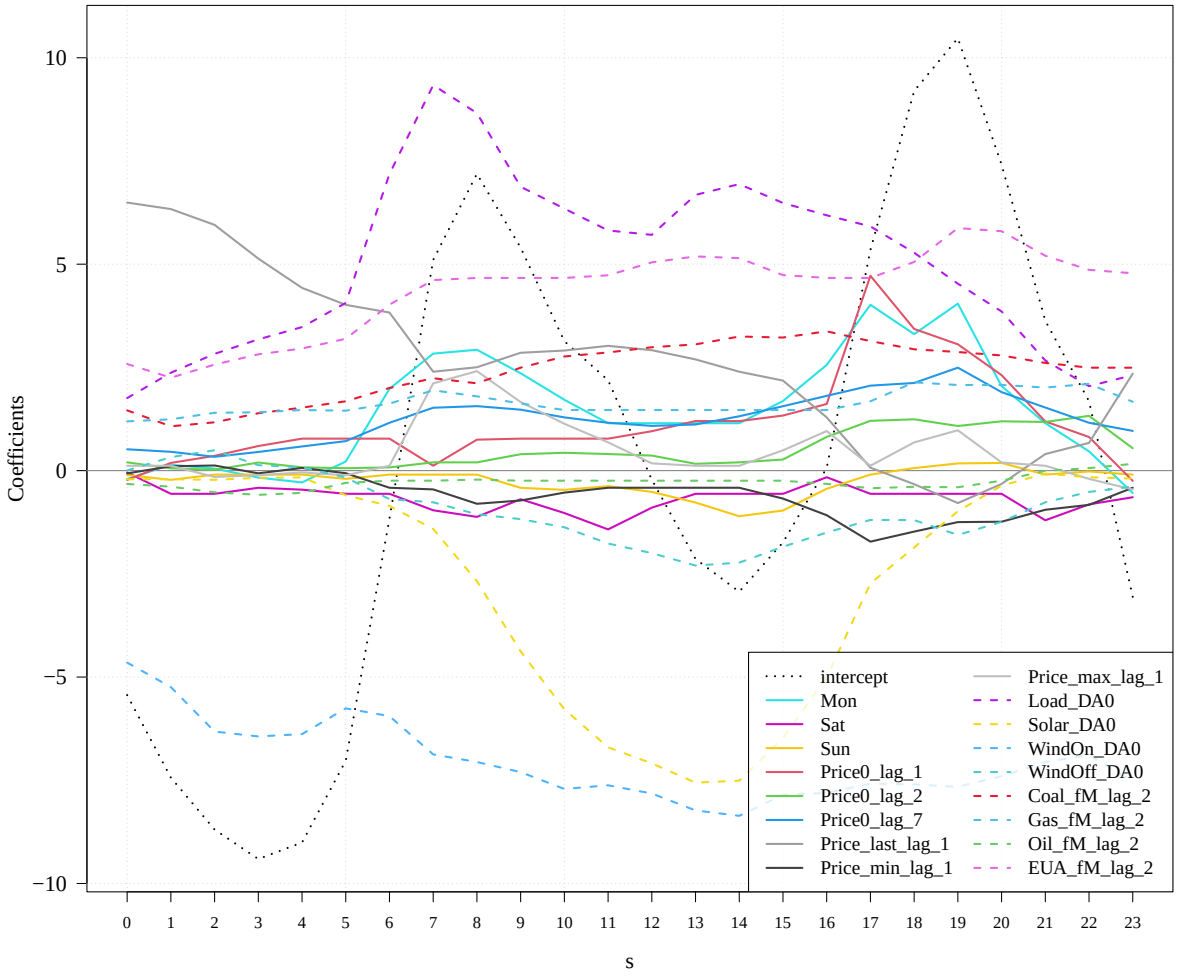


Figure 47: Estimated parameters by lasso for (34) on model (111).

Moreover, we remember that we also introduced the multivariate regression estimator (28) which was essentially equivalent to the OLS solution. Now, we introduce the multiresponse elastic net estimator

$$\widehat{\widetilde{\boldsymbol{\beta}}}_{\lambda,\alpha}^{\text{mls}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^{p\times S}}{\arg\min} \sum_{d=1}^{D} \|\widetilde{\boldsymbol{Y}}_d - \widetilde{\boldsymbol{X}}_d'\boldsymbol{\beta}\|_F + \lambda\left(\frac{1-\alpha}{2}\|\boldsymbol{\beta}\|_F^2 + \alpha\sum_{j=1}^{p}\|\boldsymbol{\beta}_j\|_2\right). \tag{168}$$

which is in general different from the multivariate elastic net.

Summarizing, we observe that in the OLS setting we had three different options to receive equivalent estimators, i) based on S univariate regressions $\widehat{\boldsymbol{\beta}}_0^{\text{ls}}, \ldots, \widehat{\boldsymbol{\beta}}_{S-1}^{\text{ls}}$, ii) based on multivariate least squares $\widehat{\boldsymbol{\beta}}^{\text{mls}}$ and iii) based on a sparse and large univariate regression $\widehat{\boldsymbol{\beta}}^{\mathcal{O}}$. The interesting part is that if we leave the world of OLS and go towards standard shrinkage approaches.

10.5 Tasks

Hints and the corresponding programming part are provide in Section **??** in `R` and in Section py.7 in `python`.

1. Apply SVD/PCA on the $4S$-dimensional regressor matrix containing $X^{\text{ext}}_{d,0}, \ldots, X^{\text{ext}}_{d,S-1}$ for $\text{ext} \in \{\text{Load}, \text{Solar}, \text{WindOn}, \text{WindOff}\}$ (without scaling it) to model the electricity prices $Y_{d,s}$ for selected $s \in \mathbb{S}$. Do the same for task by considering four separate PCA's, for each external regressor, and choose the corresponding λ-tuning values based on a $\mathrm{d}_{\min}$ threshold. Compare both approaches while choosing the model based on the BIC.

2. Extend the expert model (7) with 365 day of the year dummies and estimate the model using the lasso technique. Choose the tuning parameter λ based on the BIC. Which of the dummies impact the model? Specify your regression matrix as a sparse matrix. Do you observe any improvement in the performance?

3. Implement a lasso forecasting function for model (111). Choose the tuning parameter λ based on the BIC.

4.

5. Consider the non-linear model (116) with linear splines resp. rectifiers and quantile based knots. Choose as regressor the residual load and estimate the model using lasso. Can you avoid unstable estimation behaviour for large K (large number of linear spline knots)?

6. Reconstruct the plots of Figures 44 and 45 for the ridge estimator.

11 Generalized additive models

In the section on non-linear effects, we briefly introduced Generalized additive models (GAMs) as a flexible model that allows automatic detection of non-linear effects. Often GAMs are considered in combination with shrinkage approaches to avoid overfitting problems as discussed in subsection 9.4 in the non-linear effects section 9. They are popular in energy forecasting, esp. in load forecasting literature, but they are useful in electricity price forecasting as well, [Ser11, GGN16, NZ20b].

11.1 Basics on GAMs and univariate smoothers

A GAM is an additive model that can have a very general structure. For inputs $X_{t,1}, \dots, X_{t,p}$ we may define it by

$$Y_t = \sum_{i=1}^{L} f_i(X_{t,1}, \dots, X_{t,N}) + \varepsilon_t \tag{169}$$

for prediction target Y_t. The additive terms f_i may represent arbitrary model components. Traditionally, linear and various smoothing spline terms are used for f_i. However, in recent years complicated non-linear model components like gradient boosting machines or (deep) neural networks are used as well. Here, we will focus on the traditional view, mainly because we will cover non-linear models in the electricity price forecasting context only in Section 14.

If f_i is chosen as a linear function (i.e. $f_1(X_{t,1}, \dots, X_{t,N}) = \beta_0 + \beta_1 X_{t,1} + \ldots + \beta_N X_{t,N}$), we are back to the standard linear model framework. In practice, the key feature of GAMs is the flexible usage of splines. So f_i may be a one-dimensional spline in the j-th regressor, i.e. $f_i(X_{t,j})$. This spline can be represented as a weighted sum of basis functions as covered in Subsection 9.4. Formally this is

$$f_i(X_{t,j}) = \sum_{l=1}^{L} \phi_l \xi_l(X_{t,j}) \tag{170}$$

for coefficients ϕ_l and basis functions ξ_l which are defined on the support of $X_{t,j}$. The linear spline as introduced in section (116) can be seen as special case where $\xi_l(x) = \max(x, c_l)$ with a grid point c_l. A popular choice for basis functions are cubic b-splines, as they are local basis function and simple to compute.

Next to the problem of a potentially large parameter space, there is also the problem of overfitting. This holds especially in the number of parameters in the model is large. The standard approach for the application of smoothing splines is to consider shrinkage, because shrinkage and the way of implementing it influences the smoothing spline behaviour substantially, the kind of smoothing spline is mentioned together with the smoothing approach.

Remember that our GAM model (169) is usually estimated by least squares, this is the solution of

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}=(\boldsymbol{\theta}_1,\dots,\boldsymbol{\theta}_L)}{\arg\min} \sum_{t=1}^{T} \left(Y_t - \sum_{i=1}^{L} f_i(X_{t,1}, \dots, X_{t,N}; \boldsymbol{\theta}_i) \right)^2 . \tag{171}$$

As the f_i's are essentially just linear combinations of basis function, the full problem is just a large linear model. Thus, the solution can be computed easily using ordinary least squares (13).

The estimation procedure in (171) can be adjusted easily by introducing a roughness penalty that performs shrinkage towards a smooth (in the extreme case linear) function. Usually this penalty term penalizes the roughness of the function f_i by evaluating its m'th derivative, usually the second one (i.e. $m = 2$). If, all f_i are univariate splines, then the adjustment of (171) is relatively simple, given by

$$\widehat{\boldsymbol{\theta}}_{\boldsymbol{\lambda}} = \underset{\boldsymbol{\theta}=(\boldsymbol{\theta}_1,\dots,\boldsymbol{\theta}_L)}{\arg\min} \sum_{t=1}^{T} \left(Y_t - \sum_{i=1}^{L} f_i(X_{t,i}; \boldsymbol{\theta}_i) \right)^2 + \sum_{i=1}^{L} \lambda_i \int_{\mathbb{R}} f_i^{(m)}(x_i; \boldsymbol{\theta}_i)^2 \, dx_i. \tag{172}$$

where $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_L)$ are smoothing penalty parameters and $f_i^{(m)}$ is the m-th derivative of f_i. It is easy to see that for $\boldsymbol{\lambda} = \mathbf{0}$, no penalty applies and we are back in the least squares

situation (171). If $\lambda_i \to \infty$ we receive a shrinkage with respect to the null space which is a $(m-1)$-dimensional polynomial. For the important $m = 2$ case this corresponds to a shrinkage towards a linear function, thus a linear effect. Note that this behaviour is not only relevant if $\lambda_i \to \infty$ because it determines also the limiting behaviour for the effects at and beyond the boundary of the support. For example, consider an electricity price model that depends only on the residual demand. Assume we have observed so far only values between 5GW and 50GW and estimated the GAM model using a penalized spline. Now, we have a residual load of 1GW, then the predicted effect is essentially an $(m-1)$-polynomial extrapolation of the spline fit close to 5GW. In the $m = 2$ case this is a linear extrapolation. Thus, we see that using higher orders for m might be risky. Thus, if we want that out of the support the spline effect is less distinct, we should consider $m = 1$ as it shrinkages towards a constant function. In the example this would mean that we expect about the same price effect for a 1GW residual load and the 5GW residuals load.

The mentioned penalized least squares optimization problems (172) can be solved efficiently. To understand this, we first mention again that the unpenalized model (171) is a high-dimensional linear model which can be estimated efficiently. The penalized version can be estimated efficiently as well, because the penalty term is based on the squared $\|\cdot\|_2$-norm. Thus, it is effectively a ridge type penalty. We remember that ridge regression can be solved efficiently and even provide an explicit solution. To see this we need a penalty matrix

$$\boldsymbol{S}_{m,\boldsymbol{\lambda}} = \sum_{i=1}^{L} \lambda_i \boldsymbol{S}_{m,i} \tag{173}$$

which measures the roughness of the curve by satisfying

$$\int_{\mathbb{R}} f_i^{(m)}(x_i; \boldsymbol{\theta}_i)^2 \, dx_i = \boldsymbol{\theta}_i' \boldsymbol{S}_{m,i} \boldsymbol{\theta}_i.$$

The corresponding ridge problem is

$$\widehat{\boldsymbol{\theta}}_{\boldsymbol{\lambda}} = \underset{\boldsymbol{\theta}=(\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_L)}{\arg\min} \sum_{t=1}^{T} \left(Y_t - \mathbb{X}\boldsymbol{\theta}\right)^2 + \boldsymbol{\theta}' \boldsymbol{S}_{m,\boldsymbol{\lambda}} \boldsymbol{\theta}. \tag{174}$$

with $\mathbb{X}$ as regression matrix of the spline basis.

As mentioned there are various popular spline types that are used, we briefly introduce relevant ones in the context of electricity price modelling and forecasting. We will consider three core types of splines, that can be augmented with smoothing penalty, (173). We know already two particular examples, B-splines, and cyclic/periodic B-splines.

- B-splines, or so called natural basis splines. A standard way to implement splines. It can be numerically efficiently implemented, as they are local basis functions. Natural basis splines are build on a polynomial basis of a certain degree or order (in this context it always holds order= degree+1). Most popular are cubic B-splines with degree of 3. Natural basis splines are constructed based on a set of knots, usually this is chosen equidistant across the data space.

- cyclic/periodic B-splines. They are a special type, of B-splines that are defined on a periodic support, as illustrated in Figure (28). Of course, this is suitable for periodic data, e.g. to model annual seasonalities.

- Thin plate smoothing splines (TPSS) this is a very smart but also complex way of providing a spline basis. Here, we construct for every data point a basis function. To avoid computational problems, some dimension reduction or rank approximation is considered in the estimation approach. See [Woo03] for more information.

It is important to note that thin plate smoothing splines have usually higher computational demand than cubic splines. However, TPSS come with some theoretical background on the optimality of choice of the basis. Therefore, it is a plausible default choice of the basis. However, in practice the differences between TPSS and B-splines are often marginal. Therefore, computational costs give the advantage to the B-splines.

The above listed splines basis which characterize ξ in (170) can be augmented by smoothing penalties. Here, there are two popular alternatives: The approach introduces by (174) is usually referred as a penalized spline shrinkes to the null space an $(m-1)$-dimensional polynomial. If B-splines are used with (174) we usually refer this a P-spline. If we consider this approach for cyclic/periodic P-splines, it is usually referred as (173). However, as discussed above the shinkage approach can only shrink towards the null space. Sometime, it would be preferable to shrink to 0 (precisely the zero-function). To achieve this we add an extra shrinkage parameter λ_0 to the penalty matrix $\boldsymbol{S}_{m,\boldsymbol{\lambda}}$:

$$\boldsymbol{S}_{0,m,\boldsymbol{\lambda}} = \boldsymbol{S}_{m,\boldsymbol{\lambda}} + \lambda_0 \boldsymbol{I} \tag{175}$$

and apply the same procedure as before. If λ_0 goes to ∞ we receive a shrinkage towards 0.

In GAMs however, we have the additional problem that the potentially many tuning parameters λ_i have to be determined. With many smoothing terms it would be computationally infeasible to provide a suitable tuning for each λ_i separately, in the direct approach. Luckily, there is an efficient way to tackle this problem based on iteratively reweigthed least squares (IRLS).

Without going into detail we want to mention that during the iterations a so called deviance (score) is a key part in the optimization. This deviance score can be expression as $\|\widehat{\boldsymbol{\epsilon}}\|_2^2$ in an IRLS iteration where $\widehat{\boldsymbol{\epsilon}}$ is a vector of scaled pseudo-errors that appears during IRLS. The deviance can be now use in two but very related optimization schemas, they are based on the generalized cross validation (GCV) score or the unbiased risk estimator (UBRE) score which are useful in model selection, and provide an efficient way to receive the optimal λ_i values. The GCV and UBRE scores in an IRLS iteration is defined as

$$\mathrm{GCV}_\gamma = n \frac{\|\widehat{\boldsymbol{\epsilon}}\|_2^2}{\left(n - \gamma \operatorname{tr}(\boldsymbol{H})\right)^2} \tag{176}$$

$$\mathrm{UBRE}_\gamma = \frac{1}{n}\|\widehat{\boldsymbol{\epsilon}}\|_2^2 + \sigma^2 + \gamma \frac{2}{n} \operatorname{tr}(\boldsymbol{H})\, \sigma^2 \tag{177}$$

where $\widehat{\boldsymbol{\epsilon}}$ is a vector of scaled pseudo-errors during the IRLS iterations, $\boldsymbol{H}$ is the hat-matrix of the IRLS iteration[21], and tr is the trace operator. Further in (176) and (177) there is a smoothness tuning parameter $\gamma > 0$ and σ^2 in (177) which is the variance of the error term. The latter is usually estimated by $\widehat{\sigma}^2 = \frac{\mathrm{RSS}}{n - \mathrm{tr}(\boldsymbol{H})}$.

A particularly important role has the effective degree of freedom of the

$$\mathrm{edof} = \operatorname{tr}(\boldsymbol{H}). \tag{178}$$

The effective degree of freedom is a suitable measure for the 'effective' use of parameters. Very wiggly and rough curves have a higher effective degree of freedom than very smooth curves. Note that $\boldsymbol{H}$ is a matrix with some structure. Each smoothing term i is associated with certain parameters and a corresponding block $\boldsymbol{H}_i$ in the matrix H. As only the trace of H is relevant, it allows us to decompose the effective degree of freedom to

$$\mathrm{edof} = \sum_{i=1}^{L} \mathrm{edof}_i = \sum_{i=1}^{L} \operatorname{tr}(\boldsymbol{H}_i). \tag{179}$$

Thus, we can interpret the parameter usage for every additive term individually. To get a better understanding: Suppose for instance that the smoothed curve of f_i is almost linear (e.g. because the corresponding λ_i is very large) - then the edof will be just above 1 - which is plausible as a linear effect contributes with 1 parameter to the overall equation.

γ is often chosen as $\gamma = 1$ because it corresponds to the (asymptotic) AIC. This gets slightly clearer when remembering that leave-one-out crossvalidation is asymptotically equivalent to AIC optimization. In (177) γ can be interpret as $\kappa/2$ in the GIC equation (38).

Additionally, we want to mention that there is next to GCV and UBRE score minimization (176) and (177) also restricted maximum likelihood estimation (REML) is popular.

Concerning implementation issues `R` provides very efficient implementations in the `mgcv` package. The function `gam` is the standard estimation function. However, there is also the

[21] In standard linear regression setting the hat matrix is $\boldsymbol{X}'(\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}$, in simple ridge it is $\boldsymbol{X}'(\boldsymbol{X}'\boldsymbol{X}+\lambda\boldsymbol{I})^{-1}\boldsymbol{X}$.

function `bam` (big generalized additive model) which has reduced complexity but provides a more efficient estimation algorithm for large data sets. Unfortunately, `python` does not have a GAM package with similar degree of flexibility, complexity and efficiency, but `statsmodels.gam` provides basic smoothers.

11.2 Multivariate smoothing terms in GAMs

We have briefly studied GAMs for univariate smoothing terms. Next to univariate splines (170), also multivariate one with rather small depth are popular in practice, especially bivariate splines. Also in electricity price forecasting they can be very useful.

A bivariate spline can be expressed by

$$f_i(X_{t,j_1}, X_{t,j_2}) = \sum_{l=1}^{L} \phi_l \xi_l(X_{t,j_1}, X_{t,j_2}) \tag{180}$$

in essentially the same way as univariate splines with L-dimensional parameter vector $(\phi_1, \ldots, \phi_L)'$. However, the support of f_i is 2-dimensional and covers an area. Thus, we need usually much more basis functions L to have a suitable description of f_i. A popular way to describe the bivariate basis function ξ_l in (180) is to assume a tensor structure. In the bivariate case, the spline function is a product of two univariate ones:

$$\xi_l(x_1, x_2) = \xi_{1,l}(x_1)\xi_{2,l}(x_2) \tag{181}$$

In addition, $\xi_{1,l}$ and $\xi_{2,l}$ are usually chosen in such a way that ξ_{1,l_1} interacts (=is multiplied) with each of the considered basis function ξ_{2,l_2}. This, allows to renumerated the problem such that $l = (l_1, l_2)$, and yields

$$\xi_{l_1,l_2}(x_1, x_2) = \xi_{1,l_1}(x_1)\xi_{2,l_2}(x_2). \tag{182}$$

In turn, the bivariate interaction term (180) can be expressed by

$$f_i(X_{t,j_1}, X_{t,j_2}) = \sum_{l_1=1}^{L_1} \sum_{l_2=1}^{L_2} \phi_{l_1,l_2} \xi_{1,l_1}(X_{t,j_1})\xi_{2,l_2}(X_{t,j_2}) \tag{183}$$

with $L_1 \times L_2$-dimensional parameter matrix $(\phi_{l_1,l_2})'_{l_1=1,\ldots,L_1;l_2=1,\ldots,L_2)}$. Thus, if choosing $L_1 = L_2$ the bivariate splines requires the quadratic amount of parameters than a univariate version (e.g. $10 \times 10 = 100$ parameters compared to $2 \times 10 = 20$ parameters for only univariate terms). Thus, including interactions is usually computationally costly. Therefore, it is recommended to impose interactions only where we really expect a beneficial outcome.

Obviously, tensors for bivariate splines can be easily generalized to arbitrary dimensions. However, due to large parameter space interactions depth beyond 3-way interactions are rarely considered in practice. This gets clear, if we recall that the basis would have $L_1 \times L_2 \times L_3$ basis functions (e.g. $10 \times 10 \times 10 = 1000$ parameters compared to $3 \times 10 = 30$ parameters for only univariate terms).

A useful property of tensor products is that it can be decomposed. In the bivariate case this is

$$\text{te}(x_1, x_2) = \text{ti}(x_1) + \text{ti}(x_2) + \text{ti}(x_1, x_2) \tag{184}$$

where te is tensor product of x_1 and x_2 based on certain basis functions (the ξ's) and ti is a tensor interaction. Note that in the 1-dimensional case it holds te $= ti$. To get an idea about the tensor interaction in equation (184) consider the simple linear regression example:

$$Y_t = \beta_0 + \beta_1 X_{1,t} + \beta_2 X_{2,t} + \beta_3 X_{1,t} X_{2,t} + \varepsilon_t \tag{185}$$

Even though the effects in this model are linear, we could fit a GAM model. If we use a (standard) tensor based specification with $\text{te}(x_1, x_2)$ we can estimate the overall joint effect explained by β_1, β_2 and β_3 in (185). If we add shrinkage we would jointly shrink the three effects towards the null space or zero. If we specify a model by tensor interactions as in (185) with smoothing terms $\text{ti}(x_1)$, $\text{ti}(x_2)$ and $\text{ti}(x_1, x_2)$ they would estimate the corresponding effect for β_1, β_2 and β_3. Without any shrinkage the estimation results of both approaches (te vs ti) are the same,

up to numerical issues. However, with shrinkage in place we get clear differences, as the tensor interaction variant shrinks all three terms separately towards the null space or zero. If the true model would have the structure

$$Y_t = \beta_0 + \beta_1 X_{1,t} + \beta_3 X_{1,t} X_{2,t} + \varepsilon_t \tag{186}$$

then the tensor interaction approach can shrink the effect of β_2 towards 0 while keeping the other ones in place. Indeed, there is effectively no clear disadvantage from using the tensor interaction decomposition in contrast to the standard tensor product.

An additional important property of tensor based splines is that the resulting splines are scale invariant. So if we rescale our covariates we will get the same fit. However, tensor based splines not rotation invariant. Thus, if we consider $\text{te}(x_1, x_2)$ and we rotate x_1 and x_2 in the 2-dimensional space we will receive different fits. In electricity price forecasting rotation invariance is often not a relevant property. This is usually relevant in spatial data analytics. For instance, if x_1 and x_2 are the wind direction from north and east direction, then it makes sense to consider a model that provides the same results as if we would provide with x_1 and x_2 are the wind direction from north-east and south.east direction. Still, if we want splines that are rotation invariant we can consider isotropic splines for the specification in (180). However, those splines are not scale invariant.

Another non-trivial aspect is how a shrinkage can be generalized for multivariate smoothing terms adequately. A generalization of (174) to bivariate and even higher interaction depths is possible, but a bit tricky as the roughness of the spline has to be well defined. Therefore, we discuss here on the case for the bivariate interactions, for the general case see e.g. [Woo03].

First, we notice that the penalty term $\lambda_i \int_{\mathbb{R}} f_i^{(m)}(x_i; \boldsymbol{\theta}_i)^2 \, dx$ is an L^2-distance of the considered derivative of $f_i^{(m)} = \frac{\partial^m}{\partial^m x} f_i$. Further it holds $f_i^{(m)}(x_i; \boldsymbol{\theta}_i)^2 = \|f_i^{(m)}(x_i; \boldsymbol{\theta}_i)\|_2^2$. For $m = 2$ we have in the bivariate interaction case that that

$$f_i(x_1, x_2)^{(2)} = \frac{\partial^2}{\partial^2 x_1} f_i(x_1, x_2) + 2\frac{\partial^2}{\partial x_1 \partial x_2} f_i(x_1, x_2) + \frac{\partial^2}{\partial^2 x_2} f_i(x_1, x_2).$$

So $f_i(x_1, x_2)^{(2)}$ the roughness of a bivariate f_i will be expressed by partial derivates in direction of only x_1 and x_2 and the combination of both. Formally, we can define the corresponding optimization problem by

$$\widehat{\boldsymbol{\theta}}_\lambda = \underset{\boldsymbol{\theta}=(\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_L)}{\arg\min} \sum_{t=1}^{T} \left(Y_t - \sum_{i=1}^{L} f_i(X_{t,j_{i,1}}, X_{t,j_{i,2}}; \boldsymbol{\theta}_i) \right)^2 + \sum_{i=1}^{L} \lambda_i \int_{\mathbb{R}^2} \left\| f_i^{(2)}(x_{j_{i,1}}, x_{j_{i,2}}; \boldsymbol{\theta}_i) \right\|_2^2 d\boldsymbol{x}. \tag{187}$$

The optimization and tuning parameter selection for (187) and even higher dimensional smoothing terms can be solved efficiently in the same way and interpretation as the univariate case. Especially, we can use the GCV and UBRE scores to tune our models.

11.3 GAMs in electricity price forecasting

We have covered the relevant theory on GAMs to dive into to world of applications for electricity price forecasting. There are two main field of applications where it is useful to consider GAMs. First, it helps to fit relevant non-linear effects, including interactions. Second, we can use GAMs to explore the model world between models where we assume that all parameters are constant (see e.g. (24))across $\mathbb{S}$ and those where they are vary for each $s \in \mathbb{S}$ (e.g. (7)). We will start looking at this in more detail.

Therefore, we remember that introduced models somewhat in between where some parameters where constant and some vary across $\mathbb{S}$ (e.g. model (32)), but here each parameter $\beta_{s,i}$ takes either fixed or takes S different values. However, using the overlapping mixed design framework in (34) already allowed us to explore already the space in between in combination with shrinkage estimators as lasso or ridge. Figure 168 shows the fit of such a model. There, we see that some parameters have fairly smooth transitions across S and others are still relatively wiggly. GAMs support in an easy way to perform shrinkage in a smoother manner. Note, that this does not automatically mean that smooth functions are better - but if so GAMs are more suitable to track such a behaviour than lasso/ridge based overlapping mixed design models (34).

For illustration purpose we consider a simple electricity price forecasting model with only residual load, natural gas and emission prices as input, as considered in equation (99). A corresponding linear model would be

$$Y_{d,s} = \beta_{s,0} + \beta_{s,1} X_{d,s}^{\text{ResLoad}} + \beta_{s,2} X_{d,s}^{\text{NGas}} + \beta_{s,3} X_{d,s}^{\text{EUA}} + \varepsilon_{d,s} \tag{188}$$

for each $s \in \mathbb{S}$ with $4 \times S$ parameters. This is $4 \times 24 = 96$ for hourly data. The version with all parameter constant would be

$$Y_{d,s} = \beta_0 + \beta_1 X_{d,s}^{\text{ResLoad}} + \beta_2 X_{d,s}^{\text{NGas}} + \beta_3 X_{d,s}^{\text{EUA}} + \varepsilon_{d,s} \tag{189}$$

with just 4 parameters. The GAM based smooth version is defined by

$$Y_{d,s} = \beta_0 + X_{d,s}^{\text{ResLoad}} f_1(s) + X_{d,s}^{\text{NGas}} f_2(s) + X_{d,s}^{\text{EUA}} f_3(s) + \varepsilon_{d,s} \tag{190}$$

with f_i as smoothing function. We know that limiting cases for f_i in (190) correspond to (189) and (188). In Figure we see the REML estimate of (48). It shows that some variations across $\mathbb{S}$ seems to be plausible.

However, we can also consider a model where we allow for non-linearities in the response. This is possible with just one-dimensional additive terms

$$Y_{d,s} = f_1(s) + f_2(X_{d,s}^{\text{ResLoad}}) + f_3(X_{d,s}^{\text{NGas}}) + f_4(X_{d,s}^{\text{EUA}}) + \varepsilon_{d,s} \tag{191}$$

but also with bivariate ones :

$$\begin{aligned} Y_{d,s} =& f_1(s, X_{d,s}^{\text{ResLoad}}) + f_2(s, X_{d,s}^{\text{NGas}}) + f_3(s, X_{d,s}^{\text{EUA}} \\ &+ f_4(X_{d,s}^{\text{ResLoad}}, X_{d,s}^{\text{NGas}}) + f_5(X_{d,s}^{\text{ResLoad}}, X_{d,s}^{\text{EUA}}) + f_6(X_{d,s}^{\text{NGas}}, X_{d,s}^{\text{EUA}}) + \varepsilon_{d,s} \end{aligned} \tag{192}$$

For model (191) we see the corresponding fit for several γ values in Figure 49 with potential shrinkage towards zero using cubic B-splines. There, we see that especially the residual load exhibits significant non-linearities.

For model (192) we also consider tensor parametrization with cubic B-splines and shrinkage towards zero. Figure 50 illustrates the fitted terms in a bivariate plot. Figure 51 shows the same model fitted using tensor interactions. We observe that the latter provides much more robust results.

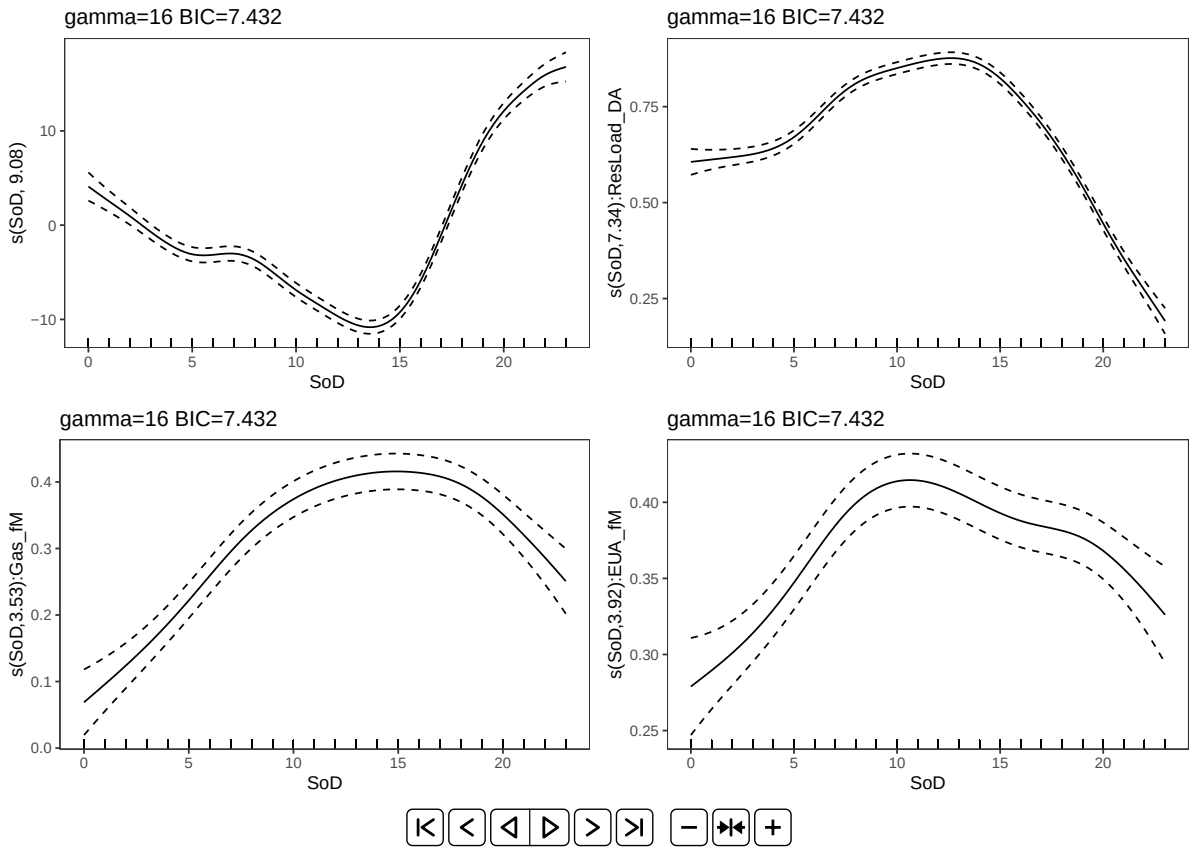


Figure 48: Estimated smoothing terms of model (190) for various γ values.

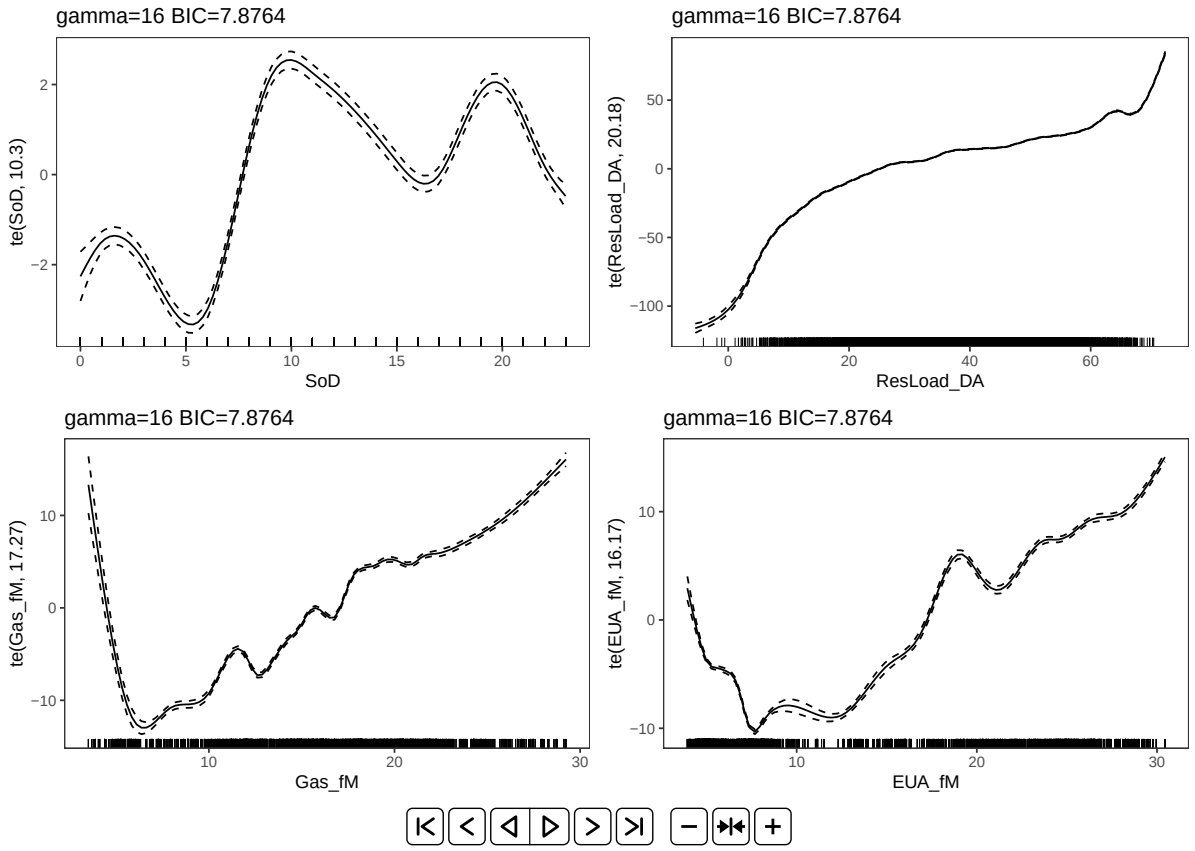


Figure 49: Estimated smoothing terms of model (191) for various γ values.

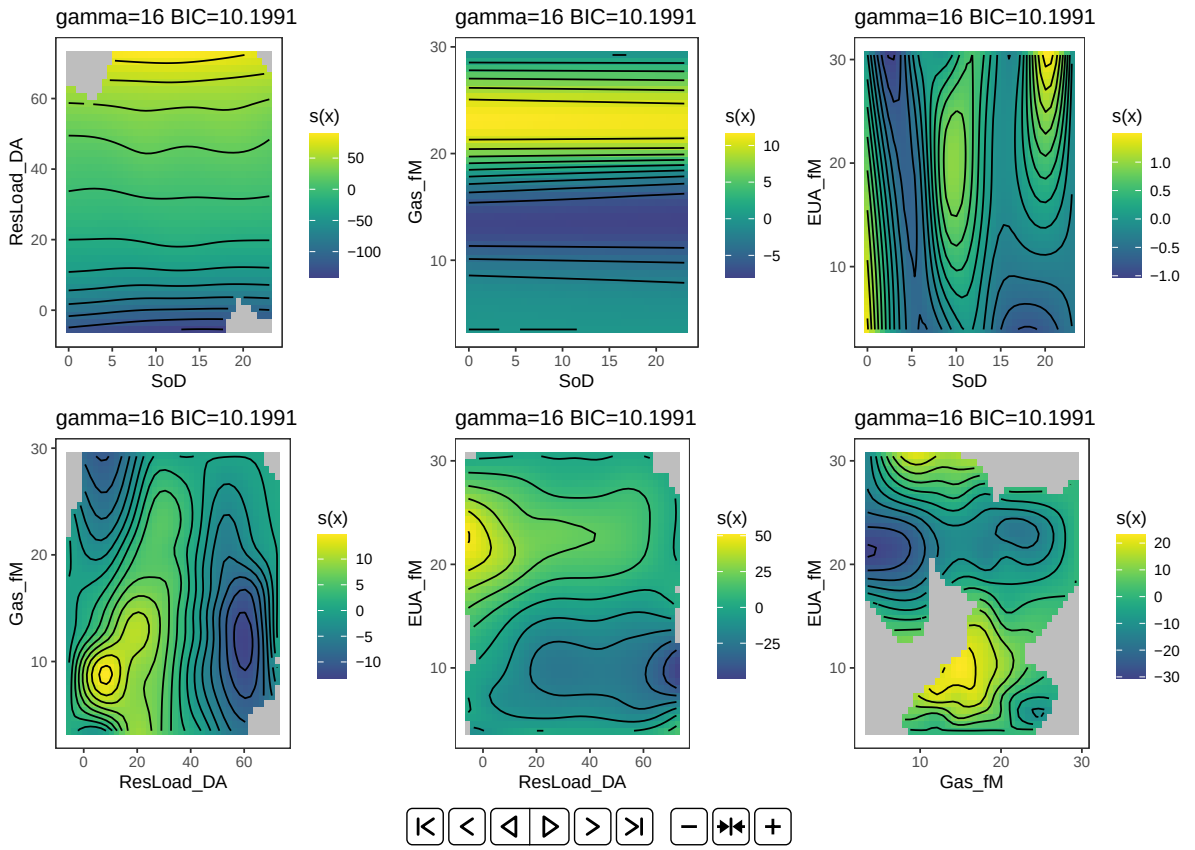


Figure 50: Estimated smoothing terms of model (192) for various γ values.

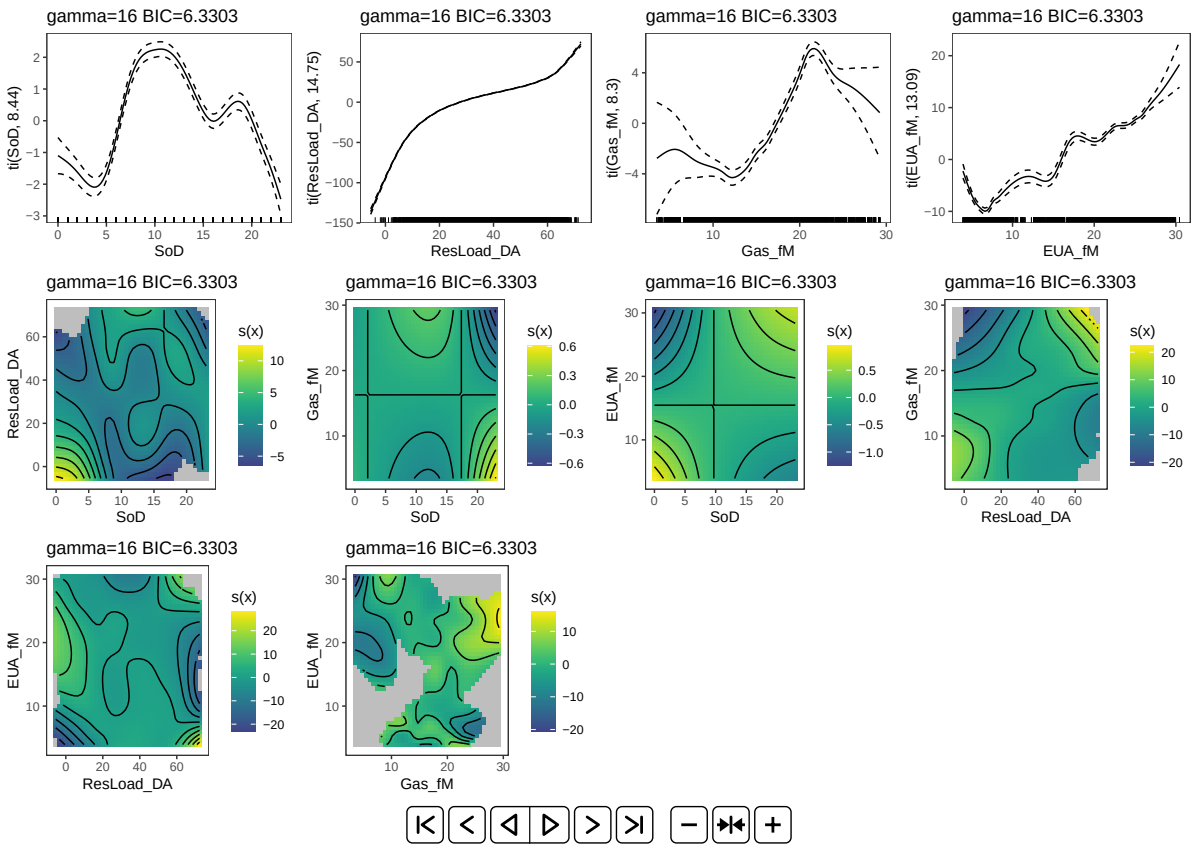


Figure 51: Estimated smoothing terms of model (192) using tensor interaction decomposition for various γ values.

12 Forecast combination

12.1 Motivation for forecasting combination

So far we have studied several tools to improve electricity price forecasting models. Still, we may have observed that there exist for a certain forecasting study multiple models that perform almost equally well. In this situation, it seems plausible to combine the forecasts to some extent to improve the forecasting accuracy. Of course, it is necessary to consider good performing forecasting models that usually are called experts in the context of forecasting combination. The idea behind this approach of combining experts is that different models make errors in different situation, if we combine them we may leverage out some forecasting errors and improve the forecasting accuracy. This approach seems to be more fruitful if the experts are different from each other, e.g. use different model structures, input data, estimation methods, etc.. Note that in general there is also the possibility to combine models, not only forecast. However, here we are focusing on forecast combination, sometimes also known as aggregation of experts. Remember that experts should be well performing as otherwise the principle garbage-in garbage-out applies.

Now, we clarify a certain situation in which we apply forecasting combination. Suppose we have $K > 1$ experts available and denote $\mathbb{K} = \{1, \dots, K\}$ the index set of all experts. For a given sample of electricity prices $\boldsymbol{Y}_1, \dots, \boldsymbol{Y}_D$ the experts provide forecasts $\widehat{\boldsymbol{Y}}_{D+1}^{(k)}$ for each $k \in \mathbb{K}$. It makes sense to look for a function g such that $g(\widehat{\boldsymbol{Y}}_{D+1}^{(1)}, \dots, \widehat{\boldsymbol{Y}}_{D+1}^{(K)})$ is a new forecast for $\boldsymbol{Y}_d$. Before we start, we slightly generalise the setting so that it matches better real data situations. We assume that we conducted a (potentially small) rolling window study for all experts with a rolling window length of N. Thus, we have the experts-provided forecasts $\boldsymbol{Y}_{D+n}^{(k)}$ given $\boldsymbol{Y}_1, \dots, \boldsymbol{Y}_{D+n-1}$ for all $k \in \mathbb{K}$ and all $n \in \{1, \dots, N\}$. We are looking for a combination function g_n which may depend on n such that $g_n(\widehat{\boldsymbol{Y}}_{D+n}^{(1)}, \dots, \widehat{\boldsymbol{Y}}_{D+n}^{(K)})$ is the desired improved forecast.

12.2 Linear combination methods

First, we fix an $s \in \mathbb{S}$ such that we are dealing only with univariate expert models. The most popular class of combination methods is the linear combination. We consider combination functions

$$g_n(\widehat{Y}_{D+n,s}^{(1)}, \dots, \widehat{Y}_{D+n,s}^{(K)}) = \sum_{k \in K} a_{n,k} \widehat{Y}_{D+n,s}^{(k)} = \boldsymbol{a}_n' \widehat{\boldsymbol{Y}}_{D+n,s}^{\mathbb{K}} \tag{193}$$

for linear combination vectors $\boldsymbol{a}_n = (a_{n,1}, \dots, a_{n,k})'$ and expert forecasts $\widehat{\boldsymbol{Y}}_{D+n,s}^{\mathbb{K}} = (\widehat{Y}_{D+n,s}^{(1)}, \dots, \widehat{Y}_{D+n,s}^{(K)})'$. The key question is: Which values for $\boldsymbol{a}_n$ should be considered.

A simple but usually good performing answer is the naive combination

$$\boldsymbol{a}_n^{\text{naive}} = \mathbf{1}/K \tag{194}$$

which weights all experts equally. If there is no further information available, this seems to be a plausible choice, as no expert is preferred to another one. However, if more information is available, e.g. due to past performance of experts, then it might be possible to find a better solution. A simple method that is considered as well is to give full weight of 1 to the expert which performs in the last time step best, and zero weight to the others. Similarly an alternative is to give the weight 1 to the expert which performed best in the recent history. This performance is usually in the target loss function, so usually the RMSE or MAE in our case.

Suppose we want to find $\boldsymbol{a}_n$ so that the variance of the forecasting error is minimised. This is

$$\begin{aligned}
\mathbb{V}\text{ar}[\boldsymbol{a}_n' \widehat{\boldsymbol{Y}}_{D+n,s}^{\mathbb{K}} - Y_{D+n,s}] &= \mathbb{V}\text{ar}[\boldsymbol{a}_n' \widehat{\boldsymbol{Y}}_{D+n,s}^{\mathbb{K}} - Y_{D+n,s} \frac{\boldsymbol{a}_n' \mathbf{1}}{\boldsymbol{a}_n' \mathbf{1}}] \\
&= \mathbb{V}\text{ar}[\boldsymbol{a}_n' (\widehat{\boldsymbol{Y}}_{D+n,s}^{\mathbb{K}} - \frac{Y_{D+n,s}}{\boldsymbol{a}_n' \mathbf{1}} \mathbf{1})] \\
&= \boldsymbol{a}_n' \, \mathbb{V}\text{ar}[\widehat{\boldsymbol{Y}}_{D+n,s}^{\mathbb{K}} - \frac{Y_{D+n,s}}{\boldsymbol{a}_n' \mathbf{1}} \mathbf{1}] \boldsymbol{a}_n
\end{aligned} \tag{195}$$

Under the additional assumption that

$$\boldsymbol{a}_n'\mathbf{1} = 1 \tag{196}$$

the expression turns to

$$\boldsymbol{a}_n' \,\mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{Y}}^{\mathbb{K}}_{D+n,s} - \frac{Y_{D+n,s}}{\boldsymbol{a}_n'\mathbf{1}}\mathbf{1}]\boldsymbol{a}_n = \boldsymbol{a}_n' \,\mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{Y}}^{\mathbb{K}}_{D+n,s} - Y_{D+n,s}\mathbf{1}]\boldsymbol{a}_n = \boldsymbol{a}_n' \,\mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{\varepsilon}}^{\mathbb{K}}_{D+n,s}]\boldsymbol{a}_n \tag{197}$$

where $\mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{\varepsilon}}^{\mathbb{K}}_{D+n,s}]$ denotes the covariance matrix of the expert forecast errors in $D+n$. For the minimisation of (195) under the linear constraint (196) we minimize the Lagrange equation

$$\mathcal{L}(\boldsymbol{a}_n, \lambda) = \boldsymbol{a}_n' \,\mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{\varepsilon}}^{\mathbb{K}}_{D+n,s}]\boldsymbol{a}_n - \lambda(\boldsymbol{a}_n'\mathbf{1} - 1) \tag{198}$$

We compute the derivatives

$$\frac{\partial}{\partial \boldsymbol{a}_n}\mathcal{L}(\boldsymbol{a}_n, \lambda) = 2\,\mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{\varepsilon}}^{\mathbb{K}}_{D+n,s}]\boldsymbol{a}_n - \lambda\mathbf{1} \tag{199}$$

$$\frac{\partial}{\partial \lambda}\mathcal{L}(\boldsymbol{a}_n, \lambda) = \boldsymbol{a}_n'\mathbf{1} - 1. \tag{200}$$

Rearranging $2\,\mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{\varepsilon}}^{\mathbb{K}}_{D+n,s}]\boldsymbol{a}_n - \lambda\mathbf{1} \stackrel{!}{=} \mathbf{0}$ yields

$$\boldsymbol{a}_n^* = \frac{\lambda}{2}\,\mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{\varepsilon}}^{\mathbb{K}}_{D+n,s}]^{-1}\mathbf{1}. \tag{201}$$

To replace λ we multiply the equation above by $\mathbf{1}'$ and receive

$$1 = \mathbf{1}'\boldsymbol{a}_n^* = \frac{\lambda}{2}\mathbf{1}'\,\mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{\varepsilon}}^{\mathbb{K}}_{D+n,s}]^{-1}\mathbf{1}$$

with $\boldsymbol{a}_n'\mathbf{1} = 1$. This yields

$$\lambda = \frac{2}{\mathbf{1}'\,\mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{\varepsilon}}^{\mathbb{K}}_{D+n,s}]^{-1}\mathbf{1}}.$$

Thus

$$\boldsymbol{a}_n^* = \frac{1}{\mathbf{1}'\,\mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{\varepsilon}}^{\mathbb{K}}_{D+n,s}]^{-1}\mathbf{1}}\,\mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{\varepsilon}}^{\mathbb{K}}_{D+n,s}]^{-1}\mathbf{1}. \tag{202}$$

The optimal solution in (193) depends only on the inverse of the covariance matrix $\mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{\varepsilon}}^{\mathbb{K}}_{D+n,s}]$ which has to be estimated. Standard approaches estimate the matrix by the sample covariance matrix either in a expanding or rolling window framework. In situations with many structural changes as in electricity price data the latter one is usually preferred.

You may observe that the optimal solution (202) has a similar structure to the global minimum variance portfolio in Markovitz portfolio theory. Indeed, there are many similarities as we could regard our set of experts as set of assets which we want to manage.

The linear constraint (196) assumption is crucial to receive equation (197) which only depends on the structure of the forecasting errors. If we drop the constraint we receive a slightly different solution. Then the variance of the forecasting error $\boldsymbol{a}_n'\widehat{\boldsymbol{Y}}^{\mathbb{K}}_{D+n,s} - Y_{D+n,s}$ can be expressed as

$$\begin{aligned}\mathbb{V}\mathrm{ar}[\boldsymbol{a}_n'\widehat{\boldsymbol{Y}}^{\mathbb{K}}_{D+n,s} - Y_{D+n,s}] &= \mathbb{V}\mathrm{ar}[\boldsymbol{a}_n'\widehat{\boldsymbol{Y}}^{\mathbb{K}}_{D+n,s}] - 2\,\mathbb{C}\mathrm{ov}[\boldsymbol{a}_n'\widehat{\boldsymbol{Y}}^{\mathbb{K}}_{D+n,s}, Y_{D+n,s}] + \mathbb{V}\mathrm{ar}[Y_{D+n,s}] \\ &= \boldsymbol{a}_n'\,\mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{Y}}^{\mathbb{K}}_{D+n,s}]\boldsymbol{a}_n - 2\boldsymbol{a}_n'\,\mathbb{C}\mathrm{ov}[\widehat{\boldsymbol{Y}}^{\mathbb{K}}_{D+n,s}, Y_{D+n,s}] + \mathbb{V}\mathrm{ar}[Y_{D+n,s}]\end{aligned} \tag{203}$$

Thus, the derivative is

$$\frac{\partial}{\partial \boldsymbol{a}_n}\,\mathbb{V}\mathrm{ar}[\boldsymbol{a}_n'\widehat{\boldsymbol{Y}}^{\mathbb{K}}_{D+n,s} - Y_{D+n,s}] = 2\,\mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{Y}}^{\mathbb{K}}_{D+n,s}]\boldsymbol{a}_n - 2\,\mathbb{C}\mathrm{ov}[\widehat{\boldsymbol{Y}}^{\mathbb{K}}_{D+n,s}, Y_{D+n,s}].$$

Setting this expression to zero leads to

$$\boldsymbol{a}_n = \mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{Y}}^{\mathbb{K}}_{D+n,s}]^{-1}\,\mathbb{C}\mathrm{ov}[\widehat{\boldsymbol{Y}}^{\mathbb{K}}_{D+n,s}, Y_{D+n,s}] \tag{204}$$

which is indeed the minimum but has substantially higher estimation risk than (202).

12.3 Advanced algorithms for expert aggregation

Advanced algorithms updated combination weights every day according to past performances. These methods can be applied using the `profoc` or `opera` package in `R`. Most of these algorithms are robust in the sense that they restrict $a_{n,k}$ to satisfy $0 \leq a_{n,k} \leq 1$ with $a_{n,1} + \dots a_{n,K} = 1$, so we are considering convex combinations. Popular algorithms are the EWA (exponentially weighted average), ML-Poly and BOA (Bernstein online aggregation). Those algorithms are usually combined with the gradient trick.

The EWA is defined by

$$a_{n,k} = \frac{a_{n-1,k} \exp(-\eta \ell(Y_{D+n,s}, \widehat{Y}^{(k)}_{D+n,s}))}{\sum_{l=1}^{K} a_{n-1,l} \exp(-\eta \ell(Y_{D+n,s}, \widehat{Y}^{(k)}_{D+n,s}))} \tag{205}$$

depending on a tuning parameter $\eta > 0$, a loss $\ell(Y_{D+n,s}, \widehat{Y}_{D+n,s})$ and an initial value $\boldsymbol{a}_0$. The initial value may be chose $\boldsymbol{a}_0 = \mathbf{1}/K$. The tuning parameter η may be regarded as (learning) rate. The loss $\ell(Y_{D+n,s}, \widehat{Y}^{(k)}_{D+n,s}))$ measures the forecasting accuracy of $\widehat{Y}^{(k)}_{D+n,s}$ to $Y_{D+n,s}$. As for the forecast evaluation we typically consider the absolute error $\ell(Y_{D+n,s}, \widehat{Y}^{(k)}_{D+n,s}) = |Y_{D+n,s} - \widehat{Y}^{(k)}_{D+n,s}|$ for ℓ_1 resp. MAE minimization, and the squared error $\ell(Y_{D+n,s}, \widehat{Y}^{(k)}_{D+n,s}) = |Y_{D+n,s} - \widehat{Y}^{(k)}_{D+n,s}|^2$ for ℓ_2 resp. MSE minimization.

12.4 Tasks

Hints and the corresponding programming part are provide in Section **??** in `R` and in Section py.8 in `python`.

1. Following a very good performance of the naive combination, try some other similar naive combinations.

2. Create a naive combination of the expert model with 3 long and 3 short rolling windows (this means 1/6 weight for each model). This approach is recommended in [HMW18]. How does it perform?

3. Create a combination of the expert model and the model with 365 days of the year dummies estimated using lasso. Try using different rolling window sizes as well. Does it improve the forecasts?

4. The solution (202) is derived by minimizing the variance under the assumption that $\boldsymbol{a}'\mathbf{1} = 1$. However, if we are measuring performance by the squared error this approach is only optimal if all experts are unbiased. If our forecasts are biased we have to minimise $\mathbb{E}[(\boldsymbol{a}'\widehat{\boldsymbol{Y}}^{\mathbb{K}}_{D+n,s} - Y_{D+n,s})^2]$. Show that in this setting it holds

$$\boldsymbol{a}^*_n = \frac{1}{\mathbf{1}'(\mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{\varepsilon}}^{\mathbb{K}}_{D+n,s}] + \mathbb{E}[\widehat{\boldsymbol{\varepsilon}}^{\mathbb{K}}_{D+n,s}]\mathbb{E}[\widehat{\boldsymbol{\varepsilon}}^{\mathbb{K}}_{D+n,s}]')^{-1}\mathbf{1}} (\mathbb{V}\mathrm{ar}[\widehat{\boldsymbol{\varepsilon}}^{\mathbb{K}}_{D+n,s}] + \mathbb{E}[\widehat{\boldsymbol{\varepsilon}}^{\mathbb{K}}_{D+n,s}]\mathbb{E}[\widehat{\boldsymbol{\varepsilon}}^{\mathbb{K}}_{D+n,s}]'))^{-1}\mathbf{1}. \tag{206}$$

13 Univariate vs. multivariate models

13.1 Introduction into the frameworks

So far we had models that model the electricity price $\boldsymbol{Y}_d$ for each product $Y_{d,s}$ for $s \in \mathbb{S}$ separately. Given the sample $\boldsymbol{Y}_1, \dots, \boldsymbol{Y}_D$ we forecast $\boldsymbol{Y}_{D+1}$ by forecasting each $Y_{D+1,s}$ separately, even though it may incorporate cross-period dependencies. Thus, we can regard the considered forecasting procedure as a multivariate forecasting technique. It is characterised by a model for the daily time series $(\boldsymbol{Y}_d)_{d\in\mathbb{Z}}$. The day-ahead forecast $\boldsymbol{Y}_{D+1}$ is just a 1-step-ahead forecast.

However, it is also possible to regard the electricity price time series $(\boldsymbol{Y}_d)_{d\in\mathbb{Z}}$ as a univariate time series. We used this procedure already to create time series plots of the full data set used. In this situation, the price series is considered as a univariate high-frequency time series. If $S = 24$ we have an hourly time series. In general, this time series is defined as $(Y_t)_{t\in\mathbb{Z}}$ with $t = (d-1)S + s$ for a given pair $(d, s) \in \mathbb{Z} \times \mathbb{S}$. This univariate time series $(Y_t)_{t\in\mathbb{Z}}$ can be back-transformed easily by defining the pair (d, s) by $d = \lceil t/S \rceil$ and $s = t-(d-1)S$. Consequently, it follows for the last observation T in the univariate framework that $T = (D-1)S+s$. For a given sample $Y_1, \dots, Y_T$, we require a S-step ahead forecast for $Y_{T+1}, \dots, Y_{T+S}$ which corresponds to $\boldsymbol{Y}_{D+1}$.

The forecasting scheme for univariate models is usually recursive. By applying a single time-shift on all inputs and replacing potentially unknown values by the forecast, the standard recursive scheme is given by:

Using the general model $\widehat{Y}_t = f(Y_{t-1}, Y_{t-2}, ...)$ and data for $t = 1, \dots, T$ we forecast

- $T+1 : \widehat{Y}_{T+1} = f(Y_T, Y_{T-1}, ...)$,
- $T+2 : \widehat{Y}_{T+2} = f(\widehat{Y}_{T+1}, Y_t, ...)$,

$\vdots$

- $T+S : \widehat{Y}_{T+S} = f(\widehat{Y}_{T+S-1}, \widehat{Y}_{T+S-2}, ...)$

in the mentioned recursive manner.

From the perspective that every day all S prices are released at once, the univariate modelling approach seems to be a bit implausible in contrast to the multivariate modelling framework. The univariate modelling framework seems to be more natural perspective for continuous time process that are measured in discrete time, such as electricity load, wind and solar generation. The regular arrival of new data corresponds to the information flow as modelled in univariate models. Another disadvantage of the univariate modelling approach is the usage of external regressors. In a recursive forecasting scheme the external regressors usually have to be modelled as well, to update them in the recursive scheme. Moreover, the univariate approach suffers from the problem of the accumulation of errors in the forecasting. The only big positive aspect of the univariate model is that it uses a lot of data (S-times as much) for the estimation of all parameters, and this can lead to smaller estimation errors and parameter uncertainty. Finally, we want to mention that it is always possible to rewrite a univariate model framework into a multivariate model framework. We discuss this later on in detail.

13.2 The univariate AR(p)

A standard example of time series model is the AR(p) process. The AR(p) model can also be assumed for the univariate time series $(Y_t)_{t\in\mathbb{Z}}$. Here, a single model is assumed for the full time series:

$$Y_t = \phi_0 + \sum_{k=1}^{p} \phi_k Y_{t-k} + \varepsilon_t \tag{207}$$

with $\mathbb{E}(\varepsilon_t) = 0$. The univariate AR($p$) has shown surprisingly good forecasting results compared to its simplicity. A key issue is that usually a large order p is required. In general an AR($14S$) with a memory of two weeks yields promising forecasting results. Usually the optimal order p is chosen by minimising an information criterion (esp. AIC) for a given grid of possible lags $0, \dots, p_{\max}$. The upper grid bound $p_{\max}$ must be chosen sufficiently large, usually it should cover a memory of a couple of weeks. One may think that a high-dimensional AR(p) process

might be overparametrised, as e.g. an AR($14S$) has with $S = 24$ in total 336(+1) parameters. However, remember that also the simple expert model (7) has 7 parameters for each of the S equations. Thus, for $S = 24$ in total $7 \times 24 = 168$ parameters which is of the same magnitude as the univariate AR($14S$).

The `ar` function in R provides a powerful tool for estimation of a high-dimensional AR process if it is estimated via solving the Yule-Walker equations or using the preferable and somewhat adjusted BURG algorithm. Both methods guarantee that the estimated solution is stationary - in contrast to the OLS and MLE methods. Of course, advanced estimation methods such as lasso or ridge can be applied as well. If the process is augmented by external regressors, non-linear effects or dummies, this is definitively the preferred way to proceed.

Remember that we know that there is a clear seasonal structure in electricity price data. But AR(p) processes are stationary, i.e. $\mathbb{E}[Y_t]$ is constant. Still, for short-term (esp. day-ahead) forecasts the seasonal structure will be approximated. However, we can adapt the AR(p) process so that $\mathbb{E}[Y_t]$ is not constant anymore. Therefore we note that every AR(p) (see eq. (207)) can be rewritten to

$$Y_t = \mu + \sum_{k=1}^{p} \psi_k (Y_{t-k} - \mu) + \varepsilon_t. \tag{208}$$

For Y_t in (208) it holds $\mathbb{E}[Y_t] = \mu$. So the representation corresponds to the demeaned parametrisation of the AR(p). In `R` this also used by default in the `ar` function.

Now, we can replace μ by a seasonal mean, e.g.

- the hour-of-the-day mean: $\mu_{\text{HoD}}(t) = \mathbb{E}[Y_t] = \mathbb{E}[Y_{t+kS}]$ for all k
- the hour-of-the-week mean: $\mu_{\text{HoW}}(t) = \mathbb{E}[Y_t] = \mathbb{E}[Y_{t+k7S}]$ for all k

Due to the periodicity $\mu_{\text{HoD}}(t)$ can be represented by S values $\mu_1, \dots, \mu_S$ and $\mu_{\text{HoD}}(t)$ by $7S$ values $\mu_1, \dots, \mu_{7S}$. The estimation can be done easily by computing the corresponding sample means. Clearly, we receive $\mu_{\text{HoD}}(t) = \mu_k$ for $k \in \{1, \dots, S\}$ and $k = t - (\lceil t/S \rceil - 1)S$, and similarly $\mu_{\text{HoW}}(t) = \mu_k$ for $k \in \{1, \dots, 7S\}$ and $k = t - (\lceil t/(7S) \rceil - 1)7S$. We have computed already the $7S$ ($= 168$ for $S = 24$) values of $\mu_{\text{HoD}}(t)$ for the electricity prices, see Figure (23). The resulting models are the AR(p) with hour-of-the-day mean (abbrv. AR-HoD) and the AR(p) with hour-of-the-week mean (abbrv. AR-HoW)

$$Y_t = \mu + \sum_{k=1}^{p} \psi_k (Y_{t-k} - \mu_{\text{HoD}}(t)) + \varepsilon_t, \tag{209}$$

$$Y_t = \mu + \sum_{k=1}^{p} \psi_k (Y_{t-k} - \mu_{\text{HoW}}(t)) + \varepsilon_t. \tag{210}$$

Usually, the AR-HoW shows more powerful forecasting performance than the AR-HoD which is itself better than the univariate AR process.

Another way to incorporate seasonal features into AR(p) models is to apply a seasonal AR(p) model, abbr. as SAR$_L$(p,q) with parameters p and q where the latter one describes seasonal linear dependencies of periodicity L. Using the backshift operator B[22] (sometimes known as lag operator) and the notation $B^K = \underbrace{B \circ B \circ \ldots \circ B}_{K\text{-times}}$ we define the model as

$$\Phi(B)\Theta(B^K)Y_t = \varepsilon_t$$

with Φ and Θ as polynomials. E.g. for $K = S$, $\Theta(z) = 1 - \theta z$ and $\Phi(z) = 1 - \phi z$ we receive

$$\begin{aligned}
\varepsilon_t &= \Phi(B)\Theta(B^K)Y_t && (211)\\
&= (1 - \phi B)(1 - \theta B^S)Y_t && (212)\\
&= (1 - \phi B - \theta B^S + \phi\theta B^{S+1})Y_t && (213)\\
&= Y_t - \phi B Y_t - \theta B^S Y_t + \phi\theta B^{S+1} Y_t && (214)\\
&= Y_t - \phi Y_{t-1} - \theta Y_{t-S} + \phi\theta Y_{t-S-1} && (215)
\end{aligned}$$

[22] It is defined by $BX_t = X_{t-1}$

which is equivalent to

$$Y_t = \phi Y_{t-1} + \theta Y_{t-S} - \phi\theta Y_{t-S-1} + \varepsilon_t. \tag{216}$$

We observe that (216) is an autoregressive time series model with two parameters. It is a special case of an AR($S+1$). Especially, it is nested in the 3-parametric sparse AR($S+1$) given by

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-S} + \phi_3 Y_{t-S-1} + \varepsilon_t. \tag{217}$$

Note that (216) is a non-linear model in the parameters due to the multiplicative parameter structure in the third term of the right hand side. Thus, the estimation usually requires non-linear optimisation methods which is usually time-consuming. In contrast, (217) is the minimal linear model which nests the $\mathrm{SAR}_S(1,1)$. Such a minimal linear model representation exists uniquely for all $\mathrm{SAR}_L(p,\, q)$-models. This minimal linear model representation has only marginally more parameters than the original $\mathrm{SAR}_L(p,\, q)$ model. In case of the $\mathrm{SAR}_S(1,1)$, the minimal linear model representation (217) has only one parameter more. Thus, the additional overestimation risk is very limited but it can be estimated by OLS. Thus, from the practical point of view seasonal AR processes are not favourable and hardly applied for forecasting purpose. The same holds for the more general class of SARIMA models in general. Moreover, as every $\mathrm{SAR}_L(p,\, q)$ model is nested into an $\mathrm{AR}(p_{\mathrm{large}})$ model, it has a constant mean that is not seasonal. Applying modifications as in (209) and (210) may eliminate this disadvantage partially.

13.3 Multivariate to univariate and vice versa

We mentioned that we can rewrite every univariate model into a multivariate model. For the naive models this is quite obvious. E.g. for the simple naive-S model (1), we have the multivariate and univariate representation:

$$Y_{d,s} = Y_{d-1,s} + \varepsilon_{d,s} \tag{218}$$

$$Y_t = Y_{t-24} + \varepsilon_t \tag{219}$$

The multivariate model can be denoted as

$$\boldsymbol{Y}_d = \boldsymbol{Y}_{d-1} + \boldsymbol{\varepsilon}_d \tag{220}$$

as well where the multivariate model notation is used.

For the more complex models this is certainly trickier, esp. for converting univariate models into multivariate ones. Converting a multivariate model into a univariate one can be always done using S cases. The result might be simplified further. For instance, for the expert model (7)

$$Y_{d,s} = \beta_{s,0} + \beta_{s,1}Y_{d-1,s} + \beta_{s,2}Y_{d-2,s} + \beta_{s,3}Y_{d-7,s} + \beta_{s,4}\mathrm{DoW}_d^1 + \beta_{s,5}\mathrm{DoW}_d^6 + \beta_{s,6}\mathrm{DoW}_d^7 + \varepsilon_{d,s} \tag{221}$$

$$Y_t = \Big\{\beta_{s,0} + \beta_{s,1}Y_{t-24} + \beta_{s,2}Y_{t-48} + \beta_{s,3}Y_{t-168} + \beta_{s,4}\mathrm{DoW}_t^1 + \beta_{s,5}\mathrm{DoW}_t^6 + \beta_{s,6}\mathrm{DoW}_t^7 \text{ if } t - (\lceil t/S \rceil - 1) = s \quad + \varepsilon_t \tag{222}$$

which both have $S \times 7$ parameters. The first one can be written in multivariate notation by

$$\boldsymbol{Y}_d = \boldsymbol{\beta}_0 + \boldsymbol{\beta}_1 \circ \boldsymbol{Y}_{d-1} + \boldsymbol{\beta}_2 \circ \boldsymbol{Y}_{d-2} + \boldsymbol{\beta}_3 \circ \boldsymbol{Y}_{d-7} + \boldsymbol{\beta}_4\mathrm{DoW}_d^1 + \boldsymbol{\beta}_5\mathrm{DoW}_d^6 + \boldsymbol{\beta}_6\mathrm{DoW}_d^7 + \boldsymbol{\varepsilon}_d \tag{223}$$

with $\circ$ as Hadamard product (also known as entrywise product).

The univariate AR(p) in (208) representation has the multivariate and univariat representations with $r = p/S$[23]:

$$\boldsymbol{Y}_d = \boldsymbol{\Psi}_0^{-1}\mu\mathbf{1} + \sum_{k=1}^{r} \boldsymbol{\Psi}_0^{-1}\boldsymbol{\Psi}_k(\boldsymbol{Y}_{d-1} - \mu\mathbf{1}) + \boldsymbol{\varepsilon}_d \tag{224}$$

$$Y_t = \mu + \sum_{k=1}^{p} \psi_k(Y_{t-k} - \mu) + \varepsilon_t \tag{225}$$

[23] without loss of generality we may assume that p/S is an integer

and

$$\boldsymbol{\Psi}_0 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ -\psi_1 & 1 & 0 & \cdots & 0 & 0 \\ -\psi_2 & -\psi_1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -\psi_{22} & -\psi_{21} & -\psi_{20} & \cdots & 1 & 0 \\ -\psi_{23} & -\psi_{22} & -\psi_{21} & \cdots & -\psi_1 & 1 \end{bmatrix} \quad \text{and} \quad \boldsymbol{\Psi}_k = \begin{bmatrix} \psi_{24(k-1)} & \psi_{24(k-1)-1} & \cdots & \psi_{24(k-1)-23} \\ \psi_{24(k-1)+1} & \psi_{24(k-1)} & \cdots & \psi_{24(k-1)-22} \\ \vdots & \vdots & \ddots & \vdots \\ \psi_{24(k-1)+23} & \psi_{24(k-1)+22} & \cdots & \psi_{24(k-1)} \end{bmatrix}$$

13.4 Tasks

Hints and the corresponding programming part are provide in Section **??** in `R` and in Section py.9 in `python`.

1. Write a forecasting function for model (208) and (210). Compare their forecasting performance with **expert.adv**.

2. Compare the computation time for estimating (208) when using estimation by

   a) Yule-Walker equations

   b) Least squares

   c) The BURG algorithms

   d) Gaussian Maximum likelihood

   Do you observe any noticeable differences in the estimated parameters? Can you confirm that BURG is the preferable algorithm?

3. Show that (207) can be written as (208).

4. Rewrite model (3) into a univariate framework.

5. Find the minimal linear model representation of $\Phi(B)\Theta(B^S)\Psi(B^{7S})Y_t = \varepsilon_t$ for $\Phi(z) = 1 - \phi_1 z - \phi_2 z^2$, $\Theta(z) = 1 - \theta_1 z$ and $\Psi(z) = 1 - \psi_1 z$.

14 Basics on non-linear models

We have seen several linear models that have non-linear effects with respect to specific regressors. But the model structure itself was always linear, as the electricity price $Y_{d,s}$ depends linearly on the parameters $\boldsymbol{\beta}_s$. This also holds for the more sophisticated high-dimensional linear models using estimation techniques like lasso as well as for generalized linear models. We know that approximating non-linear relationships by sufficiently large linear models is feasible, e.g. using Taylor approximation, Fourier approximation, wavelets, and other approximation techniques. However, sometimes this blows up the parameter space a lot which increases the estimation risk substationally. To avoid those large parameter spaces to model non-linear relationships, non-linear models can be a suitable approach. They come with the advantage that non-linear model components can be described directly.

In this chapter we will mainly cover two modern non-linear modeling approach used data science for forecasting purposes. These are decision tree learning based methods such as neural networks. However, the latter one is more popular in the energy market community.

Anyway, before we dive deeper into the world of decision trees and neural networks we consider a simple non-linear model first. This is mainly for illustration purpose to understand better the difference between a linear model with non-linear effects and a purely non-linear model, but we will also see how can be useful for improving our fundamental electricity price understanding and the relationships between data science and fundamental models.

14.1 A simple non-linear model

We know that a supply-stack model is a suitbale simple model to explain electricity price relationships. Recalling equation (95), this model is

$$Y_{d,s} = \mathrm{MO}(\mathrm{DAResLoad}_{d,s}) + \varepsilon_{d,s} \tag{226}$$

We used some simple linear assumptions on the merit order and derived linear models with non-linear effects. However, using some different assumptions on the merit-order curve will give us directly a non-linear assumption. We may assume that the merit order has a power shape (polynomial shape with real powers) :

$$\mathrm{MO}(x) = \theta_0 + \theta_2 \mathrm{sgn}(x-\theta_1)|x-\theta_1|^{\theta_3} \tag{227}$$

This gives us directly a four-parametric model for the price $Y_{d,s}$

$$\begin{aligned} Y_{d,s} &= \theta_{0,s} + \theta_{2,s}\mathrm{sgn}(\mathrm{DAResLoad}_{d,s} - \theta_{1,s})|\mathrm{DAResLoad}_{d,s} - \theta_{1,s}|^{\theta_{3,s}} + \varepsilon_{d,s} \\ &= f_{\boldsymbol{\theta},s}(\mathrm{DAResLoad}_{d,s}) + \varepsilon_{d,s} \end{aligned} \tag{228}$$

while introducing $\boldsymbol{\theta}$ and $f_{\boldsymbol{\theta},s}$ as abbriviations for the parameter vector and the model functions. As we have S delivery periods the overall number parameters is $4S$ for all $s \in \mathbb{S}$. We have to estimate the parameter vectors $\boldsymbol{\theta}_s = (\theta_{0,s}, \ldots, \theta_{3,s})$ using several different techniques.

Popular methods like least squares is an option but also maximum likelihood under a specific distribution assumption. The (non-linear) least squares estimator is given by

$$\widehat{\boldsymbol{\theta}}_s = \arg\min_{\boldsymbol{\theta}} \sum_{d=1}^{D} (Y_{d,s} - f_{\boldsymbol{\theta},s}(\mathrm{DAResLoad}_{d,s}))^2 \tag{229}$$

For linear models that we covered in the previous sections least squares was clearly the most popular choice. We want to remind, that under the normality assumption the resulting Gaussian maximum likelihood estimation method is equivalent to least squares. The same result holds for non-linear models. However, for non-linear methods the resulting optimization problem is usually non-linear (and often non-convex). For those reasons there is no computational advantage from using least squares for estimating non-linear models as we have it for linear models. Therefor it is much more attractive to use more complex objectives like non-Gaussian (=non-normal) maximum likelihood estimation techniques. If we consider a likelihood assumption for the model which is more plausible than the normal distribution assumption we should expect

some improvement in the model fit and resulting predictive accuracy. This holds especially for heavy tailed data when least squares is less reliable.

Obviously, we easily generalize the model (228) and (229) to models which do not only depend on residual load but on plenty other parameters and larger parameter vectors. As for linear models they could be summaried in an input vector $\boldsymbol{X}_{d,s} = (X_{d,s,1}, \ldots, X_{d,s,p})'$ with p as input vector dimension. In contrast to a linear relationship $\boldsymbol{\beta}_s' \boldsymbol{X}_{d,s}$ the model follows the general non-linear function f_s. Of course for $f_s(\boldsymbol{X}_{d,s}; \boldsymbol{\beta}_s) = \boldsymbol{\beta}_s' \boldsymbol{X}_{d,s}$ we are back to the linear world.

Thus, clearly all the linear models are nested in all non-linear ones, and all the theory for non-linear models also holds for linear ones. Unfortunately, this does not hold the other way round. Many nice properties of linear model do not (fully) transfer into the non-linear world. We got already an idea of this property by briefly discussing estimation/training techniques. The parameter estimation is likely the most critical problem. As mentioned for linear models, we have fast estimation algorithms available. This is e.g. explicitely OLS, but also techniques like lasso or elastic net can be considered. All those parameter estimation methods tends to be fast, and often comes with additional garuantees, e.g. that we know that we will reach a unique optimum. Unless we are in very special cases for non-linear models, the estimation or parameter training tends to be slow, and much similarly bad, comes usually with no garuantee concerning optimality. So if we find an optimum this usually on a local optimum, and for moderately or highly complex models we have essentially no chance to prove that a found local optimum is the global one. The larger the parameter space the worse this problem. Thus, a small parameter space is more beneficial in non-linear models that in linear ones.

Hence, with respect to model (228) we could also go to a multivariate world where we have to estimate one big model instead of S small models for all $s \in \mathbb{S}$ with many overall parameters. A simple example is (228) with the additional restriction that all parameters are the same across all delivery products $s \in \mathbb{S}$. This model has only 4 parameters given by:

$$\begin{aligned} Y_{d,s} &= \theta_0 + \theta_2 \text{sgn}(\text{DAResLoad}_{d,s} - \theta_1)|\text{DAResLoad}_{d,s} - \theta_1|^{\theta_3} + \varepsilon_{d,s} \\ &= f_{\boldsymbol{\theta}}(\text{DAResLoad}_{d,s}) + \varepsilon_{d,s} \end{aligned} \tag{230}$$

with corresponding least square estimation

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \sum_{d=1}^{D} (Y_{d,s} - f_{\boldsymbol{\theta}}(\text{DAResLoad}_{d,s}))^2. \tag{231}$$

We have seen a simple non-linear model for electricity prices which follows directly by the assumption of a non-linear merit order curve. Obviously, such models tend to be too trivial to yield high accuracy. There are multiple ways to get to more complex models, two popular ways in data science are either based on decision tree learning or on neural networks.

However, non-linear models with small to moderate complexity based on the merit order curve (as e.g. (228)) may be helpful for understanding electricity markets. This holds especially for those models which are based on the supply stack model. If you recall the supply stack model, you might notice that no parameter estimation is required, but plenty of external input. It is also possible to estimate just a bit of the required external input, especially those parts which are uncertain from the model or the data quality. In a standard merit order model we require information of the residual load $\text{ResLoad}_{d,s}$ (or alternatively e.g. the load, wind and solar information), the capacities of the conventional power plants (groups) $\boldsymbol{C}_{d,s}^{\text{upper}}$ and $\boldsymbol{C}_{d,s}^{\text{lower}}$ such as the corresponding fuel and emission prices $\mathbf{price}_{d,s}$. For instance we might know $\text{ResLoad}_{d,s}$ and $\mathbf{price}_{d,s}$ but the power plant information is known. If we assume that we have 2 fuel classes, one with small marginal costs, the other one with large marginal costs, then we get a model like (100) with additional unknown VolStack_1 parameter which has to be estimated. Even though such a model looks still quite simple, it is a non-linear model. Thus, the estimation tends to be demanding.

15 Decision tree learning

Decision trees are likely the most simple non-linear model that someone could consider. In can easily detect high-level (or deep) non-linear relationships between variables, like 3-way or 4-way interaction. Here, linear models are usually struggeling as non-linear approximation technique require a hugh enlargement of the parameter space. Decision trees can me trained in a fast way but decision trees provide non-continuous solutions. The latter fact is clearly unrealistic for electricity price forecasting. Therefore some advanced techniques have to be considered to make them *more* continuous. Two popular approach are random forest and gradient boosting machines which are aggregation of many trees. An alternative is the so calles MARS approach which also allows directly for linear components in the model. However, we will first study decision trees to a relevant depth to continue on more advanced decision tree learning afterwards.

15.1 Decision trees

First, we consider a simple decision tree for a univariate regression problem in the electricity price forecasting context. Assume we want to regress the electricity price on the residual load. We have seen that there is a clear non-linear relationship between those two variables. A decision tree is build by a recursive binary splitting of the residual load. The splitting is done such that a loss is minimized, typically the squared loss, so the sum of squared residuals. The splitting is done until a stopping criterion is reached. TODO

15.2 Random forests

TODO

15.3 Gradient boosting machine (GBM)

TODO

15.4 MARS

TODO

16 Artificial Neural Networks (ANN)

From the modeling perspective ANNs can described as a generalisation of generalised linear models (GLM), so it extents the linear model framework. Especially, all linear models can be regarded as ANN. For more details on ANNs and the application in `R` see [GF10].

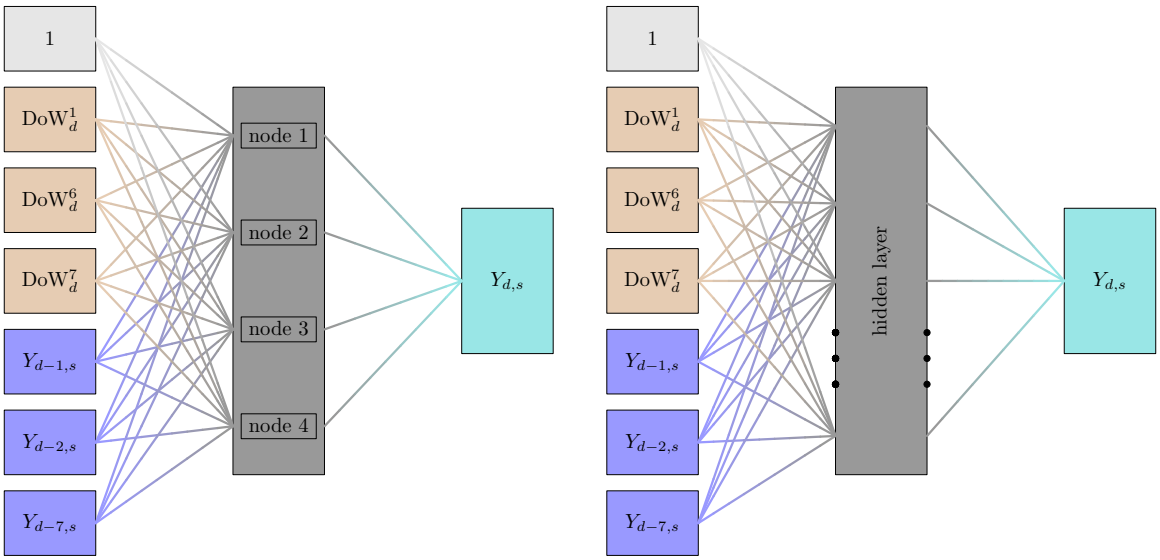
Artificial neural networks are motivated by the structure of the brain. Therefore the modeling components have names such as neurons and synapses.

The artificial neural network contains of neurons which represent an information set. The neurons are linked by synapses which represent functions that describe the information flow. This information flow can be represented by using an (algebraic) graph. In Figure 8 we have seen the neural network graph structure of the expert model. There we see an input layer of 6 neurons (Lag1, Lag2, Lag7, Mon, Sat, Sun). In a neural network representing graph the input layer on the left hand side. They are connected by synapses with an output layer. This is usually presented on the right hand side of the graph. The input layer and the output layer are essential elements in all neural network models.

16.1 Multilayer Perceptron

The most important subclass of neural nets are multilayer perceptrons (MLPs) which are so called feed-forward networks. In a multilayer perceptrons they are next to the input and the output layer multiple ordered hidden layers which contain nodes, or sometimes called neurons. Moreover, these nodes are connected with all preprocessing node, as e.g. the output node in in Figure 8 of the expert model representation.

In Figure 52 we see two graphs that represent the MLP with one layers, the first one with exactly 4 neurons in the hidden layer, the second with a stylized hidden layer. In these examples the neural nets take exactly the same input as the expert models, and the same output, namely the electricity price $Y_{d,s}$. Figure 53 shows similar graphs for MLPs with two hidden layers.

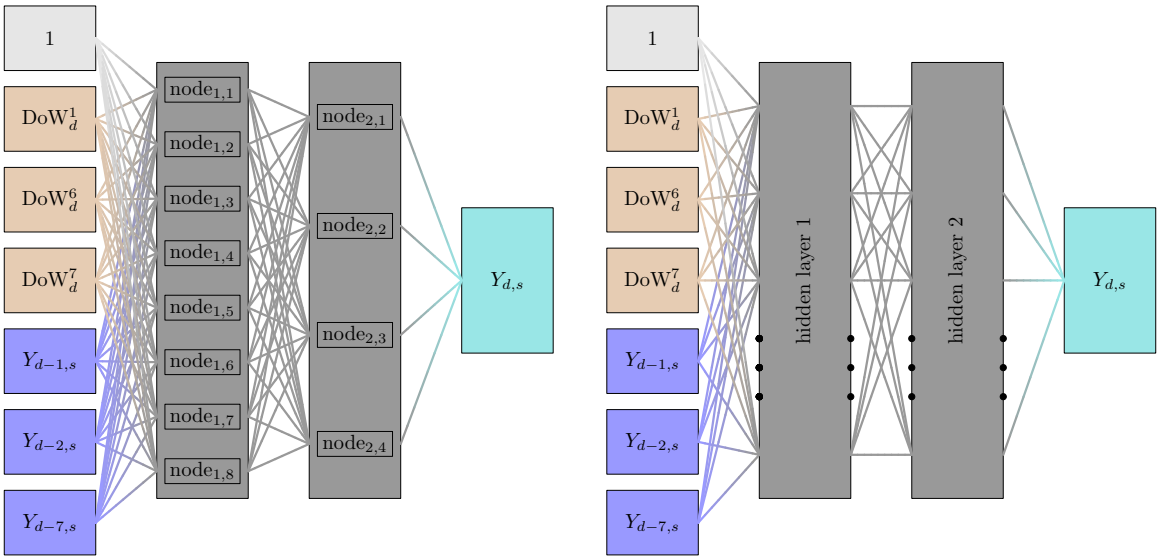


(a) MLP with one single hidden layer of 4 nodes

(b) MLP with one hidden layer of non-specified number of nodes.

Figure 52: Visualisation graph of multilayer perceptrons with one hidden layer and the input as in the expert model. The input neurons are left, hidden layer is centred and the output is right.

Again, the left one has 8 nodes in the first hidden layer and 4 in the seconde, the right one has a non-specified number of nodes in each hidden layer.



(a) MLP with two hidden layers of 8 and 4 nodes

(b) MLP with two hidden layer of non-specified number of nodes.

Figure 53: Visualisation graph of multilayer perceptrons with two hidden layers and the input as in the expert model. The input neurons are left, hidden layers are centred and the output is right.

In general we can say, the more neurons and the more hidden layers in the neural net, the more complex is the model structure. However, [HSW89] showed the universal approximation theorem. Basically it is saying that under mild regularity conditions a single hidden layer multilayer perceptron is able to model any non-linear continuous function arbitrarily well if a

sufficient amount of neurons in the hidden layer is chosen. This suggests, that more than one hidden single layer is not required in an electricity price forecasting model. Still, this only holds if we assume that the electricity price function is a continuous function in the input values. Due to the universal approximation theorem we first study MLPs with one hidden layer, and later on look at corresponding models with more hidden layers.

For the explicit model formulation of multilayer perceptrons we consider the electricity price $Y_{d,s}$ as target output and a set of regressors $\boldsymbol{X}_{d,s}$ as set of input values (without the constant). In the case of Figure (52) we would have $\boldsymbol{X}_{d,s} = (Y_{d-1,s}, Y_{d-2,s}, Y_{d-7,s}, \mathrm{DoW}_d^1, \mathrm{DoW}_d^6, \mathrm{DoW}_d^7)$. Given $\boldsymbol{X}_{d,s}$ we can write down a multilayer perceptrons model with a single hidden layer out of J neurons:

$$Y_{d,s} = f\left(\beta_{s,0} + \sum_{j=1}^{J} \beta_{s,j} f_j(\beta_{s,0,j} + \boldsymbol{\beta}'_{s,j}\boldsymbol{X}_{d,s})\right) + \varepsilon_{d,s} \tag{232}$$

The coefficients $\beta_{s,0}$, $\beta_{s,j}$, $\beta_{s,0,j}$ and $\boldsymbol{\beta}_{s,j}$ are usually called weights in the machine learning literature, as they weight the input of each corresponding neuron. However, we prefer to use the term parameters as we are in the statistical resp. forecasting context. The functions f and f_j are called link or activation functions, whereas f is the link function of the output neuron and f_j are the link functions of the hidden neurons. f is sometimes also referred as output function. They are smooth function that characterize the (potential) non-linear model structure of the neural net. In electricity price forecasting we are dealing with the real valued response $Y_{d,s}$ in a regression setting. In those situations, it is convenient to choose f as a linear output function. The link functions f_j which map the input data into hidden neurons are usually sigmoidal functions. This means monotonically increasing functions. Most the time, the same activation function is used for all J hidden neuron, so e.g. $f_j = \text{logit}$ for all j.

Historically, two very popular link functions are the logit function (well known from logistic regressions) defined by

$$\text{logit}(z) = \frac{1}{1+e^{-z}}$$

and the tangent hyperbolicus (tanh) function

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

they were popular in the 90's and the beginning of the 2000's. Some years later the ReLU (rectified linear unit) defined by

$$\text{ReLU}(x) = \max(x, 0)$$

is also known as *rectifier*. It become popular due to its robustness properties. We discussed the ReLU already in the context of non-linear function approximations and linear splines. Here, we also discussed the robustness properties, in the context of the tail index, we remember that the absolute value function such as maximum and minimum functions preserve the tail index which is an advantage for many estimation/learning algorithms. However, also ReLUs face problems, especially the so called dead ReLU problem. Still, even here are solutions available such as leaky ReLU or smooth ReLU. However, there are many other link functions available which are also used in practice. Finally, We only want to mention radial basis function as used for the activation function in neural networks as well. The most common radial basis function are Gaussian radial basis functions. They are defined by

$$\phi(z) = e^{-(\lambda z)^2}.$$

where λ is a tuning parameter that has to be specified.

Note that if in (232) next to f also f_j is chosen as a linear function as well, then the resulting neural net has a linear model structure. This model structure corresponds directly to a linear PCA based modeling approach. Here it is required that the number of hidden neuron J is less than the input variables. In other words, PCA regression (PCA on input matrix and the regression on the transformation resp. PCA approximation) can be regarded as a linear neural network with one hidden layer, where the parameters are estimated in a two-step approach.

In an MLP the hidden as introducen in equation (232) are usually fully connected. This means that each input variable is connected to each hidden neuron. Such a fully connected layer

is also called dense layer. This is also the case for the hidden layer to the output layer. However, there are also neural network structures where the hidden layers are not fully connected, so sparse connections are used. This is particularly relevant for large input spaces. In this case, the number of parameters can be reduced significantly. Still, it is up to the expertise of the modeler to decide which connections are relevant and which are not. This is not an easy task, and often the fully connected approach is used. However, also dropout techniques can be used to reduce the number of parameters. Here, the idea is to randomly drop some connections during the training process. This technique can be regarded as regularization technique which can help to reduce overfitting - due to reduction of the parameter space.

Alternatively, it is also possible to add a penalty term to the loss function which penalizes parameters, similarly as we introduced it for the lasso and ridge penalty. Here, also the ℓ_1- and ℓ_2-penalty are the most popular ones. Basically they act similarly as in a regression setting. However, the ℓ_1-penalty is not differentiable at zero, which is a problem for many optimization algorithms. Therefore, the ℓ_2-penalty is more popular in neural networks. However, the ℓ_1-penalty is still used in practice, and there are also some optimization algorithms which can handle the non-differentiability at zero. One crucial problem is that regularization only performs well if the parameters (=weights) are on the same scale. Otherwise, the regularization will be dominated by the large parameters.

Therefore, it is common to standardize the input variables to have mean zero and standard deviation one. We discussed this as a common approach for linear models as well. However, for some MLP architectures it is also common to standardize the output variable to have mean zero and standard deviation one. This is not relevant in linear models. The reason is that the activation functions are usually chosen such that the output is in a certain range. For instance, the logit function has a range of $[0, 1]$, the tanh function has a range of $[-1, 1]$. Thus, the output variable is often standardized to have mean zero and standard deviation one. If ReLUs are used as activation functions, then the output variable does not have to be standardized. However, the input variables are still standardized, by subtracting the mean and dividing by the standard deviation, or alternatively by robust standardization using the median and the median absolute deviation (MAD). The latter is often preferred if outliers are expected in the input data. In machine learning it is also popular to scale the inputs to the range $[0, 1]$ or $[-1, 1]$. This is not common in statistics, but it is also possible. However, it is important to remember that the activation functions are chosen such that the output is in a certain range.

The single hidden layer perceptron ANN as defined in equation (232) can be regarded as the standard neural network model in electricity price forecasting. In general, it can be generalised by an arbitrary integration function $g(\boldsymbol{\beta}, \boldsymbol{x}) = \beta_0 + \sum_{i=1}^{p} \beta_i x_i$ with $\boldsymbol{\beta} = (\beta_0, \dots, \beta_p)$ and $\boldsymbol{x} = (x_1, \dots, x_p)$. Using the general integration function notation g, we can rewrite

$$Y_{d,s} = f\left(\beta_{s,0} + \sum_{j=1}^{J} \beta_{s,j} f_j(g_j(\boldsymbol{\beta}_{s,j}, \boldsymbol{X}_{d,s}))\right) + \varepsilon_{d,s} \tag{233}$$

$$= f\left(g(\boldsymbol{\beta}_s, \boldsymbol{f}(\boldsymbol{g}(\boldsymbol{B}_s, \boldsymbol{X}_{d,s})))\right) + \varepsilon_{d,s} \tag{234}$$

using the activation and integration functions $\boldsymbol{f} = (f_1, \dots, f_J)'$ and $\boldsymbol{g} = (g_1, \dots, g_J)'$ for the input layer and f and g for the hidden layers. Moreover, $\boldsymbol{\beta}_s = (\beta_{s,0}, \beta_{s,1}, \dots, \beta_{s,J})'$ and $\boldsymbol{B}_s = (\boldsymbol{\beta}'_{s,1}, \dots, \boldsymbol{\beta}_{s,J})'$ characterise the parameters.

Even though the universal approximation theorem holds, often neural networks with multiple hidden layers are applied as well. The reason is not very intuitive to understand. One main argument leads to the so called encoder and decoder method that may be regarded as a non-linear PCA approach. We remember that PCA in combination with linear regression can help to reduce the estimation risk and improve forecasting accuracy. However, the resulting models stays a linear model. More importantly, the approximation of the true underlying function can only get worse due to the PCA approximation. But the estimation risk reduces mainly due to a reduces amount of parameters. In a feed-forward neural network like an MLP we can achieve the same properties and reduce estimation risk. To understand this, consider an MLP with a large 1st hidden layer which can capture all the nonlinear dependencies in the data. If we add a small 2nd hidden layer the information of the large 1st hidden layer has to go through the small 2nd layer. Thus, the information has to be condensed or compressed. This encoding principle

acts similarly as PCA, as explained above. Thus, if a large input data set is considered for a 2 layer MLP, the first hidden layer is typically larger than the second one. Sometimes a decoding is added the decoder enlarges the information space again. This is particularly relevant for networks with

We can also include an extra hidden layer to the MLP in our formal notation. For instance

$$Y_{d,s} = f\left(g(\boldsymbol{\beta}_s, \boldsymbol{f}_2(\boldsymbol{g}_2(\boldsymbol{B}_{2,s}, \boldsymbol{f}_1(\boldsymbol{g}_1(\boldsymbol{B}_{1,s}, \boldsymbol{X}_{d,s})))))\right) + \varepsilon_{d,s} \tag{235}$$

defines a multilayer perceptron with 2 hidden layers. Here $\boldsymbol{f}_1$ and $\boldsymbol{g}_1$ are the activation and integration function of in the first hidden layer and $\boldsymbol{f}_2$ and $\boldsymbol{g}_2$ those ones of the second hidden layer. The parameter matrix $\boldsymbol{B}_{1,s}$ characterise the weights of the input variables to the first hidden layer, the parameter matrix $\boldsymbol{B}_{2,s}$ the weights from the first hidden layer to the second one. And finally, $\boldsymbol{\beta}_s$ are the corresponding weights for the 2 layer of neurons which give the information to the output layer.

The understanding of equation (232) is essential to understand the advantages and disadvantages of neural networks in general. As mentioned, the main advantage of neural networks is that they are able to model non-linear relationships between the input variables and the response variable, but this comes at some cost. A crucial point for the usage of neural network is the specification of the structure of the neural network. If at least one activation function is chosen non-linear then the resulting model is non-linear. Thus the estimation of the parameters requires for non-linear optimization. In neural network setting this process is usually called training of the neural net, learning of the neural net or calibration of the neural net. But from the statistical point of view it essentially means the estimation of the parameters, as an optimal set of parameters has to be determined. This optimal set of parameters has to minimize a certain predefined loss function.

For real values responses in a regression type setting such as the forecasting of electricity prices $Y_{d,s}$ the standard loss function is again the $\|\cdot\|_1$-loss (absolute loss) or $\|\cdot\|_2$-loss (squared loss) function. The quadratic loss in corresponds directly to the least square loss, which is used in the OLS estimation of linear model. It minimized the (R)MSE and predicts the mean of the underlying prediction distribution. The absolute loss minimizes the MAE and predicts the median of the distribution of the response variable.

So far we discussed the design of MLPs and also touched questions around parameter uncertainty and overfitting. However, we know already that optimization problem is in general non-linear, and typically not convex.

A standard but relatively efficient training algorithm is based on backpropagation. This is a iterative updating algorithm procedure related to online gradient descent. Here, we provide the neural net a starting solution and in every update iteration the current solution is adapted so that the loss function gets smaller. The computation of the gradient is done by backpropagation. This means that the gradient is computed by propagating the error from the output layer to the input layer, based on the the chain rule. The backpropagation algorithm is a standard algorithm for neural nets and is implemented in many software packages, such as `keras tensorflow` in `python` and `R`. The gradient is computed on a small chunk of the data the so called batch of batch size B. In OGD (see (59)) we evaluated only the last error, so the batch size was only $B = 1$. In contrast solving the OLS problem using the full in-sample data of size $B = D$ is equivalent to a batch size of D. In backpropagation algorithms, the batch size B is typically chosen between 32 and 256 due to internal computational efficiencies. The larger the batch size the more accurate is the gradient, but the more expensive is the computation. The batch size is a tuning parameter which has to be chosen by the modeler or the hyperparameter tuning algorithm. Note, that batches cycle through the data, thus first evaluating batch $1, \ldots, B$, then $B+1, \ldots, 2B$, etc. until it reaches the sample size D. This full cycle is called an epoch. In neural network settings it is popular to use multiple epochs, so the gradient is computed multiple times. This helps stabilizing the gradient and the resulting parameters. Thus, next to the batch size B the number of epochs E is often regarded as a tuning parameter. The larger the number of epochs the more accurate is the gradient, but the more expensive is the computation. Similarly as in OGD (see (59)) the backpropagation algorithm requires a learning rate parameter η also called step size. The learning rate $\eta > 0$ has to be tuned as well. Typically it takes values clearly smaller than 1. Summarizing, there are usually three relevant hyperparameters for

backpropagation based learning algorithms for neural networks. Those are the batch size B, the number of epochs E and the learning rate η.

However, it is not guaranteed that the algorithm converges to a global optimum. In fact, it is likely that the algorithm converges to a local optimum. This is a major drawback of neural nets. However, there are some techniques which can help to overcome this problem. One technique is to restart the algorithm with different starting values. Another technique is to use a committee of neural nets or ensemble of neural nets. Here, we train multiple neural nets with different starting values and average the results. This is similar to the idea of model averaging. However, the averaging is done on the model level and not on the parameter level.

The simple fact that neural nets require non-linear estimation methods, which tend to be crucially slower than linear model algorithms, is the major draw-back of the neural network modeling approach.

So far we only considered MLP with a single output node. However, it is also possible to consider MLPs with multiple output nodes. This is particularly relevant if we have multivariate outputs. For instance, if we want to forecast the electricity price for each delivery product $s \in \mathbb{S}$ we have S output nodes. This is perfectly in line with the multivariate regression setting. Remember in the OLS case, the estimation of S univariate models, one for each delivery product $s \in \mathbb{S}$ was equivalent to the estimation of a (single) S-dimensional multivariate model due to the special linear regression properties. For lasso techniques this did not hold anymore. The same holds for neural nets. However, we can also consider a single MLP with S output nodes, [Mar20, LMDSW21]. Figure (54) illustrates the corresponding graph for multiple output nodes, and the same input nodes as in Figure 22. Obviously, the same can be generalized to multiple hidden layers as well.

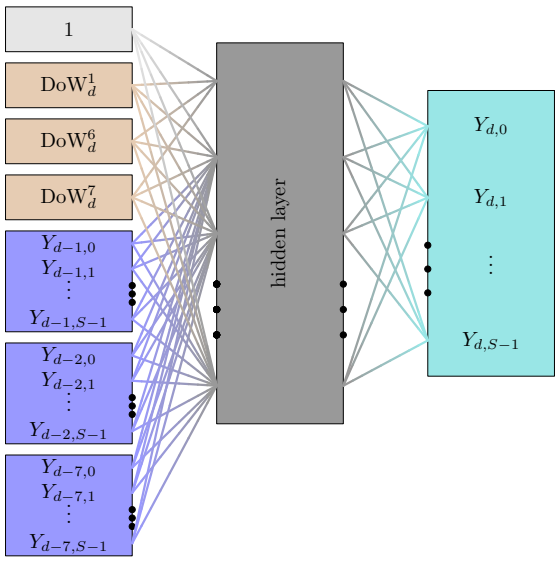


Figure 54: Network visualisation of an MLP with a single hidden layer, expert model inputs as in Figure 22 and S model outputs.

TODO summary box of typical tuning parameters for MLPs

16.2 Recurrent neural networks

References

[BFMV19] Alessandro Brusaferri, Lorenzo Fagiano, Matteo Matteucci, and Andrea Vitali. Day ahead electricity price forecast by narx model with lasso based features selection. In 2019 IEEE 17th International Conference on Industrial Informatics (INDIN), volume 1, pages 1051–1056. IEEE, 2019.

[CCS13] René Carmona, Michael Coulon, and Daniel Schwarz. Electricity price modeling and asset valuation: a multi-fuel structural approach. Mathematics and Financial Economics, 7(2):167–202, 2013.

[Die15] Francis X Diebold. Comparing predictive accuracy, twenty years later: A personal perspective on the use and abuse of diebold–mariano tests. Journal of Business & Economic Statistics, 33(1):1–1, 2015.

[EHJT04] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. The Annals of statistics, 32(2):407–499, 2004.

[Fan97] Jianqing Fan. Comments on wavelets in statistics: A review by a. antoniadis. Statistical Methods & Applications, 6(2):131–138, 1997.

[FHHT07] Jerome Friedman, Trevor Hastie, Holger Höfling, and Robert Tibshirani. Pathwise coordinate optimization. The annals of applied statistics, 1(2):302–332, 2007.

[FHT10] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. Journal of statistical software, 33(1):1, 2010.

[FL01] Jianqing Fan and Runze Li. Variable selection via nonconcave penalized likelihood and its oracle properties. Journal of the American statistical Association, 96(456):1348–1360, 2001.

[FM20] Carlo Fezzi and Luca Mosetti. Size matters: Estimation sample length and electricity price forecasting accuracy. The Energy Journal, 41(4), 2020.

[GF10] Frauke Günther and Stefan Fritsch. neuralnet: Training of neural networks. The R journal, 2(1):30–38, 2010.

[GGN16] Pierre Gaillard, Yannig Goude, and Raphaël Nedellec. Additive models and robust aggregation for gefcom2014 probabilistic electric load and electricity price forecasting. International Journal of forecasting, 32(3):1038–1050, 2016.

[GXT+18] Wei Guo, Tao Xu, Keming Tang, Jianjiang Yu, and Shuangshuang Chen. Online sequential extreme learning machine with generalized regularization and adaptive forgetting factor for time-varying system prediction. Mathematical Problems in Engineering, 2018, 2018.

[HMW18] Katarzyna Hubicka, Grzegorz Marcjasz, and Rafał Weron. A note on averaging day-ahead electricity price forecasts across calibration windows. IEEE Transactions on Sustainable Energy, 10(1):321–323, 2018.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. Neural networks, 2(5):359–366, 1989.

[KGMF12] Dogan Keles, Massimo Genoese, Dominik Möst, and Wolf Fichtner. Comparison of extended mean-reversion and time series models for electricity spot price simulation considering negative prices. Energy Economics, 34(4):1012–1032, 2012.

[LM15] Johannes Lederer and Christian Müller. Don't fall for tuning parameters: tuning-free variable selection in high dimensions with the trex. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 29, 2015.

[LMDSW21] Jesus Lago, Grzegorz Marcjasz, Bart De Schutter, and Rafał Weron. Forecasting day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark. Applied Energy, 293:116983, 2021.

[LNHW15] Bidong Liu, Jakub Nowotarski, Tao Hong, and Rafał Weron. Probabilistic load forecasting via quantile regression averaging on sister forecasts. IEEE Transactions on Smart Grid, 8(2):730–737, 2015.

[LTTT14] Richard Lockhart, Jonathan Taylor, Ryan J Tibshirani, and Robert Tibshirani. A significance test for the lasso. Annals of statistics, 42(2):413, 2014.

[Mar20] Grzegorz Marcjasz. Forecasting electricity prices using deep neural networks: A robust hyper-parameter selection scheme. Energies, 13(18):4605, 2020.

[MRNF21] D Stasinopoulos Mikis, A Rigby Robert, Georgikopoulos Nikolaos, and De Bastiani Fernanda. Principal component regression in gamlss applied to greek–german government bond yield spreads. Statistical Modelling, page 1471082X211022980, 2021.

[MTW06] Adam Misiorek, Stefan Trueck, and Rafal Weron. Point and interval forecasting of spot electricity prices: Linear vs. non-linear time series models. Studies in Nonlinear Dynamics & Econometrics, 10(3), 2006.

[NW16] Jakub Nowotarski and Rafał Weron. On the importance of the long-term seasonal component in day-ahead electricity price forecasting. Energy Economics, 2016.

[NZ20a] Michał Narajewski and Florian Ziel. Econometric modelling and forecasting of intraday electricity prices. Journal of Commodity Markets, 19:100107, 2020.

[NZ20b] Michał Narajewski and Florian Ziel. Ensemble forecasting for intraday electricity prices: Simulating trajectories. Applied Energy, 279:115801, 2020.

[Ser11] Francesco Serinaldi. Distributional modeling and short-term forecasting of electricity prices by generalized additive models for location, scale and shape. Energy Economics, 33(6):1216–1226, 2011.

[UNW16] Bartosz Uniejewski, Jakub Nowotarski, and Rafał Weron. Automated variable selection and shrinkage for day-ahead electricity price forecasting. Energies, 9(8):621, 2016.

[UW18] Bartosz Uniejewski and Rafał Weron. Efficient forecasting of electricity spot prices with expert and lasso models. Energies, 11(8):2039, 2018.

[UWZ17] Bartosz Uniejewski, Rafał Weron, and Florian Ziel. Variance stabilizing transformations for electricity spot price forecasting. IEEE Transactions on Power Systems, 33(2):2219–2229, 2017.

[Wer14] Rafał Weron. Electricity price forecasting: A review of the state-of-the-art with a look into the future. International Journal of Forecasting, 30(4):1030–1081, 2014.

[Woo03] Simon N Wood. Thin plate regression splines. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 65(1):95–114, 2003.

[Z+10] Cun-Hui Zhang et al. Nearly unbiased variable selection under minimax concave penalty. The Annals of statistics, 38(2):894–942, 2010.

[ZCA16] Florian Ziel, Carsten Croonenbroeck, and Daniel Ambach. Forecasting wind power–modeling periodic and non-linear effects under conditional heteroscedasticity. Applied Energy, 177:285–297, 2016.

[Zie16] Florian Ziel. Forecasting electricity spot prices using lasso: On capturing the autoregressive intraday structure. IEEE Transactions on Power Systems, 31(6):4977–4987, 2016.

[Zou06] Hui Zou. The adaptive lasso and its oracle properties. Journal of the American statistical association, 101(476):1418–1429, 2006.

[ZSH15] Florian Ziel, Rick Steinert, and Sven Husmann. Efficient modeling and forecasting of electricity spot prices. Energy Economics, 47:98–111, 2015.

17 Test data and selected results

Here, we illustrate the test data used in the manuscript, and the performance of selected models in the forecasting study.

17.1 Evaluation results: validation data

	MAE	skill	RMSE	skill
expert	7.70	78.81	11.86	75.78
expert_mix	7.68	78.61	11.83	75.59
expert_mv	7.94	81.27	12.11	77.38
naive	9.77	100.00	15.65	100.00

Table 1: MAE and RMSE of selected models at the validation set from 2018-10-04 to 2020-10-02.

	MAE	skill	RMSE	skill
expert_neff=00002	17.25	172.50	32.38	206.90
expert_neff=00004	12.06	120.60	21.85	139.62
expert_neff=00008	9.84	98.40	16.62	106.20
expert_neff=00016	8.62	86.20	13.79	88.12
expert_neff=00032	8.06	80.60	12.66	80.89
expert_neff=00064	7.80	78.00	12.20	77.96
expert_neff=00128	7.68	76.80	11.98	76.55
expert_neff=002	17.25	172.50	32.38	206.90
expert_neff=00256	7.63	76.30	11.86	75.78
expert_neff=004	12.06	120.60	21.85	139.62
expert_neff=00512	7.63	76.30	11.83	75.59
expert_neff=008	9.84	98.40	16.62	106.20
expert_neff=01024	7.64	76.40	11.81	75.46
expert_neff=016	8.62	86.20	13.79	88.12
expert_neff=02048	7.64	76.40	11.81	75.46
expert_neff=024	7.64	76.40	11.81	75.46
expert_neff=032	8.06	80.60	12.66	80.89
expert_neff=048	7.64	76.40	11.81	75.46
expert_neff=064	7.80	78.00	12.20	77.96
expert_neff=128	7.68	76.80	11.98	76.55
expert_neff=256	7.63	76.30	11.86	75.78
expert_neff=512	7.63	76.30	11.83	75.59
expert_neff=Inf	7.66	76.60	11.81	75.46
naive	10.00	100.00	15.65	100.00

Table 2: MAE and RMSE of selected models at the validation set from 2018-10-04 to 2020-10-02.

	MAE	skill	RMSE	skill
expert_mlrmbo	7.55	75.50	11.64	74.38
naive	10.00	100.00	15.65	100.00

Table 3: MAE and RMSE of selected models at the validation set from 2018-10-04 to 2020-10-02.

	MAE	skill	RMSE	skill
naive	9.77	100.00	15.65	100.00
sarimax	7.45	76.25	11.51	73.55
sarmax	7.74	79.22	11.86	75.78

Table 4: MAE and RMSE of selected models at the validation set from 2018-10-04 to 2020-10-02.

	MAE	skill	RMSE	skill
expert.last	6.62	67.76	10.61	67.80
naive	9.77	100.00	15.65	100.00
sarimax.last	6.35	64.99	10.23	65.37

Table 5: MAE and RMSE of selected models at the validation set from 2018-10-04 to 2020-10-02.

	MAE	skill	RMSE	skill
expertadv_online	5.17	51.70	8.19	52.33
expertadv_online_diff	5.67	56.70	9.01	57.57
expertadv_online_fullfuels	5.14	51.40	8.16	52.14
expertadv_online_fullfuels_diff	5.73	57.30	9.07	57.96
expertadv_online_nofuels	5.80	58.00	9.12	58.27
expertadv_online_nofuels_diff	5.66	56.60	9.00	57.51
naive	10.00	100.00	15.65	100.00

Table 6: MAE and RMSE of selected models at the validation set from 2018-10-04 to 2020-10-02.

	MAE	skill	RMSE	skill
trafo_expert_asinh_rob	7.48	76.56	11.73	74.95
trafo_expert_asinh_std	7.46	76.36	11.69	74.70
trafo_expert_atan_rob	8.01	81.99	40.33	257.70
trafo_expert_atan_std	7.52	76.97	11.88	75.91
trafo_expert_boxcox_rob	7.53	77.07	11.73	74.95
trafo_expert_boxcox_std	7.53	77.07	11.73	74.95
trafo_expert_glogistic_rob	9.90	101.33	20.28	129.58
trafo_expert_glogistic_std	9.02	92.32	18.95	121.09
trafo_expert_id_rob	7.70	78.81	11.86	75.78
trafo_expert_id_std	7.70	78.81	11.86	75.78
trafo_expert_Ksigma_rob	7.51	76.87	11.69	74.70
trafo_expert_Ksigma_std	7.53	77.07	11.66	74.50
trafo_expert_Ksigmalog_rob	7.56	77.38	11.68	74.63
trafo_expert_Ksigmalog_std	7.60	77.79	11.72	74.89
trafo_expert_logistic_rob	7.58	77.58	12.97	82.88
trafo_expert_logistic_std	7.43	76.05	11.68	74.63
trafo_expert_mlog_rob	7.50	76.77	11.71	74.82
trafo_expert_mlog_std	7.50	76.77	11.70	74.76
trafo_expert_poly_rob	7.74	79.22	12.04	76.93
trafo_expert_poly_std	7.66	78.40	11.96	76.42
trafo_expert_probit_rob	7.60	77.79	12.13	77.51
trafo_expert_probit_std	7.49	76.66	11.92	76.17
trafo_expert_tanh_rob	7.78	79.63	12.38	79.11
trafo_expert_tanh_std	7.60	77.79	12.10	77.32
trafo_expert_tprobit_rob	7.73	79.12	12.40	79.23
trafo_expert_tprobit_std	7.52	76.97	11.94	76.29
trafo_naive	9.77	100.00	15.65	100.00

Table 7: MAE and RMSE of selected models at the validation set from 2018-10-04 to 2020-10-02.

	MAE	skill	RMSE	skill
expert-adv	5.02	51.38	7.51	47.99
expert-adv-lassoBIC	4.90	50.15	7.48	47.80
expert-adv-mlassoBIC	5.12	52.41	7.65	48.88
naive	9.77	100.00	15.65	100.00

Table 8: MAE and RMSE of selected models at the validation set from 2018-10-04 to 2020-10-02.

17.2 Graphs of considered training and test data

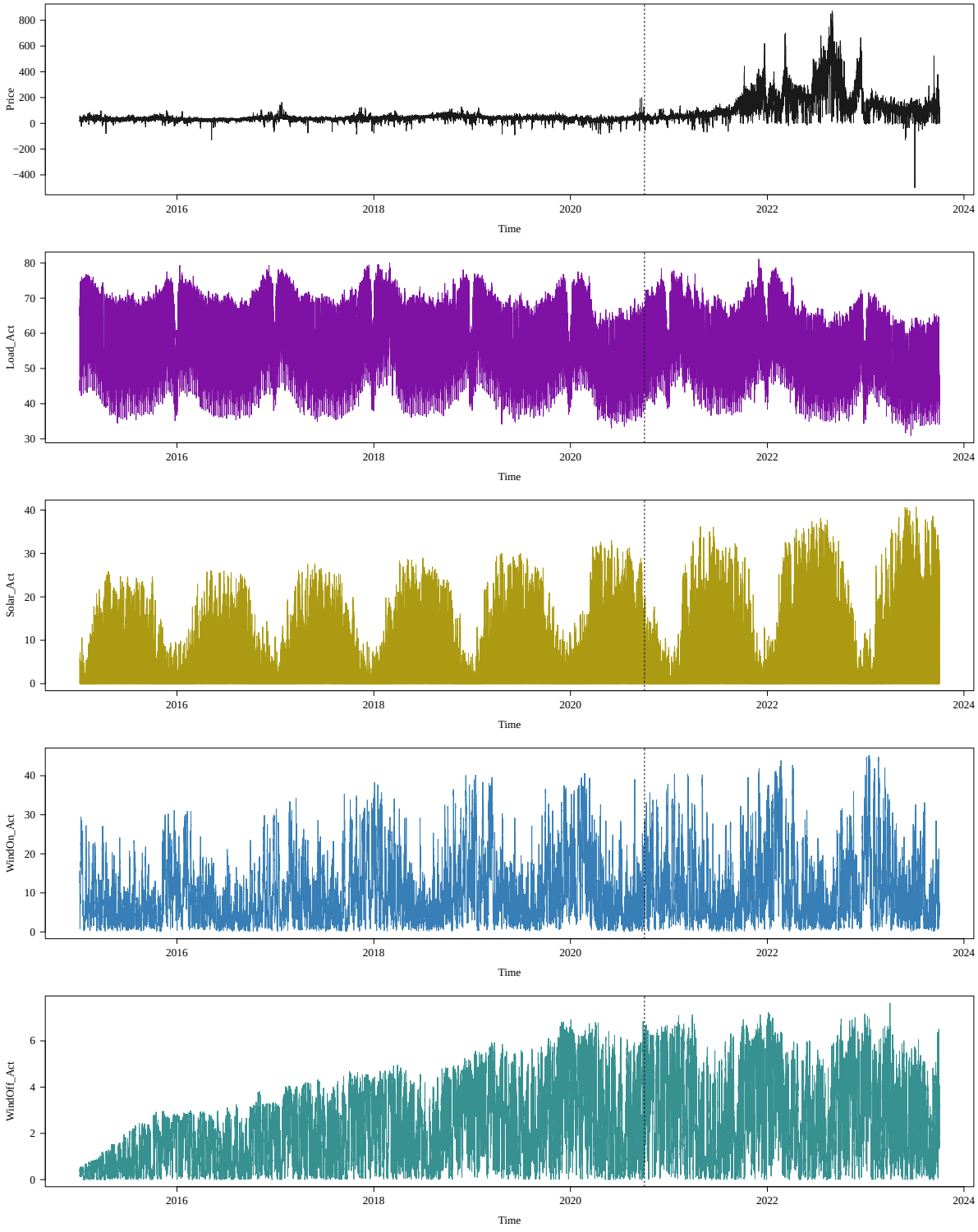
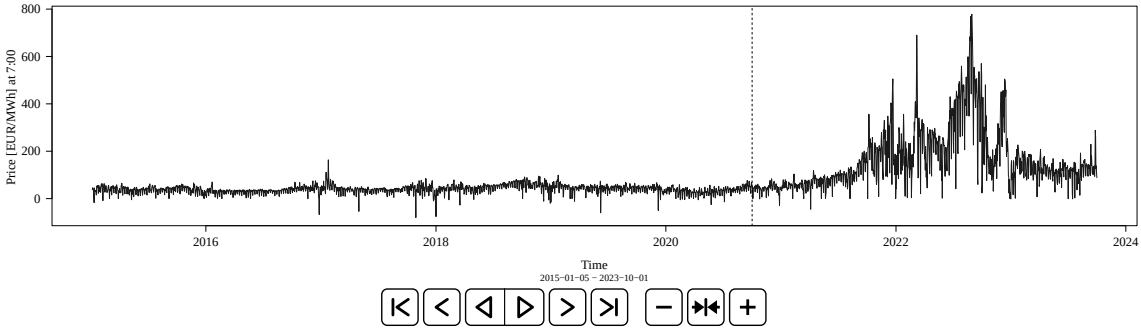
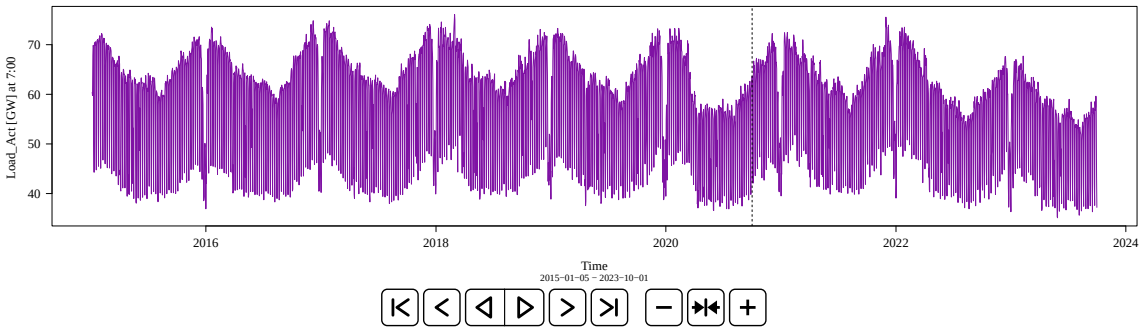


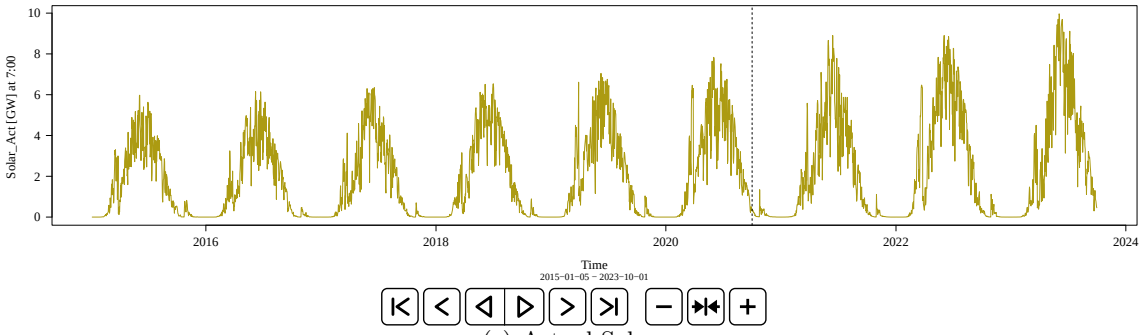
Figure 55: Time series plot of electricity price and considered fundamentals (load, solar, wind onshore and offshore) on training and test data.



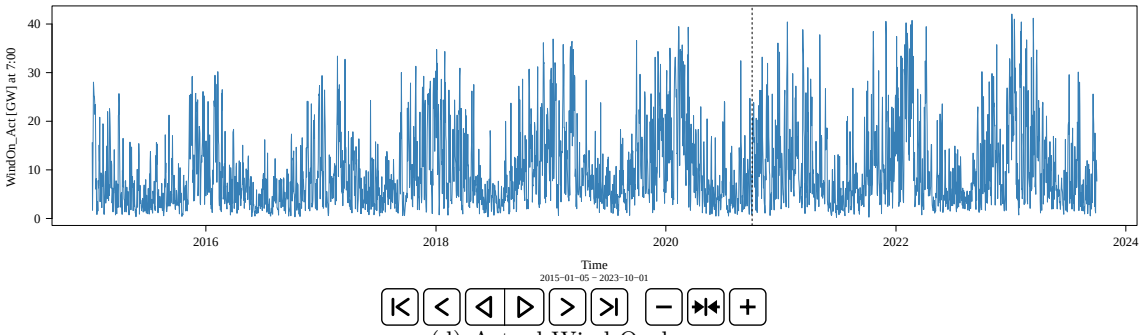
(a) Prices



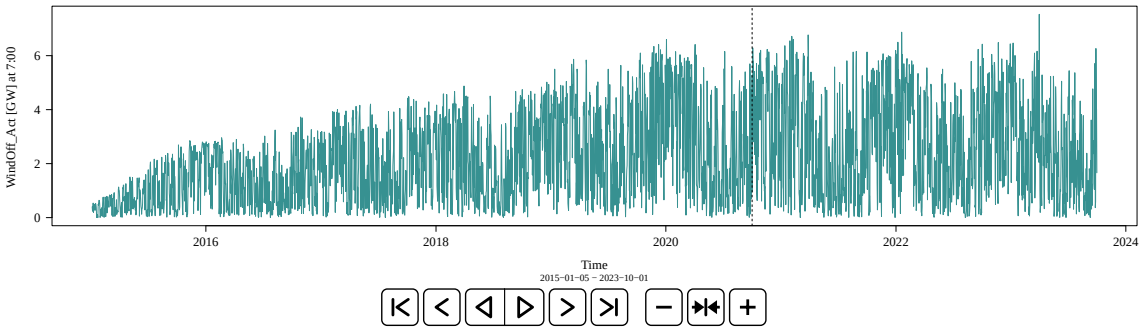
(b) Actual Load



(c) Actual Solar



(d) Actual Wind Onshore



(e) Actual Wind Offshore

Figure 56: Time series plot of electricity price and considered fundamentals (load, solar, wind onshore and offshore) for each $s \in \mathbb{S}$ on training and test data.

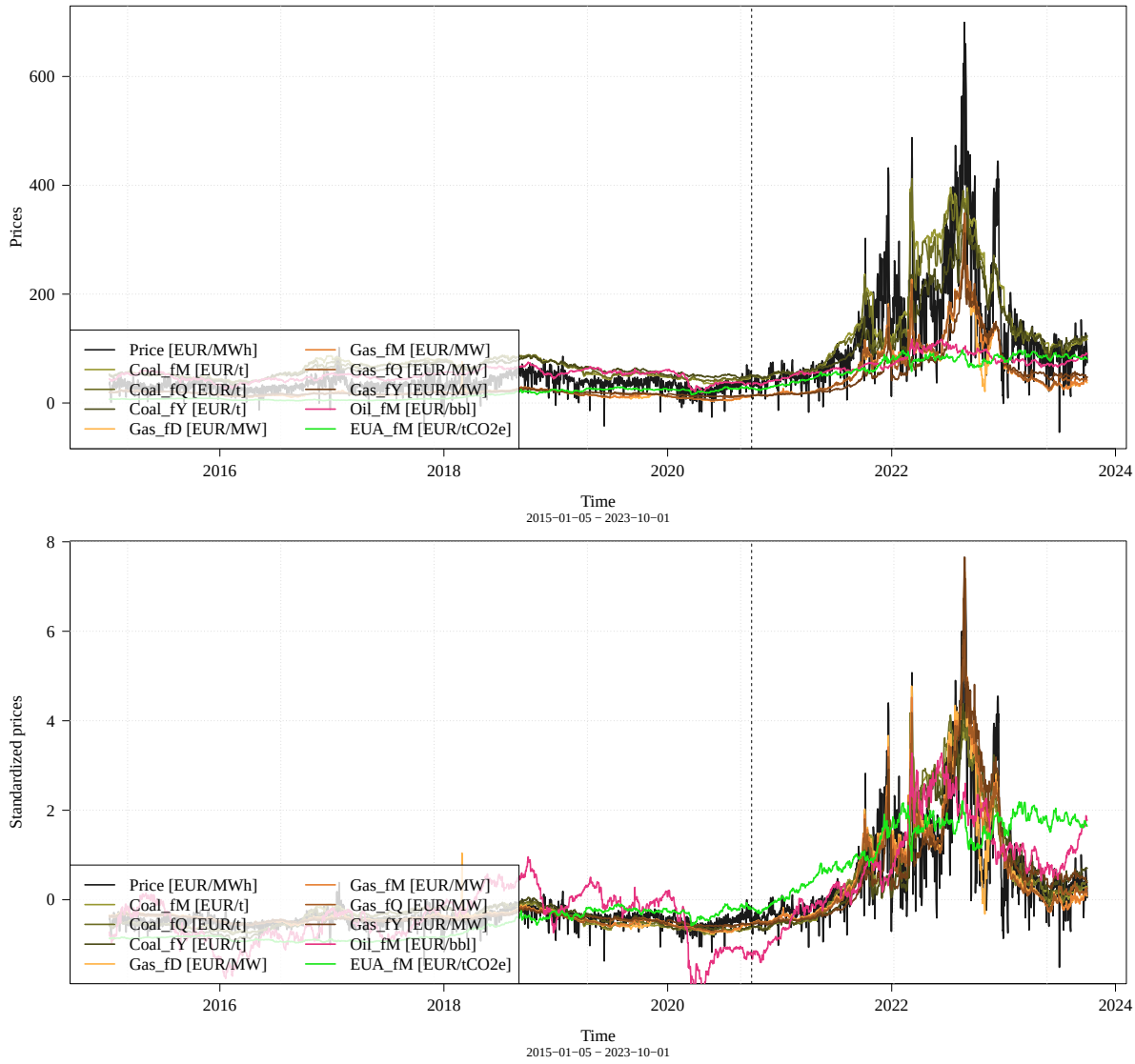


Figure 57: Correlation forecasts of day-ahead forecast and actual fundamentals and the electricity price on training and test data.

17.3 Evaluation results: test data

Do you really want to see the test data?

	MAE	skill	RMSE	skill
expert	31.83	83.39	50.05	80.08
expert_mix	31.76	83.21	49.97	79.95
expert_mv	32.43	84.96	50.87	81.39
naive	38.17	100.00	62.50	100.00

Table 9: MAE and RMSE of selected models at the test set from 2020-10-03 to 2023-10-02.

	MAE	skill	RMSE	skill
expert_neff=00002	64.81	170.55	122.98	196.77
expert_neff=00004	43.66	114.89	74.55	119.28
expert_neff=00008	36.64	96.42	58.74	93.98
expert_neff=00016	33.32	87.68	52.56	84.10
expert_neff=00032	31.77	83.61	50.07	80.11
expert_neff=00064	31.31	82.39	49.30	78.88
expert_neff=00128	31.32	82.42	49.26	78.82
expert_neff=002	64.81	170.55	122.98	196.77
expert_neff=00256	31.40	82.63	49.49	79.18
expert_neff=004	43.66	114.89	74.55	119.28
expert_neff=00512	31.47	82.82	49.82	79.71
expert_neff=008	36.64	96.42	58.74	93.98
expert_neff=01024	31.67	83.34	50.19	80.30
expert_neff=016	33.32	87.68	52.56	84.10
expert_neff=02048	31.89	83.92	50.46	80.74
expert_neff=024	31.67	83.34	50.19	80.30
expert_neff=032	31.77	83.61	50.07	80.11
expert_neff=048	31.89	83.92	50.46	80.74
expert_neff=064	31.31	82.39	49.30	78.88
expert_neff=128	31.32	82.42	49.26	78.82
expert_neff=256	31.40	82.63	49.49	79.18
expert_neff=512	31.47	82.82	49.82	79.71
expert_neff=Inf	32.19	84.71	50.76	81.22
naive	38.00	100.00	62.50	100.00

Table 10: MAE and RMSE of selected models at the test set from 2020-10-03 to 2023-10-02.

	MAE	skill	RMSE	skill
expert_mlrmbo	31.66	83.32	50.17	80.27
naive	38.00	100.00	62.50	100.00

Table 11: MAE and RMSE of selected models at the test set from 2020-10-03 to 2023-10-02.

	MAE	skill	RMSE	skill
naive	38.17	100.00	62.50	100.00
sarimax	31.50	82.53	49.58	79.33
sarmax	33.11	86.74	53.99	86.38

Table 12: MAE and RMSE of selected models at the test set from 2020-10-03 to 2023-10-02.

	MAE	skill	RMSE	skill
expert.last	26.53	69.50	42.77	68.43
naive	38.17	100.00	62.50	100.00
sarimax.last	25.39	66.52	40.68	65.09

Table 13: MAE and RMSE of selected models at the test set from 2020-10-03 to 2023-10-02.

	MAE	skill	RMSE	skill
expertadv_online	22.32	58.74	36.45	58.32
expertadv_online_diff	23.14	60.89	38.04	60.86
expertadv_online_fullfuels	22.23	58.50	36.42	58.27
expertadv_online_fullfuels_diff	23.59	62.08	39.02	62.43
expertadv_online_nofuels	23.51	61.87	38.07	60.91
expertadv_online_nofuels_diff	22.92	60.32	37.79	60.46
naive	38.00	100.00	62.50	100.00

Table 14: MAE and RMSE of selected models at the test set from 2020-10-03 to 2023-10-02.

	MAE	skill	RMSE	skill
trafo_expert_asinh_rob	33.12	86.77	51.30	82.08
trafo_expert_asinh_std	32.52	85.20	50.49	80.78
trafo_expert_atan_rob	74.28	194.60	1102.05	1763.28
trafo_expert_atan_std	38.95	102.04	117.78	188.45
trafo_expert_boxcox_rob	31.94	83.68	49.95	79.92
trafo_expert_boxcox_std	32.02	83.89	50.04	80.06
trafo_expert_glogistic_rob	48.51	127.09	105.84	169.34
trafo_expert_glogistic_std	35.44	92.85	66.03	105.65
trafo_expert_id_rob	31.83	83.39	50.05	80.08
trafo_expert_id_std	31.83	83.39	50.05	80.08
trafo_expert_Ksigma_rob	45.08	118.10	76.85	122.96
trafo_expert_Ksigma_std	34.41	90.15	54.96	87.94
trafo_expert_Ksigmalog_rob	33.03	86.53	51.34	82.14
trafo_expert_Ksigmalog_std	31.93	83.65	50.07	80.11
trafo_expert_logistic_rob	57.42	150.43	150.02	240.03
trafo_expert_logistic_std	39.55	103.62	101.40	162.24
trafo_expert_mlog_rob	32.29	84.60	50.27	80.43
trafo_expert_mlog_std	32.18	84.31	50.16	80.26
trafo_expert_poly_rob	34.50	90.39	53.00	84.80
trafo_expert_poly_std	34.39	90.10	52.51	84.02
trafo_expert_probit_rob	52.02	136.29	87.83	140.53
trafo_expert_probit_std	40.27	105.50	65.12	104.19
trafo_expert_tanh_rob	53.53	140.24	89.76	143.62
trafo_expert_tanh_std	42.23	110.64	68.19	109.10
trafo_expert_tprobit_rob	46.35	121.43	76.85	122.96
trafo_expert_tprobit_std	37.80	99.03	59.51	95.22
trafo_naive	38.17	100.00	62.50	100.00

Table 15: MAE and RMSE of selected models at the test set from 2020-10-03 to 2023-10-02.

	MAE	skill	RMSE	skill
expert-adv	22.51	58.97	35.55	56.88
expert-adv-lassoBIC	22.45	58.82	35.63	57.01
expert-adv-mlassoBIC	23.15	60.65	36.10	57.76
naive	38.17	100.00	62.50	100.00

Table 16: MAE and RMSE of selected models at the test set from 2020-10-03 to 2023-10-02.

Tutorials in R

R Tutorials in R

R.1 Read the data and data preparation

After studying the subsequent subsections which cover the content from Section 3, you should be able to solve the Tasks 3.5.

Before we start reading the electricity price data in R, we change the language settings. As we work in English, we change the R output system language to English. This can be done by evaluating

```
Sys.setlocale(locale = "en_US.utf8") # English US Linux
Sys.setlocale(locale = "en_US") # English US Mac
Sys.setlocale(locale = "English_United States.1252") # English US Windows
```

in the console. If you are not sure what is your operating system, just run all commands. We change the language settings to guarantee the same output concerning the times and dates, e.g. the weekdays are named `Monday`, `Tuesday`, etc.

Now, we want to read the data to R and prepare the data for modelling purpose. There is no universal standard of the given data format. This has always to be checked by the modeller. However, in this tutorial every time series data is given with a time label. This time label gives always the time in the universal time code (UTC) which corresponds to the UK winter time.

During this course we will mainly work for illustration purpose with the German EPEX day-ahead prices. First, we read the provided file *Germany.csv*[24] into R using the command

```
data <- read.csv("../data/DE_utc.csv") # read data
head(data) # check first entries / data format
str(data) # display the structure of the data
```

and observe the structure of the provided data. We see in the first column the time label, in the next ones the EPEX price of Germany/Luxembourg, the corresponding trading volumes, day-ahead forecasts of power generation (solar, wind onshore and offshore) and of load, the actual values of the mentioned generation and load, and finally the prices of the European Union emission allowance and the fuels: coal, oil and natural gas.

We then define our test set as being the last 3 years of the data and remove it from the data set. We will be using this data only at the very end of the tutorial to test the goodness of our models after we have chosen them. We can not look at this data to train our models, since our models should not be able to adapt to this data as the future is not known in real life.

```
N_test <- 3 * 365 * 24
idx <- 1:(nrow(data) - N_test)
data <- data[idx, ]
```

In this tutorial, we work for illustration purpose mainly with the German EPEX prices which is given in the 2nd column.

Let us store it as `price`, and let us extract the name of the column in order to use it later on for plotting.

```
id_select <- 2 # indicator for EPEX DE/AT column
price <- data[, id_select] # uses id_select data
price <- data[, "Price"] # alternatively select by name
```

The first column provides the time in UTC. Using the `strptime` command we can convert a character string to time-date format. Here, we require conversion rules to interpret time-date strings (see `?strptime`):

```
# create time vector
time_utc <- as.POSIXct(
    strptime(
        data[, 1],
        format = "%Y-%m-%d %H:%M:%S",
        tz = "UTC"
    )
) # or: data[, "DateTime"] |> lubridate::as_datetime(tz = "UTC")
```

[24] eventually we either have to change the working folder before (using `setwd()`) or copy the file into the working folder (see `getwd()`)

Note that `strptime` converts a character string into the time class `POSIXlt`. However, there is another time class `POSIXct` which is more convenient for us to work with, therefore we apply the function `as.POSIXct`. At this point the difference between `POSIXct` and `POSIXlt` is not important for us, for more details see `?POSIXlt` and `?as.POSIXct`.

We can easily create a time series plot:

```
# simple time series plot
plot(time_utc, price,
    type = "l",
    ylab = "Price [EUR/MWh]",
    xlab = "Time"
)
```

Remember that this data (so the vector `price`) is not clock-change adjusted.

Now, we have to get deeper into the problem of time formation. Even though this seems to be a bit nasty, every modeler and forecaster that deals with intraday data has to face this problem. So lets have a closer look into the structure of the `POSIXct` class. It is a class that essentially codes time on a second precision in UTC. Every time point is represented by an integer number, that increases by 1 for every second, starting counting at 1st January 1970 00:00:00 UTC. For some mathematics it is suitable to work with these representing numbers instead of the more complex structured `POSIXlt` class, therefore we save it under `time_numeric`. Furthermore, we convert the time `time_numeric` into the local time using `as.POSIXct`. Note that most of the European countries use Central European Time (CET), as time zone such as Germany and Austria. However, UK, Ireland and Portugal use Western European Time (WET) and the European countries Finland, the Baltic States, Ukraine, Moldova, Romania, Bulgaria, Greece and Cyprus use Eastern European Time (EET). We convert the time using:

```
time_numeric <- as.numeric(time_utc) # time as numeric
head(time_numeric)
local_time_zone <- "CET" # local time zone abbrv.
time_lt <- as.POSIXct(time_numeric, origin = "1970-01-01", tz = local_time_zone)
# or data[, "DateTime"] |> as_datetime(tz = "CET")
head(time_lt)
head(time_utc)
```

When we model our prices, we want to make use of the regular seasonal pattern in our data, especially the fact that we observe S prices every day. Due to the clock change we are facing a problem here. Therefore, we introduced in the section 3 the object $Y_{d,s}$ which gives the clock-change adjusted price on day d and period s. In order to make use of it, we have to consider a clock-change transformation for the prices and the time. First, we consider the transformation for the time. Initially we compute S given the data[25].

```
S <- 24 * 60 * 60 / as.numeric(
    names(
        which.max(
            table(
                diff(time_numeric)
            )
        )
    )
)
```

Now we want to introduce a time vector that every day has 24 hours. This is the case for `time_utc`, but not for `time_lt` (here some days have 23 or 25 hours). We can check it running the following code.

```
# check on which days the number of products is other than 24
only_days_utc <- as.Date(time_utc, tz = "UTC")
table(only_days_utc)[(table(only_days_utc) != 24)]
only_days_lt <- as.Date(time_lt, tz = "CET")
table(only_days_lt)[(table(only_days_lt) != 24)]
```

Unfortunately, `time_utc` does not take into account the local time zone. Therefore, we will introduce a new 'fake' local time-zone vector called `time_s`, matching our requirement to have

[25] Alternatively `S` could be provided manually by `S<-24`.

S values each day, starting every day at 0:00. This time vector will be stored as UTC, but in fact it is not(!) UTC. Given the time vector `time_s`, we will also store the corresponding dates as `dates_s`:

```
# Save the start and end-time
start_end_time <- strptime(
  format(c(time_lt[1], time_lt[length(time_lt)]), "%Y-%m-%d %H:%M:%S",
  tz = "CET"
  ),
  format = "%Y-%m-%d %H:%M:%S",
  tz = "UTC"
)
print(start_end_time_fake)

time_fake_numeric <- seq(
    from = as.numeric(start_end_time_fake[1]),
    to = as.numeric(start_end_time_fake[length(start_end_time_fake)]),
    by = 24 * 60 * 60 / S
) # 'fake' local time
time_fake <- as.POSIXct(time_fake_numeric, origin = "1970-01-01", tz = "UTC")
dates <- unique(as.Date(time_fake)) # , tz=local_time_zone))
```

We see that the transformation looks already relative complex. For the transformation of the price vector this is even more complex. Our transformation rule interpolates the missing hour in March linearly and it averages the double entries in October. This can be quite complicated, as this transformation is influenced by the time zone and S. Therefore, we simply import a function `DST.trafo` from the provided `DST.trafo.R` file which must be copied into the working folder. The function `DST.trafo` requires a vector (or matrix) of data in UTC format as argument `X`, a UTC time vector as argument `Xtime` and a time zone vector as argument `Xtz` (where WET/CET/EET are supported). For more information, please explore the `DST.trafo.R` file. The output will be an array, with the data/period format as introduced above. The first dimension will be the length of the number of supplied dates, the second will be S and the third dimension will be the number of provided time series (one in our case). As we only proceed with one time series of electricity prices, we save the 2-dimensional array of prices as `price_DST`:

```
source("../DST.trafo.R")

price_DST_arr <- DST.trafo(
    X = price,
    Xtime = time_utc,
    Xtz = local_time_zone
) # may take a while

dim(price_DST_arr)

price_DST <- price_DST_arr[, , 1]
dim(price_DST)
```

Similarly as above, we can create a time series plot

```
price_ts <- as.numeric(t(price_DST))
plot(time_fake, price_ts,
type = "l",
ylab = "Electricity Price [EUR/MWh]",
xlab = "Time",
las = 1,
cex.lab = 1.3,
cex.axis = 1.3
)
```

which is almost the same as this one created above using `time_utc` and `price`, but clock-change adjusted.

In practice, of course the amount of available data is limited. When we start in the next section with simple forecast models, we will only use a restricted set of available data. So we suppose to have the first $D = 2 \times 365 = 730$ days (usually 2 years) of EPEX.DE+AT data available. Now, the target is to do a forecast for the next day $D + 1$ or, in the given situation,

day $D + 1 = 2 \times 365 + 1 = 731$. Then, we can easily create an index set `index` that covers the available range of days.

```
D <- 365 * 2 # 2 years of price data
index <- 1:D

# Inspect available data
head(price_DST[index, ])

plot(dates[index],
price_DST[index, 1], # col index 1 corresponds to hour 0
type = "o", ylab = "Electricity Price at 00:00 [EUR/MWh]", xlab = "Time"
)
```

The latter plot only shows the prices for the hour 0:00-1:00.

R.2 Naive and expert-type models

R.2.1 Naive models

In this section, we want to write functions that take as input the available/provided data, in our case `price_DST[index,]` with the provided index set `index`. As output, we expect the forecast of the next days values. So within the function we will use the object `Y` as representative for provided data `price_DST[index,]` with the same structure:

```
Y <- price_DST[index, ]
## then e.g.
Y[nrow(Y), ]
# gives the value of the last day.
```

The last line gives the last `S` (or S) values of `Y`. This can be directly used to define a forecasting function for the **naive-S** model (1):

```
forecast_naive_s <- function(Y) {
  forecast <- Y[nrow(Y) - 1 + 1, ] ## return the last row
  return(forecast)
}

forecast_naive_s(
Y = price_DST[index, ]
) # run the forecast

tail(price_DST[index, ], ) # inspect
tail(price_DST[index, ], 1) # inspect
```

`forecast_naive_s` is a function that takes the input `Y`. The last line specifies by default the output object of any function, in our case `Y[dim(Y)[1],]` which are the last (observed) `S` values in `Y`. If we apply `forecast_naive_s` to `price_DST[index,]` we directly get the last `S` values of `price_DST[index,]`.

Similarly we can define the function for the `naive7S` model (2) which returns the prices from one week before the forecasted day as the price forecast:

```
forecast_naive7_s <- function(Y) {
  ## return the 6th last row, because we need -7 from the perspective of t+1.
  ## from t perspective this is just -6
  forecast <- Y[nrow(Y) - 7 + 1, ]
  return(forecast)
}
```

We can compare it with the `naiveS` model:

```
tail(price_DST[index, ], 7) # inspect
forecast_naive_s(price_DST[index, ]) # inspect
forecast_naive7_s(price_DST[index, ]) # inspect
```

The implementation of the slightly more sophisticated `naive` model (3) requires a little more effort. Here we have to evaluate the weekday of the day d that we want to forecast. Therefore, we need some time/date information on the input matrix `Y`. Within the forecasting function that

we will create, we will use the object `days` as representative for the date vector that corresponds to `Y`. When we call the function `days`, we will get the value `dates_s[index]` and as before, `Y` will be `price_DST[index,]`. For evaluating the weekday information of the provided dates `days`, we use the function `format` (see `?strptime` or alternatively `?format.POSIXct`). The `%a` specifier returns the weekday shortcut (Mon, Tue, ..., Sun) of the input vector. So if we apply this to our full dates vector, we receive:

```
  days <- dates[index]
  format(days, "%a")
  # or lubridate::wday(days, label = TRUE)
```

However, precisely we do not need to evaluate all dates in the `naive` model (3), but only the date of the day to forecast. This we can do so using:

```
  format(days[length(days)] + 1, "%a") # or lubridate::wday(days[length(days)] +1,
      label = TRUE)
```

Now we have all ingredients to define the `forecast.naive` function for model (3):

```
forecast_naive <- function(Y, days) {
  ## days = dates[index]
  # Get weekday name of the day to forecast
  weekday <- lubridate::wday(days[length(days)] + 1, label = TRUE)
  if (weekday %in% c("Mon", "Sat", "Sun")) { # if Mo, Sat..
    forecast <- Y[dim(Y)[1] - 7 + 1, ] # ... take values a week ago
  } else {
    forecast <- Y[dim(Y)[1] - 1 + 1, ] # .. else last day values
  }
  output <- list("forecasts" = forecast, "coefficients" = NULL)
  return(output)
}

forecast_naive(price_DST[index, ], days = dates[index])
```

Note that within the `if` command we use the function `%in%` which checks if the first object is contained in the second one which is in our case `c("Mon", "Sat", "Sun")`, so the days representing character strings.

R.2.2 Expert type models

Now we want to implement simple forecasting methods that are based on the estimation of a statistical model. The modelling approach that we consider most of the time is based on having S separate models, one model for each period of the day. Thus, we have to estimate S models and compute S forecasts. However, for illustration purpose we start with a single model for a single period `s` which is represented by $s \in \mathbb{S}$ in our notation. We simply choose `s = 6`, this matches the 6th period with the delivery of electricity between 5:00 and 6:00.

```
# Choose an hour of a day
s <- 6 # id for period
```

As mentioned before, we will first consider the linear model (15) that contains only day-of-the-week effects. Therefore, we have to define the weekday-dummies, so vectors that are always 0 except for the corresponding weekdays where they are 1. Moreover, we used the index set `index` before to specify the provided data range. Now, it is suitable to define an extended index set that covers the data range as `index` before, but also the day that we want to forecast. This is useful as for the evaluation we have to specify the week-day dummies not only for all the provided data, but also for the forecasted day. So we define an extended index set and compute the corresponding extended set of dates:

```
# Define extended days vector: including the day to forecast
days_ext <- c(dates[index], dates[index[length(index)]] + 1)
```

Now, we want to define the weekday dummies. We have already used the `%a` argument of the format function for dates. In this exercise, we consider a slightly more sophisticated approach using the `%w` argument (see `?strptime`). It returns the weekday of a time or date with values from 0 to 6 with "0" representing a Sunday, "1" a Monday up to "6" for the Saturday. Note that

the function returns character vectors, which we change into numerical values to allow easier handling.

```
weekdays_num <- as.numeric(format(days_ext, "%w")) ## 0==Sun, 1==Mon, etc.
# alternatively: lubridate::wday(days_ext) ## 1==Sun, 2==Mon, etc.
```

The latter line gives us the weekdays of the last values of days_ext.

Now we define a matrix of all dummies. For convenience, we want to have the order Mon, Tue, ... Sun, so we consider in the line below a vector `c(1:6,0)` which matches this ordering:

```
# Week-day dummy matrix:
WD <- t(sapply(weekdays_num, "==", c(1:6, 0))) + 0 ## now first Mon, then Tue
dimnames(WD) <- list(NULL, c("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"))
tail(WD)
```

The weird looking `+ 0` term transforms the resulting vector from logical (only `TRUE` or `FALSE`)[26] to numeric which contains numbers (`FALSE` $\to 0$ and `TRUE` $\to 1$). The second line gives the 2nd dimension of the matrix names that match the weekdays.

For the simple model in equation (15) we only require the Monday, Saturday and Sunday column. Thus, we define the index set for the active weekdays in the expert model:

```
expert_wd <- c(1, 6, 7) # 1=Mon, 6=Sat, 7=Sun
```

As a more convenient alternative, we could have also used the `lubridate` package to do the same thing, with a different ordering of the weekdays. Note that the last variable `expert_wd]` has the same weekday ordering.

```
# Get weekday names
lubridate::wday(days_ext) ## 1==Sun, 2==Mon, etc.

# Week-day dummy matrix:
WD <- t(sapply(weekdays_num, "==", c(1:7))) + 0
dimnames(WD) <- list(NULL, c("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"))
tail(WD)

# Define expert lags
expert_wd <- c(2, 7, 1) # 2=Mon, 7=Sat, 1=Sun
```

Now we can define the regression matrix `XREG` (defined as $\mathcal{X}_s$ in (16))

```
# Define regressor matrix
XREG <- cbind(intercept = 1, WD[, expert_wd])
head(XREG, 14)
```

It contains in the first column only 1's which model the intercept, the latter ones contain the selected weekday dummies. Similarly we define the response vector `YREG` (defined as $\mathcal{Y}_s$ in (16)):

```
# Define response vector
YREG <- Y[index, s]
```

Note that we evaluate `Y` on the index set `index` as we do not know the value for the day that we want to forecast. Now we can use the `lm` command to estimate the linear model. However, we regress `YREG` only on `XREG` except the last row:

```
model_lm <- lm(YREG ~ XREG[-dim(XREG)[1], ] - 1)
```

We write the `-1`, because the `lm` command assumes by default that no intercept is provided. However, as our regression matrix `XREG` (or $\mathcal{X}_s$) contains the intercept (the column of ones) we should call `lm` with the `-1` in the regression formula. We can evaluate the summary statistics:

```
summary(model_lm)
```

The model coefficients (the estimated parameters $\widehat{\boldsymbol{\beta}}_s^{\text{ls}}$) can be called by:

```
model_lm$coef
```

Instead of using the `lm` command we could have used the analytical formula (13) to estimate the parameters:

[26] more precisely a logical also allows `NA` but not `NaN`, `Inf` or `-Inf`

```
model_OLS <- solve(t(XREG[-dim(XREG)[1], ]) %*%
XREG[-dim(XREG)[1], ]) %*%
t(XREG[-dim(XREG)[1], ]) %*%
YREG
```

With the model coefficients we can receive the forecast by

```
model_lm$coef %*% XREG[dim(XREG)[1], ]
```

Formally `model_lm$coef` is the estimated parameter vector $\widehat{\boldsymbol{\beta}}$ and `XREG[dim(XREG)[1],]` is the $\boldsymbol{X}_{D+1,s}$ which is in our case the last line of the regression matrix `XREG` (as we extended the index set in advance). Note that this is only the forecast for hour `s`. Obviously, we have to do S forecasts (for each period s of the day) to receive a full day-ahead forecast.

Below we do not use the function `lm` to estimate a linear model, but the much faster function `lm.fit` (see `?lm.fit`). The function works similarly, so we can estimate a model by providing the regressor matrix and the response vector as above:

```
model_lmfit <- lm.fit(XREG[-dim(XREG)[1], ], YREG)
```

The model coefficients that can be called in the same way as above:

```
model_lmfit$coef ## estimated parameters
```

So the forecast is given by

```
model_lmfit$coef %*% XREG[dim(XREG)[1], ] ## forecast
```

Now, we can do a quick microbenchmark run to check which function is the fastest.

```
  # Now compare computation time
microbenchmark::microbenchmark(
    lm = lm(YREG ~ XREG[-dim(XREG)[1], ] - 1),
    analytic = solve(
        t(XREG[-dim(XREG)[1], ]) %*%
            XREG[-dim(XREG)[1], ]
    ) %*%
        t(XREG[-dim(XREG)[1], ]) %*%
        YREG,
    lm.fit = lm.fit(XREG[-dim(XREG)[1], ], YREG)
)
```

Now we want to turn towards the full expert model (7). Similarly to `expert_wd`, we define a set of considered index lags:

```
expert_lags <- c(1, 2, 7) ## expert lag specification
```

Now we consider the following function:

```
# Function for getting 'lagged' vectors
get_lagged <- function(lag, Z) {
  return(
  c(rep(NA, lag), Z[(1 + lag):(length(Z) + 1) - lag])
  )
}
```

`get_lagged` simply lags the input vector `Z` by the input number lag while the length of the output vector is increased by 1. This increasing of the length by 1 simplifies our life later on for the forecasting, the reason is the same as above where we extended the index vector. Of course, we do not know the observations before our first observation in the vector `Z`. So when we lag our vector, we assume that the value before the first observation is unknown and will be represented by an `NA` – the native representation in `R` of a missing value. For instance, have a look at

```
get_lagged(lag=3, Z=c(1,2,5,6,2,-3,0,9))
```

Using the `sapply` command on `get_lagged` with `Y[index,s]`, we get directly the required lagged vectors for our regression problem:

```
# Get lagged regressors using sapply:
XLAG <- sapply(expert_lags, get_lagged, Z = price_DST[index, s])
dimnames(XLAG) <- list(NULL, paste("lag", expert_lags))
head(XLAG, 10)
tail(XLAG, 10)
```

Similarly as above, we can now specify the linear regression. Now it contains, the intercept, the lagged (autoregressive) effects (`XLAG`) and the selected weekday dummies (`WD[,expert_wd]`):

```
XREG<- cbind(1, XLAG, WD[,expert_wd])
YREG<- price_DST[index,s]
```

We can estimate the model and have a look at the summary output

```
model_expert_lm <- lm(YREG ~ XREG[-dim(XREG)[1], ] - 1)
summary(model_expert_lm)
```

Again, we get the forecast by

```
  model_expert_lm$coef %*% XREG[dim(XREG)[1],]
```

If we want to use `lm.fit` as above using

```
model_expert_lmfit <- lm.fit(XREG[-dim(XREG)[1], ], YREG)
```

we will get an error message. This is because the regressor matrix contains `NA`'s. Whereas the `lm` function can deal with such kind of data, the `lm.fit` function is not able to do so. Thus, we have to handle it manually if we want to benefit from the `lm.fit`'s performance. Therefore, we define the index set containing the indices of non-`NA` values of `XREG`. This we do by

```
act_index<- !apply(is.na(XREG),1, any)[-dim(XREG)[1]]
# alternatively: act_index <- which((rowSums(is.na(XREG[-nrow(XREG), ])) == 0))
```

To understand it, we have to have look at the function from the inner to the outer commands. `is.na(XREG)` gives a matrix with `TRUE` and `FALSE`, depending if the entry is an `NA` or not. `apply(is.na(XREG),1, any)` gives a vector, the corresponding element is `TRUE` if at least one cell in a particular row took an `NA` value, it is `FALSE` if all elements are non-`NA`. The `!` (exclamation mark) inverts the logical vector and `[-dim(XREG)[1]]` drops the last entry of the vector. Thus, `act_index` selects all available (meaning non-`NA`) entries. With `act_index` we can estimate the model using `lm.fit`:

```
model_expert_lmfit_noNA<- lm.fit(XREG[act_index, ], YREG[act_index] )
```

and compute the corresponding forecast:

```
model_expert_lmfit_noNA$coef %*% XREG[dim(XREG)[1],]
```

Finally, we want to write a forecasting function that takes as input only the available data (including the time/date information) and provides the values of the forecast as output. So we define

```
# Define forecasting function
forecast_expert <- function(Y,
days,
expert_wd = c(2, 7, 1),
expert_lags = c(1, 2, 7)) {
  # Init object to store forecasts
  forecast <- numeric()
  coefs <- matrix(
  nrow = length(expert_wd) + length(expert_lags) + 1,
  ncol = S
  )

  # Get dim per day
  S <- dim(Y)[2]

  # Create regressors: weekday dummies including the day to forecast
  days_ext <- c(days, days[length(days)] + 1)
  weekdays_num <- lubridate::wday(days_ext) ## 1==Sun, 2==Mon, etc.
```

```
  WD <- t(sapply(weekdays_num, "==", 1:7)) + 0 ## first Sun, then Mon...
  dimnames(WD) <- list(
  NULL,
  c("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat")
  )

  # Loop through all hours
  for (s in 1:S) {
    # Create regressors: lag matrix
    XLAG <- sapply(expert_lags, get_lagged, Z = Y[, s])
    dimnames(XLAG) <- list(NULL, paste("lag", expert_lags, sep = ""))

    # Create full regressor matrix
    XREG <- cbind(intercept = 1, XLAG, WD[, expert_wd])

    # Create response vector
    YREG <- Y[, s]

    # Remove NA's and last
    act_index <- which((rowSums(is.na(XREG[-nrow(XREG), ])) == 0))

    # Fit
    model <- lm.fit(XREG[act_index, ], YREG[act_index])

    # Store results
    forecast[s] <- model$coef %*% XREG[dim(XREG)[1], ]
    coefs[, s] <- model$coef
  }
  dimnames(coefs) <- list(names(model$coef), 1:S)
  output <- list("forecasts" = forecast, "coefficients" = coefs)
  return(output)
}
```

To understand, how `forecast_expert` works, it helps to specify

```
  Y <- price_DST[index, ]
  days <- dates[index]
```

and run the code within the function manually. The first lines prepare the estimation, so e.g. weekday-dummy matrix `WD` is defined (which does not depend on s or `s`) and the expert model (`expert_wd` and `expert_lags`) are specified. The main part is the loop from 1 to S, for each period of the day. Within this loop we have to state the model with its regressor matrix `XREG` and the response vector `YREG`. Then, the model gets estimated and forecasted as above. We can use the function in the same way as for the naive model.

```
forecast <- forecast_expert(
    Y = price_DST[index, ],
    days = dates[index]
)
```

We can also visualize the forecasts

```
# Get forecasts
f_expert <- forecast_expert(price_DST[index, ], days = dates[index])$f
f_naive <- forecast_naive(price_DST[index, ], days = dates[index])$f
df <- cbind(price_DST[index[length(index)] + 1, ], f_naive, f_expert)

# Plot results
plot(1:S - 1, df[, 1],
ylab = "Price in EUR/MWh",
col = "white",
ylim = range(df),
xlab = "s",
sub = format(dates[tail(index, 1)] + 1)
)
for (i in 1:dim(df)[2]) {
  lines(1:S - 1, df[, i], type = "o", lty = 3, pch = 20 + i, bg = 1 + i)
}
grid()
legend("topleft",
```

```
c("actual price", "forecast naive", "forecast expert"),
pch = 20 + 1:dim(df)[2],
pt.bg = 1 + 1:dim(df)[2],
lty = 3,
bg = rgb(1, 1, 1, .8)
)
```

We can also compute the forecasting error

```
# View forecasting error
price_DST[index[length(index)] + 1, ] - f_naive
price_DST[index[length(index)] + 1, ] - f_expert
mean(abs(price_DST[index[length(index)] + 1, ] - f_naive))
mean(abs(price_DST[index[length(index)] + 1, ] - f_expert))
```

R.2.3 Multivariate models

Now we want to build multivariate models. We begin by creating model (24) where the p-dimensional parameter vector $\boldsymbol{\beta}$ does not depend on s. First, we define `get_lagged` – a function we use through this section for creating lagged regressors.

```
get_lagged <- function(lag, Z) {
  return(
      c(rep(NA, lag), Z[(1 + lag):(length(Z) + 1) - lag])
  )
}
```

Then, we save the dimensions of the input object and create an extended `days` vector which includes the day to forecast:

```
D <- dim(Y)[1]
S <- dim(Y)[2]
# Days vector including the day to forecast
days_ext <- c(days, days[length(days)] + 1)
```

Now, we prepare the weekday dummies:

```
weekdays_num <- lubridate::wday(days_ext, week_start = 1)
WD <- t(sapply(weekdays_num, "==", 1:7)) + 0

dimnames(WD) <- list(
    NULL,
    c("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun")
)
```

For convenience, we store the expert specification in two separate objects:

```
expert_wd <- c(1, 6, 7) # 1=Mon, 2=Tue, ..., 7=Sun
expert_lags <- c(1, 2, 7) # lags
```

Now we start creating our regression matrix. First, we do it for each s. Afterwards we merge the S matrices together:

```
xreg_list <- list()
xreg_oos_list <- list()
yreg_list <- list()

# Loop through hour
for (s in 1:S) {
    # Reg matrix: lags
    xlag <- sapply(expert_lags, get_lagged, Z = Y[, s])
    dimnames(xlag) <- list(NULL, paste0("lag", expert_lags))

    # Reg matrix: full
    xreg_temp <- cbind(intercept = 1, WD[, expert_wd], xlag)

    # Store reg matrix
    xreg_list[[s]] <- xreg_temp[1:D, ]

    # Store oos reg matrix
```

```
        xreg_oos_list[[s]] <- xreg_temp[D + 1, ]

        # Store response
        yreg_list[[s]] <- Y[, s]
    }
```

Now that we have lists holding all S regression matrices, we can bind them together by using `do.call` or the `reduce` function from the `purrr` package.

```
  # Combine reg matrices and response to single objects
  xreg <- do.call(rbind, xreg_list) # alt: purrr::reduce(xreg_list, .f = rbind)
  xreg_oos <- do.call(rbind, xreg_oos_list) # alt: purrr::reduce(xreg_oos_list, .f =
      rbind)
  yreg <- do.call(c, yreg_list) # alt: purrr::reduce(yreg_list, .f = c)
```

That is, we only have to detect missings in the `xreg` matrix and remove them prior of estimation. Afterwards we are able to predict using the estimated coefficients and the `xreg_oos` object which holds the out of sample data.

```
    act_index <- which((rowSums(is.na(xreg)) == 0))
    model <- lm.fit(xreg[act_index, ], yreg[act_index])
    forecast <- as.numeric(xreg_oos %*% model$coef)
```

Next, we create model (32), where the autoregressive parameters are constant while the parameters corresponding to the weekdays can vary over s. This is more complicated since our regression matrix is much larger. First, we define the regression matrix which does not vary over s:

```
  # Init reg matrix
  xreg <- matrix(0,
      nrow = D * S,
      ncol = length(expert_wd) * S + length(expert_lags) + 24
  )

  colnames(xreg) <- c(
    paste0("lag_", seq_along(expert_lags)),
    unlist(map(seq_len(S), ~ paste0(c("int", "mon", "sat", "sun"), "_", .x)))
    # alt: lapply(seq_len(S), function(x) paste0(c("int", "mon", "sat", "sun"), "_",
        x))
)
```

This contain 24 sub-matrices. Some of these matrices will be filled, some of them will remain untouched. We then define other objects which will be needing and fill later in the loop:

```
  # Init One-Step-Ahead obs
  xreg_oos <- tail(xreg, 24) * 0

  # Init response
  yreg <- numeric(length = D * S)

  # Reg matrix weekdays
  wday_and_intercept <- cbind(intercept = 1, WD[, expert_wd])
```

Now, we are going to fill all objects using a loop:

```
# Loop through hours
for (s in 1:S) {
    # Set depenend variable
    yreg[(D * (s - 1) + 1):(D * s)] <- Y[, s]

    # Set intercept and weekdays
    xreg[
        (D * (s - 1) + 1):(D * s),
        (3 + (s - 1) * 4 + 1):(3 + (s - 1) * 4 + 4)
    ] <- wday_and_intercept[1:D, ]

    # Set out of sample values
    xreg_oos[
        s,
        (3 + (s - 1) * 4 + 1):(3 + (s - 1) * 4 + 4)
```

```
    ] <- wday_and_intercept[D + 1, ]

    # Reg matrix: lags
    xlag <- sapply(expert_lags, get_lagged, Z = Y[, s])
    dimnames(xlag) <- list(NULL, paste("lag", expert_lags, sep = ""))

    # Reg matrix: full, add lags
    xreg[(D * (s - 1) + 1):(D * s), 1:3] <- xlag[1:D, ]

    # Set out of sample values, add lags
    xreg_oos[s, 1:3] <- xlag[D + 1, ]
}
```

Here, for every additional hour we filled in the next D rows $(D * (s-1)+1)$ and the next corresponding 4 columns $(D * (s-1)+1)$ with the same weekday dummies and intercept. The first 3 columns are initially skipped and later filled in, since these are reserved for the lags. Notice how there are 4×24 different columns corresponding to different weekday dummies and intercepts for each hour, but only 3 columns corresponding to the lags for all hours. That is, we have a regression matrix with $24 + 24 * 3 + 3 = 99$ columns, and $D * S$ rows. The out-of-sample data is correspondingly filled in.

Then we remove the NAs, estimate the model and compute the forecast:

```
# Index without NA's
act_index <- which((rowSums(is.na(xreg)) == 0))

# Fit model
model <- lm.fit(xreg[act_index, ], yreg[act_index])

# Compute forecast
forecast <- as.numeric(xreg_oos %*% model$coef)
```

You may have noticed that our regression matrix `xreg` contains mainly zeros. In fact, $\approx 94.6\%$ of all entries are zero. Hence, we can save this information more efficiently using a so-called sparse matrix.

Now, let's convert `xreg` into a sparse matrix and estimate the model again:

```
    xreg_sparse <- Matrix(xreg, sparse = TRUE)
    model_coef <- MatrixModels:::lm.fit.sparse(
        xreg_sparse[act_index, ],
        yreg[act_index]
    )
    forecast <- as.numeric(xreg_oos %*% model_coef)
```

In the tutorial, we write functions to apply both of the above models. We also write a function that uses sparse matrices. The performance of these functions can be easily compared using a microbenchmark:

```
    microbenchmark(
        mixexpert = forecast_mixexpert(Y, days),
        mixexpert_sparse = forecast_mixexpert_sparse(Y, days)
    )
```

It shows that the sparse version needs less than 50% of the computation time of the non-sparse version. We can use the `provfis` package to obtain even more details about the execution times:

```
    # Even more inside into mixexpert:
    profvis({
        forecast_mixexpert(Y, days)
    })

    # Even more inside into mixexpert_sparse:
    profvis({
        forecast_mixexpert_sparse(Y, days)
    })
```

Inspecting the output shows that `lm.fit` needs 140ms out of the 210ms total execution time. `lm.fit.sparse` only needs 10 out of 70ms for execution.

R.3 Forecasting and Evaluation in R

R.3.1 Forecasting study design

Now we want to do the forecasting study in R. First, we define the required constants: the sample size N_window and the out-of-sample size N_eval. Using this information, we save the target dates of the out-of-sample forecasting study accessing the `dates_s` vector:

```
# Define forecasting study bounds
N <- dim(price_DST)[1]
N_window <- 730 # define rolling window size
N_eval <- N - N_window + 1 # size of forecasting study
S = 24 # no of hours per day

# Get out-of-sample dates
oos_dates <- dates[N_window] + 1:N_eval
# + 1 since last day to be forecasted is not included in historical data;
# we will remove this from error calculation later

# Define model names
model_names <- c("true", "naive", "expert", "mv_expert", "mv_expert_mix")
n_models <- length(model_names)
```

The last two lines define the number of models that we want to compare in the forecasting study and provide the model names. Here we want to compare all models that we have created already. Since we do not wish to create redundancy and type out every forecasting function again, we load all functions that we defined before by sourcing the `00_function.R` file:

```
source("00_functions.R")
```

It is a reasonable idea to analyze the forecast values and the estimated coefficients of our model. The naive model has no coefficients, so that we will analyze only those of the more advanced models.

Now, we want to carry out the forecasting study. Therefore, we need objects that store all relevant information of our forecasting study. Therefore, we create one 3-dimensional array to assign our forecasts and a list containing three arrays to store the estimated coefficients.

```
forecasts <- array(, dim = c(N, S, n_models))
dimnames(forecasts) <- list(format(oos_dates), 1:S, model_names)

coeffs_list <- list()
coeffs_list[["expert"]] <- array(dim = c(7, N, S))
coeffs_list[["mv_expert"]] <- array(dim = c(7, N))
coeffs_list[["mv_expert_mix"]] <- array(dim = c(99, N))
```

These objects will be filled during the study with the corresponding forecasting errors and coefficients.

Using our forecasting functions, we can write down the code for the forecasting study easily:

```
# Do rolling window forecasting study
# Loop through all days to forecast
for (n in 1:N_eval) {
    # test: n = N_eval
    index <- 1:N_window + n - 1 ## defining index set for input data (rolling
        window range)

    # Actual price (missing for last day)
    if (n < N_eval) { # skip last day
        forecasts[n, , "true"] <- price_DST[max(index) + 1, ]
    }

    # Naive model (no coefs)
    forecasts[n, , "naive"] <- forecast_naive(
        price_DST[index, ],
        days = dates[index]
    )$forecasts

    # Expert model
    fc_expert <- forecast_expert(
```

```
        price_DST[index, ],
        days = dates[index]
    )

    forecasts[n, , "expert"] <- fc_expert$forecasts
    coeffs_list[["expert"]][, n, ] <- fc_expert$coefficients

    # MV expert model
    fc_expert_mv <- forecast_mvexpert(
        price_DST[index, ],
        days = dates[index]
    )

    forecasts[n, , "mv_expert"] <- fc_expert_mv$forecasts
    coeffs_list[["mv_expert"]][, n] <- fc_expert_mv$coefficients

    # MV expert mixed model
    fc_mv_expert_mix <- forecast_mixexpert_sparse(
        Y = price_DST[index, ],
        days = dates[index]
    )

    forecasts[n, , "mv_expert_mix"] <- fc_mv_expert_mix$forecasts
    coeffs_list[["mv_expert_mix"]][, n] <- fc_mv_expert_mix$coefficients

    # Show progress
    cat("->", round((n / N_eval) * 100), "%", "\r")
} # n
```

The first line defines the index set `index` that is used to give the observed sample `price_DST[index,]` to the forecasting functions. Then, we compute the forecasts, the corresponding errors and return the coefficients. As we have N_eval days in the out-of-sample forecasting study, the loop runs from 1 to N_eval.

Now we do need to do some post-processing. Specifically, naming the arrays that store the coefficients and calculate the forecasting errors.

```
dimnames(coeffs_list[["expert"]]) <- list(
  rownames(fc_expert$coefficients),
  NULL,
  paste0("S", 1:S)
)

dimnames(coeffs_list[["mv_expert"]]) <- list(
    rownames(fc_expert_mv$coefficients),
    NULL
)

dimnames(coeffs_list[["mv_expert_mix"]]) <- list(
    names(fc_mv_expert_mix$coefficients),
    NULL
)

# Note that we do have a forecast for dim(forecasts)[1] but we don't know the
# true value. Thus we can't calculate errros for it:
tail(forecasts)

errors <- sweep(
    x = forecasts[-dim(forecasts)[1], , ],
    MARGIN = 1:2,
    FUN = "-",
    forecasts[-dim(forecasts)[1], , "true"]
)

dimnames(errors) <- list(
    NULL,
    NULL,
    dimnames(forecasts)[[3]]
)
```

R.3.2 Coefficient analysis

Having saved the model's coefficients in the forecasting study, we can understand the impact of the regressors on the price formation in our data. The best way to do it is to view a plot of the coefficients over time. Below is an example of inspecting the coefficients of the expert model for $s = 11$.

```
# Plot expert model coefficients over time
s <- 11 # select hour
ts.plot(t(coeffs_list[["expert"]][, , s]), col = 1:7)

legend("topleft",
    dimnames(coeffs_list[["expert"]])[[1]],
    col = 1:7, bg = rgb(1, 1, 1, .8), lwd = 2
)
```

We receive a plot of the coefficients for the product $s = 11$. The intercept and the weekday dummies change over time, but the autoregressive coefficients seem relatively stable, close to 0. This is naturally not true. The impression comes from the difference in scale of both types of regressors. We didn't scale any of our regressors, so comparing and drawing both these types on one plot doesn't make much sense. Thus, we split them into two plots.

```
par(mfrow = c(2, 1))
# We will split the plot to inspect lags and dummies seperately:
# Plot DoW dummies of expert model
which.coefs <- 5:7 # DoW coefficients
ts.plot(t(coeffs_list[["expert"]][which.coefs, , s]),
    col = seq_along(which.coefs)
)
legend("topleft",
    dimnames(coeffs_list[["expert"]])[[1]][which.coefs],
    col = seq_along(which.coefs), bg = rgb(1, 1, 1, .8), lwd = 2
)

# Plot intercept & lags of expert model
which.coefs <- 1:4 # intercept and lags coefs
ts.plot(t(coeffs_list[["expert"]][which.coefs, , s]),
    col = seq_along(which.coefs) + 3
)
legend("topleft", dimnames(coeffs_list[["expert"]])[[1]][which.coefs],
    col = seq_along(which.coefs) + 3, bg = rgb(1, 1, 1, .8), lwd = 2
)
```

Using the code above, we can observe the changes of the coefficients over time. It may be time-consuming, but it is worth spending time on this exercise to gain insights regarding structural breaks, changes in the dependency structure, etc. For instance, running the code above, we observe pretty stable parameter values until a few days before $d = 500$. There we see a sudden change in the coefficient values, which may suggest some structural break. Then, until around $d = 700$, the importance of lagged values increases while it decreases for the dummies.

R.3.3 MAE and RMSE

The MAE and RMSE can be computed for all models at once. With the error array `errors` from the out-of-sample forecasting study, we receive the MAE and RMSE by

```
mae <- sort(
    apply(abs(errors), 3, mean)
)
rmse <- sort(
    sqrt(apply(errors^2, 3, mean))
)
```

where we utilize the `apply` command.

Of course, we can also evaluate with an hourly resolution:

```
mae_hourly <- apply(abs(errors), c(2, 3), mean)
rmse_hourly <- sqrt(apply(errors^2, c(2, 3), mean))
```

Using the obtained matrix, we can plot it over the products. This way we can easily check the forecasting performance of our models for all products.

```
ts.plot(rmse_hourly, col = seq_along(model_names), lwd = 2)
legend("topleft",
    model_names,
    col = seq_along(model_names),
    bg = rgb(1, 1, 1, .8), lwd = 2
)
```

R.3.4 The DM-Test

We define a very general function `dm_test` that carries out the Diebold-Mariano test. It takes as input two error vectors of dimension N_eval or matrices of dimension $N_eval \times S$ (for the multivariate tests) that represent the errors of forecasts A and B. Moreover, it takes the optional parameters `hmax` which can be ignored for our purpose[27] and a parameter `power` which is 1 for the $\|\cdot\|_1$ test and 2 for the $\|\cdot\|_2$ test.

The function is given by

```
dm_test <- function(error_a, error_b, hmax = 1, power = 1) {
  # As dm_test with alt hypothesis H_1 == "less": model a is better than b
  ## error_x must be a N_eval-1 x S matrix (-1 since last oos day is unknown)
  ## or an N_eval-1 x 1 vector (for each hour separately); then apply()
  ## won't do anything

  # Calc loss of model a and b (L-power norm) across all hours
  loss_a <- apply(abs(as.matrix(error_a))^power, 1, sum)^(1 / power)
  loss_b <- apply(abs(as.matrix(error_b))^power, 1, sum)^(1 / power)

  # Calculate delta
  delta <- loss_a - loss_b
  sum(delta)
  # Estimate variance of delta
  delta_var <- var(delta) / length(delta)

  # Calc test statistic
  statistic <- mean(delta, na.rm = TRUE) / sqrt(delta_var)

  # Calc p-value
  delta_length <- length(delta)
  k <- ((delta_length + 1 - 2 * hmax +
      (hmax / delta_length) * (hmax - 1)) / delta_length)^(1 / 2)
  statistic <- statistic * k
  p_value <- pt(statistic, df = delta_length - 1)

  # Return results
  return(list(stat = statistic, p.val = p_value))
}
```

We see that the first two lines define `loss_a` and `loss_b` the vector of loss functions with elements that represent $L_{A,i}$ and $L_{B,i}$. The third line defines the loss difference called `delta`. The fourth line computes the variance of the loss difference and the fifth line computes the DM-test statistic `statistic` by taking the mean of `delta` and dividing it by its standard deviation. The next three lines adjust for the finite sample bias. The object `p_value` computes the p-value of the DM-test. Finally, `dm_test` returns a list with two elements, the test statistic, and the p-value. We can easily apply the `dm_test` by

```
dm_test(errors[, , 2], errors[, , 3])
dm_test(errors[, , 3], errors[, , 2])
```

Moreover, we can evaluate the DM-test for each period of the day separately by

```
p_values <- numeric(S)
for (s in 1:S) {
```

[27] The parameter `hmax` determines the largest significant autocorrelation lag of the difference series, which is only required if we forecast more than a day ahead.

```
    p_values[s] <- dm_test(errors[, s, 5], errors[, s, 4], power = 2)$p.val
  }
```

We test model 5 (the multivariate expert-mix model) against model 4 (the fully multivariate model). The results can be analyzed using a simple plot.

```
  plot(p_values, type = "h", lwd = 2)
  abline(h = 0.05, lty = 2, col = 2)
```

It seems intuitive to use the `dm_test` function to test each model against all other models. To do so, we define a list `dm_results` which is filled with two dataframes that hold the p_values and the test statistics, respectively.

```
  dm_results_df <- data.frame(
    matrix(
        ncol = length(model_names_wo_true),
        nrow = length(model_names_wo_true)
    )
)

dimnames(dm_results_df) <- list(model_names_wo_true, model_names_wo_true)

dm_results <- list(
    "p_val" = dm_results_df,
    "t_stat" = dm_results_df
)

dm_results$p_val[, "type"] <- "p_val"
dm_results$t_stat[, "type"] <- "t_stat"
```

Now we can compute the desired tests using two nested for-loops:

```
  for (mod_a in seq_along(model_names_wo_true)) {
    for (mod_b in seq_along(model_names_wo_true)) {
        if (mod_a == mod_b) {
            dm_results[["p_val"]][mod_a, mod_b] <- NA
            dm_results[["t_stat"]][mod_a, mod_b] <- NA
        } else {
            dm <- dm_test(errors_wo_true[, , mod_a], errors_wo_true[, , mod_b])
            dm_results[["p_val"]][mod_a, mod_b] <- dm$p.val
            dm_results[["t_stat"]][mod_a, mod_b] <- dm$stat
        }
    }
}
```

The code above fills the list `dm_results`. That said, we have gathered the desired data. However, visualizing it required some more data wrangling:

```
  dm_results_tible <- dm_results %>%
  # We rbind both dataframes together
  reduce(.f = rbind) %>%
  # Convert them to a tibble. Its basically a nice dataframe
  as_tibble() %>%
  # Add a columns mod_a
  mutate(mod_a = c(
      model_names_wo_true,
      model_names_wo_true
  )) %>%
  # And bring the tibble to into a longer format
  pivot_longer(!c(type, mod_a),
      names_to = "mod_b", values_to = "val"
  ) %>%
  # Finally bring the tibble to into a wider format wrt type
  pivot_wider(names_from = type, values_from = val)
```

Notice, that we are utilizing the `%>%` operator from the `dplyr` package. It simply puts the output of the preceding expression into the function which follows as the first argument. That means we do not have to nest various functions and /or save intermediate results which we do not need. The other function calls are well documented within the code itself. The resulting object `dm_results_tible` is a `tibble`. We do not need to cover details here. Just treat them as

if they are dataframes and you will be fine[28]. It contains 4 columns `model_a`, `model_b`, `p_val` and `t_stat`. We can easily visualize our results using the `ggplot2` package:

```
# P-values
dm_results_tible %>%
    ggplot(aes(x = mod_b, y = reorder(mod_a, desc(mod_a)), fill = p_val)) +
    geom_raster()
# Test statistics
dm_results_tible %>%
    ggplot(aes(x = mod_b, y = reorder(mod_a, desc(mod_a)), fill = t_stat)) +
    geom_raster()
```

R.3.5 Online Learning

We will briefly cover how (58) can be implemented in R. In fact, we have implemented (58) using the `RcppArmadillo` package. It allows us to write Functions in C++ and call them from withing an R session. We will skip the details here and instead refer to the `mrls.cpp` file which you can find on moodle. The following command makes the C++ function available in R:

```
Rcpp::sourceCpp("mrls.cpp")
```

The `mrls` function expects various arguments. Some of them are vectors which have to be created upfront. Therefore, we create the follwoing wrapper function in R which conviniently creates the required inputs for us.

```
# Define Online-Learning function in R (wrapper for mrls())
fast_rec_ls <- function(y, # n-dimensional response vector (e.g: price)
                        x, # (n+1) x p regressor matrix incl. element not observed
                            yet (e.g.: weekday dummies, autoregressive elements)
                           (should be: nrow(x) = length(y) + 1)
                        start = 30, # starting point of validation/test data
                        rfac = 5, # empirical value to calculate refit times (do
                            not change)
                        eps = 1e-6, # cutoff value for weights during refit: if w
                            = rho^(...) < eps => w = 0
                        lambda = 1e-7, # ridge regularization at refit, to make
                            sure no error in case of n < p (less observations than
                             regressors) due to exponential decay
                        rho = c(1 - 2^(-seq(1, log(length(y), 2), 1)), 1)) { #
                            sequence of forgetting parameters (no forgetting: rho
                            = 1)

    # Define rho (forgetting parameter) matrix: rows = weight for observations;
        columns = different rho values
    rho_seq <- matrix(, length(y), length(rho))
    for (i in seq_along(rho)) {
        rho_seq[, i] <- rho[i]^((length(y) - 1):0)
    }
    colnames(rho_seq) <- rho

    # Define refit parameter: determines how often (after how many model fits) the
         Gramian is fully recomputed for each rho parameter
    ## refit = 1: no online learning; refit should decrease with rho, since high
        forgetting might lead to numerical instability (accumulation of rounding
        errors)
    refit_by <- floor(pmax(pmin(length(y), 1 / (1 - rho)) * rfac, 1))
    return(mrls(y = y, x = x, start = start, rhoseq = rho_seq, refitseqby = refit_
        by, eps = eps, lambda = lambda))
}
```

The `fast_rec_ls` function will return a $(n - start + 1)$x$(nrho)$ matrix of forecasts. This means, that a forecast is computed for which we have not yet observed the realization. Next, we can easily write a function that carries out a forecasting study. The core of this function is a loop running over all $s \in S$ to compute forecasts for all S hours of the day.

```
# Forecasting Study using the online approach
```

[28] You can read about the advantages of tibbles over data.frames here: `https://tibble.tidyverse.org/`

```r
fs_expert <- function(Y = price_DST, # price matrix (n x 24)
                      days = dates, # dates vector (n)
                      N_forecast = 730, # no of days to forecast from end of dataset
                          (evaluation + test points)
                      rho = c(1 - 2^(-seq(1, log(dim(Y)[1], 2), 1)), 1), # sequence
                         of forgetting parameters (no forgetting: rho = 1)
                      expert_wd = c(1, 6, 7), # weekday dummies
                      expert_lags = c(1, 2, 7),
                      ...) { # lags (autoregressive values)

    # Get dimension of prices per day (24 for hourly products)
    S <- dim(Y)[2]

    # Get no of days in dataset
    N_days <- dim(Y)[1]

    # Init object to store forecasts
    forecast <- array(, dim = c(N_forecast, S, length(rho))) # 1D: days to forecast;
         2D: hours; 3D: rho values

    # Days vector including the day to forecast
    days_ext <- c(days, days[length(days)] + 1)
    N_days_ext <- length(days_ext)

    # Prepare weekday dummies
    weekdays_num <- lubridate::wday(days_ext, week_start = 1) # transform days into
        weekday codes: 1==Mon, 2==Tue, etc. due to week_start = 1
    WD <- t(sapply(weekdays_num, "==", 1:7)) + 0 # generate weekday matrix
    dimnames(WD) <- list(days_ext, c("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
        )) # set names of WD cols

    # Run (online) forecast for each hour
    for (s in 1:S) { # loop through hours
        # test: s = 1

        # Prepare lags:
        xlag <- sapply(expert_lags, get_lagged, Z = Y[, s]) # generate lags of Y in
            columns; get_lagged() function autmatically returns +1 for the last
            value to forecast

        # Prepare regressor matrix:
        xreg <- cbind(intercept = 1, WD[, expert_wd], xlag) # add intercept, weekday
             dummies to xlag

        # Prepare response vector:
        yreg <- Y[, s] # get price values for the hour

        # Remove NAs
        act_index <- which((rowSums(is.na(xreg)) == 0)) # get indexes of xreg rows
            without NAs

        # Get length of act_index
        N_act <- length(act_index)

        # Run forecast
        forecast[, s, ] <- fast_rec_ls(
            y = yreg[act_index[-N_act]], # y: prices without last value, since last
                value is not known (is NA), to be forecasted
            x = xreg[act_index, ], # x: regressors with corresponding last value, to
                 forecast unknown last y
            start = N_act - N_forecast, # start from the point such that from that
                point to end we have N_forecast left (N_actual != N_days_ext because
                 we removed some obs due to NAs when getting lagged values)
            # alt equivalent: start = N_days_ext - N_forecast - max(expert_lags); N_
                days_ext = nrow(xreg)
            rho = rho # sequence of forgetting parameters
        )
    }

    # Calculate effective sample size
```

```
    neff <- round(1 / (1 - rho), 3)

    # Set names
    dimnames(forecast) <- list(
        format(tail(days_ext, dim(forecast)[1])),
        1:S,
        paste("neff=", neff, sep = "")
    )

    # Return result
    return(forecast)
}
```

Next we will carry out the forecasting study:

```
  # Check computation time
  system.time(
      forecast_mrls <- fs_expert(Y = price_DST, days = dates, N_forecast = length(
          dates) - 730 + 1)
  )
```

The whole study takes less than half a second, although it is carried out multiple times with different forgetting factors. For the case of $\rho = 1$ we obtain an expanding window forecasting study. That is, the first forecast is equivalent to that of the expert model which we used in the forecasting study of subsection R.3.1. We can check by loading the forecasts and compare them to the newly obtained ones:

```
  # Load forecasts object created in tutorial 4: 04_Forecast_Evaluation.R
  load("out/04_forecasts_study.rds")

  # Show first 5 forecasts of expanding window model for rho=1, for hours 9-15
  head(forecast_mrls[, 10:16, 12], 5) # the 12th rho value is the one with rho = 1

  # Show first 5 forecasts of rolling window model for rho=1, for hours 9-15
  head(forecasts[, 10:16, "expert"], 5)

  # Note: Values are the same (up to 4th decimal place) for the first forecast,
  # then it's different because expanding window model uses the last data point
  #  while rolling window discards the last datapoint
```

The forecasting errors can be created by substracting the true values by using `sweep`:

```
  errors_rec_ls <- sweep(
    x = forecast[-dim(forecasts)[1], , ],
    MARGIN = 1:2,
    FUN = "-",
    forecasts[-dim(forecasts)[1], , "true"]
)
```

Now, calculating the hourly RMSE is straightforward:

```
  rmse_hourly_rec_ls <- sqrt(
    apply(errors_rec_ls^2, 2:3, mean, na.rm = TRUE)
  ) # RMSE
```

This yields a matrix containing RMSE values for each $s \in S$ and each effective sample size $n_{\text{eff}}(\varrho)$. We can easily aggregate over S by taking the sum of each column:

```
  colMeans(rmse_hourly_rec_ls)
```

The results show that an effective sample size of 2048 already yields the best performance in RMSE.

R.4 Hyperparameter tuning with `mlrMBO` in `R`

To find out which of the regressors considered so far (autoregressive and weekday dummy components) are best, one could try out all possible combinations and compare the results. For example, we could try including only the first lag. Then try including the first two lags. Then only the second lag, etc. However, this is not feasible for a large number of regressors. Another

possibility is to try out a few random combinations of regressors. The problem with that is, that we might overlook a good combination of regressors due to the random search. Ideally we want something inbetween of these two extremes. The *mlrMBO* package is based on Bayesian optimization. This means that it will initially choose some random regressors, and then it will use a black-box algorithm to nudge the regressors into the right direction, meaning it will minimize a loss function. It is not guaranteed to find the global optimum, however it is very likely to find a good combination of regressors in most real-life applications.

The package is available on CRAN and can be installed by:

```
library(mlrMBO)
```

Some dependencies may be necessary.

In short, `mlrMBO` minimizes (or maximizes) a function by iteratively trying out different input values. The initial choice of the inputs are random, but the subsequent choices are based on the previous results.[29]. The function to be minimized is called the *objective function*. In our case, the objective function is the RMSE of the expert model on the evaluation set. Thus, we define a function that is essentially the same as the one from the previous section, in that it undergoes the forecasting study using the `mrls.cpp` function, but it returns the loss instead of the forecasts. Also, the expert lags and weekday dummies, which are argument of the function, are defined as booleans instead of integers, so this:

```
active_weekdays = as.logical(c(1, 0, 0, 0, 0, 1, 1)), # active weekdays for expert
    model
active_lags = as.logical(c(1, 1, 0, 0, 0, 0, 1)), # active lags for expert model
```

instead of

```
  active_weekdays = c(1,6,7), # active weekdays for expert model
  active_lags = c(1,2,7), # active lags for expert model
```

So, if we want lags 1, 2 and 7, we set the first, second and seventh element of the vector to 1, and the rest to 0. We do this because this format is needed for the mlrMBO package, since it will try to minimize the function by changing the values of the arguments, effectively "activating" or "deactivating" the regressors.

We first define the forecasting study design parameters

```
# Define training, eval, test data
N_data <- dim(price_DST_arr)[1] # length of data in no of days
N_test <- 365 * 0 # 0 years of test data
N_eval <- 365 * 3 # 3 years of eval data
N_train <- N_data - N_test - N_eval
N_window <- 730 # window size

# Define response, days
data <- price_DST
days <- dates

# Define variables to be used
rho <- 1 # forgetting parameter
expert_wd <- c(1, 6, 7) # weekday dummies for expert model
active_weekdays <- as.logical(c(1, 0, 0, 0, 0, 1, 1)) # active weekdays for expert
    model
active_lags <- as.logical(c(1, 1, 0, 0, 0, 0, 1)) # active lags for expert model

test_loss <- numeric(0) # init test loss vector
forecast_mlrmbo <- list() # init forecast list
save_forecast <- TRUE
```

Then we define the loss function, which will serve as the objective function for mlrMBO:

```
# Define loss function (to be minimized by mlrMBO later)
fs_expert_loss <- function(data = price_DST,
                           days = dates,
                           N_eval = 365 * 3, # no of days to evaluate
```

[29]For more information on how the algorithm works see the official documentation page: https://cran.r-project.org/web/packages/mlrMBO/vignettes/mlrMBO.html

```
                        N_test = 365 * 0, # no of days to test
                        N_window = 730, # max initial window size
                        rho = 1, # forgetting parameter
                        active_weekdays = as.logical(c(1, 0, 0, 0, 0, 1, 1)), #
                            active weekdays for expert model
                        active_lags = as.logical(c(1, 1, 0, 0, 0, 0, 1)), #
                            active lags for expert model
                        save_forecast = TRUE) { # save forecast yes/no

# Forecast amount put in arg
N_forecast <- N_eval + N_test

# Trim data
Y <- tail(data, N_forecast + N_window)
days <- tail(days, N_forecast + N_window)

# Get lags
expert_lags <- seq_along(active_lags)[active_lags] # get lags to be used
expert_wd <- seq_along(active_weekdays)[active_weekdays] # get weekday dummies
    to be used

# Get dimension of prices per day (24 for hourly products)
S <- dim(Y)[2]

# Get no of days in dataset
N_days <- dim(Y)[1]

# Init object to store forecasts
forecast <- array(, dim = c(N_forecast, S, length(rho))) # 1D: days to forecast;
     2D: hours; 3D: rho values

# Days vector including the day to forecast
days_ext <- c(days, days[length(days)] + 1)
N_days_ext <- length(days_ext)

# Prepare weekday dummies
weekdays_num <- lubridate::wday(days_ext, week_start = 1) # transform days into
    weekday codes: 1==Mon, 2==Tue, etc. due to week_start = 1
WD <- t(sapply(weekdays_num, "==", 1:7)) + 0 # generate weekday matrix
dimnames(WD) <- list(as.character(days_ext), c("Sun", "Mon", "Tue", "Wed", "Thu"
    , "Fri", "Sat")) # set names of WD cols

# Run (online) forecast for each hour
for (s in 1:S) { # loop through hours
    # test: s = 1

    # Prepare lags:
    xlag <- sapply(expert_lags, get_lagged, Z = Y[, s]) # generate lags of Y in
        columns; get_lagged() function autmatically returns +1 for the last
        value to forecast
    if (length(xlag) == 0) xlag <- numeric(0)

    # Prepare regressor matrix:
    xreg <- cbind(intercept = 1, WD[, expert_wd], xlag) # add intercept, weekday
         dummies to xlag

    # Prepare response vector:
    yreg <- Y[, s] # get price values for the hour

    # Remove NAs
    act_index <- which((rowSums(is.na(xreg)) == 0)) # get indexes of xreg rows
        without NAs

    # Get length of act_index
    N_act <- length(act_index)

    # Run forecast
    forecast[, s, ] <- fast_rec_ls(
        y = yreg[act_index[-N_act]], # y: prices without last value, since last
            value is not known (is NA), to be forecasted
```

```
            x = as.matrix(xreg[act_index, ]), # x: regressors with corresponding
                last value, to forecast unknown last y; nrow of x = length of y + 1
            start = N_act - N_forecast, # start from the point such that from that
                point to end we have N_forecast left (N_actual != N_days_ext because
                 we removed some obs due to NAs when getting lagged values)
            # alt equivalent: start = N_days_ext - N_forecast - max(expert_lags); N_
                days_ext = nrow(xreg)
            rho = rho # sequence of forgetting parameters
        )

        # Run equivalanet forecast using lm.fit (for testing)
        # start <- N_act - N_forecast
        # for (i in 1:N_forecast) {
        #     model <- lm.fit(
        #         as.matrix(xreg[act_index, ])[1:(start + i - 1), ] %>% as.matrix(),
        #         yreg[act_index[-N_act]][1:(start + i - 1)] %>% as.matrix()
        #     )

        #     forecast_lmfit[i] <- model$coefficients %*% as.matrix(xreg[act_index,
            ])[start + i - 1 + 1, ]
        #     betas_lmfit[i] <- model$coefficients
        # }
        # forecast[, s, 1] - forecast_lmfit
    }
    # Calculate effective sample size
    neff <- round(1 / (1 - rho), 3)

    # Set names
    dimnames(forecast) <- list(
        format(tail(days_ext, dim(forecast)[1])),
        1:S,
        paste("neff=", neff, sep = "")
    )

    # Save forecast
    if (save_forecast) forecast_mlrmbo[[length(forecast_mlrmbo) + 1]] <<- forecast

    # Calculate error
    error <- rbind(tail(Y, N_forecast - 1), t(rep(NA, S))) - forecast[, , 1] ## add
        an NA row for Y since the last forecast corresponds to the last value of Y,
        which is not known (NA)

    # Calculate MSE per day (MSE across hours)
    loss <- as.numeric(apply(error * error, 1, mean))

    # Calculate RMSE for evaluation set only (first N_eval values of the loss vector
        , because loss vector only includes eval and test data)
    eval_loss <- sqrt(mean(loss[1:N_eval], na.rm = TRUE))

    # Calculate RMSE for test set only (next N_test no of values after N_eval, i.e.
        up to N_forecast; N_eval+N_test = N_forecast)
    test_loss[length(test_loss) + 1] <<- sqrt(mean(loss[(N_eval + 1):(N_eval + N_
        test)], na.rm = TRUE))

    # Return the evaluation loss (which will be the objective of mlrMBO to minimize)
    return(eval_loss)
}
```

Now we can use this loss function to define the objective function for mlrMBO, which we will call `obj_fun`. It has 4 arguments that need to be defined: First, the name of the model under `name=`. Second, the loss function itself under `fn=`. This function takes in a single argument that can be a vector. In our case this vector will be a named vector `x` with all relevant inputs for our model: rho, window length, the lags, the weekday dummies from 1 through 7. These then get passed into our loss function as defined above. The third argument is the parameter set, which defines the search space for the inputs under `para.set=`. Here, the type and names of the input paramters of `x` need to be specified. The fourth argument is `minimize=TRUE`, which tells mlrMBO that we want to minimize the loss function. The code for this is as follows:

```
# Define loss function as objective for mlrMBO
```

```
obj_fun <- smoof::makeSingleObjectiveFunction(
    # Name of obj_fun
    name = "lin_mod",

    # Define function
    fn = function(x) {
        # The function will take in a vector x of named parameters
        # e.g: expert model will be:
        # x = c(1, 730,
        #   1, 1, 0, 0, 0, 0, 1,
        #   1, 0, 0, 0, 0, 1, 1)
        # names(x) = c("rho", "window",
        # "lag1", "lag2", "lag3", "lag4", "lag5", "lag6", "lag7",
        # "wd1", "wd2", "wd3", "wd4", "wd5", "wd6", "wd7")

        # Get weekday dummy parameters (active weekday dumies), convert to logical
        x_wd <- (x[paste("wd", 1:7, sep = "")]) |> as.logical()

        # Get lag parameters (active lags), convert to logical
        x_lag <- (x[paste("lag", 1:8, sep = "")]) |> as.logical()

        # Define loss that uses the parameters above
        loss <- fs_expert_loss(
            data = price_DST,
            days = dates,
            N_eval = 365 * 2,
            N_test = 365 * 0,
            rho = x["rho"],
            N_window = x["window"],
            active_weekdays = x_wd,
            active_lags = x_lag,
            save_forecast = TRUE
        )
        return(loss)
    },

    # Define parameter types and bounds
    par.set = makeParamSet(
        # Define rho range on exponential grid
        makeNumericParam("rho", lower = -13, upper = -3, trafo = function(x) {
            return(1 - 2^x)
        }),

        # Define size range
        makeIntegerParam("window", lower = 14, upper = dim(price_DST)[1]),

        # Define lags range
        makeIntegerVectorParam("lag", len = 8, lower = 0, upper = 1),

        # Define weekday dummies range
        makeIntegerVectorParam("wd", len = 7, lower = 0, upper = 1)
    ),
    minimize = TRUE
)
```

For the rho (ρ) parameter we defined a function $f(x) = 1-2^x$ which takes in values from -13 upto -3. This exponential function makes sure that the actual `rho` parameter that gets passed to the loss function is between 0.875 and 0.9998. `mlrMBO` is allowed to choose values between -13 and -3 because we want to make sure that many more values are tried out closer towards 1 than towards 0. The implicit assumption of doing so is that some form of exponential decay (forgetting) will be the best choice for the parameter, but not too much. For $rho = 0.875$ the effective sample size would be $8 = 1/1(-0.875)$, meaning that our model effectively uses only the equivalent of a window length of 8 historical observations to fit the model. Clearly this can not be enough for our model, since it needs some history to be able to learn all of the relevant effects correctly.

The `window` parameter is allowed to take values between 14 and the length of the dataset. The lower bound is 14 because we want to make sure that the model has at least two weeks of

data to learn from. The upper bound is the length of the dataset because we want to make sure that the model can use all of the data.

Then we generate a random initial combination of regressors, which will be the starting point for the optimization algorithm. We store this in an object called `design`:

```
  # Generate initial random design
  N_params <- getNumberOfParameters(obj_fun) # get no of parameters
  N_points <- as.integer(1 + sqrt(N_params) * 2) # empirical number of initial
      points

  set.seed(12345) # set seed for reproducibility of sims
  design <- generateDesign(
      n = N_points,
      par.set = getParamSet(obj_fun),
      fun = lhs::randomLHS
  ) # generate design
```

Since this design is randomly generated, we also include our expert model as part of the design, to make sure that the algorithm will not miss it. We do this by adding the expert model to the design.

```
# Add the expert model to the design
design <- design %>% add_row(design[1, ] * 0, .before = 1) # add empty row at
    beginning
design[1, c("rho", "window", "lag1", "lag2", "lag7", "wd1", "wd6", "wd7")] <- c(-13,
     dim(price_DST)[1], 1, 1, 1, 1, 1, 1)
```

After doing this we fix a few control parameters that will define which optimization algorithm is going to be used and how it will run. For example, the `setMBOControlTermination()` function determines when the algorithm should stop searching. We specify that we want it to stop searching for the optimum either when the number of iterations equal to 100 + the number of paramters is reached, or when 100 seconds have passed. We do so by specifying $max.evals = N\_params + 100$ and $time.budget = 100$ inside of the function.

```
  # Generate control objects
control <- makeMBOControl(propose.points = 1) # create object
control <- setMBOControlTermination(control, max.evals = N_params + 100, time.budget
     = 100) # setup termination criteria
control <- setMBOControlInfill(control_obj, crit = makeMBOInfillCritEI())
control <- setMBOControlMultiPoint(control, method = "cl", cl.lie = min)

# Make learner
mbo_learner <- makeLearner("regr.randomForest", predict.type = "se") # alt: , se.
    method = "jacknife")

# Run MBO optimizer
run <- mbo(fun = obj_fun, learner = mbo_learner, design = design, control = control,
     show.info = TRUE)
## if you get NaN errors check design parameter, could be out of ranges
```

The last command starts the learning process, which finishies if one of the termination criteria is met. We can then plot the results and print out the 10 best combinations that the optimization algorithm has found:

```
# Plot results
plot(run)

# Show optimal path sorted, replace rho with effective sample size
path_matrix <- run$opt.path$env$path %>% arrange(y)
path_matrix[, "rho"] <- 1 - 2^path_matrix[, "rho"]
cbind("N_eff" = 1 / (1 - path_matrix[, "rho"]), path_matrix) %>%
    round(4) %>%
    head(10)
```

We notice that the regressors of our expert model are always chosen, while the others sometimes are and sometimes aren't.

R.5 Correlation structures in R

R.5.1 Sample ACF and PACF

With R it is effortless to compute sample autocorrelation and sample partial autocorrelation functions. To do that we have to use the commands `acf` and `pacf`. R computes the corresponding sample ACF or PACF and creates plots, as seen before.

```
# Setup forecasting study design
N_window <- 730 # rolling window length
N_forecast <- dim(price_DST)[1] - N_window + 1 # size of forecasting study
S <- 24 # products per day

# Correlations in R: ACF, PACF examples
s <- 18 # for hour 17:00

## Plot
par(mfrow = c(1, 1))
acf(price_DST[, s], lag.max = 50, main = "Sample ACF: hour 17")
pacf(price_DST[, s], lag.max = 50, main = "Sample PACF: hour 17")
```

The `acf` and `pacf` have an optional parameter `lag.max` which we can use to set the maximal lag that has to be calculated. If we do not want to have the ACF or PACF plot, but instead want to work with the (partial) autocorrelations explicitly, we should use the `plot=FALSE` option. Then, `acf` and `pacf` create a list of output variables where the list element called `acf` contains the relevant information:

```
s <- 18
sample_acf <- acf(price_DST[, s], plot = FALSE, lag.max = 10)
sample_pacf <- pacf(price_DST[, s], plot = FALSE, lag.max = 10)
sample_acf$acf
sample_pacf$acf
```

Note that the first value of `sample.acf$acf` is always one (which corresponds to lag 0). For the pacf `sample.acf$pacf` the first element corresponds to lag 1 and is equal to the second element of `sample.acf$acf`.

A forecasting study can show if it is worth including a specific lag or not. Let's take a look at the plot of the PACF function for hour 18.

```
s<- 18
par(mfrow=c(1,1))
pacf(Y[,s], main = "Sample PACF: hour 18")
```

We observe that lags 1,2,4,5,6,7 seem to contain some significant information regarding the price value. Let us now perform an exemplary forecasting study for $s = 18$ to compare the forecasting performance of the expert model with a model that extends the expert model by adding the lags 4,5,6. For the purpose of this exercise, we will use the `forecast.expert` function with different `expert_lags` arguments. The study is analogous to the one performed before, so we are not going into details of the code again.

```
# Setup forecasting study design (cont'd)
oos_dates <- dates[D + 1:N_forecast] ## dates to do the forecasting study on
number_of_models <- 3
model_names <- c("true", "expert", "expert_extended")
expert_wd <- c(2, 7, 1) # weekday dummies: Mo, Sa, Su
expert_lags <- c(1, 2, 7) # lags: 1, 2, 7 (standard expert model)
expert_lags_extended <- 1:8 # lags: 1 - 8 (extended expert model)

# Setup objects to store forecasts
forecasts <- array(dim = c(N_forecast, S, number_of_models))
dimnames(forecasts) <- list(oos_dates, 1:S, model_names)

# Run study
for (n in 1:N_forecast) {
    # Rolling window index
    index <- 1:N_window + n - 1
    prices <- as.matrix(price_DST[index, ])
```

```
    # True model
    if (n < N_forecast) forecasts[n, , "true"] <- price_DST[max(index) + 1, ]

    # Model 1 - expert model
    forecasts[n, , "expert"] <- forecast_expert(
        Y = prices, days = dates[index],
        expert_wd = expert_wd,
        expert_lags = expert_lags
    )$f

    # Model 2 - extended expert model
    forecasts[n, , "expert_extended"] <- forecast_expert(prices,
        days = dates[index],
        expert_wd = expert_wd,
        expert_lags = expert_lags_extended
    )$f

    cat("->", round((n / N_forecast) * 100, 2), "% done", "\r")
}

# Calculate errors
errors <- sweep(
    x = forecasts[-dim(forecasts)[1], , ],
    MARGIN = 1:2,
    FUN = "-",
    forecasts[-dim(forecasts)[1], , "true"]
)
```

By running the following code, we calculate the MAE and RMSE values.

```
# Calculate MAE
mae <- apply(abs(errors), 2:3, mean)
ts.plot(mae[, 2] - mae[, 3], col = 1:3, type = "o")
title(main = "Difference expert - expert_extended")
abline(h = 0)
colMeans(mae)

# Calculate RMSE
rmse <- sqrt(apply(errors^2, 2:3, mean))
ts.plot(rmse[, 2] - rmse[, 3], col = 1:3, type = "o")
title(main = "Difference expert - expert_extended")
abline(h = 0)
colMeans(rmse)
```

We see that the additional lags result in lower error measures. This indicates that the results can still be improved. Another possibility of the utility check of the other lags is the coefficient analysis.

R.5.2 Cross-period dependencies

For computing sample cross-period dependencies $\widehat{\mathbb{C}\mathrm{or}}(Y_{d,s}, Y_{d-k,l})$, we write a function `get_cpacf`. It takes a price matrix `Y` and the lag `k` (or k) as input. The output is an $S \times S$ matrix with the corresponding $\widehat{\mathbb{C}\mathrm{or}}(Y_{d,s}, Y_{d-k,l})$ values for $s, l \in \{1, \ldots, S\}$. `get_cpacf` is given by

```
# Define function to calculate cross-period correlations
# (this is the correl between a certain hour and all previous hours)
get_cpacf <- function(y, k = 1) {
    cpacf <- matrix(, S, S)
    for (s in 1:S) {
        for (l in 1:S) {
            y_s <- y[(k + 1):length(y[, s]), s]
            y_l_lagged <- y[(k + 1):length(y[, s]) - k, l]
            cpacf[s, l] <- cor(y_s, y_l_lagged)
        }
    }
    return(cpacf)
}
```

First, we create the matrix `cpacf`. Then, we have two loops with the index variables s and l over $1:S$. The inner three lines only assign the two vectors that correspond to $Y_{d,s}$ and $Y_{d-k,l}$ of and compute the sample cross-period auto-correlations $\widehat{\mathbb{C}\text{or}}(Y_{d,s}, Y_{d-k,l})$.

We can easily compute the cross-period correlations for any k, e.g. $k = 1$ and look at the corresponding $S \times S$ matrix:

```
  k <- 1
  cpacf <- get_cpacf(tail(price_DST, 5 * D), k = k)
  round(cpacf, 2)
```

However, the interpretation of the results is quite tricky. It is much easier to plot the results in a matrix plot as given in Figure 18. To create such graph, we use the package `plotrix`:

```
library(plotrix)
```

The full code to create a plot as in Figure 18 is given by:

```
  if (!interactive) {
    pdf(paste("out/cpacf_k=", k, ".pdf", sep = ""), width = 8, height = 6)
}
par(mar = c(4.4, 4.4, 1, 4), family = "Times") ## plotting area and font
cred <- c(.5, 0, 0, 0, 1, 1, 1) ## red
cgreen <- c(.5, 0, 1, 1, 1, 0, 0) ## green
cblue <- c(.5, 1, 1, 0, 0, 0, 1) ## blue
crange <- c(-1, 1) ## range
## colors in plot:
matrix_red <- approxfun(
    seq(crange[1], crange[2], length = length(cred)), cred
)(cpacf)
matrix_green <- approxfun(
    seq(crange[1], crange[2], length = length(cgreen)), cgreen
)(cpacf)
matrix_blue <- approxfun(
    seq(crange[1], crange[2], length = length(cblue)), cblue
)(cpacf)

plot_cols <- rgb(matrix_red, matrix_green, matrix_blue, alpha = .5)

## colors for legend:
crange_legend <- seq(crange[1], crange[2], length = 100)
legend_red <- approxfun(
    seq(crange[1], crange[2], length = length(cred)), cred
)(crange_legend)
legend_green <- approxfun(
    seq(crange[1], crange[2], length = length(cgreen)), cgreen
)(crange_legend)
legend_blue <- approxfun(
    seq(crange[1], crange[2], length = length(cblue)), cblue
)(crange_legend)
legend_cols <- rgb(legend_red, legend_green, legend_blue, alpha = .5)
## create plot, without axis
color2D.matplot(cpacf,
    cellcolors = plot_cols,
    show.values = 2,
    vcol = rgb(0, 0, 0),
    vcex = .6,
    axes = FALSE,
    xlab = "l",
    ylab = "s"
)
axis(1, 1:S - .5, 1:S, cex.axis = .8) ## draw x-axis
axis(2, S:1 - .5, 1:S, cex.axis = .8, las = 2) ## draw y-axis
legend_labels <- formatC(crange[1] + (crange[2] - crange[1]) *
    seq(0, 1, length = 11), format = "f", digits = 2)
## draw legend based on 'par' plotting range coordinates
pardat <- par()
color.legend(
    pardat$usr[2] + 0.5, 0,
    pardat$usr[2] + 1, pardat$usr[2],
    paste0(" ", legend_labels),
```

```
    legend_cols,
    align = "rb", gradient = "y"
)
if (!interactive) {
    dev.off() ## closes/saves pdf
}
```

We can define a function that plots the correlation matrix automatically:

```
mat_plot <- function(mat,
                     labx = NULL,
                     laby = NULL,
                     plot_s_x = TRUE,
                     plot_s_y = TRUE,
                     ylab = "s",
                     xlab = "l", ...) {
    par(mar = c(4.4, 4.4, 1, 5.2), family = "Times") ## plotting area and font
    cred <- c(.5, 0, 0, 0, 1, 1, 1) ## red
    cgreen <- c(.5, 0, 1, 1, 1, 0, 0) ## green
    cblue <- c(.5, 1, 1, 0, 0, 0, 1) ## blue
    crange <- c(-1, 1) ## range
    ## colors in plot:
    matrix_red <- approxfun(
        seq(crange[1], crange[2], length = length(cred)), cred
    )(mat)
    matrix_green <- approxfun(
        seq(crange[1], crange[2], length = length(cgreen)), cgreen
    )(mat)
    matrix_blue <- approxfun(
        seq(crange[1], crange[2], length = length(cblue)), cblue
    )(mat)
    colors_plot <- rgb(matrix_red, matrix_green, matrix_blue, alpha = .5)
    ## colors for legend:
    crange_leg <- seq(crange[1], crange[2], length = 100)
    legend_red <- approxfun(
        seq(crange[1], crange[2], length = length(cred)), cred
    )(crange_leg)
    legend_green <- approxfun(
        seq(crange[1], crange[2], length = length(cgreen)), cgreen
    )(crange_leg)
    legend_blue <- approxfun(
        seq(crange[1], crange[2], length = length(cblue)), cblue
    )(crange_leg)
    colors_legend <- rgb(legend_red, legend_green, legend_blue, alpha = .5)
    ## create plot, without axis
    plotrix::color2D.matplot(mat,
        cellcolors = colors_plot,
        show.values = 2,
        vcol = rgb(0, 0, 0),
        vcex = .8,
        axes = FALSE,
        xlab = xlab,
        ylab = ylab,
        cex.lab = 1.3, ...
    )
    if (is.null(labx)) labx <- 1:dim(mat)[2] - plot_s_x
    if (is.null(laby)) laby <- 1:dim(mat)[1] - plot_s_y
    axis(1, 1:dim(mat)[2] - .5, labx, cex.axis = 1, las = 3)
    axis(2, dim(mat)[1]:1 - .5, laby, cex.axis = 1, las = 2) ## draw y-axis
    ## compute legend labels
    lableg <- formatC(crange[1] +
        (crange[2] - crange[1]) *
            seq(0, 1, length = 11), format = "f", digits = 2)
    ## draw legend based on 'par' plotting range coordinates
    pardat <- par()
    color.legend(pardat$usr[2] + 0.5, 0, pardat$usr[2] + 1, pardat$usr[4],
        paste(" ", lableg, sep = ""), colors_legend,
        align = "rb", gradient = "y"
    )
}
```

R.5.3 Partial correlations

Now we want to look at the code required to create a plot as in Figure 20(a). Therefore, we require a general partial correlation function which gives an estimator for $\rho_{X,Y|\boldsymbol{Z}} = \mathbb{C}\text{or}(X, Y|\boldsymbol{Z})$. We consider the function `pcor` for this purpose:

```
pcor <- function(x, y, z) {
  cor(
      lm.fit(cbind(1, z), y)$res,
      lm.fit(cbind(1, z), x)$res
  )
}
```

`pcor` takes three inputs, `x`, `y` and `z`. `x` and `y` are observed vectors of the same lenght. The object z is a matrix with the corresponding observed z values.

For illustration purpose we compute first $\mathbb{C}\text{or}(Y_{t,s}, Y_{t-3,s}|Y_{t-1,s}, Y_{t-2,s})$ with $s = 2$:

```
  maxlag <- 3
  index <- (1 + maxlag):(3 * 365)
  index_ext <- 1:(3 * 365)

  s <- 2
  x <- price_DST[index, s]
  y <- price_DST[index - 3, s]
  z <- cbind(price_DST[index - 1, s], price_DST[index - 2, s])
  pcor(x, y, z)
  pacf(price_DST[index_ext, s], lag = 3, plot = FALSE)
```

We observe that `pcor(x,y, z)` approximatively matches the partial autocorrelation at lag 3. The difference lies in the fact that the R function uses a different formula for the calculation, while our function is more general as it allows conditioning not only on lags, but on any other regressors.

As in the example, `z` can be an arbitrary information matrix. For the sample cross-period partial autocorrelations as illustrated in Figure 20 we have to condition on prices of the same day. As an example, we compute the sample cross-period partial autocorrelations $\widehat{\rho}_{Y_{d,s},Y_{d-1,S-1}|Y_{d-1,s}}$ of Figure 20(a). We store the sample cross-period partial autocorrelations in a matrix $\texttt{cp}_p acf_d iag$:

```
  cp_pacf_diag <- matrix(, S, S)
  for (s in 1:S) {
      for (l in 1:S) {
          x <- price_DST[index, s]
          y <- price_DST[index - 1, l]
          z <- cbind(price_DST[index - 1, s])
          cp_pacf_diag[s, l] <- pcor(x, y, z)
      }
  }
  round(cp_pacf_diag, 2)
```

We call it "diag" because we condition on the same hour of the previous day, so hence the "diagonal" of the cross-period correlation matrix.

The corresponding plot is given by

```
  if (!interactive) {
    pdf("out/partial_diagonal.pdf", width = 8, height = 6)
}
mat_plot(cp_pacf_diag)
title(main = "Cor(s,l-1|s-1)")
if (!interactive) dev.off() ## closes/saves pdf
```

We can also compute the sample the sample cross-period partial autocorrelations conditioned on the last value of the previous day $\widehat{\rho}_{Y_{d,s},Y_{d-1,S-1}|Y_{d-1,S},Y_{d-1,s}}$ of Figure 20(c).

```
  # Calculate PAC for all hours, conditioned on previous day's last hour
  ## Call it "last"
  cp_pacf_last <- matrix(, S, S)
  for (s in 1:S) {
      for (l in 1:S) {
```

```
        x <- price_DST[index, s]
        y <- price_DST[index - 1, l]
        z <- cbind(price_DST[index - 1, S])
        cp_pacf_last[s, l] <- pcor(x, y, z)
    }
}
round(cp_pacf_last, 2)

## Plot
if (!interactive) {
    pdf(paste("out/partial_last.pdf", sep = ""), width = 10, height = 6)
}
mat_plot(cp_pacf_last)
title(main = "Cor(s,l-1|S-1)")
if (!interactive) dev.off()
```

Furthermore, we can also compute the sample the sample cross-period partial autocorrelations conditioned on the previous day's, last value of the previous day $\widehat{\rho}_{Y_{d,s},Y_{d-1,S-1}|Y_{d-1,S}}$ of Figure 20(b).

```
# Calculate PAC for all hours, conditioned on previous day's same AND last hour
## Call it "diag_last"
cp_pacf_diag_last <- matrix(, S, S)
for (s in 1:S) {
    for (l in 1:S) {
        x <- price_DST[index, s] ## response
        y <- price_DST[index - 1, l] ## the object to analyse.
        z <- cbind(
            price_DST[index - 1, s],
            price_DST[index - 1, S]
        )
        cp_pacf_diag_last[s, l] <- pcor(x, y, z)
    }
}

## Plot
if (!interactive) {
    pdf(paste("out/partial_expert_last.pdf", sep = ""), width = 10, height = 6)
}
mat_plot(cp_pacf_diag_last)
title(main = "Cor(s,l-1|s-1, S-1)")
if (!interactive) dev.off()
```

However, for strict exploration of the dependency structure, we should condition on the full 'actually considered model', e.g., model (67):

```
# Setup conditioning for expert model lags
maxlag <- 7 # max lag to consider
index <- (1 + maxlag):(3 * 365) # index for lags for manual calculation
index_ext <- 1:(3 * 365) # index for pacf() function

# PAC of current hour to all hours conditioned on expert model lags: 1,2,7 + last
# hour of previous day
weekdays_num <- lubridate::wday(dates)
WD <- t(sapply(weekdays_num, "==", 1:7)) + 0
cp_pacf_expert_last <- matrix(, S, S)

for (s in 1:S) {
    for (l in 1:S) {
        x <- price_DST[index, s] # response
        y <- price_DST[index - 1, l] # the object to analyse
        z <- cbind(
            price_DST[index - 1, s],
            price_DST[index - 2, s],
            price_DST[index - 7, s],
            price_DST[index - 1, S], # capital S = 24th hour
            WD[index, c(2, 7, 1)]
        )
        cp_pacf_expert_last[s, l] <- pcor(x, y, z)
    }
```

```
}

## Plot
if (!interactive) {
    pdf(paste("out/partial_expert_last.pdf", sep = ""), width = 10, height = 6)
}
mat_plot(cp_pacf_expert_last)
if (!interactive) dev.off()
```

R.5.4 Unit roots

To check if the price exhibits unit-root-type behavior we can fit a SARIMAX model on it, as specified in (64). To do so, we first define a function that forecasts the price using the arima0() function in R, which needs the arguments order and seasonal to be specified. The order argument needs 3 values specified: the number of sequential AR (autoregressive) components, the number of differencing required, which corresonds to the unit root order, and the number of sequential MA (moving average) components required. The seasonal argument requires a list, where the first component is the same as for the order argument and the second component is the period. We will use order = c(1, 1, 1) and seasonal = list(order = c(0, 0, 1), period = 7). The function is defined as follows:

```
# Define function to forecast expert model using SARIMAX
## Fix arguments to run function line by line
Y <- price_DST[index, ]
days <- dates[index]

forecast_sarimax <- function(Y,
                             days,
                             wd = c(2, 7, 1),
                             order = c(1, 1, 1),
                             seasonal = list(
                                 order = c(0, 0, 1),
                                 period = 7
                             ),
                             method = "CSS") {
    # Setup object to store forecasts & coefs
    forecast <- numeric()
    coefs <- matrix(
        nrow = order[1] + order[3] + length(wd) + seasonal$order[1] + seasonal$
            order[3],
        ncol = S
    )

    # Get number of products per day
    S <- dim(Y)[2]

    # Get days vector including the day to forecast (oos)
    days_ext <- c(days, days[length(days)] + 1)

    # Week-day dummy matrix:
    weekdays.num <- lubridate::wday(days_ext) ## 1==Sun, 2==Mon, etc.
    WD <- t(sapply(weekdays.num, "==", 1:7)) + 0
    dimnames(WD) <- list(
        NULL,
        c("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat")
    )

    # Loop through hours
    for (s in 1:S) {
        mod <- arima0(Y[, s],
            xreg = WD[-dim(WD)[1], wd],
            order = order,
            seasonal = seasonal,
            method = "CSS"
        )
```

```
        coefs[, s] <- mod$coef
        forecast[s] <- predict(mod,
            newxreg = tail(WD[, wd], 1), n.ahead = 1, se = FALSE
        )
    }
    dimnames(coefs) <- list(names(mod$coef), 1:S)
    output <- list("forecasts" = forecast, "coefficients" = coefs)
    return(output)
}
```

We can then conduct a forecasting study that compares the expert model, expert model extended by the first 8 lags, the expert.last model and the SARIMAX model. The code is given by:

```
# Forecasting study with expert_last and sarimax
oos_dates <- dates[N_window + 1:N_forecast] ## dates to do the forecasting study on
number_of_models <- 5
model_names <- c(
    "true",
    "expert",
    "expert_extended",
    "expert_last",
    "sarimax"
)
expert_wd <- c(2, 7, 1)
expert_lags <- c(1, 2, 7)
expert_lags_extended <- 1:8

## Setup objects to store forecasts
forecasts <- array(dim = c(N_forecast, S, number_of_models))
dimnames(forecasts) <- list(oos_dates, 1:S, model_names)

## Run forecast
for (n in 1:N_forecast) {
    # test: n = 1368+1
    # Define rolling window index
    index <- 1:N_window + n - 1

    # Get prices
    prices <- as.matrix(price_DST[index, ])

    # True model
    if (n < N_forecast) forecasts[n, , "true"] <- price_DST[max(index) + 1, ]

    # Expert model
    forecasts[n, , "expert"] <- forecast_expert(
        Y = prices,
        days = dates[index],
        expert_wd = expert_wd,
        expert_lags = expert_lags
    )$f

    # Extended expert model, extended with lags 1-8
    forecasts[n, , "expert_extended"] <- forecast_expert(prices,
        days = dates[index],
        expert_wd = expert_wd,
        expert_lags = expert_lags_extended
    )$f

    # Expert.last model
    forecasts[n, , "expert_last"] <- forecast_expert_last(prices,
        days = dates[index],
        expert_wd = expert_wd,
        expert_lags = expert_lags_extended
    )$f

    # Expert SARIMAX model
    forecasts[n, , "sarimax"] <- forecast_sarimax(prices,
        days = dates[index]
    )$f
```

```
    cat("->", round((n / N_forecast) * 100, 2), "% done", "\r")
} # n
```

We can then calculate the errors (RMSE) and plot the difference between the expert model and the SARIMAX model for every hour.

```
# Calculate errors
errors <- sweep(
    x = forecasts[-dim(forecasts)[1], , ],
    MARGIN = 1:2,
    FUN = "-",
    forecasts[-dim(forecasts)[1], , "true"]
)

# Calculate RMSE
rmse <- sqrt(apply(errors^2, 2:3, mean))

# Plot Expert vs. Sarimax RMSE
ts.plot(rmse[, "expert"] - rmse[, "sarimax"], col = 1:3, type = "o")
abline(h = 0)
colMeans(rmse)
title(main = "Difference RMSE Expert - Sarimax for every hour")
```

We see that the SARIMAX model is always better than the expert model, meaning that taking unit root into account improves our model.

R.6 Seasonal structures in R

R.6.1 Weekly seasonality

We illustrate the methods using the electricity prices price_DST on the index set index as done for the expert model above:

```
  # Setup parameters for calculation
  N_window <- 730 # window length
  index <- seq_len(N_window) # window index

  Y <- price_DST[index, ] # get prices for window
  colnames(Y) <- 0:(S - 1) # set column names
  days <- dates[index] # get dates
```

For computing the weekly sample mean as illustrated in Figure 23 we apply the commands

```
  wmean <-
  apply(Y,
      MARGIN = 2, # Apply FUN to every column (hour)
      FUN = function(x) { # X ist one column of Y
          tapply(x, # Tapply groups according to Index and applies FUN
              INDEX = lubridate::wday(days, week_start = 1, label = TRUE, week_start
                  = 1),
              FUN = mean
          )
      }
  )
```

The core of the above code is the tapply function. It groups data according to an index and then applies a function to each group. In our case, we want to group according to the weekday. However, we can't use tapply directly, calculate all hours at once. Therefore we wrap it in an apply statement which provides the columns of Y to tapply one by one.

The next lines create a plot of wmean as given in Figure 23:

```
  # Plot weekday means
  if (!interactive) {
      pdf("out/04_seasonal_structures/weekly_mean_sample.pdf",
          width = 8, height = 6
      )
```

```
}
par(mar = c(4.9, 4.4, 1, 4), family = "Times") ## plotting area and font
plot(0:(S - 1),
    wmean[1, ],
    col = "white",
    ylim = range(wmean),
    cex.axis = 1.3,
    cex.lab = 1.3,
    xaxt = "n",
    xlab = "S",
    ylab = "weekly mean price in EUR/MWh",
    las = 2,
    sub = paste(min(days), max(days), sep = " to ")
)
grid()
axis(1, 0:(S - 1), 0:(S - 1), cex.axis = 1.3)
color <- rainbow(7)
for (i in 1:7) {
    lines(0:(S - 1), wmean[i, ], type = "o", lwd = 2, pch = i, col = color[i])
}
legend("topleft",
    rownames(wmean),
    lwd = 2, col = color, pch = 1:7, cex = 1.2, bg = "transparent"
)
if (!interactive) dev.off()
```

For computing the explicit solution of the periodic mean (or weekly mean) of the expert model given in equation (70) we use the code:

```
# Calculating the periodic mean of the expert model for s = 9
s <- 9 # index for hour 08:00

## Run expert model
expert_model <- forecast_expert(Y[index, ], days[index])

## Calculate implicit means via matrix algebra formula from the lecture
## Formula: A_s * m_s = c_s => solve for m_s
b <- expert_model$coef[, s] # get betas
A <- diag(1 - b[4], 7) # diagnoal of A matrix; b[4] corresponds to beta_s,3 in
    lecture
diag(A[c(2:7, 1), ]) <- -b[2] # b[2] corresponds to beta_s,1 in lecture
diag(A[c(3:7, 1:2), ]) <- -b[3] # b[3] corresponds to beta_s,2 in lecture
c_vec <- b[1] + c(b[5], 0, 0, 0, 0, b[6], b[7]) # c_vec corresponds to c_s in
    lecture;
## b[1] corresponds to beta_0 in lecture
```

There we store b as the vector of the estimated parameter of the expert model and assign the matrix $\boldsymbol{A}$ as A and $\boldsymbol{c}$ as c_vec.

The solution is now given by

```
solve(A) %*% c_vec # sol
ve for m_s
```

We observe that all 7 values are different. This shows, that even though we do not have a Tuesday, Wednesday, Thursday or Friday dummy in the model the means are usually different. The reason is the combined autoregressive structure. Still, in the very simple model (15) without autoregressive parameters, the mean for the electricity price on Tuesday, Wednesday, Thursday and Friday at a certain hour is the same.

Remember that above we computed the mean only for an expert model for only one time period (here an hour) s. For computing the weekly means for all hours we can use for instance the code:

```
# Compute implicit expert model means for all hours
expert_mean <- wmean # we recycle the wman matrix to preserve dims and names

for (s in 1:S) {
    b <- expert_model$coef[, s] ## beta note that index sets start counting at 1
```

```
    A <- diag(1 - b[4], 7)
    diag(A[c(2:7, 1), ]) <- -b[2]
    diag(A[c(3:7, 1:2), ]) <- -b[3]
    c_vec <- b[1] + c(b[5], 0, 0, 0, 0, b[6], b[7]) # c_vec corresponds to c_s in
        lecture;
    expert_mean[, s] <- solve(A) %*% c_vec
}

expert_mean
```

The computation of the conditional correlation with the weekday dummies of model (67) can be done using

```
# Partial correlations for expert_last specification
## PCOR of price and weekday dummies conditioned on lagged prices, weekday dummies
    1,6,7
weekdays_num <- lubridate::wday(dates, week_start = 1)
WD <- t(sapply(weekdays_num, "==", 1:7)) + 0
wd_pacf <- matrix(, S, 7)
maxlag <- 7
index <- index + maxlag

for (s in 1:S) {
    for (l in 1:7) {
        x <- price_DST[index, s] ## response
        y <- WD[index, l] ## the object to analyse: Weekday dummies
        z <- cbind(
            price_DST[index - 1, s],
            price_DST[index - 2, s],
            price_DST[index - 7, s],
            price_DST[index - 1, S],
            WD[index, c(1, 6, 7)]
        )
        wd_pacf[s, l] <- pcor(x, y, z)
    }
}
```

As in the previous section, we create a matrix type plot of the 24×7 matrix and analyse the correlation:

```
# Plot PACF
if (!interactive) {
    pdf("out/04_seasonal_structures/pacfs_wd.pdf",
        width = 8, height = 6
    )
}
par(mar = c(4.4, 4.4, 1, 7), family = "Times") ## plotting area and font
cred <- c(.5, 0, 0, 0, 1, 1, 1) ## red
cgreen <- c(.5, 0, 1, 1, 1, 0, 0) ## green
cblue <- c(.5, 1, 1, 0, 0, 0, 1) ## blue
crange <- c(-1, 1) ## range

## colors in plot:
matrix_red <- approxfun(
    seq(crange[1], crange[2], length = length(cred)), cred
)(wd_pacf)
matrix_green <- approxfun(
    seq(crange[1], crange[2], length = length(cgreen)), cgreen
)(wd_pacf)
matrix_blue <- approxfun(
    seq(crange[1], crange[2], length = length(cblue)), cblue
)(wd_pacf)
colorplot <- rgb(matrix_red, matrix_green, matrix_blue, alpha = .5)

## colors for legend:
crange_legend <-
    seq(crange[1], crange[2], length = 100)
legend_red <- approxfun(
    seq(crange[1], crange[2], length = length(cred)), cred
)(crange_legend)
```

```
  legend_green <- approxfun(
      seq(crange[1], crange[2], length = length(cgreen)), cgreen
  )(crange_legend)
  legend_blue <- approxfun(
      seq(crange[1], crange[2], length = length(cblue)), cblue
  )(crange_legend)
  legend_colors <- rgb(legend_red, legend_green, legend_blue, alpha = .5)

  ## create plot, without axis
  color2D.matplot(wd_pacf,
      cellcolors = colorplot,
      show.values = 2,
      vcol = rgb(0, 0, 0),
      vcex = .6,
      axes = FALSE,
      xlab = "l",
      ylab = "s"
  )
  axis(1, 1:S - .5, 1:S, cex.axis = .8) ## draw x-axis
  axis(2, S:1 - .5, 1:S, cex.axis = .8, las = 2) ## draw y-axis

  ## compute legend labels
  lableg <- formatC(
      crange[1] + (crange[2] - crange[1]) * seq(0, 1, length = 11),
      format = "f", digits = 2
  )

  ## draw legend based on 'par' plotting range coordinates
  pardat <- par()
  color.legend(
      pardat$usr[2] + 0.5, 0, pardat$usr[2] + 1,
      pardat$usr[4], paste(" ", lableg, sep = ""), legend_colors,
      align = "rb", gradient = "y"
  )
  if (!interactive) dev.off() ## closes/saves pdf
```

R.6.2 Annual seasonality

For defining the four-season indicator-based basis, we have to specify the four seasons. Therefore, we create a list which contains the months that are contained by a certain season.

```
  # Months grouped by seasons: spring, summer, autumn, winter
  month_list <- list(c(3, 4, 5), c(6, 7, 8), c(9, 10, 11), c(12, 1, 2))
```

For computing the means within a season, we use the code:

```
  season_names <- sapply(month_list, FUN = function(x) {
    paste(month.abb[x], collapse = "+")
})

amean <- array(,
    dim = c(S, length(month_list)),
    dimnames = list(1:24, season_names)
)

for (j in 1:dim(amean)[2]) {
    sindex <- lubridate::month(days) %in% month_list[[j]]
    amean[, j] <- colMeans(Y[sindex, ])
}
```

There, we first create the $S \times 4$ matrix amean which shall contain the corresponding mean values. Within the loop we compute the mean values for each season and each period of the day.

We can plot the results:

```
  if (!interactive) pdf("out/annual_mean_sample.pdf", width = 8, height = 6)
  par(mar = c(4.9, 4.4, 1, 4), family = "Times") ## plotting area and font
  plot(0:(S - 1),
```

```
    amean[, 1],
    col = "white",
    ylim = range(amean),
    cex.axis = 1.3,
    cex.lab = 1.3,
    xaxt = "n",
    xlab = "S",
    ylab = "annual mean price in EUR/MWh",
    las = 2,
    sub = paste(min(days), max(days), sep = " to ")
)
grid()
axis(1, 0:(S - 1), 0:(S - 1), cex.axis = 1.3)
color <- rainbow(dim(amean)[2])
for (i in 1:dim(amean)[2]) {
    lines(0:(S - 1),
        amean[, i],
        type = "o",
        lwd = 2,
        pch = i,
        col = color[i]
    )
}
legend("topleft",
    season_names,
    lwd = 2,
    col = color,
    pch = 1:dim(amean)[2],
    cex = 1.2,
    bg = "transparent"
)
if (!interactive) dev.off()
```

For the creation of smooth annual basis functions, we first define the constant $A = 365.24$

```
A<- 365.24 ## annual periodicity
```

Then, we define an index set Aindex where we want to compute the basis functions:

```
a_index <- 0:floor(2 * A + 1)
```

Now we can create the Fourier basis using

```
m <- 2
fourier_basis <- matrix(1, length(a_index), 2 * m + 1)
for (i in 1:m) {
    fourier_basis[, 2 * i] <- sin(a_index * 2 * pi * i / A)
    fourier_basis[, 2 * i + 1] <- cos(a_index * 2 * pi * i / A)
}
head(fourier_basis)
```

And plot the results

```
# Plot
if (!interactive) {
    pdf("out/04_seasonal_structures/annual_Fourier.pdf",
        width = 8, height = 5
    )
}
par(mar = c(4.9, 4.4, 1, 4), family = "Times") ## plotting area and font
plot(a_index, a_index,
    ylim = c(-1, 1.3),
    col = "white",
    xaxt = "n",
    xlab = "d",
    ylab = expression(B[i](d)),
    las = 2,
    cex.lab = 1.3,
    cex.axis = 1.3,
```

```
    main = "Annual Fourier"
  )
  grid(nx = NA, ny = NULL)
  xleg <- c(0, 1 / 2 * A, A, 3 / 2 * A, 2 * A)
  abline(v = xleg, lty = 3, col = grey(.5))
  axis(1, xleg, paste(xleg))
  color <- rainbow(m * 2)
  for (i in 1:(m * 2)) {
      lines(a_index,
          fourier_basis[, 1 + i],
          pch = i, col = color[i], lwd = 2, lty = i %% 2 + 1
      )
  }
  legend("topright",
      paste(c("sin(2pi*", "cos(2pi*"), sort(c(1:m, 1:m)), "/A)", sep = ""),
      lwd = 2, ncol = m, col = color, lty = 1:6 %% 2 + 1, bg = rgb(1, 1, 1, .8)
  )
  if (!interactive) dev.off()
```

For the B-spline basis we consider the use of the function get.pbas:

```
get_pbas <- function(b_index,
                     period = 24,
                     knot_distance = period / 4,
                     ord = 4) {
    ## ord=4 --> degree = 3 --> cubic splines
    ## knot_distance = equidist grid spacing
    lb <- 1
    ub <- period
    knots <- seq(lb - (0) * knot_distance,
        ub + (1) * knot_distance,
        by = knot_distance
    )
    derivs <- numeric(period)
    ## some stuff adjusted from pbc-package to be faster
    n_knots <- length(knots)
    a_knots <- c(
        knots[1] - knots[n_knots] + knots[n_knots - (ord - 1):1],
        knots, knots[n_knots] + knots[1:degree + 1] - knots[1]
    )
    basis_interior <- splines::splineDesign(
        knots = a_knots,
        x = seq_len(period), ord, derivs
    )
    basis_interior_l <- basis_interior[, 1:(ord - 1), drop = FALSE]
    basis_interior_r <- basis_interior[,
        (ncol(basis_interior) - ord + 2):ncol(basis_interior),
        drop = FALSE # keep dataframe even if only one col gets selected
    ]
    basis <- cbind(basis_interior[,
        -c(
            1:(ord - 1),
            (ncol(basis_interior) - ord + 2):ncol(basis_interior)
        ),
        drop = FALSE
    ], basis_interior_l + basis_interior_r)
    output <- t(array(t(basis), dim = c(dim(basis)[2], length(b_index))))
    return(output)
}
```

get.pbas is a function that takes an index set as input, such as the period period, the knot difference dK and the order of the B-spline ord. Note that the order of a B-spline is always the degree + 1. So we need an order of 4 for a cubic B-spline. The knot difference dK should be period/K if we want to get K basis functions.

Thus, the code for creating a periodic cubic B-spline basis with $K = 6$ basis functions and periodicity $A = 365.24$ is given by

```
  K <- 6 ##
```

```
bspline_basis <- get_pbas(a_index, period = A, knot_distance = A / K, ord = 4)
head(bspline_basis)
ts.plot(bspline_basis, col = rainbow(K))
```

We can plot the basis using

```
if (!interactive) pdf("out/annual_Bspline_cubic.pdf", width = 8, height = 5)
par(mar = c(4.9, 4.4, 1, 4), family = "Times") ## plotting area and font
plot(a_index,
    a_index,
    ylim = c(0, 1),
    col = "white",
    xaxt = "n",
    xlab = "d",
    ylab = expression(B[i](d)),
    las = 2, cex.lab = 1.3,
    cex.axis = 1.3
)
grid(nx = NA, ny = NULL)
xleg <- c(0, 1 / 2 * A, A, 3 / 2 * A, 2 * A)
abline(v = xleg, lty = 3, col = grey(.5))
axis(1, xleg, paste(xleg))
color <- rainbow(K)
for (i in 1:K) {
    lines(a_index,
        bspline_basis[, i],
        pch = i, col = color[i], lwd = 2, lty = i %% 2 + 1
    )
}
legend("topright",
    paste("i=", 1:K, sep = ""),
    lwd = 2, ncol = m, col = color, lty = 1:6 %% 2 + 1, bg = rgb(1, 1, 1, .8)
)
if (!interactive) dev.off()
```

R.7 External regressors in R

So far in our models we have only been using the information that is available in the data that we forecast, i.e. the autoregressive effects, seasonal structure and cross-period correlations. But the intuition suggests that there is definitely more information available in the external regressors. The most intuitive and at the same time popular in the electricity price forecasting are electricity demand, specific types of energy generation, fuel prices and price values from other markets. In this section we will take a look at full data and check the impact of some external regressors on the prices. First, we read the provided data using the code below.

```
data <- read.csv("../data/DE_utc.csv") # read data

# Now define index sets for training and test set
# The test set will cover (about) 3 years (3x365 days)
N_test <- 3 * 365 * 24
idx <- 1:(nrow(data) - N_test)
data <- data[idx, ]

head(data)
```

We see that the dataset contains many regressors, but not all of them are available before the Day-Ahead Auction, e.g. the actual load. Since we are interested in Day-Ahead prices, we consider only the data where the prices are available. After that, we prepare the dataset as we learned in the previous tutorials and plot the day-ahead prices for illustration purposes.

```
# Replace NAs with last known values
data <- zoo::na.locf(data, na.rm = FALSE)

# Extract time data
```

```r
time_utc <- data[, "DateTime"] |> as_datetime(tz = "UTC") # UTC time
time_lt <- data[, "DateTime"] |> as_datetime(tz = "CET") # local time (CET/CEST)

# Save the start and end-time
S = 24
start_end_time <- strptime(
    format(c(time_lt[1], time_lt[length(time_lt)]), "%Y-%m-%d %H:%M:%S",
        tz = "CET"
    ),
    format = "%Y-%m-%d %H:%M:%S",
    tz = "UTC"
)

# Create fake local time
time_fake_numeric <- seq(
    from = as.numeric(start_end_time[1]),
    to = as.numeric(start_end_time[length(start_end_time)]),
    by = 24 * 60 * 60 / S
)

time_fake <- as.POSIXct(time_fake_numeric, origin = "1970-01-01", tz = "UTC")

dates <- unique(as.Date(time_fake))

# Load function
source("DST.trafo.R")

# Apply function on data
data_DST_arr <- DST.trafo(
    X = data[, -1],
    Xtime = time_utc,
    Xtz = "CET"
) # may take a while

dimnames(data_DST_arr) <- list(as.character(dates), 1:S, colnames(data)[-1])

# Set solar <10 MWh to 0
data_DST_arr[, , c("Solar_Act")]
idx_small_solar <- (data_DST_arr[, , c("Solar_Act")] %>% colSums()) < 10
data_DST_arr[, idx_small_solar, c("Solar_Act")] <- 0
data_DST_arr[, idx_small_solar, c("Solar_DA")] <- 0

# Get price data
price_DST <- data_DST_arr[, , 1]
```

In this exercise we focus only on the external regressors such as the day-ahead forecasts of load, wind and solar generation. Thus, we extract them from the data.

```r
# Get data
dimnames(data_DST_arr)[[3]] # show names
## day-ahead price
price <- data_DST_arr[, , "Price"]
## day-ahead load
load <- data_DST_arr[, , "Load_DA"]
## day-ahead wind onshore
windon <- data_DST_arr[, , "WindOn_DA"]
## day-ahead wind offshore
windoff <- data_DST_arr[, , "WindOff_DA"]
## day-ahead solar
solar <- data_DST_arr[, , "Solar_DA"]
## day-ahead residual load
residualload <- load - windon - windoff - solar
```

Using the following code, we recreate Figure 29.

```r
# Plot each variable
s <- 10
par(mfrow = c(6, 1), mar = c(4.9, 4.9, 1, 1), family = "Times")
```

```
plot(dates, price[, s],
    type = "l", ylab = paste("Price in EUR/MWh at s =", s), xlab = "Time",
    main = paste("Price in EUR/MWh at s =", s)
)

plot(dates, load[, s],
    type = "l", ylab = paste("Load in MW at s =", s), xlab = "Time", col = "
        purple",
    main = paste("Load in MW at s =", s)
)

plot(dates, solar[, s],
    type = "l", ylab = paste("Solar in MW at s =", s), xlab = "Time", col = "gold"
        ,
    main = paste("Solar in MW at s =", s)
)

plot(dates, windon[, s],
    type = "l", ylab = paste("Wind_on in MW at s =", s), xlab = "Time", col = "
        dodgerblue",
    main = paste("Wind_on in MW at s =", s)
)

plot(dates, windoff[, s],
    type = "l", ylab = paste("Wind_off in MW at s =", s), xlab = "Time", col = "
        turquoise",
    main = paste("Wind_off in MW at s =", s)
)

plot(dates, residualload[, s],
    type = "l", ylab = paste("Residual load in MW at s =", s), xlab = "Time",
    main = paste("Residual load in MW at s =", s)
)
```

With the following code, we explore the price-fundamentals relationship.

```
# Bi-plot of price and variables
par(mfrow = c(2, 4), mar = c(4.9, 4.9, 1, 1), family = "Times")
plot(load[, s], price[, s],
    type = "p", ylab = paste("Price in EUR/MWh at s =", s),
    xlab = "Day-ahead Load",
    col = lubridate::wday(dates[1:7]),
    pch = 19
)

## day-ahead wind onshore
plot(windon[, s], price[, s],
    type = "p", ylab = paste("Price in EUR/MWh at s =", s),
    xlab = "Day-ahead Wind onshore",
    pch = 19
)
legend("topleft",
    as.character(lubridate::wday(dates[1:7], label = TRUE)),
    pch = 19, col = lubridate::wday(dates[1:7]),
)

## day-ahead wind offshore
plot(windoff[, s], price[, s],
    type = "p", ylab = paste("Price in EUR/MWh at s =", s),
    xlab = "Day-ahead Wind offshore",
    pch = 19
)
## day-ahead solar
plot(solar[, s], price[, s],
    type = "p", ylab = paste("Price in EUR/MWh at s =", s),
    xlab = "Day-ahead Solar",
    pch = 19
)
## day-ahead residual load
plot(residualload[, s], price[, s],
```

```
    type = "p", ylab = paste("Price in EUR/MWh at s =", s),
    xlab = "Day-ahead Residual load",
    col = lubridate::wday(dates[1:7]),
    pch = 19
)
legend("topleft",
    as.character(lubridate::wday(dates[1:7], label = TRUE)),
    pch = 19, col = lubridate::wday(dates[1:7]),
)
```

In the plots above, we observe that the residual load seems to explain the price better than the load. We can compare both regressors quickly as below.

```
par(mfrow = c(1, 2))
plot(load[, s], price[, s],
type = "p", ylab = paste("Price in EUR/MWh at s =", s),
xlab = "Day-ahead Load",
pch = 19
)

plot(residualload[, s], price[, s],
  type = "p", ylab = paste("Price in EUR/MWh at s =", s),
  xlab = "Day-ahead Residual load",
  pch = 19
)
par(mfrow = c(1, 1))

# Correlation load - price
cor(cbind(load[, s], price[, s]))

# Correlation residual load - price
cor(cbind(residualload[, s], price[, s]))
```

Often visual analysis may be misleading and not that precise. Thus, we calculate the covariance of both regressors with the prices. We see that the values are higher for the residual load.

Now, let us implement model (93) and estimate it using OLS. First, we set the environment and prepare the regression matrix and the response vector.

```
# Define get_lagged function
get_lagged_dt <- function(z, lags = 1, give_names = TRUE) {
    data.table::rbindlist(
        list(
            data.table::data.table(z),
            as.list(
                rep.int(
                    NA,
                    ifelse(is.null(dim(z)[2]), 1, dim(z)[2])
                )
            )
        )
    )[, data.table::shift(.SD, lags, give.names = give_names),
        .SDcols = seq_len(ifelse(is.null(dim(z)[2]), 1, dim(z)[2]))
    ]
}

# Define forecast expert extended model
## To test function, fix arguments:
dat <- data_DST_arr
days <- dates
expert_wd <- c(2, 7, 1)
price_s_lags <- c(1, 2, 7)
fuel_lags <- c(2)

forecast_expert_ext <- function(dat,
                                days,
                                expert_wd = c(2, 7, 1),
                                price_s_lags = c(1, 2, 7),
                                fuel_lags = c(2)) {
    # Remove the last Y observation (oos)
```

```r
# Bcause this is what we want to forecast, the model is not allowed to use it
dat[dim(dat)[1], , "Price"] <- NA

# Get no of products per day
S <- dim(dat)[2]

# Define object to store forecast
forecast <- numeric(S)

# Get days vector incl day to forecast
days_ext <- c(days, days[length(days)] + 1)

# Prepare weekday dummies
weekdays_num <- lubridate::wday(days_ext) ## 1==Sun, 2==Mon, etc.
wd <- t(sapply(weekdays_num, "==", 1:7)) + 0
dimnames(wd) <- list(
    NULL,
    c("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat")
)

# External regressors - names for subsetting:
da_forecast_names <- c("Solar_DA", "WindOn_DA", "WindOff_DA", "Load_DA")
# fuel_names <- c("EUA_fM", "Coal_fM", "Coal_fQ", "Coal_fY", "Oil_fM", "Gas_fD
    ", "Gas_fM", "Gas_fQ", "Gas_fY")
fuel_names <- c("EUA_fM", "Coal_fM", "Oil_fM", "Gas_fM")

# Prepare fuel lags (lag of 2, since closing price it not known for t, but for
     t-1)
## Note: y_t+1 = y_t + fuel_t-1, because fuel_t is the closing price, so it is
     not
## known until the next day; we don't know it at t, so we need to take the
    closing price
## of 2 days before the response (t+1) => t+1-2 = t-1
mat_fuels <- get_lagged_dt(dat[, 1, fuel_names], fuel_lags)

# Delete last observation since it corresponds to oos price
mat_fuels <- mat_fuels[-nrow(mat_fuels), ]

# Get price of last hour of yesterday
price_last <- matrix(dat[, S, "Price"]) # get price
price_last <- get_lagged_dt(price_last, 1) # lag it
price_last <- price_last[-nrow(price_last), ] # remove last obs (NA)

dimnames(price_last) <- list(NULL, "Price_last") # define names

# Object to store coefficients
coefs <- matrix(
    nrow = length(expert_wd) + length(price_s_lags) +
        length(fuel_names) * length(fuel_lags) +
        length(da_forecast_names) + 2, #+2 for: intercept, price_last
    ncol = S
)

# Loop through hours
for (s in 1:S) {
    # test: s = 1

    # Get actual price
    acty <- matrix(dat[, s, "Price"], dimnames = list(NULL, "Price"))

    # Get price lags
    mat_price_lags <- get_lagged_dt(acty, price_s_lags)
    mat_price_lags <- mat_price_lags[-nrow(mat_price_lags), ] # remove last
        obs (NA)

    # Get DA forecasts
    mat_da_forecasts <- dat[, s, da_forecast_names]

    # Create full reg matrix
    regmat <- as.matrix(
```

```
            cbind(
                y = dat[, s, "Price"], # response is first col of reg matrix
                intercept = 1,
                wd[, expert_wd],
                mat_price_lags,
                price_last,
                mat_da_forecasts,
                mat_fuels
            )
        )

        # Index without NA and without last row (since it corresponds to oos price
            )
        act_index <- which((rowSums(is.na(regmat[-nrow(regmat), ])) == 0))

        # Fit model
        model <- lm.fit(regmat[act_index, -1], regmat[act_index, 1])
        model$coef[is.na(model$coef)] <- 0 ## deal with singularities

        coefs[, s] <- model$coef
        forecast[s] <- model$coef %*% regmat[dim(regmat)[1], -1]
    }

    rownames(coefs) <- names(model$coef)

    output <- list("forecasts" = forecast, "coefficients" = coefs)

    return(output)
}

# Run model
N_window <- 730
index <- 1:N_window

model <- forecast_expert_ext(
    dat = data_DST_arr[c(index, max(index) + 1), , ],
    days = dates[index]
)

print(model$forecasts)
```

Note that this function needs one more observation than usual. This is only due to the fact, that we include forecasts as regressors which do not have to be lagged. Therefore, we need the forecast corresponding to the day which we want to forecast (which is still $D+1$). We want to stress here that we only include this extra observation to build the dependend variables for forecasting correctly and we only need this due to the day-ahead forecasts included in our data. Importantly we do not use the observation at $D+1$ for estimation and for evaluation.

We can easly estimate the model using scaled regressors by replacing the estimation part as follows:

```
# Scale regmatrix
regmat_scaled <- scale(regmat[act_index, ])

# Fit model
model <- lm.fit(
    regmat_scaled[, -1], regmat_scaled[, 1]
)
```

That is, we can use the coef matrix to recreate Figure 37.

```
# Plot coefficients of scaled model
par(mfrow = c(1, 1)) # Setup frame
coefmat <- model$coefficients # get coefs
coef_names <- rownames(coefmat) # get coef names

ss <- 1:S - 1 # days vector
lty <- (seq_along(coef_names) > 7) + 1 # line types
```

```
actcol <- rainbow(length(coef_names)) # colors
par(mar = c(4.6, 4.6, 1, 1), family = "Times") # plotting area and font
plot(ss, ss,
    ylim = range(coefmat),
    type = "n",
    col = actcol, lwd = 2,
    xlab = "s", ylab = "Coefficients", xaxt = "n", las = 2,
    cex.lab = 1.3, cex.axis = 1.3
)
grid()
abline(0, 0, col = rgb(.5, .5, .5))
axis(1, ss, ss)
for (i.p in 1:dim(coefmat)[1]) {
    lines(ss, coefmat[i.p, ], col = actcol[i.p], lwd = 2, lty = lty[i.p])
}
legend("topright",
    coef_names,
    lwd = 2,
    col = actcol, bg = rgb(1, 1, 1, .8), cex = 1.2, ncol = 2, lty = lty
)
```

R.8 Non-linear effects in R

For the purpose of estimating the tail index, we use the hill() function from the evir package in R.

```
library(evir)
?hill
```

The function plots an estimator of the tail index α depending on the order statistics of the data. It is quite important for interpretation of the plot to set the end parameter correctly. This parameter sets the highest number of order statistics, for which the tail index α will be calculated and plotted. Setting it too high or too low will result in a bad estimation. Based on a rule of thumb, we want to set the end parameter in such a way that the α estimate looks more or less constant. Using this rule of thumb we found that $2\sqrt{N}$ is a sufficient value for the parameter, where N is the length of the data that we explore.

As an example we consider the t-distribution with different degrees of freedom k. Remember that for the t-distribution it holds that $\alpha = k$. First, we plot the histograms of scaled data generated from the t-distribution and one for the scaled electricity prices.

```
# Plot histogram of a t-distributed random variable for different df
## Note: for t-distribution alpha = degrees of freedom (df)
## Note: from the variance formula: tdf/(tdf-2) it is clear that for tdf=2 the
    variance does not exist

# Set params
set.seed(12345)
par(mfrow = c(2, 2))

tdf <- 4
hist(rt(length(price), df = tdf) / sqrt(tdf / (tdf - 2)), # division by stdev to
    standardize
    breaks = "Scott", freq = FALSE,
    main = paste("t with df =", tdf), xlab = "x"
) ## mean 0, variance 1
tdf <- 3
hist(rt(length(price), df = tdf) / sqrt(tdf / (tdf - 2)), # division by stdev to
    standardize
    breaks = "Scott", freq = FALSE,
    main = paste("t with df =", tdf), xlab = "x"
) ## mean 0, variance 1
tdf <- 2.2
hist(rt(length(price), df = tdf) / sqrt(tdf / (tdf - 2)), # division by stdev to
    standardize
```

```
    breaks = "Scott", freq = FALSE,
    main = paste("t with df =", tdf), xlab = "x"
) ## mean 0, variance 1
hist(scale(price),
    breaks = "Scott", freq = FALSE,
    main = "Scaled price series", xlab = "x"
) # mean 0, variance 1
```

We see clearly that the lower the number of degrees of freedom, the heavier the tails. Moreover, the histogram of the scaled price series looks similarly to the histograms of the t-distribution with $k = 3$ and $k = 4$. Now, we want to estimate the tail index using the Hill estimator.

```
# Estimator tail index with Hill-estimator for different df
set.seed(12345)
par(mfrow = c(2, 2))
tdf <- 4
hill(rt(length(price), df = tdf),
    end = 2 * sqrt(length(price))
)
abline(h = tdf, col = 4)
title(paste("Hill for t with df =", tdf))

tdf <- 3
hill(rt(length(price), df = tdf),
    end = 2 * sqrt(length(price))
)
abline(h = tdf, col = 4)
title(paste("Hill for t with df =", tdf))

tdf <- 2.2
hill(rt(length(price), df = tdf),
    end = 2 * sqrt(length(price))
)
abline(h = tdf, col = 4)
title(paste("Hill for t with df =", tdf))

## price data
hill((price - median(price)), start = log(length(price)), end = 2 * sqrt(length(
    price)))
abline(h = c(3, 4), col = 4, lty = 2)
title(paste("Hill for price data"))
```

We see that the estimation works correctly, i.e. the tail index is estimated to be close to the true one. Let us note that the Hill estimator works very well with lower tail indices and not that good with higher ones. Anyway, in the range of interest the performance is satisfying.

Alternatively, we can also estimate the tail index by fitting a t-distribution to our price data using maximum likelihood. This can be done via the gamlss package. The estimated degrees of freedom are ca 3.

```
# Alt method: using maximum likelihood to fit a t-distribution
model <- gamlss(price ~ 1, family = "SST")
predict(model, what = "tau", type = "response")

MASS::fitdistr(price, "t") # show t-distribution
## Note: the df is the tail index. here: ca 3
```

Now we want to reconstruct Figure 40. Let us note that we use there (among others) a quadratic effect of our price series. We checked that the tail index of the prices is higher than 4, so the quadratic effect's tail index is higher than 2. This means that the variance exists, and we will not run into any trouble, except likely the slow convergence. Remember that we want to create a simple model regressing the prices on the residual load and one additional term consisting of a transformation of the residual load. We will prepare the data first:

```r
# Get data
index <- 1:dim(data_DST_arr)[1]
days <- dates[index]
Y <- data_DST_arr[index, , "Price"] ## price
X <- data_DST_arr[index, , "Load_Act"] -
    apply(
        data_DST_arr[index, , c("Solar_Act", "WindOn_Act", "WindOff_Act")],
        c(1, 2), sum
    ) ## residual load

# Generate data frame
df <- as.data.frame(cbind(price = Y[index, s], resload = X[index, s]))

plot_price_resload <- function() {
  par(mfrow = c(1, 1), mar = c(4.9, 4.4, 1, 1), family = "Times")
  plot(df$resload, df$price,
      cex.axis = 1.3, cex.lab = 1.3,
      xlab = expression(X["d,s"]),
      ylab = expression(Y["d,s"]),
      las = 1,
      sub = paste(min(days), max(days), sep = " to ")
  )
  grid()
}
plot_price_resload()
```

Now we are ready to create the plots.
We will begin with the quadratic term and continiue with the $|x-20|$ and $|x-50|$ terms.

```r
# Linear Model
plot_price_resload()
mod <- lm(price ~ resload, data = df)
lines(df_resload_oos_range$resload,
    predict(mod, newdata = df_resload_oos_range),
    type = "l", lwd = 2, col = 2
)

# Linear Model with quadratic term
mod <- lm(price ~ poly(resload, 2), data = df)
lines(df_resload_oos_range$resload, predict(mod, newdata = df_resload_oos_range),
    type = "l", lwd = 2, col = 3)

# Linear Model with absolue value term at x=20
mod <- lm(price ~ resload + I(abs(resload - 20)), data = df)
lines(df_resload_oos_range$resload, predict(mod, newdata = df_resload_oos_range),
    type = "l", lwd = 2, col = 4)
legend("topleft",
    c("linear model", "with add. quadr. term", paste("with add. |x-", 20, "| term"
        , sep = "")),
    title = paste("s =", s - 1), lwd = 2, col = c(2, 3, 4), pch = NA, cex = 1.2,
    bg = rgb(1, 1, 1, .8)
)

# Linear Model with absolue value term at x=50
plot_price_resload() # new plot
mod <- lm(price ~ resload, data = df) # linear model
lines(df_resload_oos_range$resload, predict(mod, newdata = df_resload_oos_range),
    type = "l", lwd = 2, col = 2) # plot linear model

mod <- lm(price ~ resload + I(abs(resload - 50)), data = df)
lines(df_resload_oos_range$resload, predict(mod, newdata = df_resload_oos_range),
    type = "l", lwd = 2, col = 4)
legend("topleft",
    c("linear model", paste("with add. |x-", 50, "| term", sep = "")),
    title = paste("s =", s - 1), lwd = 2, col = c(2, 4), pch = NA, cex = 1.2,
    bg = rgb(1, 1, 1, .8)
)
```

Let us note that in the code above we simply fit two models for each plot.

The only thing worth closer attention is the I() function. The function makes the operators like +, -, * preserve their arithmetical meaning in the formula. Without the I() function they would be treated as formula operators, e.g. to point the interactions between the regressors.

R.9 Dealing with overfitting in linear models in R

R.9.1 Dimension reduction using SVD/PCA

In the lecture, we learned about the dimension reduction technique using principal component analysis (PCA) which is strongly related to the singular-value decomposition (SVD) method for matrices. To understand PCA, we will apply the method manually step-by-step for our expert model. To do so, we first setup our data and parameters and generate the regressor matrix as usual

```
# Setup params
s <- 10
N_window <- 730
index <- 8:N_window

# Subset data
dat <- data_DST_arr[index, , ]
dat[dim(dat)[1], , "Price"] <- NA
days <- dates[head(index, -1)]

# Setup expert params
expert_wd <- c(2, 7, 1) # Mon, Sat, Sun
expert_lags <- c(1, 2, 7)
fuel_lags <- c(2)

# Define object to store forecasts
forecast <- numeric(S)

# Define extended days vector, including the day to forecast
days_ext <- c(days, days[length(days)] + 1) # including the day to forecast

# Prepare regression matrix parts that are the same for all hours:
# Prepare weekday dummy matrix
weekdays_num <- lubridate::wday(days_ext)
wd <- t(sapply(weekdays_num, "==", 1:7)) + 0
dimnames(wd) <- list(
    NULL,
    c("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat")
)

wd <- wd[, expert_wd]

# Prepare lagged fuel prices
xfuel <- get_lagged_dt(dat[, 1, c("EUA_fM", "Coal_fM", "Oil_fM", "Gas_fM")], fuel_
    lags)

# Prepare last day's price, min and max price of last day
xprice <- cbind(
    dat[, S, "Price"],
    apply(dat[, , "Price"], 1, min),
    apply(dat[, , "Price"], 1, max)
)

dimnames(xprice) <- list(NULL, paste("Price", c("last", "min", "max"), sep = "_"))
xprice <- get_lagged_dt(xprice, 1)

# Prepare regression matrix parts that differ across hours:
# Get prices
acty <- as.matrix(dat[, s, "Price"])
dimnames(acty) <- list(NULL, paste("Price", s, sep = ""))

# Get lagged prices
xprice_s_lag <- get_lagged_dt(acty, expert_lags)
```

```
# Get DA forecasts
ext_names <- c("Solar_DA", "WindOn_DA", "WindOff_DA", "Load_DA")
xda_forecast <- dat[, s, ext_names]
dimnames(xda_forecast) <- list(NULL, ext_names)

Xreg <- as.matrix(cbind(
    y = dat[, s, "Price"], # response is first column of Xreg
    intercept = 1,
    wd,
    xprice_s_lag[-nrow(xprice_s_lag), ],
    xprice[-nrow(xprice), ],
    xda_forecast,
    xfuel[-nrow(xfuel), ]
)) ## regressor matrix

# Remove NAs & last value
act_index <- which(!apply(is.na(Xreg), 1, any)[-dim(Xreg)[1]]) # without NA's and
    last

# Scale all cols that are not constant
act_col <- apply(Xreg[act_index, ], 2, sd) > 0
act_col[1] <- FALSE # do not scale response
Xreg_scaled <- scale(Xreg[act_index, act_col])
```

Remember that we require scaled regressors for PCA, since this advanced estimation method evaluated the variation of the individual variables. In the last chunk we scale all columns and leave the intercept out. In this design we included the response as the first column of the regressor matrix, so we exclude this from the scaling as well. Now, we perform the SVD decomposition on the scaled regressor matrix as in Equation 145.

```
# Do SVD decomposition on scaled Xreg
Xreg_scaled_SVD <- svd(Xreg_scaled) #  X = U D V',
# alt: Xreg_scaled.pca<- princomp(Xreg_scaled)
```

Next, we can calculate the scores matrix as in Equation 146:

```
# Calc scores
scores_matrix <- Xreg_scaled_SVD$u %*% diag(Xreg_scaled_SVD$d)
```

Then we can calculate all possible PCA models, i.e. we calculate $\widehat{\gamma}_{s,\lambda}^{\text{ls-PCA}}$, for every λ as in Equation 148 to get the PCA path like so:

```
# This is the step-by-step way, we have a faster solution down below
# Define object to store PCA path
pca_path_matrix <- matrix(0, ncol(Xreg_scaled), ncol(Xreg_scaled))

# Loop over all columns of Xreg_scaled
for (idx_col in 1:ncol(Xreg_scaled)) {
    # test: idx_col = 1
    model <- lm(scale(Xreg[act_index, 1]) ~ scores_matrix[, 1:idx_col] - 1) # run
        PCA model
    pca_path_matrix[1:idx_col, idx_col] <- model$coef # store coefs
}
pca_path_matrix # Note: see orthogonal design
```

However, this is redundant because since the resulting matrix is orthogonal, all the smaller models are nested in the biggest model. In other words, all columns of the PCA path matrix are just subsets of the last column. Thus, we can simply calculate the model and then subset it to get the PCA path matrix:

```
# Simpler and much faster: as SVD is orthogonal, just run complete OLS
scaled_y <- scale(Xreg[act_index, 1]) # scale y
mod <- lm(scaled_y ~ scores_matrix - 1) # run model on whole matrix

# Generate PCA path matrix
gpca_path_matrix <- array(mod$coef, dim = rep(length(mod$coef), 2))
gpca_path_matrix[lower.tri(gpca_path_matrix)] <- 0
## Note: this is the same as the manual solution above: pca_path_matrix
```

We can now calculate the betas from the PCA path matrix as in $\widehat{\widehat{\boldsymbol{\beta}}}^{\text{ls-PCA}}_{s,\lambda} = \mathcal{V}_{s,\lambda}\widehat{\boldsymbol{\gamma}}^{\text{ls-PCA}}_{s,\lambda}$, for every λ:

```
# Get betas from PCA path
sol_pca_path_matrix <- Xreg_scaled_SVD$v %*% gpca_path_matrix
```

Now we can plot the PCA path as in figure 43:

```
# Plot
ts.plot(t(cbind(0, sol_pca_path_matrix)), col = color_codes, lwd = 2, lty =
    linetype_codes, xlab = "beta_k")
legend("topleft", paste("k=", 1:p, ",", dimnames(Xreg_scaled)[[2]][act_svd], sep =
     ""), title = paste("s =", s - 1), col = color_codes, lwd = 2, bg = rgb(1, 1,
    1, .8), ncol = 2, lty = linetype_codes, cex = 1.2)
```

But note that these values are scaled. So if we want to calculate predictions, we have to rescale the betas as shown in equations following equation 17:

```
# Get forecasts
# Unscale betas
Xreg_scaled_center <- attr(Xreg_scaled, "scaled:center")
Xreg_scaled_scale <- attr(Xreg_scaled, "scaled:scale")

betas_unscaled <- rbind(
    scale_center_y - scale_scale_y * t(Xreg_scaled_center) %*% solve(diag(Xreg_
        scaled_scale)) %*% sol_pca_path_matrix,
    scale_scale_y * solve(diag(Xreg_scaled_scale)) %*% sol_pca_path_matrix
)

# Calculate forecasts by multiplying last row of Xreg with betas_unscaled
forecasts <- Xreg[nrow(Xreg), -1] %*% betas_unscaled
data_DST_arr[index[length(index)], s, "Price"] # compare to true value
```

We can plot the in-sample fit of the different PCA models:

```
# Plot in-sample error
plot(colMeans(abs(Xreg[act_index, 1] - fitted_values)), type = "o", xlab = "p (
    columns of scores matrix)", ylab = "MAE", main = paste("PCA component in-
    sample error for s =", s - 1))
```

To get the best model, we can use information criteria as in. For example, here we use the BIC to select the optimal lambda:

```
# Get model via BIC
errors_by_lambda <- Xreg[act_index, 1] - fitted_values
rss <- colSums(errors_by_lambda * errors_by_lambda) # residual sum of squares
K <- colSums(gpca_path_matrix != 0) # number of active parameters () (degrees of
    freedom)
D <- dim(Xreg[act_index, ])[1] ## number of observations
kappa <- log(D) # BIC= log(D) # HQC= 2*log(log(D)) # AIC= 2
ic <- log(rss) + K * kappa / D # See (38) in the script
pca_path_opt <- as.matrix(gpca_path_matrix[, which.min(ic)]) ## IC-chosen beta
lambda_opt <- which.min(ic)
abline(v = lambda_opt, col = "blue") # plot optimal p
legend("topright", "BIC optimum", col = "blue", lwd = 2, bg = rgb(1, 1, 1, .8),
    cex = 1.2)
```

R.9.2 LASSO and Ridge estimators

More complicated models require advanced estimation techniques. In the lecture, we have learned about two methods that are very popular in the electricity price forecasting - lasso and ridge estimation. In this section, we will use the glmnet implementation in R to model and forecast electricity prices.

```
library(glmnet)
? glmnet
```

The package contains all the tools that we need for successful lasso or ridge estimation. First, let us take a look at the glmnet function from the package glmnet. The function performs an elastic net regularization in accordance with equation (157). As mentioned during the lecture, if we set the α parameter to 1, then we get the lasso estimation, if we set it to 0, we get the ridge estimation. Thus, using the glmnet function we can perform both lasso and ridge regression. Let us note that we provide the scaled response $\widetilde{\boldsymbol{Y}}_s$ and scaled regression matrix $\widetilde{\mathcal{X}}_s$ to the function, similarly as we would do for the function lm.fit. Moreover, the glmnet function can standardize the given regressors itself. However, for better interpretation of the fitted coefficients, it is advised to perform the scaling ourselves. A very crucial parameter is λ. We can either leave the choice of considered λ sequence to the algorithm or specify one. The latter one is the recommended option.

As an example, we work on the prices of product $s = 10$. For the first exercise, we simply take the expert model (7). Before, we estimated it using OLS, but now we will use the lasso and ridge estimators. To do it, we build the response vector and the regression matrix, as we did before.

```
# Model setup
s <- 10
N_window <- 730
index <- 8:N_window
Y <- price_DST[index, ]
days <- dates[index]
yreg <- Y[, s] ## response vector

# Days vector extended by day to forecast
days_ext <- c(days, days[length(days)] + 1)

# Weekday Dummies
weekdays_num <- lubridate::wday(days_ext)
wd <- t(sapply(weekdays_num, "==", 1:7)) + 0

dimnames(wd) <- list(
    NULL,
    c("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat")
)

# Expert weekday dummies
expert_wd <- c(2, 7, 1) # Mon, Sat, Sun

# Expert Lags
expert_lags <- c(1, 2, 7)

# Get lagged prices
xlag <- sapply(expert_lags, get_lagged, Z = Y[, s])
dimnames(xlag) <- list(NULL, paste("lag", expert_lags, sep = ""))

# Generate regression matrix
xreg <- cbind(xlag, wd[, expert_wd])
```

Now, we fit the models for both scaled and unscaled regressors.

```
# Get index of non-NA rows & remove last row (to forecast)
act_index <- which((rowSums(is.na(xreg[-dim(xreg)[1], ])) == 0))

# Fit unscaled models
model <- lm.fit(xreg[act_index, ], yreg[act_index]) # OLS
model_ridge <- glmnet(xreg[act_index, ], yreg[act_index], alpha = 0) # Ridge
model_lasso <- glmnet(xreg[act_index, ], yreg[act_index], alpha = 1) # Lasso

# Scaled models, relevant for interpretation
xreg_scaled <- scale(xreg[act_index, ])
yreg_scaled <- scale(yreg[act_index])
model_scaled <- lm.fit(xreg_scaled, yreg_scaled) # OLS
model_ridge_scaled <- glmnet(
    xreg_scaled,
```

```
    yreg_scaled,
    alpha = 0,
    standardize = F
) # ridge
model_lasso_scaled <- glmnet(
    xreg_scaled,
    yreg_scaled,
    alpha = 1,
    standardize = F
) # lasso
```

Using the fitted models, we could perform forecasting as we did before. The only problem is that we have obtained many coefficient sets, which depend on λ. The selection of the λ is key for successful lasso/ridge estimation. We handle this problem later in this section. Now, let us analyse the coefficients dependent on the values of λ and the $||\cdot||_1$ norm of the coefficient vector. To do this, we reproduce Figures 44 and 45 for the lasso estimator. For better interpretation, we use the model fitted to the scaled regressors.

```
# Plot LASSO path
beta <- model_lasso_scaled$beta # coefs
abs_beta <- apply(abs(beta), 2, sum) # vector abs (L1) norm, for x-axis
parameter_names <- dimnames(xreg_scaled)[[2]] # reg names
p <- dim(beta)[1] # dimension
col_p <- rainbow(p) # color
pch <- rep(1:6, length = p) # point type
```

Using the code above, we extract the coefficient matrix, the lambda values and the dimensions of the coefficient matrix. Using the data, we can reconstruct Figures 44 and 45.

```
par(mar = c(4.9, 4.9, 1, 4), family = "Times") ## plotting area and font
plot(abs_beta, beta[1, ] * NA,
    col = 1, ylim = range(beta), xlim = range(abs_beta),
    xlab = expression(paste("||", bold(beta)["s"], "||"[1])),
    ylab = expression(hat(beta)["s,k"]),
    cex.lab = 1.3, cex.axis = 1.3,
    sub = paste0(min(days), " to ", max(days), " at hour s=", s)
)
grid()
for (i in 1:p) {
    lines(abs_beta, beta[i, ],
        col = col_p[i], lwd = 2, pch = pch[i],
        type = "o"
    )
}
legend("topleft", paste0("k=", 1:p, ",", parameter_names),
    col = col_p, pch = pch, lwd = 2, bg = rgb(1, 1, 1, .8), ncol = 2
)
```

As mentioned before, the key to the successful lasso/ridge estimation is the proper choice of λ parameter. The preferable method of tuning the λ parameter is the k-fold cross-validation due to high robustness. On the other hand, this method causes a large computational effort which we can not always afford. Package glmnet consists of a function cv.glmnet which can perform a k-fold cross-validation or block-wise k-fold cross-validation.

```
?cv.glmnet
```

For standard k-fold cross-validation, the nfolds parameter is key, whereas for the block-wise version, the foldid is key. As an example, we perform a 10-fold cross-validation as below.

```
# Set seed
set.seed(1234)

# Set number of folds
nfold <- 10
```

```
# Do CV and get optimal lambda
mod0cv <- cv.glmnet(
    xreg[act_index, ],
    yreg[act_index],
    alpha = 1,
    nfold = nfold
)
print(mod0cv$lambda.min)

# Fit LASSO using optimal lambda
mod0 <- glmnet(
    xreg[act_index, ],
    yreg[act_index],
    alpha = 1,
    lambda = mod0cv$lambda.min
)
print(mod0$beta)

# Do forecast
beta_vec <- c(mod0$a0, as.numeric(mod0$beta)) # intercept, other coefs
c(1, xreg[dim(xreg)[1], ]) %*% beta_vec
```

The coefficients obtained this way can be easily used for the purpose of forecasting. We see that the cv.glmnet function makes it very easy to perform a cross-validation exercise, but it is recommended to perform the cross-validation manually for better understanding and control of the study. When the cross-validation causes too heavy computational effort, then it is reasonable to use the information criteria. In the code below, we use the most conservative information criteria, the BIC. It is especially useful when dealing with data containing very many regressors as it does not allow for overfitting.

```
# Fit lasso
model_lasso <- glmnet(
    xreg[act_index, ],
    yreg[act_index],
    alpha = 1,
    standardize = TRUE
)

# Get betas
beta <- as.matrix(model_lasso$beta)
beta <- rbind(intercept = model_lasso$a0, beta) # add intercept

# Get best model via BIC
errors_by_lambda <- yreg[act_index] -
    (cbind(1, xreg[act_index, ]) %*% as.matrix(beta)) # residuals
rss <- colSums(errors_by_lambda * errors_by_lambda) # residual sum of squares
K <- colSums(beta != 0) # number of active parameters () (degrees of freedom)
D <- dim(xreg[act_index, ])[1] ## number of observations
kappa <- log(D) # BIC= log(D) # HQC= 2*log(log(D)) # AIC= 2
ic <- log(rss) + K * kappa / D # See (38) in the script
lambda_opt <- which.min(ic)
beta_opt <- as.matrix(beta[, which.min(ic)]) ## IC-chosen beta

# View
model_lasso$lambda[lambda_opt]
beta_opt
```

This way we obtain the best coefficients in terms of BIC. Note that in the code above we can easily change BIC to any other information criteria. It is useful as well to plot the IC in order to get an intuition of its behavior.

Lastly we are going to estimate a lasso approach the expert.adv model 111. Note that we are using some of the objects defined above for creating the regression matrix.

```
# Model setup
```

```r
s <- 10
N_window <- 730
index <- 8:N_window
Y <- price_DST[index, ]
days <- dates[index]
yreg <- Y[, s] ## response vector

# Days vector extended by day to forecast
days_ext <- c(days, days[length(days)] + 1)

# Weekday Dummies
weekdays_num <- lubridate::wday(days_ext)
wd <- t(sapply(weekdays_num, "==", 1:7)) + 0

dimnames(wd) <- list(
    NULL,
    c("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat")
)

# Expert weekday dummies
expert_wd <- c(2, 7, 1) # Mon, Sat, Sun

# Expert Lags
expert_lags <- c(1, 2, 7)

# Get lagged prices
xlag <- sapply(expert_lags, get_lagged, Z = Y[, s])
dimnames(xlag) <- list(NULL, paste("lag", expert_lags, sep = ""))

# Add external variables:
# Get Fuels
xfuels <- get_lagged_dt(
    data_DST_arr[index, 1, c("EUA_fM", "Coal_fM", "Oil_fM", "Gas_fM")], 2
)

# Last price, min, max price
xprice_temp <- cbind(
    data_DST_arr[index, S, "Price"],
    apply(data_DST_arr[index, , "Price"], 1, min),
    apply(data_DST_arr[index, , "Price"], 1, max)
)
dimnames(xprice_temp) <- list(NULL, c("last", "min", "max")) # TODO: forecast
    expert ext

# Lag prices
xprice <- get_lagged_dt(xprice_temp, 1)

# Get extended index incl. day to forecast
index_ext <- c(index, max(index) + 1)

# Get DA forecasts
ext_names <- c("Solar_DA", "WindOn_DA", "WindOff_DA", "Load_DA")
ext <- data_DST_arr[index_ext, s, ext_names]

# Generate regression matrix
xreg <- as.matrix(cbind(y = c(yreg, NA), wd[, expert_wd], xlag, xprice, ext,
    xfuels))

# Get index of non-NA rows & remove last row (to forecast)
act_index <- which((rowSums(is.na(xreg[-dim(xreg)[1], ])) == 0))

# Scaled lasso/ridge, only relevant for interpretation
xreg_scaled <- scale(xreg[act_index, -1])
yreg_scaled <- scale(yreg[act_index])

# Fit LASSO
model_lasso_scaled <- glmnet(
    xreg_scaled,
    yreg_scaled,
    alpha = 1, # Lasso
```

```
    standardize = F
)
```

The figures can be created in the same way as already done above.

Tutorials in python

py Tutorials in python

Note: the following python code was last updated in 2022 and might differ from the R tutorials which are up-to-date. To reproduce the results from the R tutorials you may need to change some of the code, such as imported file names, data formats, data used, etc.

py.1 Getting started with simple models in python

After studying the subsequent subsections which cover the content from Section 3, you should be able to solve the Tasks 3.5.

py.1.1 Read the data and data preparation

We start our work with loading the packages that we will use in this script. This is a common and recommended practice in python to load all the used packages in the very beginning of the script.

```
# Load packages, functions
from my_functions import DST_trafo
import locale
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Before we start reading the electricity price data in python, we change the language settings. As we work in English, we change the python output system language to English. This can be done by evaluating

```
locale.setlocale(locale.LC_ALL, 'en_US.utf8') # English US Linux
```

for Linux systems,

```
locale.setlocale(locale.LC_ALL, 'EN_US') # English US Mac
```

for Mac systems and by

```
# English US Windows
locale.setlocale(locale.LC_ALL,  locale="English_United States.1252")
```

for Windows systems in the console. We change the language settings to guarantee the same output concerning the times and dates, e.g. the weekdays are named Monday, Tuesday, etc.

Now, we want to read the data to python and prepare the data for modelling purpose. There is no universal standard of the given data format. This has always to be checked by the modeller. However, in this tutorial every time series data is given with a time label. This time label gives always the time in the universal time code (UTC) which corresponds to the UK winter time.

During this course we will mainly work for illustration purpose with the German EPEX day-ahead prices. First, we read the provided file *DE_utc.csv*[30] into python using the command.

```
data = pd.read_csv("../data/Germany.csv")
data  # shows first and last entries
data.head()  # shows first entries
data.info() # display the structure of the data
```

and observe the structure of the provided data. We see in the first column the time label, in the next ones the EPEX price of Germany/Luxembourg, the corresponding trading volumes, day-ahead forecasts of power generation (solar, wind onshore and offshore) and of load, the actual values of the mentioned generation and load, and finally the prices of the European Union emission allowance and the fuels: coal, oil and natural gas.

In this tutorial, we work for illustration purpose mainly with the German EPEX prices which is given in the 2nd column. Let us store it as price.

```
id_select = 1
price = data.iloc[:, id_select]
# alt using column name
price = data.loc[:, "Price"]
price
```

The first column provides the time in UTC. Using the pd.to_datetime command we can convert a character string to date-time format.

```
time_utc = pd.to_datetime(data["DateTime"], utc=True, format="%Y-%m-%d %H:%M:%S")
time_utc
```

We can easily create a time series plot:

```
# Plot time series
plt.plot(time_utc, price)
plt.ylabel("Price [EUR/MWh]")
plt.xlabel("Time")
plt.show()
```

Remember that this data (so the vector price) is not clock-change adjusted.

Now, we have to get deeper into the problem of time formation. Even though this seems to be a bit nasty, every modeller and forecaster that deals with intraday data has to face this problem. So lets have a closer look into the structure of the datetime class. It is a class that essentially codes time on a nanosecond precision in UTC. Every time point is represented by an integer number, that increases by 1 for every nanosecond, starting counting at 1st January 1970 00:00:00 UTC. For some mathematics it is suitable to work with these representing numbers instead of the more complex structured datetime class, therefore we save it under time_numeric. Furthermore, we convert the time time_utc into the local time using dt.tz_convert(). Note that most of the European countries use Central European Time (CET), as time zone such as Germany. However, UK, Ireland and Portugal use Western European Time (WET) and the European countries Finland, the Baltic States, Ukraine, Moldova, Romania, Bulgaria, Greece and Cyprus use Eastern European Time (EET). We convert the time using:

```
time_numeric = pd.to_numeric(time_utc) # time as numeric
time_numeric  # the number of ns (10^-9 s) since 1970-01-01

local_time_zone = "CET"  # local time zone abbrv
time_lt = time_utc.dt.tz_convert(local_time_zone)
time_lt
```

When we model our prices, we want to make use of the regular seasonal pattern in our data, especially the fact that we observe S prices every day.

[30] eventually we either have to change the working folder before (using `os.chdir()`) or copy the file into the working folder (see `os.getcwd()`)

Due to the clock change we are facing a problem here. Therefore, we introduced in Section 3 the object $Y_{d,s}$ which gives the clock-change adjusted price on day d and period s. In order to make use of it, we have to consider a clock-change transformation for the prices and the time. First, we consider the transformation for the time. Initially we compute S given the data[31].

```
S = int(24 * 60 * 60 * 10 ** 9 / time_numeric.diff().max())
```

Now we want to introduce a time vector that every day has 24 hours. This is the case for time_utc, but not for time_lt (here some days have 23 or 25 hours). We can check it running the following code.

```
only_days_utc = time_utc.dt.date
only_days_utc_tab = only_days_utc.value_counts().sort_index()
only_days_utc_tab[only_days_utc_tab != 24]

# now the same for local time
only_days_lt = time_lt.dt.date
only_days_lt_tab = only_days_lt.value_counts().sort_index()
only_days_lt_tab[only_days_lt_tab != 24]
```

Unfortunately, time_utc does not take into account the local time zone. Therefore, we will introduce a new 'fake' local time-zone vector called time_S, matching our requirement to have S values each day, starting every day at 0:00. This time vector will be stored as UTC, but in fact it is not (!) UTC.

```
# save the start and end-time
start_end_time_S = time_lt.iloc[[0, -1]].dt.tz_localize(None).dt.tz_localize("UTC"
    )

# creating 'fake' local time
start_end_time_S_num = pd.to_numeric(start_end_time_S)
time_S_numeric = np.arange(
  start=start_end_time_S_num.iloc[0],
  stop=start_end_time_S_num.iloc[1]+24*60*60*10**9/S,
  step=24*60*60*10**9/S
)

# 'fake' local time
time_S = pd.Series(pd.to_datetime(time_S_numeric, utc=True))
dates_S = pd.Series(time_S.dt.date.unique())
```

We see that the transformation looks already relative complex. For the transformation of the price vector this is even more complex. Our transformation rule interpolates the missing hour in March linearly and it averages the double entries in October. This can be quite complicated, as this transformation is influenced by the time zone and S. Therefore, we simply import a function DST_trafo from the provided my_functions.py file (we did it in the very beginning of the section) which must be copied into the working folder. The function DST_trafo requires a vector (or matrix) of data in UTC format as argument X, a UTC time vector as argument Xtime and a time zone string as argument tz (where only CET is supported at the moment). For more information, please explore the my_functions.py file. The output will be an array, with the data/period format as introduced above.

The first dimension will be the length of the number of supplied dates, the second will be S and the third dimension will be the number of provided time series (one in our case). As we only proceed with one time series of electricity prices, we save the 2-dimensional array of prices as price_S:

```
price_S_out = DST_trafo(X=price, Xtime=time_utc, tz=local_time_zone)
price_S_out.shape

# save as dataframe
price_S = pd.DataFrame(price_S_out[..., 0])
```

[31] Alternatively S could be provided manually by S=24.

```
price_S
```

Similarly as above, we can create a time series plot

```
price_ts = price_S.values.flatten()
plt.rcParams.update({'font.size': 15})
plt.figure(figsize=(10, 5))
plt.plot(time_S, price_ts)
plt.xlabel("Time", fontsize=16)
plt.ylabel("Electricity Price [EUR/MWh]", fontsize=16)
plt.tight_layout()
plt.savefig("out/pricets.pdf")
plt.show()
```

which is almost the same as this one created above using time_utc and price, but clock-change adjusted. We can also prepare the plots for each hour, similarly as in Figure 7.

```
os.makedirs("out/pricets_S", exist_ok=True)
plt.rcParams.update({'font.size': 15})
for s in range(S):
  plt.figure(figsize=(10, 5))
  plt.plot(dates_S, price_S.iloc[:, s])
  plt.xlabel("Time", fontsize=16)
  plt.ylabel("Electricity Price at "+str(s)+":00 [EUR/MWh]",
             fontsize=16)
  plt.tight_layout()
  plt.savefig("out/pricets_S/pricets_S" + str(1000 + s+1) + ".pdf")
  plt.close()
```

In practice, of course the amount of available data is limited. When we start in the next section with simple forecast models, we will only use a restricted set of available data. So we suppose to have the first $D = 2 \times 365 = 730$ days (usually 2 years) of data available. Now, the target is to do a forecast for the next day $D + 1$ or, in the given situation, day $D + 1 = 2 \times 365 + 1 = 731$. Then, we can easily create an index set index that covers the available range of days.

```
D = 730
index = np.arange(D)
# the available data is:
price_S.iloc[index]
plt.plot(price_S.iloc[index, 0])
plt.scatter(dates_s[index], price_S.iloc[index, 0],
            facecolors='none', edgecolors='C0')
plt.ylabel("Price at 0:00 [EUR/MWh]")
plt.xlabel("Time")
plt.show()
```

The latter plot only shows the prices for the hour 0:00-1:00.

py.1.2 Naive models

Now, in addition to the packages used in Section py.1.1, we load the following.

```
from calendar import day_abbr
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
```

In this section, we want to write functions that take as input the available/provided data, in our case price_S.iloc[index] with the provided index set index. As output, we expect the forecast of the next days values. So within the function we will use the object Y as representative for provided data price_S[index,] with the same structure:

```
Y = price_S.iloc[index]
# then e.g.
Y.iloc[-1]
```

The last line gives the last S (or S) values of Y. This can be directly used to define a forecasting function for the **naive-S** model (1):

```
def forecast_naive_s(Y):
  return Y.values[-1]  # return the last row

forecast_naive_s(Y)
Y.tail()
```

forecast_naive_s is a function that takes the input Y. The return statement specifies the output object of any function, in our case Y.values[-1] which are the last (observed) S values in Y. If we apply forecast_naive_s to price_S.iloc[index] we directly get the last S values of price_S.iloc[index].

Similarly we can define the function for the naive7S model (2) which returns the prices from one week before the forecasted day as the price forecast:

```
def forecast_naive_7s(Y):
  return Y.values[-7]
```

We can compare it with the naiveS model:

```
Y.tail(8)
pd.DataFrame((forecast_naive_s(Y), forecast_naive_7s(Y)),
             index=["naive_s", "naive_7s"])
```

The implementation of the slightly more sophisticated naive model (3) requires a little more effort. Here we have to evaluate the weekday of the day d that we want to forecast. Therefore, we need some time/date information on the input matrix Y. Within the forecasting function that we will create, we will use the object days as representative for the date vector that corresponds to Y. For evaluating the weekday information of the provided dates days, we use the method dt.strftime() of the datetime class. The %a specifier returns the weekday shortcut (Mon, Tue, ..., Sun) of the input vector. So if we apply this to our full dates vector, we receive:

```
days = pd.to_datetime(dates_S[index], utc=True)
days.dt.strftime("%a")
```

However, precisely we do not need to evaluate all dates in the naive model (3), but only the date of the day to forecast. This we can do using:

```
(days.iloc[-1] + pd.DateOffset(1)).strftime("%a")
```

As it is more efficient to compare the numbers rather than characters, we will use the weekday() method to get the weekday information in numeric format. It returns the weekday of a time or date with values from 0 to 6 with "0" representing a Monday, "1" a Tuesday up to "6" for the Sunday. We add 1 to it for a better interpretation. Now we have all ingredients to define the forecast_naive function for model (3):

```
def forecast_naive(Y, days):
  weekday = (days.iloc[-1] + pd.DateOffset(1)
             ).weekday() + 1  # 1 - Mon, ... , 7 - Sun
  if weekday in [1, 6, 7]:
    # if Mon, Sat, Sun, use the price from previous week
    forecast = Y.values[-7]
  else:  # otherwise, use the price of yesterday
    forecast = Y.values[-1]
  return {"forecasts": forecast, "coefficients": None}

forecast_naive(price_S.iloc[index], days=dates_S[index])
```

Note that within the if command we use the function in which checks if the first object is contained in the second one which is in our case [1, 6, 7], so the days representing numbers.

py.1.3 Expert type models

Now we want to implement simple forecasting methods that are based on the estimation of a statistical model. The modelling approach that we consider most of the time is based on having S separate models, one model for each period of the day. Thus, we have to estimate S models and compute S forecasts. However, for illustration purpose we start with a single model for a single period s which is represented by $s \in \mathbb{S}$ in our notation. We simply choose s = 5, this matches the 6th period with the delivery of electricity between 5:00 and 6:00 (recall that in python we count from 0).

```
s = 5  # id for period / hour of the day
```

As mentioned before, we will first consider the linear model (15) that contains only day-of-the-week effects. Therefore, we have to define the weekday-dummies, so vectors that are always 0 except for the corresponding weekdays where they are 1. Moreover, we used the index set index before to specify the provided data range. Now, it is suitable to define an extended index set that covers the data range as index before, but also the day that we want to forecast. This is useful as for the evaluation we have to specify the week-day dummies not only for all the provided data, but also for the forecasted day. So we define an extended index set and compute the corresponding extended set of dates:

```
days_ext = days.append(pd.Series(days.iloc[-1] + pd.DateOffset(1),
                       index=[index[-1]+1]))
days_ext
```

Now, we want to define the weekday dummies. We have already used the weekday() method of the datetime class for dates. Here, we use it again, but we say additionally that it is a method of the datetime class as we use it on a pandas series.

```
weekdays_num = days_ext.dt.weekday + 1
weekdays_num
```

The latter row gives us the weekdays of the last values of days.ext. Now we define a matrix of all considered dummies. For convenience, we want to have the order Mon, Tue, ... Sun. We generate the weekday dummies only for the considered weekdays given in expert_wd list. For the simple model in equation (15) we only require the Monday, Saturday and Sunday column. Thus, we define the index set for the active weekdays in the expert model.

```
expert_wd = [1, 6, 7]  # 1=Mon, 6=Sat, 7=Sun
WD = pd.DataFrame([(weekdays_num == x) + 0 for x in expert_wd]
                  ).transpose()
WD.columns = [day_abbr[i-1] for i in expert_wd]
WD
```

The weird looking + 0 term transforms the resulting vector from logical (only True or False) to numeric which contains numbers (False $\to 0$ and True $\to 1$).

Now we can define the regression matrix XREG (defined as $\mathcal{X}_s$ in (16))

```
# regression matrix
XREG = pd.concat([pd.Series(np.ones(WD.shape[0]), name="intercept"),
              WD], axis=1)
XREG
```

It contains in the first column only 1's which model the intercept, the latter ones contain the selected weekday dummies. Similarly we define the response vector YREG (defined as $\mathcal{Y}_s$ in (16)):

```
YREG = Y.iloc[index, s] # response vector
YREG
```

Note that we evaluate Y on the index set index as we do not know the value for the day that we want to forecast. Now we can use the sm.OLS command to estimate the linear model. However, we regress YREG only on XREG except the last row:

```
mod1 = sm.OLS(YREG.values, XREG.iloc[:-1]).fit()
```

We can evaluate the summary statistics:

```
mod1.summary()
```

The model coefficients (the estimated parameters $\widehat{\boldsymbol{\beta}}_s^{\text{ls}}$) can be called by:

```
mod1.params
```

Instead of using the sm.OLS command we could have used the analytical formula (13) to estimate the parameters:

```
mod1a = np.linalg.inv(XREG.iloc[:-1].transpose()@XREG.iloc[:-1]
                      ) @ XREG.iloc[:-1].transpose() @ YREG.values
mod1a
```

With the model coefficients we can receive the forecast by

```
mod1.params @ XREG.iloc[-1]
```

Formally mod1.params is the estimated parameter vector $\widehat{\boldsymbol{\beta}}$ and XREG.iloc[-1] is the $\boldsymbol{X}_{D+1,s}$ which is in our case the last line of the regression matrix XREG (as we extended the index set in advance). Note that this is only the forecast for hour s. Obviously, we have to do S forecasts (for each period s of the day) to receive a full day-ahead forecast.

Below we do not use the function sm.OLS to estimate a linear model, but the LinearRegression from the more popular sklearn package. The function works similarly, so we can estimate a model by providing the regressor matrix and the response vector as above:

```
mod1b = LinearRegression(fit_intercept=False).fit(X=XREG.iloc[:-1],
                                                  y=YREG)
```

Let us note the fit_intercept parameter. By default it is set to True, so we set it to False as we already have the intercept in the XREG matrix. The model coefficients that can be called similarly as above:

```
mod1b.coef_
```

So the forecast is given by

```
mod1b.coef_ @ XREG.iloc[-1]
```

Now, we can do a quick microbenchmark run to check which function is the fastest.

```
# Now compare computation time
%timeit -o sm.OLS(YREG.values, XREG.iloc[:-1]).fit()
%timeit -o np.linalg.inv(XREG.iloc[:-1].transpose() @ XREG.iloc[:-1]
                         ) @ XREG.iloc[:-1].transpose() @ YREG.values
%timeit -o LinearRegression(fit_intercept=False).fit(X=XREG.iloc[:-1],
                                                     y=YREG)
```

No we want to turn towards the full expert model (7). Similarly to expert_wd, we define a set of considered index lags:

```
expert_lags = [1, 2, 7]
```

Now we consider the following function:

```
# function for getting 'lagged' vectors
def get_lagged(lag, Z):
  return np.concatenate((np.repeat(np.nan, lag), Z[:(len(Z) + 1-lag)]))
```

get_lagged simply lags the input vector Z by the input number lag while the length of the output vector is increased by 1. This increasing of the length by 1 simplifies our life later on for the forecasting, the reason is the same as above where we extended the index vector. Of course, we do not know the observations before our first observation in the vector Z. So when we lag our vector, we assume that the value before the first observation is unknown and will be represented by an NA -- the native representation in R of a missing value. For instance, have a look at

```
get_lagged(lag=3, Z=np.array([1, 2, 5, 6, 2, -3, 0, 9]))
```

Using the list comprehension with get_lagged and YREG, we get directly the required lagged vectors for our regression problem:

```
XLAG = pd.DataFrame(np.transpose([get_lagged(lag=lag, Z=YREG)
                                  for lag in expert_lags]),
columns=["lag "+str(lag) for lag in expert_lags])
XLAG
```

Similarly as above, we can now specify the linear regression. Now it contains, the intercept, the lagged (autoregressive) effects (XLAG) and the selected weekday dummies:

```
XREG = pd.concat([pd.Series(np.ones(WD.shape[0]), name="intercept"),
                  XLAG, WD], axis=1)
YREG  = price_S.iloc[index, s]
```

Now if we want to estimate the models as above using

```
mod2 = sm.OLS(YREG.values, XREG.iloc[:-1]).fit()
mod2b = LinearRegression(fit_intercept=False).fit(X=XREG.iloc[:-1],
                                                  y=YREG)
```

we will get an error message. This is because the regressor matrix contains NaN's. To handle this problem, we define the index set containing the indices of non-NaN values of XREG. This we do by

```
act_index = np.array(~ np.isnan(XREG).any(axis=1))
act_index[-1] = False  # no NAs and no last obs
```

To understand it, we have to have a look at the function from the inner to the outer commands. np.isnan(XREG) gives a matrix with True and False, depending if the entry is a NaN or not. np.isnan(XREG).any(axis=1) gives a vector, the corresponding element is True if at least one cell in a particular row took a NaN value, it is False if all elements are non-NaN. The~ (tilde) inverts the logical vector and the last line drops the last entry of the vector. Thus, act_index selects all available (meaning non-NaN) entries. With act_index we can estimate the model using the above functions

```
mod2 = sm.OLS(YREG[act_index[:-1]].values, XREG.iloc[act_index]).fit()
mod2.summary()

mod2b = LinearRegression(fit_intercept=False).fit(
  X=XREG.iloc[act_index], y=YREG[act_index[:-1]])
```

and compute the corresponding forecast:

```
print(mod2.params @ XREG.iloc[-1])
print(mod2b.coef_ @ XREG.iloc[-1])
```

Finally, we want to write a forecasting function that takes as input only the available data (including the time/date information) and provides the values of the forecast as output. So we define

```
def forecast_expert(Y, days, expert_wd=[1, 6, 7],
  expert_lags=[1, 2, 7]):
  S = Y.shape[1]
  forecast = np.repeat(np.nan, S)

  days_ext = days.append(pd.Series(days.iloc[-1] + pd.DateOffset(1),
                                   index=[index[-1]+1]))
  # preparation of weekday dummies including the day to forecast
  weekdays_num = days_ext.dt.weekday+1  # 1 = Mon, 2 = Tue, ..., 7 = Sun
  WD = np.transpose([(weekdays_num == x) + 0 for x in expert_wd])

  # preparation of lags :
  def get_lagged(lag, Z):
    return np.concatenate((np.repeat(np.nan, lag), Z[:(len(Z) + 1-lag)]))

  coefs = np.empty((S, len(expert_wd)+len(expert_lags)+1))

  for s in range(S):
    # prepare the Y vector
    YREG = Y.iloc[:, s].values

    # get lags
    XLAG = np.transpose([get_lagged(lag=lag, Z=YREG)
                         for lag in expert_lags])

    # combine to X matrix
    XREG = np.column_stack((np.ones(Y.shape[0]+1), WD, XLAG))

    act_index = ~ np.isnan(XREG).any(axis=1)
    act_index[-1] = False  # no NAs and no last obs
    model = LinearRegression(fit_intercept=False).fit(
      X=XREG[act_index], y=YREG[act_index[:-1]])

    forecast[s] = model.coef_ @ XREG[-1]
    coefs[s] = model.coef_

    regressor_names = ["intercept"]+[
      day_abbr[i-1]
      for i in expert_wd]+["lag "+str(lag)
      for lag in expert_lags]

    coefs_df = pd.DataFrame(coefs, columns=regressor_names)

  return {"forecasts": forecast, "coefficients": coefs_df}
```

To understand, how forecast_expert works, it helps to specify

```
Y = price_S.iloc[index]
days = pd.to_datetime(dates_S[index], utc=True)
```

and run the code within the function manually.

The first lines prepare the estimation, so e.g. weekday-dummy matrix WD is defined (which does not depend on s or s) and the expert model (expert_wd and expert_lags) are specified. The main part is the loop from 1 to S, for each period of the day. Within this loop we have to state the model with its regressor matrix XREG and the response vector YREG. Then, the model gets estimated and forecasted as above. We can use the function in the same way as for the naive model.

```
forecast_expert(Y=price_S.iloc[index],
days=pd.to_datetime(dates_S[index], utc=True))
```

We can also visualize the forecasts

```
f_expert = forecast_expert(
price_S.iloc[index], days=pd.to_datetime(dates_S[index], utc=True))
f_naive = forecast_naive(
price_S.iloc[index], days=pd.to_datetime(dates_S[index], utc=True))
df = pd.DataFrame([price_S.iloc[index[-1] + 1], f_naive["forecasts"],
f_expert["forecasts"]],
index=["actual", "naive", "expert"]).transpose()

plt.rcParams.update({'font.size': 15})
plt.figure(figsize=(10, 5))
markers = ['o', 's', 'D']
colors = ['tab:red', 'tab:green', 'tab:blue']
for i in range(df.shape[1]):
plt.plot(df.iloc[:, i], linestyle='dotted', marker=markers[i],
color='black', markerfacecolor=colors[i])
plt.ylabel("Price in [EUR/MWh]")
plt.xlabel("s")
plt.title(dates_S[index[-1]+1])
plt.tight_layout()
plt.grid()
plt.legend(['actual price', 'forecast naive', 'forecast expert'])
plt.show()
```

We can also compute the forecasting error

```
price_S.iloc[index[-1] + 1] - f_naive["forecasts"]
price_S.iloc[index[-1] + 1] - f_expert["forecasts"]
```

In the python code we additionally recreate Figures 9 and 10, but we do not cover it in this script.

py.1.4 Multivariate models

Now we want to build multivariate models. We begin by creating model (24) where the p-dimensional parameter vector $\boldsymbol{\beta}$ does not depend on s. First, in addition to the packages used before, we load the following module

```
from scipy import sparse
```

Now, we create the forecasting functions forecast_mvexpert1, forecast_mixexpert and forecast_mixexpert_sparse, and we define Y and days to run the code line by line.

```
Y = price_S.iloc[index]
days = pd.to_datetime(dates_S[index], utc=True)
```

We save the dimensions of the input object and create an extended days vector which includes the day to forecast. For convenience, we store the expert specification in two seperate objects and based on it the names of utilized regressors are created.

```
D = Y.shape[0]
S = Y.shape[1]

# expert specification
expert_wd = [1, 6, 7]  # 1=Mon, 2=Tue, ..., 7=Sun
expert_lags = [1, 2, 7]  # lags

regressor_names = (
["intercept"]
+ [day_abbr[i - 1] for i in expert_wd]
+ ["lag " + str(lag) for lag in expert_lags]
)

# days vector including the day to forecast
days_ext = days.append(
pd.Series(days.iloc[-1] + pd.DateOffset(1), index=[index[-1] + 1])
)
```

Now, we prepare the weekday dummies:

```
weekdays_num = days_ext.dt.weekday + 1  # 1 = Mon, 2 = Tue, ..., 7 = Sun
WD = np.transpose([(weekdays_num == x) + 0 for x in expert_wd])
```

We define again the get_lagged function.

```
def get_lagged(lag, Z):
return np.concatenate((np.repeat(np.nan, lag), Z[: (len(Z) + 1 - lag)]))
```

At this point we start creating our regression matrix. First, we do it for each s. Afterwards we merge the S matrices together:

```
XREGlist = []
YREGlist = []

# prepare regressors
for s in range(S):
YREGlist.append(Y.iloc[:, s].values)
XLAG = np.transpose([get_lagged(lag=lag, Z=YREGlist[s])
for lag in expert_lags])
XREGlist.append(np.column_stack((np.ones(D + 1), WD, XLAG)))
```

Now that we have lists holding all S regression matrices, we can bind them together by using the concatenate function from the numpy package.

```
XREG = np.concatenate([array[:D] for array in XREGlist])
XREGoos = np.concatenate([array[D:] for array in XREGlist])
YREG = np.concatenate(YREGlist)
```

Again, we have to detect missings in the XREG matrix and remove them prior to the estimation. Afterwards we make a prediction using the estimated coefficients and the XREGoos object which holds the out of sample data.

```
act_index = ~np.isnan(XREG).any(axis=1)

model = LinearRegression(fit_intercept=False).fit(
X=XREG[act_index], y=YREG[act_index]
)

forecast = XREGoos @ model.coef_
coefs_df = pd.DataFrame([model.coef_], columns=regressor_names)
```

As said, we save it as a function and use it in the following way

```
forecast_mvexpert1(Y=price_S.iloc[index],
days=pd.to_datetime(dates_S[index], utc=True))
```

Next, we create model (32), where the autoregressive parameters are constant while the parameters corresponding to the weekdays can vary over s. This is more complicated since our regression matrix is much larger. First, we repeat the previous steps until we define the objects, which store the regression matrix. In this case we have regressors that vary and that do not vary over s. We fill them similarly as before.

```
# list holding regressors that are constant
XBREGlist = []
# list holding regressors that vary across 0,...,S-1
XREGlist = []
# list holding corresponding dependend variable
YREGlist = []

for s in range(S):
YREGlist.append(Y.iloc[:, s].values)
XLAG = np.transpose([get_lagged(lag=lag, Z=YREGlist[s])
for lag in expert_lags])
XBREGlist.append(XLAG)
XREGlist.append(np.column_stack((np.ones(D + 1), WD)))
```

Then, we name the regressors and define the regression matrix XREG. We fill it with zeros first, and then we fill the non-zero values in a loop. We do it similarly for XREGoos, but for YREG we use the concatenate function as before.

```
regressor_names = ["lag " + str(lag) for lag in expert_lags]
regr = ["intercept"] + [day_abbr[i - 1] for i in expert_wd]
regressor_names += [
regressor + "_s" + str(s) for s in range(S) for regressor in regr
]

const_regr_n = np.concatenate(XBREGlist).shape[1]
vary_regr_n = np.concatenate(XREGlist, axis=-1).shape[1]

# matrix holding all regressors
XREG = np.zeros((D * S, const_regr_n + vary_regr_n))
XREG[:, :const_regr_n] = np.concatenate([array[:D]
    for array in XBREGlist])

# regressor matrix out of sample used for forecasting
XREGoos = np.zeros((S, const_regr_n + vary_regr_n))
XREGoos[:, :const_regr_n] = np.concatenate(
[array[D:] for array in XBREGlist])

YREG = np.concatenate(YREGlist)

for s in range(S):
XREG[
    s * D: (s + 1) * D,
    (s * int(vary_regr_n / S) + const_regr_n): (
    (s + 1) * int(vary_regr_n / S) + const_regr_n
    ),
  ] = XREGlist[s][:D]
XREGoos[
    s,
    (s * int(vary_regr_n / S) + const_regr_n): (
    (s + 1) * int(vary_regr_n / S) + const_regr_n
    ),
  ] = XREGlist[s][D:]
```

We have now a regression matrix with $24+24*3+3=99$ columns, and $D*S$ rows. We estimate the model in analogue to the procedure above:

```
# index without NA's
act_index = ~np.isnan(XREG).any(axis=1)

model = LinearRegression(fit_intercept=False).fit(
X=XREG[act_index], y=YREG[act_index]
)

# compute forecast
forecast = XREGoos @ model.coef_
coefs_df = pd.DataFrame([model.coef_], columns=regressor_names)
```

We save it as a function and use it in the following way

```
forecast_mixexpert(Y, days)
```

You may have noticed that our regression matrix XREG contains mainly zeros. In fact, $\approx 94.6\%$ of all entries are zero. Hence, we can save this information more efficiently using a so-called sparse matrix. Now, let's convert XREG into a sparse matrix and estimate the model again:

```
XREG_sparse = sparse.lil_matrix(XREG[act_index])

model = LinearRegression(fit_intercept=False).fit(
X=XREG_sparse, y=YREG[act_index])

# compute forecast
forecast = XREGoos @ model.coef_
```

We write the above adjustment as a new function forecast_mixexpert_sparse which we can call as every other function. The performance of these functions can be easily compared using %timeit:

```
%timeit -o forecast_mvexpert1(Y, days)
%timeit -o forecast_mixexpert(Y, days)
%timeit -o forecast_mixexpert_sparse(Y, days)
```

It shows, that the sparse version needs less than 50% of the computation time of the non-sparse version.

py.2 Forecasting and Evaluation in python

py.2.1 Forecasting study design

Now we want to do the forecasting study in Python. First, we load additional functions.

```
from scipy.stats import t
from forecast_functions import *
```

The t class from scipy.stats module contains all necessary methods to use with the t-distribution. The forecast_functions.py file contains all written by us forecast functions, which we now load not to create redundancy and type out every forecasting function again. Then, we define the required constants: the sample size D and the out-of-sample size N. Using this information, we save the target dates of the out-of-sample forecasting study accessing the dates_S vector:

```
D = 730
N = price_S.shape[0] - D
oos_dates = dates_S[D:N+D]

model_names = ["true", "naive", "expert", "mv_expert", "mv_expert_mix"]
n_models = len(model_names)
```

The last two lines define the number of models that we want to compare in the forecasting study and provide the model names. Here we want to compare all models that we have created already. It is a reasonable idea to analyze the forecast values and the estimated coefficients of our model. The naive model has no coefficients, so that we will analyze only these of the more advanced models.

Now, we want to carry out the forecasting study. Therefore, we need objects that store all relevant information of our forecasting study. Therefore, we create one 3-dimensional array to assign our forecasts and a dictionary containing three arrays to store the estimated coefficients.

```
forecasts = np.empty((N, S, n_models))
forecasts[::] = np.nan
dim_names = [oos_dates.values, np.arange(S), model_names]

coeffs_list = {}
coeffs_list["expert"] = np.full((7, N, S), np.nan)
coeffs_list["mv_expert"] = np.full((7, N), np.nan)
coeffs_list["mv_expert_mix"] = np.full((99, N), np.nan)
```

These objects will be filled during the study with the corresponding forecasting errors and coefficients.

Using our forecasting functions, we can write down the code for the forecasting study easily:

```
for n in range(N):
# 'model' 1 is the true price to forecast (prediction target)
forecasts[n, :, 0] = price_S.iloc[D+n]

Y = price_S.iloc[n:D+n]
days = pd.to_datetime(dates_S[n:D+n], utc=True)

forecasts[n, :, 1] = forecast_naive(Y, days)["forecasts"]

fc_expert = forecast_expert(Y, days)
forecasts[n, :, 2] = fc_expert["forecasts"]
coeffs_list["expert"][:, n] = fc_expert["coefficients"].transpose()

fc_expert_mv = forecast_mvexpert(Y, days)
forecasts[n, :, 3] = fc_expert_mv["forecasts"]
coeffs_list["mv_expert"][:, n] = fc_expert_mv["coefficients"]

fc_mv_expert_mix = forecast_mixexpert_sparse(Y, days)
forecasts[n, :, 4] = fc_mv_expert_mix["forecasts"]
coeffs_list["mv_expert_mix"][:, n] = fc_mv_expert_mix["coefficients"]

print("\r-> "+str(np.round((n+1)/N*100, 4))+"% done", end='')
```

In the loop, we compute the forecasts, the corresponding errors and return the coefficients. As we have N days in the out-of-sample forecasting study, the loop runs from 0 to $N-1$ (as Python starts counting from 0).

Now we do need to do some post-processing. Specifically, we calculate the forecasting errors.

```
# Note that we can calculate a forecast for n = N but we don't know the
# true value. Thus we can't calculate errros for it.
errors = forecasts - forecasts[:, :, 0:1]
```

py.2.2 Coefficient analysis

Having saved the model's coefficients in the forecasting study, we can understand the impact of the regressors on the price formation in our data. The best way to do it is to view a plot of the coefficients over time. Below is an example of inspecting the coefficients of the expert model for $s = 10$.

```
s = 10
plt.plot(np.transpose(coeffs_list["expert"][..., s]))
plt.legend(fc_expert["coefficients"].columns,
bbox_to_anchor=(1.05, 1))
plt.title('Expert coefficients of hour '+str(s))
plt.xlabel("Time")
plt.show()
```

We receive a plot of the coefficients for the product $s = 10$. The intercept and the weekday dummies change over time, but the autoregressive coefficients seem relatively stable, close to 0. This is naturally not true. The impression comes from the difference in scale of both types of regressors. We didn't scale any of our regressors, so comparing and drawing both these types on one plot doesn't make much sense. Thus, we split them into two plots.

```
fig, axs = plt.subplots(2, sharex=True)
for coef in range(4):
axs[0].plot(np.transpose(coeffs_list["expert"][coef, :, s]),
color="C"+str(coef))
for coef in range(4, coeffs_list["expert"].shape[0]):
axs[1].plot(np.transpose(coeffs_list["expert"][coef, :, s]),
color="C"+str(coef))
fig.legend(fc_expert["coefficients"].columns,
bbox_to_anchor=(1.15, 0.9))
fig.suptitle('Expert coefficients of hour '+str(s))
plt.xlabel("Time")
plt.show()
```

Using the code above, we can observe the changes of the coefficients over time. It may be time-consuming, but it is worth spending time on this exercise to gain insights regarding structural breaks, changes in the dependency structure, etc. For instance, running the code above, we observe pretty stable parameter values until a few days before $d = 500$. There we see a sudden change in the coefficient values, which may suggest some structural break.
Then, until around $d = 700$, the importance of lagged values increases while it decreases for the dummies.

py.2.3 MAE and RMSE

The MAE and RMSE can be computed for all models at once. With the error array errors from the out-of-sample forecasting study, we receive the MAE and RMSE by

```
MAE = np.mean(np.abs(errors), (0, 1))
RMSE = np.sqrt(np.mean(errors**2, (0, 1)))
pd.DataFrame([MAE, RMSE], columns=model_names, index=["MAE", "RMSE"])
```

Of course, we can also evaluate with an hourly resolution:

```
MAE_hourly = np.mean(np.abs(errors), 0)
RMSE_hourly = np.sqrt(np.mean(errors**2, 0))
```

Using the obtained matrix, we can plot it over the products. This way we can easily check the forecasting performance of our models for all products.

```
plt.plot(RMSE_hourly)
plt.title("RMSE over hours")
plt.legend(model_names, ncol=3, loc=[0.05, 0.1])
plt.xlabel("Hour of the day")
plt.show()
```

py.2.4 The DM-Test

We define a very general function dm_test that carries out the Diebold-Mariano test. It takes as
input two error vectors of dimension N or matrices of dimension $N \times S$ (for the multivariate tests) that represent the errors of forecasts A and B. Moreover, it takes the optional parameters hmax which can be ignored for our purpose[32] and a parameter power which is 1 for the $\|\cdot\|_1$ test and 2 for the $\|\cdot\|_2$ test. The function is given by

```
def dm_test(error_a, error_b, hmax=1, power=1):
# as dm_test with alternative == "less"
loss_a = (np.abs(error_a)**power).sum(1)**(1/power)
loss_b = (np.abs(error_b)**power).sum(1)**(1/power)
delta = loss_a - loss_b
# estimation of the variance
delta_var = np.var(delta) / delta.shape[0]
statistic = delta.mean() / np.sqrt(delta_var)
delta_length = delta.shape[0]
k = ((delta_length + 1 - 2 * hmax + (hmax / delta_length)
* (hmax - 1)) / delta_length)**(1 / 2)
statistic = statistic * k
p_value = t.cdf(statistic, df=delta_length-1)

return {"stat": statistic, "p_val": p_value}
```

We see that the first two lines define loss_a and loss_b the vector of loss functions with elements that represent $L_{A,i}$ and $L_{B,i}$. The third line defines the loss difference called delta. The fourth line computes the variance of the loss difference and the fifth line computes the DM-test statistic statistic by taking the mean of delta and dividing it by its standard deviation. The next three lines adjust for the finite sample bias. The object p_value computes the p-value of the DM-test. Finally, dm_test returns a list with two elements, the test statistic, and the p-value.

[32] The parameter `hmax` determines the largest significant autocorrelation lag of the difference series, which is only required if we forecast more than a day ahead.

We can easily apply the dm_test by

```
dm_test(errors[..., 1], errors[..., 2])
dm_test(errors[..., 2], errors[..., 1])
```

Moreover, we can evaluate the DM-test for each period of the day separately by

```
p_values = np.full(S, np.nan)
for s in range(S):
p_values[s] = dm_test(errors[:, s:s+1, 4],
errors[:, s:s+1, 3], power=2)["p_val"]
```

We test model 5 (the multivariate expert-mix model) against model 4 (the fully multivariate model). The results can be analyzed using a simple plot.

```
plt.vlines(x=np.arange(S), ymin=0, ymax=p_values, lw=2)
plt.axhline(y=0.05, ls='--', color="tab:red")
plt.ylabel("P-values")
plt.xlabel("Hour")
plt.show()
```

It seems intuitive to use the dm_test function to test each model against all other models. To do so, we define a dictionary dm_results which is filled with two data frames that hold the p_values and the test statistics, respectively.

```
errors_wo_true = errors[..., 1:]
model_names_wo_true = [str(i+1)+"_"+model_names[i+1]
for i in range(len(model_names)-1)]

dm_results_df = pd.DataFrame(
columns=model_names_wo_true, index=model_names_wo_true,
dtype=np.float64)

dm_results = {"p_val": dm_results_df.copy(),
"t_stat": dm_results_df.copy()}
dm_results["p_val"].loc[:, "type"] = "p_val"
dm_results["t_stat"].loc[:, "type"] = "t_stat"
```

Now we can compute the desired tests using two nested for-loops:

```
for i_a in range(len(model_names_wo_true)):
mod_a = model_names_wo_true[i_a]
for i_b in range(len(model_names_wo_true)):
mod_b = model_names_wo_true[i_b]
if mod_a != mod_b:
dm = dm_test(errors_wo_true[..., i_a], errors_wo_true[..., i_b])
dm_results["p_val"].loc[mod_a, mod_b] = dm["p_val"]
dm_results["t_stat"].loc[mod_a, mod_b] = dm["stat"]
```

The code above fills the dictionary dm_results. That said, we have gathered the desired data. We can easily visualize our results using the matplotlib package:

```
# P-values
plt.rcParams.update({'font.size': 12})
fig = plt.figure(figsize=(5, 5))
plt.matshow(dm_results["p_val"].iloc[:, :-1],
fignum=fig.number, cmap=plt.cm.RdYlGn.reversed(),
vmin=0, vmax=0.1)
plt.xticks(ticks=np.arange(
errors_wo_true.shape[-1]), labels=model_names_wo_true, rotation=45)
plt.yticks(ticks=np.arange(
errors_wo_true.shape[-1]), labels=model_names_wo_true)
cb = plt.colorbar()
cb.ax.tick_params(labelsize=14)
plt.title("P-values of DM Test")
plt.show()

# test statistics
plt.rcParams.update({'font.size': 12})
fig = plt.figure(figsize=(5, 5))
plt.matshow(dm_results["t_stat"].iloc[:, :-1],
fignum=fig.number, cmap=plt.cm.seismic)
plt.xticks(ticks=np.arange(
errors_wo_true.shape[-1]), labels=model_names_wo_true, rotation=45)
plt.yticks(ticks=np.arange(
errors_wo_true.shape[-1]), labels=model_names_wo_true)
cb = plt.colorbar()
cb.ax.tick_params(labelsize=14)
plt.title("Test statistics of DM Test")
plt.show()
```

py.3 Correlation structures in python

py.3.1 Sample ACF and PACF

With Python it is quite easy to plot sample autocorrelation and sample partial autocorrelation functions. Therefore, we have to use the commands plot_acf and plot_pacf from statsmodels.graphics.tsa. Python computes the corresponding sample ACF or PACF and creates plots, as seen before.

```
s = 1
fig, axs = plt.subplots(2, sharex=True)
sm.graphics.tsa.plot_acf(price_S.values[:, s], lags=50, ax=axs[0],
title="Sample ACF: hour 2", marker=None)
sm.graphics.tsa.plot_pacf(price_S.values[:, s], lags=50, ax=axs[1],
title="Sample PACF: hour 2", marker=None)
plt.show()
```

The plot_acf and plot_pacf have an optional parameter lags which we can use to set the maximal lag that has to be calculated. If we do not want to have the ACF or PACF plot, but instead want to work with the (partial) autocorrelations explicitly, we should use the acf and pacf functions from statsmodels.tsa.stattools.

```
s = 17
sample_acf = sm.tsa.stattools.acf(price_S.values[:, s], nlags=10,
fft=False)
sample_acf

sample_pacf = sm.tsa.stattools.pacf(price_S.values[:, s], nlags=10)
sample_pacf
```

Note that the first value of sample_acf and sample_pacf is always one (which corresponds to lag 0).

A forecasting study can show if it is worth including a specific lag or not. Let's take a look at the plot of the PACF function for hour 18.

```
s = 17
sm.graphics.tsa.plot_pacf(price_S.values[:, s],
title="Sample PACF: hour 18", marker=None)
plt.show()
```

We observe that lags 1 to 7 seem to contain some significant information regarding the price value. Let us now perform an exemplary forecasting study for $s = 17$ to compare the forecasting performance of the expert model with a model that extends the expert model by adding the lags 3, 4, 5, 6. For the purpose of this exercise, we will use the forecast_expert function with different expert_lags arguments. The study is analogous to the one performed before, so we are not going into details of the code again.

```
D = 365*2
N = 365*3
oos_dates = dates_S[D:N+D]
model_names = ["true", "expert", "expert_extended"]
number_of_models = len(model_names)
expert_wd = [1, 6, 7]
expert_lags = [1, 2, 7]
expert_lags_extended = [1, 2, 3, 4, 5, 6, 7]

forecasts = np.full((N, S, number_of_models), np.nan)
dim_names = [oos_dates.values, np.arange(S), model_names]

for n in range(N):
# 'model' 1 is the true price to forecast (prediction target)
forecasts[n, :, 0] = price_S.iloc[D+n]

Y = price_S.iloc[n:D+n]
days = pd.to_datetime(dates_S[n:D+n], utc=True)

fc_expert = forecast_expert(Y=Y, days=days, expert_wd=expert_wd,
expert_lags=expert_lags)
forecasts[n, :, 1] = fc_expert["forecasts"]

fc_expert_extended = forecast_expert(Y=Y, days=days,
expert_wd=expert_wd,
expert_lags=expert_lags_extended)
forecasts[n, :, 2] = fc_expert_extended["forecasts"]

print("\r-> "+str(np.round((n+1)/N*100, 4))+"% done", end='')
```

By running the following code, we calculate the MAE and RMSE values.

```
errors = forecasts - forecasts[:, :, 0:1]
MAE_hourly = np.mean(np.abs(errors), 0)
RMSE_hourly = np.sqrt(np.mean(errors**2, 0))
```

We can now plot the differences in MAE and RMSE over hours and compare the overall results.

```
plt.plot(MAE_hourly[:, 1] - MAE_hourly[:, 2])
plt.title("MAE difference over hours")
plt.xlabel("Hour of the day")
plt.show()

plt.plot(RMSE_hourly[:, 1] - RMSE_hourly[:, 2])
plt.title("RMSE difference over hours")
plt.xlabel("Hour of the day")
plt.show()

pd.DataFrame([MAE_hourly.mean(0), RMSE_hourly.mean(0)],
columns=model_names, index=["MAE", "RMSE"])
```

We see that the additional lags result in lower error measures. This indicates that the results can still be improved. Another possibility of the utility check of the other lags is the coefficient analysis.

py.3.2 Cross-period dependencies

For computing sample cross-period dependencies $\widehat{\mathbb{C}\text{or}}(Y_{d,s}, Y_{d-k,l})$, we write a function get_cpacf. It takes a price matrix Y and the lag k (or k) as input. The output is an $S \times S$ matrix with the corresponding $\widehat{\mathbb{C}\text{or}}(Y_{d,s}, Y_{d-k,l})$ values for $s, l \in \{1, \ldots, S\}$. get_cpacf is given by

```
def get_cpacf(y, k=1):
S = y.shape[1]
n = y.shape[0]
cpacf = np.full((S, S), np.nan)
for s in range(S):
for l in range(S):
y_s = y[k:n, s]
y_l_lagged = y[:(n-k), l]
cpacf[s, l] = np.corrcoef(y_s, y_l_lagged)[0, 1]
return cpacf
```

First, we create the matrix cpacf. Then, we have two loops with the index variables s and l over range(S). The inner three lines only assign the two vectors that correspond to $Y_{d,s}$ and $Y_{d-k,l}$ and compute the sample cross-period auto-correlations $\widehat{\mathbb{C}\text{or}}(Y_{d,s}, Y_{d-k,l})$.

We can easily compute the cross-period correlations for any k, e.g. $k = 1$ and look at the corresponding $S \times S$ matrix:

```
k = 1
cpacf = get_cpacf(price_S.tail(5*D).values, k=k)
pd.DataFrame(cpacf).round(2)
```

However, the interpretation of the results is quite tricky. It is much easier to plot the results in a matrix plot as given in Figure 18. The full code to create a plot similar as in Figure 18 (the Figure was obtained using R) is given by:

```
plt.rcParams.update({'font.size': 12})
fig, ax = plt.subplots(figsize=(15, 15))
cax = ax.matshow(cpacf, cmap=plt.cm.rainbow, vmin=-1, vmax=1)
cb = fig.colorbar(cax, fraction=0.046, pad=0.04)
for (i, j), z in np.ndenumerate(cpacf):
ax.text(j, i, '{:0.2f}'.format(z), ha='center', va='center')
plt.xticks(ticks=np.arange(S), labels=np.arange(S), rotation=45)
plt.yticks(ticks=np.arange(S), labels=np.arange(S))
plt.tight_layout()
plt.xlabel("l")
plt.ylabel("s")
plt.savefig("out/cpacf_k="+str(k)+".pdf")
plt.show()
```

py.3.3 Partial correlations

Now we want to look at the code required to create a plot as in Figure 20(a). Therefore, we require a general partial correlation function which gives an estimator for $\rho_{X,Y|\boldsymbol{Z}} = \mathbb{C}\text{or}(X, Y|\boldsymbol{Z})$. We consider the function pcor for this purpose:

```
def pcor(x, y, z):
XREG = np.column_stack((np.ones(z.shape[0]), z))
model_y = LinearRegression(fit_intercept=False).fit(X=XREG, y=y)
model_x = LinearRegression(fit_intercept=False).fit(X=XREG, y=x)
cor = np.corrcoef(y - model_y.predict(XREG),
x - model_x.predict(XREG))[0, 1]
return cor
```

pcor takes three inputs, x, y and z. x and y are observed vectors of the same lenght. The object z is a matrix with the corresponding observed z values.

For illustration purpose we compute first $\mathbb{C}\text{or}(Y_{t,s}, Y_{t-3,s}|Y_{t-1,s}, Y_{t-2,s})$ with $s = 1$:

```
maxlag = 3
index = np.arange(maxlag, 3*365)
index_ext = np.arange(3*365)

s = 1
x = price_S.iloc[index, s]
y = price_S.iloc[index-3, s]
z = np.column_stack((price_S.iloc[index-1, s],
price_S.iloc[index-2, s]))

pcor(x, y, z)
sm.tsa.pacf(price_S.iloc[index_ext, s], nlags=3)
```

We observe that pcor(x, y, z) matches the partial autocorrelation at lag 3.

As in the example, z can be an arbitrary information matrix. For the sample cross-period partial autocorrelations as illustrated in Figure 20 we have to condition on prices of the same day. As an example, we compute the sample cross-period partial autocorrelations $\widehat{\rho}_{Y_{d,s},Y_{d-1,S-1}|Y_{d-1,s}}$ of Figure 20(a). We store the sample cross-period partial autocorrelations in a matrix cp_pacf_diag:

```
cp_pacf_diag = np.full((S, S), np.nan)
for s in range(S):
for l in range(S):
x = price_S.iloc[index, s]
y = price_S.iloc[index-1, l]
z = price_S.iloc[index-1, s]
cp_pacf_diag[s, l] = pcor(x, y, z)

pd.DataFrame(cp_pacf_diag).round(2)
```

The corresponding plot is given by

```
plt.rcParams.update({'font.size': 12})
fig, ax = plt.subplots(figsize=(15, 15))
cax = ax.matshow(cp_pacf_diag, cmap=plt.cm.rainbow, vmin=-1, vmax=1)
cb = fig.colorbar(cax, fraction=0.046, pad=0.04)
for (i, j), z in np.ndenumerate(cp_pacf_diag):
ax.text(j, i, '{:0.2f}'.format(z), ha='center', va='center')
plt.xticks(ticks=np.arange(S), labels=np.arange(S), rotation=45)
plt.yticks(ticks=np.arange(S), labels=np.arange(S))
plt.tight_layout()
plt.xlabel("l")
plt.ylabel("s")
plt.savefig("out/partial_diagonal.pdf")
plt.show()
```

We repeat the above using two other z matrices. In the first case, we take the last product of yesterday

```
z = price_S.iloc[index-1, S-1]
```

and in the second we combine the two previous z.

```
z = np.column_stack(
(price_S.iloc[index-1, s], price_S.iloc[index-1, S-1])
)
```

However, for strict exploration of the dependency structure, we should condition on the full 'actually considered model', e.g., model (67):

```
maxlag = 7
index = np.arange(maxlag, 3*365)
index_ext = np.arange(3*365)

days = pd.to_datetime(dates_S, utc=True)
weekdays_num = days.dt.weekday+1
expert_wd = [1, 6, 7]
WD = np.transpose([(weekdays_num == x) + 0 for x in expert_wd])

cp_pacf_expert_last = np.full((S, S), np.nan)
for s in range(S):
for l in range(S):
x = price_S.iloc[index, s]
y = price_S.iloc[index-1, l]
z = np.column_stack(
(price_S.iloc[index-1, s], price_S.iloc[index-2, s],
price_S.iloc[index-7, s], price_S.iloc[index-1, S-1],
WD[index])
)
cp_pacf_expert_last[s, l] = pcor(x, y, z)
```

py.4 Seasonal structures in python

py.4.1 Weekly seasonality

Now, we need additionally month_abbr function from calendar package.

```
from calendar import month_abbr
```

We illustrate the methods using the electricity prices price_S on the index set index as done for the expert model above:

```
D = 730
index = np.arange(D)

Y = price_S.iloc[index]
days = pd.to_datetime(dates_S[:D], utc=True)
Y.index = days
```

For computing the weekly sample mean as illustrated in Figure 23 we apply the commands

```
wmean = Y.groupby(by=Y.index.strftime("%A"), sort=False).mean()
days_order = [4, 5, 6, 0, 1, 2, 3]
wmean = wmean.iloc[days_order]
```

The core of the above code is the groupby method. It groups data according to a given vector and then using mean() method we calculate the average in each group. In our case, we want to group according to the weekday. As a result we get a data frame of average prices per weekday and per hour.

The next lines create a plot of wmean similarly as given in Figure 23:

```
plt.rcParams.update({'font.size': 12})
plt.figure(figsize=(10, 5))
markers = ["o", "^", "+", "x", "D", "v", "h"]
for day in range(len(markers)):
plt.plot(wmean.columns, wmean.iloc[day], marker=markers[day],
markerfacecolor='none')
plt.ylabel("Weekly mean price in EUR/MWh")
plt.xlabel("s, "+str(Y.index.min().date())+" to "+str(
Y.index.max().date()))
plt.tight_layout()
plt.grid()
plt.legend(wmean.index)
os.makedirs("out/04_seasonal_structures", exist_ok=True)
plt.savefig("out/04_seasonal_structures/weekly_mean_sample.pdf")
plt.show()
```

For computing the explicit solution of the periodic mean (or weekly mean) of the expert model given in equation (70) we use the code:

```
s = 5  # id for period
expert_model = forecast_expert(Y, days)
b = expert_model["coefficients"].iloc[s]  # beta
A = np.diag(np.repeat(1-b[-1], 7))
rng = np.arange(7)
A[rng, rng-1] = -b[-3]
A[rng, rng-2] = -b[-2]
vec = b[0] + np.array([b[1], 0, 0, 0, 0, b[2], b[3]])
```

There we store b as the vector of the estimated parameter of the expert model and assign the matrix $\boldsymbol{A}$ as A and $\boldsymbol{c}$ as vec. The solution is now given by

```
np.linalg.inv(A) @ vec
```

We observe that all 7 values are different. This shows, that even though we do not have a Tuesday, Wednesday, Thursday or Friday dummy in the model the means are usually different. The reason is the combined autoregressive structure. Still, in the very simple model (15) without autoregressive parameters, the mean for the electricity price on Tuesday, Wednesday, Thursday and Friday at a certain hour is the same.

Remember that above we computed the mean only for an expert model for only one time period (here an hour) s. For computing the weekly means for all hours we can use for instance the code:

```
# we recycle the wmean to preserve dims and names
expert_mean = wmean.copy()

for s in range(S):
b = expert_model["coefficients"].iloc[s]  # beta
A = np.diag(np.repeat(1-b[-1], 7))
rng = np.arange(7)
A[rng, rng-1] = -b[-3]
A[rng, rng-2] = -b[-2]
vec = b[0] + np.array([b[1], 0, 0, 0, 0, b[2], b[3]])
# mean
expert_mean.iloc[:, s] = np.linalg.inv(A) @ vec
```

The computation of the conditional correlation with the weekday dummies of model (67) can be done using

```
weekdays_num = pd.to_datetime(dates_S, utc=True).dt.weekday+1
WD = np.transpose([(weekdays_num == x) + 0 for x in range(1, 8)])
maxlag = 7
index = index + 7

wd_pacf = np.full((S, 7), np.nan)
for s in range(S):
for l in range(7):
x = price_S.iloc[index, s]
y = WD[index, l]
z = np.column_stack(
(price_S.iloc[index-1, s], price_S.iloc[index-2, s],
price_S.iloc[index-7, s], price_S.iloc[index-1, S-1],
WD[index][:, [0, 5, 6]])
)
wd_pacf[s, l] = pcor(x, y, z)
```

As in the previous section, we create a matrix type plot of the 24×7 matrix and analyse the correlation:

```
plt.rcParams.update({'font.size': 12})
fig, ax = plt.subplots(figsize=(8, 7))
cax = ax.matshow(wd_pacf, cmap=plt.cm.rainbow, vmin=-1, vmax=1,
aspect='auto')
cb = fig.colorbar(cax)
for (i, j), z in np.ndenumerate(wd_pacf):
ax.text(j, i, '{:0.2f}'.format(z), ha='center', va='center')
plt.xticks(ticks=np.arange(7), labels=expert_mean.index, rotation=45)
plt.yticks(ticks=np.arange(S), labels=np.arange(S))
plt.tight_layout()
plt.xlabel("l")
plt.ylabel("s")
plt.savefig("out/04_seasonal_structures/pacfs_wd.pdf")
plt.show()
```

py.4.2 Annual seasonality

For defining the four-season indicator-based basis, we have to specify the four seasons. Therefore, we create a list which contains the months that are contained by a certain season.

```
month_list = [[3, 4, 5], [6, 7, 8], [9, 10, 11], [12, 1, 2]]
```

For computing the means within a season, we use the code:

```
season_names = ["+".join([month_abbr[month_num]
for month_num in months])
for months in month_list]

amean = np.full((S, len(month_list)), np.nan)
for j in range(len(month_list)):
sindex = Y.index.month.isin(month_list[j])
amean[:, j] = Y[sindex].mean(0)
```

There, we first create the $S \times 4$ matrix amean which shall contain the corresponding mean values. Within the loop we compute the mean values for each season and each period of the day.

We can plot the results:

```
plt.rcParams.update({'font.size': 12})
plt.figure(figsize=(10, 5))
markers = ["o", "^", "+", "x"]
for j in range(len(month_list)):
plt.plot(np.arange(S), amean[:, j],
marker=markers[j], markerfacecolor='none')
plt.ylabel("Annual mean price in EUR/MWh")
plt.xlabel("s, "+str(Y.index.min().date())+" to "+str(
Y.index.max().date()))
plt.tight_layout()
plt.grid()
plt.legend(season_names)
plt.savefig(
"out/04_seasonal_structures/annual_mean_sample.pdf")
plt.show()
```

For the creation of smooth annual basis functions, we first define the constant $A = 365.24$

```
A = 365.24  # annual periodicity
```

Then, we define an index set a_index where we want to compute the basis functions:

```
a_index = np.arange(np.floor(2*A+2))
```

Now we can create the Fourier basis using

```
m = 2
fourier_basis = np.ones((len(a_index), 2*m+1))
for i in range(m):
fourier_basis[:, 2*i+1] = np.sin(a_index*2*np.pi*(i+1)/A)
fourier_basis[:, 2*i+2] = np.cos(a_index*2*np.pi*(i+1)/A)

pd.DataFrame(fourier_basis)
```

And plot the results

```
plt.rcParams.update({'font.size': 12})
plt.figure(figsize=(10, 5))
for i in range(1, fourier_basis.shape[1]):
plt.plot(fourier_basis[:, i], linestyle='--' if i % 2 == 1 else '-')
plt.ylabel("$B_i(d)$")
plt.xlabel("d")
plt.tight_layout()
plt.grid()
plt.legend([fun+"($2\pi\\frac{"+str(i+1)+"}{A})$" for i in range(2)
    for fun in ["sin", "cos"]])
plt.savefig(
"out/04_seasonal_structures/annual_Fourier.pdf")
plt.show()
```

For the B-spline basis we consider the use of the function get_pbas:

```
def get_pbas(Bindex, period=365.24, dK=365.24/6, order=4):
# ord=4 --> cubic splines
# dK = equidistance distance
# support will be 1:n
n = len(Bindex)
stp = dK
x = np.arange(1, period)  # must be sorted!
lb = x[0]
ub = x[-1]
knots = np.arange(lb, ub+stp, step=stp)
degree = order-1
Aknots = np.concatenate(
(knots[0] - knots[-1] + knots[-1-degree:-1], knots,
knots[-1] + knots[1:degree+1] - knots[0]))

from bspline import Bspline
bspl = Bspline(Aknots, degree)
basisInterior = bspl.collmat(x)
basisInteriorLeft = basisInterior[:, :degree]
basisInteriorRight = basisInterior[:, -degree:]
basis = np.column_stack(
(basisInterior[:, degree:-degree],
basisInteriorLeft+basisInteriorRight))
ret = basis[np.array(Bindex % basis.shape[0], dtype="int"), :]
return ret
```

get_pbas is a function that takes an index set as input, such as the period period, the knot difference dK and the order of the B-spline order. Note that the order of a B-spline is always the degree + 1. So we need an order of 4 for a cubic B-spline. The knot difference dK should be period/K if we want to get K basis functions.

Thus, the code for creating a periodic cubic B-spline basis with $K = 6$ basis functions and periodicity $A = 365.24$ is given by

```
K = 6
bspline_basis = get_pbas(a_index, period=A, dK=A/K, order=4)
pd.DataFrame(bspline_basis)
```

We can plot the basis using

```
plt.rcParams.update({'font.size': 12})
plt.figure(figsize=(10, 5))
for i in range(bspline_basis.shape[1]):
plt.plot(bspline_basis[:, i], linestyle='--' if i % 2 == 0 else '-')
plt.ylabel("$B_i(d)$")
plt.xlabel("d")
plt.tight_layout()
plt.grid()
plt.ylim([0, 1])
plt.legend(["i="+str(i+1) for i in range(bspline_basis.shape[1])],
ncol=bspline_basis.shape[1])
plt.savefig(
"out/04_seasonal_structures/annual_Bspline_cubic.pdf")
plt.show()
```

py.5 External regressors in python

So far in our models we have been only using the information that is available in the data that we forecast, i.e. the autoregressive effects, seasonal structure and cross-period correlations. But the intuition suggests that there is definitely more information available in the external regressors. The most intuitive and at the same time popular in the electricity price forecasting are electricity demand, specific energy generation, fuel prices and price values from other markets. In this section we will take a look at full data and check the impact of some external regressors on the prices. First, we read the provided data using the code below.

```
data = pd.read_csv("../data/Germany.csv")
```

We see that the dataset contains many regressors, but not all of them are available before the Day-Ahead Auction, e.g. the actual load. Since we are interested in Day-Ahead prices, we consider only the data with the prices available. After that, we prepare the dataset as we learned in the previous tutorials and plot the day-ahead prices for illustration purpose. In this exercise we focus ourselves only on the external regressors such as the day-ahead forecasts of load, wind and solar generation. Thus, we extract them from the data.

```
data_array = DST_trafo(X=data.iloc[:, 1:], Xtime=time_utc,
tz=local_time_zone)
data_array.shape

data.columns

# day-ahead price
price = data_array[:, :, data.columns[1:] == "Price"][..., 0]
# day-ahead load
load = data_array[:, :, data.columns[1:] == "Load_DA"][..., 0]
# day-ahead wind onshore
windon = data_array[:, :, data.columns[1:] == "WindOn_DA"][..., 0]
# day-ahead wind offshore
windoff = data_array[:, :, data.columns[1:] == "WindOff_DA"][..., 0]
# day-ahead solar
solar = data_array[:, :, data.columns[1:] == "Solar_DA"][..., 0]
# day-ahead residual load
residualload = load - windon - windoff - solar
```

Using the following code, we recreate Figure 29.

```
s = 9
fig, axs = plt.subplots(6, figsize=(15, 15), sharex=True)

axs[0].plot(dates_S, price[:, s], linewidth=0.5, color="C0")
axs[0].set_ylabel("Price in EUR/MWh at s = "+str(s))

axs[1].plot(dates_S, load[:, s], linewidth=0.5, color="C1")
axs[1].set_ylabel("Load in MW at s = "+str(s))

axs[2].plot(dates_S, windon[:, s], linewidth=0.5, color="C2")
axs[2].set_ylabel("Wind_on in MW at s = "+str(s))

axs[3].plot(dates_S, windoff[:, s], linewidth=0.5, color="C3")
axs[3].set_ylabel("Wind_off in MW at s = "+str(s))

axs[4].plot(dates_S, solar[:, s], linewidth=0.5, color="C4")
axs[4].set_ylabel("Solar in MW at s = "+str(s))

axs[5].plot(dates_S, residualload[:, s], linewidth=0.5, color="C5")
axs[5].set_ylabel("Residual load in MW at s = "+str(s))

plt.tight_layout()
plt.xlabel("Time")
plt.show()
```

With the following code, we explore the price-fundamentals relationship.

```
s = 9
fig, axs = plt.subplots(5, figsize=(15, 12.5))

colors = ["C"+str(i) for i in range(7)]
weekdays = pd.to_datetime(dates_S).dt.weekday.values

# day-ahead load
for day in np.unique(weekdays):
axs[0].scatter(load[weekdays == day, s], price[weekdays == day, s],
color=colors[day], facecolors='none')
axs[0].set_ylabel("Price in EUR/MWh")
axs[0].set_xlabel("Day-ahead Load, s = "+str(s))

# day-ahead wind onshore
axs[1].scatter(windon[:, s], price[:, s], color='C0', facecolors='none')
axs[1].set_ylabel("Price in EUR/MWh")
axs[1].set_xlabel("Day-ahead Wind onshore, s = "+str(s))

# day-ahead wind offshore
axs[2].scatter(windoff[:, s], price[:, s], color='C0',
facecolors='none')
axs[2].set_ylabel("Price in EUR/MWh")
axs[2].set_xlabel("Day-ahead Wind offshore, s = "+str(s))

# day-ahead solar
axs[3].scatter(solar[:, s], price[:, s], color='C0', facecolors='none')
axs[3].set_ylabel("Price in EUR/MWh")
axs[3].set_xlabel("Day-ahead Solar, s = "+str(s))

# day-ahead residual load
for day in np.unique(weekdays):
axs[4].scatter(residualload[weekdays == day, s],
price[weekdays == day, s],
color=colors[day], facecolors='none')
axs[4].set_ylabel("Price in EUR/MWh")
axs[4].set_xlabel("Day-ahead Residual Load, s = "+str(s))

plt.legend([day_abbr[i] for i in range(7)])
plt.tight_layout()
plt.show()
```

In the plots above, we observe that the residual load seems to explain the price better than the load. We can compare both regressors quickly as below.

```
s = 9
fig, axs = plt.subplots(1, 2, figsize=(15, 7), sharey=True)

axs[0].scatter(load[:, s], price[:, s])
axs[0].set_xlabel("Day-ahead Load, s = "+str(s))

axs[1].scatter(residualload[:, s], price[:, s])
axs[1].set_xlabel("Day-ahead Residual Load, s = "+str(s))

plt.ylabel("Price in EUR/MWh")
plt.tight_layout()
plt.show()
```

Often visual analysis may be misleading and not that precise. Thus, we calculate the covariance of both regressors with the prices. We see that the values are higher for the residual load.

```
np.corrcoef(load[:, s], price[:, s])
np.corrcoef(residualload[:, s], price[:, s])
```

Now, let us implement model (93) and estimate it using OLS. We write a very similar forecast function as before.

```
def forecast_expert_ext(dat, days, reg_names, wd=[1, 6, 7],
price_s_lags=[1, 2, 7], fuel_lags=[2]):
# dat.shape[0] = D+1, days.shape[0] = D
S = dat.shape[1]
forecast = np.repeat(np.nan, S)

days_ext = days.append(pd.Series(days.iloc[-1] + pd.DateOffset(1),
index=[len(days)]))
# preparation of weekday dummies including the day to forecast
weekdays_num = days_ext.dt.weekday+1  # 1 = Mon, 2 = Tue, ..., 7 = Sun
WD = np.transpose([(weekdays_num == x) + 0 for x in wd])

# Names for subsetting:
da_forecast_names = ["Solar_DA", "WindOn_DA", "WindOff_DA", "Load_DA"]

fuel_names = ["EUA", "API2_Coal", "Brent_oil", "TTF_Gas"]

# preparation of lags:
def get_lagged(Z, lag):
return np.concatenate((np.repeat(np.nan, lag), Z[:(len(Z)-lag)]))

# Lag of 2 as end of day data... at d-1
mat_fuels = np.concatenate([np.apply_along_axis(
get_lagged, 0, dat[:, 0, reg_names.isin(fuel_names)], lag=l)
for l in fuel_lags], axis=-1)
price_last = get_lagged(
Z=dat[:, S-1, reg_names == "Price"][..., 0], lag=1)

coefs = np.empty((S, len(wd)+len(price_s_lags) +
len(fuel_names)*len(fuel_lags)+len(da_forecast_names)+2))

for s in range(S):
# prepare the Y vector
acty = dat[:, s, reg_names == "Price"][..., 0]

# get lags
mat_price_lags = np.transpose([get_lagged(lag=lag, Z=acty)
for lag in price_s_lags])
mat_da_forecasts = dat[:, s, reg_names.isin(da_forecast_names)]

# combine all regressors to a matrix
regmat = np.column_stack(
(acty,  np.ones(acty.shape[0]), WD, mat_price_lags,
price_last, mat_da_forecasts, mat_fuels))
# hide the prices of the last day (the one to be forecasted)
regmat[-1, 0] = np.nan

act_index = ~ np.isnan(regmat).any(axis=1)

model = LinearRegression(fit_intercept=False).fit(
X=regmat[act_index, 1:], y=regmat[act_index, 0])

# deal with singularities
model.coef_[np.isnan(model.coef_)] = 0

forecast[s] = model.coef_ @ regmat[-1, 1:]
coefs[s] = model.coef_

regressor_names = ["intercept"]+[
day_abbr[i-1] for i in wd]+["Price lag "+str(lag)
for lag in price_s_lags]+[
"Price last lag 1"]+da_forecast_names+[
fuel+" lag "+str(lag) for lag in fuel_lags for fuel in fuel_names]

coefs_df = pd.DataFrame(coefs, columns=regressor_names)

return {"forecasts": forecast, "coefficients": coefs_df}
```

And then simply call it with

```
model = forecast_expert_ext(
data_array[:D+1],
pd.to_datetime(dates_S)[:D],
data.columns[1:])
```

Note that here we give additionally reg_names variable which delivers the names of regressors available in dat. We do it this way as it is not very convenient to name numpy arrays.

Then, we can easily estimate the model using scaled regressors by scaling the regression matrix

```
regmat_mean = regmat[act_index].mean(axis=0)
regmat_sd = regmat[act_index].std(axis=0)
regmat_scaled = (regmat[act_index] - regmat_mean)/regmat_sd
```

The scaled regression matrix can now be used for modelling. The forecasting however is a little bit different. Here we need to remember to scale the new observations with the previously calculated values. In the end, we need to scale the forecast back to the original scale of the prices.

```
model = LinearRegression(fit_intercept=False).fit(
X=regmat_scaled[:, 1:], y=regmat_scaled[:, 0])

# deal with singularities
model.coef_[np.isnan(model.coef_)] = 0

forecast[s] = (model.coef_ @ ((regmat[-1, 1:] -
regmat_mean[1:])/regmat_sd[1:])
)*regmat_sd[0] + regmat_mean[0]
```

Now, we can use the coef matrix to recreate Figure 37.

```
model = forecast_expert_ext_scaled(
dat=data_array[:D+1],
days=pd.to_datetime(dates_S)[:D],
reg_names=data.columns[1:])

coefmat = model["coefficients"]
coef_names = coefmat.columns

lty = ["-" if i < 7 else '--' for i in range(len(coef_names))]
actcol = plt.get_cmap('gist_rainbow')(np.linspace(0, 1, len(coef_names))
)

plt.rcParams.update({'font.size': 14})
plt.figure(figsize=(15, 10))
for i_p in range(coefmat.shape[1]):
plt.plot(coefmat.iloc[:, i_p], linestyle=lty[i_p], color=actcol[i_p])

plt.grid()
plt.axhline(0, color='grey')
plt.legend(coef_names, ncol=2)
plt.xlabel("s")
plt.ylabel("Coefficients")

plt.tight_layout()
plt.show()
```

py.6 Non-linear effects in python

In purpose of estimation of the tail index, we need a hill estimator. The currently available python packages unfortunately do not provide it. Thus, we write a function based on the hill() function from evir package in R.

```
def hill(data, start=14, end=None, abline_y=None, ci=0.95, ax=None):
"""Hill estimator translation from R package evir::hill

Plot the Hill estimate of the tail index of heavy-tailed data.

Parameters
----------
data : array_like
data vector
start : int
lowest number of order statistics at which to plot a point
end : int, optional
highest number of order statistics at which to plot a point
abline_y : float, optional
value to be plotted as horizontal straight line
ci : float
probability for asymptotic confidence band
ax : Axes, optional
the Axes in which to plot the estimator
"""
ordered = np.sort(data)[::-1]
ordered = ordered[ordered > 0]
n = len(ordered)
k = np.arange(n)+1
loggs = np.log(ordered)
avesumlog = np.cumsum(loggs)/k
xihat = np.hstack([np.nan, (avesumlog-loggs)[1:]])
alphahat = 1/xihat
y = alphahat
ses = y/np.sqrt(k)
if end is None:
end = n-1
x = np.arange(np.min([end, len(data)-1]), start, -1)
y = y[x]
qq = norm.ppf(1 - (1-ci)/2)
u = y + ses[x] * qq
l = y - ses[x] * qq
if ax is None:
fig, ax = plt.subplots()
ax.plot(x, y, color='black', linewidth=1)
ax.plot(x, u, color='red', linestyle='--', linewidth=1)
ax.plot(x, l, color='red', linestyle='--', linewidth=1)
if abline_y is not None:
ax.axhline(abline_y, color='C0', linewidth=1)
ax.set_ylabel('alpha (CI, p = '+str(ci)+")")
ax.set_xlabel("Order Statistics")
```

The function plots an estimator of the tail index α depending on the order statistics of the data. It is quite important for interpretation of the plot to set the end parameter correctly. The parameter sets the highest number of order statistics, for which the tail index α will be calculated and plotted. Setting it too high or too low will result in a bad estimation. Based on a rule of thumb, we want to set the end parameter in such a way that the α estimate looks more or less constant. Using this rule of thumb we found that $2\sqrt{N}$ is sufficient value for the parameter, where N is the length of the data that we explore.

As an example we consider the t distribution with different degrees of freedom k. Let us remind that for t distribution holds $\alpha = k$. First, we plot the histograms of the considered data generated from the t distribution and one for scaled data with electricity prices.

```
s = 13
price = price_S.iloc[:, s]

fig, axs = plt.subplots(2, 2, figsize=(10, 6))
tdf = 4  # mean 0, variance 1
axs[0, 0].hist(t.rvs(size=len(price), df=tdf) /
np.sqrt(tdf/(tdf-2)), bins='scott', density=True)
axs[0, 0].set_title("t with df = "+str(tdf))
axs[0, 0].set_xlabel("x")
axs[0, 0].set_ylabel("Density")

tdf = 3  # mean 0, variance 1
axs[0, 1].hist(t.rvs(size=len(price), df=tdf) /
np.sqrt(tdf/(tdf-2)), bins='scott', density=True)
axs[0, 1].set_title("t with df = "+str(tdf))
axs[0, 1].set_xlabel("x")
axs[0, 1].set_ylabel("Density")

tdf = 2.2  # mean 0, variance 1
axs[1, 0].hist(t.rvs(size=len(price), df=tdf) /
np.sqrt(tdf/(tdf-2)), bins='scott', density=True)
axs[1, 0].set_title("t with df = "+str(tdf))
axs[1, 0].set_xlabel("x")
axs[1, 0].set_ylabel("Density")

# mean 0, variance 1
axs[1, 1].hist((price - price.mean())/price.std(), bins='scott',
density=True)
axs[1, 1].set_title("Scaled price series")
axs[1, 1].set_xlabel("x")
axs[1, 1].set_ylabel("Density")

plt.tight_layout()
plt.show()
```

We see clearly that the lower the number of degrees of freedom, the heavier the tails. Moreover, the histogram of the scaled price series looks similarly to the histogram of t distribution with $k = 4$.

Now, we want to estimate the tail index using the Hill estimator.

```
fig, axs = plt.subplots(2, 2, figsize=(10, 6))

tdf = 4
hill(t.rvs(size=len(price), df=tdf), end=int(2*np.sqrt(len(price))),
abline_y=tdf, ax=axs[0, 0])
axs[0, 0].set_title("t with df = "+str(tdf))

tdf = 3
hill(t.rvs(size=len(price), df=tdf), end=int(2*np.sqrt(len(price))),
abline_y=tdf, ax=axs[0, 1])
axs[0, 1].set_title("t with df = "+str(tdf))

tdf = 2.2
hill(t.rvs(size=len(price), df=tdf), end=int(2*np.sqrt(len(price))),
abline_y=tdf, ax=axs[1, 0])
axs[1, 0].set_title("t with df = "+str(tdf))

# price data
hill(price, start=int(np.log(len(price))),
end=int(2*np.sqrt(len(price))),
abline_y=4, ax=axs[1, 1])
axs[1, 1].set_title("Price series")

plt.tight_layout()
plt.show()
```

We see that the estimation works correctly, i.e. the tail index is estimated to be close to the true one. Let us note that the Hill estimator works very well with lower tail indices and not that good with higher ones. Anyway, in the range of the particular interest of ours the performance is satisfying.

Now we want to reconstruct Figure 40. Let us note that we use there (among others) a quadratic effect of our price series. We checked that the tail index of the prices is higher than 4, so the quadratic effect's tail index is higher than 2. This means that the variance exists, and we will not run into any trouble, except likely the slow convergence. Let us remind that we want to create a simple model regressing the prices on the residual load and one additional term consisting of a transformation of the residual load. We prepare the data first:

```
index = np.arange(data_array.shape[0])
days = pd.to_datetime(dates_S[index])
Y = data_array[index, :, id_price]
X = data_array[index, :, id_f[-1]] - data_array[index][..., id_f[:-1]].sum(-1)

df = np.transpose([Y[:, s], X[:, s]])
xrng = np.ptp(df[:, -1])
dfg = np.linspace(df[:, -1].min()-xrng, df[:, -1].max()+xrng, 1000)
```

Now we are ready to create the plots.
We will begin with the quadratic term and continiue with the $|x-20|$ and $|x-50|$ terms.

```
plt.scatter(df[:, -1], df[:, 0], color='black', facecolor='none')
plt.grid()
plt.xlabel("$X_{d,s}$")
plt.ylabel("$Y_{d,s}$")
plt.title(str(dates_S.min())+" to "+str(dates_S.max()))

# linear model
mod = LinearRegression(fit_intercept=True).fit(X=df[:, -1:], y=df[:, 0])
plt.plot(dfg, dfg * mod.coef_ + mod.intercept_, color='red')

# Model with add. quadr. term
mod = LinearRegression(fit_intercept=True).fit(
X=np.hstack([df[:, -1:], df[:, -1:]**2]), y=df[:, 0])
plt.plot(dfg, np.column_stack([dfg, dfg**2]) @
mod.coef_ + mod.intercept_, color='green')

plt.legend(["linear model", "with add. quadr. term"],
title="s = "+str(s))
plt.xlim([df[:, -1].min(), df[:, -1].max()])
plt.ylim([df[:, 0].min(), df[:, 0].max()])
plt.tight_layout()
plt.show()

plt.scatter(df[:, -1], df[:, 0], color='black', facecolor='none')
plt.grid()
plt.xlabel("$X_{d,s}$")
plt.ylabel("$Y_{d,s}$")
plt.title(str(dates_S.min())+" to "+str(dates_S.max()))

# linear model
mod = LinearRegression(fit_intercept=True).fit(X=df[:, -1:], y=df[:, 0])
plt.plot(dfg, dfg * mod.coef_ + mod.intercept_, color='red')

# Model with add. shifted absolute term
mod = LinearRegression(fit_intercept=True).fit(
X=np.hstack([df[:, -1:], np.abs(df[:, -1:]-20)]), y=df[:, 0])
plt.plot(dfg, np.column_stack([dfg, np.abs(dfg-20)]) @
mod.coef_ + mod.intercept_, color='C0')

plt.legend(["linear model", "with add. |x-20| term"],
title="s = "+str(s))
plt.xlim([df[:, -1].min(), df[:, -1].max()])
plt.ylim([df[:, 0].min(), df[:, 0].max()])
plt.tight_layout()
plt.show()

plt.scatter(df[:, -1], df[:, 0], color='black', facecolor='none')
plt.grid()
plt.xlabel("$X_{d,s}$")
plt.ylabel("$Y_{d,s}$")
plt.title(str(dates_S.min())+" to "+str(dates_S.max()))

# linear model
mod = LinearRegression(fit_intercept=True).fit(X=df[:, -1:], y=df[:, 0])
plt.plot(dfg, dfg * mod.coef_ + mod.intercept_, color='red')

# Model with add. shifted absolute term
mod = LinearRegression(fit_intercept=True).fit(
X=np.hstack([df[:, -1:], np.abs(df[:, -1:]-50)]), y=df[:, 0])
plt.plot(dfg, np.column_stack([dfg, np.abs(dfg-50)]) @
mod.coef_ + mod.intercept_, color='C0')

plt.legend(["linear model", "with add. |x-20| term"],
title="s = "+str(s))
plt.xlim([df[:, -1].min(), df[:, -1].max()])
plt.ylim([df[:, 0].min(), df[:, 0].max()])
plt.tight_layout()
plt.show()
```

py.7 Advanced estimation techniques in python

More complicated models require advanced estimation techniques. In the lecture, we have learned about two methods that are very popular in the electricity price forecasting -- lasso and ridge estimation. In this section, we will use the sklearn implementation in python in order of modelling and forecasting of the electricity prices.

```
from sklearn import linear_model
```

The package contains all the tools that we need for successful lasso or ridge estimation. Particularly interesting for us are Lasso and Ridge classes. Let us note that we provide the scaled response $\widetilde{\boldsymbol{Y}}_s$ and scaled regression matrix $\widetilde{\mathcal{X}}_s$ to the fit functions, similarly as we would do for the class LinearRegression. Moreover, the lasso and ridge functions can standardize the given regressors itself. However, for better interpretation of the fitted coefficients, it is advised to perform the scaling ourselves. A very crucial parameter is λ (α in the sklearn implementation). We can either leave the choice of considered λ sequence to the algorithm or specify one. The latter one is the recommended option.

As an example, we work on the prices of product $s = 9$. For the first exercise, we simply take the expert model (7). Before, we estimated it using OLS, but now we will use the lasso and ridge estimators. To do it, we build the response vector and the regression matrix, as we did before.

```
s = 9
D = 730
index = np.arange(7, D)
Y = price_S.iloc[index]
days = pd.to_datetime(dates_S[index])

YREG = Y.iloc[:, s].values  # response vector

# Expert specification

days_ext = days.append(pd.Series(days.iloc[-1] + pd.DateOffset(1),
index=[len(days)]))

# weekday dummies
expert_wd = [1, 6, 7]
weekdays_num = days_ext.dt.weekday+1  # 1 = Mon, 2 = Tue, ..., 7 = Sun
WD = np.transpose([(weekdays_num == x) + 0 for x in expert_wd])

# lags
expert_lags = [1, 2, 7]


def get_lagged(Z, lag):
return np.concatenate((np.repeat(np.nan, lag), Z[:(len(Z) + 1-lag)]))


XLAG = np.transpose([get_lagged(lag=lag, Z=YREG)
for lag in expert_lags])
regressor_names = ["lag "+str(lag) for lag in expert_lags] + \
[day_abbr[i-1] for i in expert_wd]

# Regression matrix
XREG = np.column_stack((XLAG, WD))

act_index = ~ np.isnan(XREG).any(axis=1)
act_index[-1] = False  # no NAs and no last obs
```

Now, we fit the models for both scaled and unscaled regressors.

```
# unscaled models - not recommended for lasso/ridge
model = linear_model.LinearRegression(fit_intercept=False).fit(
X=XREG[act_index], y=YREG[act_index[:-1]])

lambdas = 2**(np.linspace(0.5, -10, 100))
coefs_ridge = np.full((lambdas.shape[0], XREG.shape[1]+1), np.nan)
coefs_lasso = np.full((lambdas.shape[0], XREG.shape[1]+1), np.nan)
for l in range(lambdas.shape[0]):
model_ridge = linear_model.Ridge(alpha=lambdas[l]).fit(
X=XREG[act_index], y=YREG[act_index[:-1]]
)
coefs_ridge[l] = np.hstack([model_ridge.intercept_, model_ridge.coef_])
model_lasso = linear_model.Lasso(alpha=lambdas[l]).fit(
X=XREG[act_index], y=YREG[act_index[:-1]]
)
coefs_lasso[l] = np.hstack([model_lasso.intercept_, model_lasso.coef_])
```

```
# scaled lasso/ridge
XREG_mean = np.mean(XREG[act_index], axis=0)
XREG_std = np.std(XREG[act_index], axis=0)
XREG_scaled = (XREG[act_index] - XREG_mean)/XREG_std

YREG_mean = np.mean(YREG[act_index[:-1]], axis=0)
YREG_std = np.std(YREG[act_index[:-1]], axis=0)
YREG_scaled = (YREG[act_index[:-1]] - YREG_mean)/YREG_std

model_scaled = linear_model.LinearRegression(fit_intercept=False).fit(
X=XREG_scaled, y=YREG_scaled)

coefs_ridge_scaled = np.full((lambdas.shape[0], XREG.shape[1]), np.nan)
coefs_lasso_scaled = np.full((lambdas.shape[0], XREG.shape[1]), np.nan)
for l in range(lambdas.shape[0]):
model_ridge_scaled = linear_model.Ridge(alpha=lambdas[l]).fit(
X=XREG_scaled, y=YREG_scaled
)
coefs_ridge_scaled[l] = model_ridge_scaled.coef_
model_lasso_scaled = linear_model.Lasso(alpha=lambdas[l]).fit(
X=XREG_scaled, y=YREG_scaled
)
coefs_lasso_scaled[l] = model_lasso_scaled.coef_
```

Using the fitted models, we could perform forecasting as we did before. The only problem is that we have obtained many coefficient sets, dependent on λ. The selection of the λ is the key for successful lasso/ridge estimation. We handle this problem later in this section. Now, let us analyse the coefficients dependent on the values of λ and the $||\cdot||_1$ norm of the coefficient vector. To do this, we reproduce Figures 44 and 45 for lasso estimator. For better interpretation, we use the model fitted to the scaled regressors.

```
beta = coefs_lasso_scaled
abs_beta = np.abs(beta).sum(1)
p = beta.shape[1]
```

Using the code above, we extract the coefficient matrix, the lambda values and the dimensions of the coefficient matrix. Using the data, we can reconstruct Figures 44 and 45.

```
col_p = plt.get_cmap('gist_rainbow')(np.linspace(0, 1, p))
markers = ["o", "^", "+", "x", "D", "v"]
plt.rcParams.update({'font.size': 14})
plt.figure(figsize=(8, 8))
for i in range(p):
plt.plot(abs_beta, beta[:, i], marker=markers[i], color=col_p[i],
markerfacecolor='none')
plt.grid()

plt.ylabel("$\\hat{\\beta}_{s,k}$")
plt.xlabel("$||\\beta_s||_1$")
plt.title(str(dates_S[index].min())+" to " +
str(dates_S[index].max())+" at hour s="+str(s))
plt.legend(["k="+str(i+1)+", "+regressor_names[i] for i in range(p)], ncol=2)

plt.tight_layout()
plt.show()
```

As mentioned before, the key to the successful lasso/ridge estimation is the proper choice of λ parameter. The preferable method of tuning the λ parameter is the k-fold cross-validation due to high robustness. On the other hand, this method causes a large computational effort which we not always can afford. Package sklearn consists of a function LassoCV, which may perform a k-fold cross-validation or block-wise k-fold cross-validation. For standard k-fold cross-validation, the nfolds parameter is the key, whereas for the block-wise version, the foldid is the key. As an example, we perform a 10-fold cross-validation as below.

```
np.random.seed(1234)
nfold = 10

mod0cv = linear_model.LassoCV(alphas=lambdas, cv=nfold).fit(
X=XREG_scaled, y=YREG_scaled
)
mod0cv.alpha_  # the best lambda according to CV

# calculate the forecast
np.hstack([1, (XREG[-1]-XREG_mean)/XREG_std]) @ np.hstack(
[mod0cv.intercept_, mod0cv.coef_]) * YREG_std + YREG_mean
```

The coefficients obtained this way can be easily used in the purpose of forecasting. We see that the LassoCV function makes it very easy to perform a cross-validation exercise, but it is recommended to perform the cross-validation manually for better understanding and control of the study. When the cross-validation causes too heavy computational effort, then it is reasonable to use the information criteria. In the code below, we use the most conservative information criteria, the BIC. It is especially useful when dealing with data containing very many regressors as it does not allow for overfitting.

```
model_lasso = linear_model.Lasso(fit_intercept=False).path(
X=np.column_stack([np.ones(XREG_scaled.shape[0]), XREG_scaled]),
y=YREG_scaled, alphas=lambdas
)
beta = model_lasso[1]
fitted_values = np.column_stack(
[np.ones(XREG_scaled.shape[0]), XREG_scaled]) @ beta
allRES = YREG_scaled[:, None] - fitted_values
rss = (allRES**2).sum(axis=0)  # residual sums of square
df = (beta != 0).sum(axis=0)  # number of parameters
nobs = XREG_scaled.shape[0]  # number of observations
kappa = np.log(nobs)  # BIC= log(nobs) # HQC= 2*log(log(nobs)) # AIC= 2
ic = np.log(rss) + kappa * df / nobs
BIC_min = np.argmin(ic)
beta_opt = beta[:, BIC_min]  # IC-chosen beta
```

This way we obtain the best coefficients in terms of BIC. Note that in the code above we can easily change BIC to any other information criteria. It is useful as well to plot the IC in order to get an intuition of its behavior.

Lastly we are going to estimate a lasso approach the expert.adv model (111). Note that we are using some of the objects defined above for creating the regression matrix.

```
# weekday dummies
expert_wd = list(range(1, 8))
WD = np.transpose([(weekdays_num == x) + 0 for x in expert_wd])

# expert specification
xprice_temp = np.column_stack([
data_array[index, S-1, reg_names == "Price"],
data_array[index, :, reg_names == "Price"].min(1),
data_array[index, :, reg_names == "Price"].max(1)
])
xprice = np.apply_along_axis(get_lagged, 0, xprice_temp, lag=1)


fuel_lags = [2]
fuel_names = ["EUA", "API2_Coal", "Brent_oil", "TTF_Gas"]

xfuels = np.concatenate([np.apply_along_axis(
get_lagged, 0, data_array[index][:, 0, reg_names.isin(fuel_names)], lag=l)
for l in fuel_lags], axis=-1)

# externals
index_ext = np.append(index, index[-1]+1)
da_forecast_names = ["Solar_DA", "WindOn_DA", "WindOff_DA", "Load_DA"]

ext = data_array[index_ext][:, s, reg_names.isin(da_forecast_names)]
regressor_names += da_forecast_names

XREG = np.column_stack([
XLAG, xprice, WD, ext, xfuels
])

regressor_names = ["lag "+str(lag) for lag in expert_lags] + \
["last", "min", "max"] + [day_abbr[i-1] for i in expert_wd] + \
da_forecast_names + fuel_names

# Model estimation
act_index = ~ np.isnan(XREG).any(axis=1)
act_index[-1] = False  # no NAs and no last obs

# scaled lasso/ridge
XREG_mean = np.mean(XREG[act_index], axis=0)
XREG_std = np.std(XREG[act_index], axis=0)
XREG_scaled = (XREG[act_index] - XREG_mean)/XREG_std

YREG_mean = np.mean(YREG[act_index[:-1]], axis=0)
YREG_std = np.std(YREG[act_index[:-1]], axis=0)
YREG_scaled = (YREG[act_index[:-1]] - YREG_mean)/YREG_std

model_lasso_scaled = linear_model.Lasso(fit_intercept=False).path(
X=XREG_scaled, y=YREG_scaled, alphas=lambdas
)
```

The figures can be created in the same way as described above.

py.8 Forecasting combination in python

Combining forecasts of models using different rolling window sizes or even forecasts of different models can be very useful and significantly lower the error. In the following section, we will learn how to implement these in python. As an example will serve us the well-known extended expert model (67). We consider 4 variants of the model, each having the same regressors, but different length of the rolling window: 730, 112, 84 and 56 days. First, we set some arguments.

```
D = 365 * 2  # in.sample size
N = 365 * 3  # size of forecasting study
oos_dates = dates_S[D:D+N]  # dates to do the forecasting study on
K = 4  # number of experts
S = price_S.shape[1]
model_names = ["true", "expert", "expert_8w", "expert_12w", "expert_16w"]
```

We use similar study as previously. As mentioned, we use 4 variants of the expert model. Using the forecast_expert_last function, we calculate forecasts of all models.

```
forecasts = np.full((N, S, K+1), np.nan)
dim_names = [oos_dates.values, np.arange(S), model_names]

for n in range(N):
# true
forecasts[n, :, 0] = price_S.iloc[D+n]

Y = price_S.iloc[n:D+n]
days = pd.to_datetime(dates_S[n:D+n], utc=True)

# expert
forecasts[n, :, 1] = forecast_expert_last(Y, days)["forecasts"]

# expert_8w
forecasts[n, :, 2] = forecast_expert_last(
Y.tail(7*8), days.tail(7*8))["forecasts"]

# expert_12w
forecasts[n, :, 3] = forecast_expert_last(
Y.tail(7*12), days.tail(7*12))["forecasts"]

# expert_16w
forecasts[n, :, 4] = forecast_expert_last(
Y.tail(7*16), days.tail(7*16))["forecasts"]

print("\r-> "+str(np.round((n+1)/N*100, 4))+"% done", end='')
```

Then, we create an error array for a product $s = 12$ and we calculate the RMSE.

```
s = 12
error = forecasts[:, s, 1:] - forecasts[:, s, :1]
pd.DataFrame([np.sqrt((error**2).mean(0))], columns=model_names[1:])
```

We see that reducing the rolling window length didn't yield any improvement in terms of RMSE. Let us now measure the RMSE of the smallest errors of the models over all days. That is to say, on every day we choose the smallest error, and then we calculate the RMSE. In the real life this is unachievable, but it will give us an important information -- whether there is any space for improvement or not.

```
np.sqrt((error**2).min(1).mean())
```

We see that the results could be improved. Now, let us implement the naive combination (194).

```
weights_naive = np.ones(K)/K
combination_naive = np.expand_dims(
weights_naive, axis=0) @ forecasts[:, s, 1:].transpose()
np.sqrt(((forecasts[:, s, 0] - combination_naive)**2).mean())
```

We see that this simple approach yields a better performance than all of the short-term models. However, it does not outperform the usual expert last model.

```
weights_naive_ls = np.hstack([0.5, np.ones(K-1)/(2*(K-1))])
# same weight on long and short term
combination_naive_ls = np.expand_dims(
weights_naive_ls, axis=0) @ forecasts[:, s, 1:].transpose()
np.sqrt(((forecasts[:, s, 0] - combination_naive_ls)**2).mean())
```

With the code above, we can achieve even lower error. We see clearly that combining forecasts makes sense. Let us now combine the forecasts choosing the model to use based on $d-1$ performance of them.

```
weights = np.zeros((N, K))
weights[0] = 1/K
min_index = (error**2).argmin(1)
for i in range(1, N):
weights[i, min_index[i-1]] = 1
predictions = (weights * forecasts[:, s, 1:]).sum(1)
np.sqrt(((forecasts[:, s, 0] - predictions)**2).mean())
```

We see that choosing the forecasts this way worsens the results in comparison with the standard expert model. Another possible approach is to generalize the above to use more than one day of past performance.

```
n_init = 28  # best on the last n_init values, req. 3 min, just coding issues
lowest_expanding = np.zeros(error.shape[0], dtype=int)
lowest_rolling = np.zeros(error.shape[0], dtype=int)
w_expanding = np.zeros((N, K))
w_rolling = np.zeros((N, K))
error = forecasts[:, s, 1:] - forecasts[:, s, :1]
for i_N in range(n_init, N):
lowest_expanding[i_N] = np.sqrt((error[:i_N]**2).sum(0)).argmin()
lowest_rolling[i_N] = np.sqrt((error[(i_N-n_init):i_N]**2).sum(0)).argmin()
w_expanding[i_N, lowest_expanding[i_N]] = 1
w_rolling[i_N, lowest_rolling[i_N]] = 1
# replace initial part by naive
w_expanding[:n_init] = w_rolling[:n_init] = 1/K

comb_expanding = (w_expanding * forecasts[:, s, 1:]).sum(1)
comb_rolling = (w_rolling * forecasts[:, s, 1:]).sum(1)

print(np.sqrt(((forecasts[:, s, 0] - comb_expanding)**2).mean()))
print(np.sqrt(((forecasts[:, s, 0] - comb_rolling)**2).mean()))
```

Unfortunately, this method yields even worse forecasts. Now, we turn ourselves to the more complicated combination model, i.e. the one described by equation (202).

```
n_init = int(D / 2)  # min K required for estimation of (inverse) covariance
sigma_inv_expanding = np.zeros((N, K, K))
sigma_inv_rolling = np.zeros((N, K, K))
w_expanding = np.zeros((N, K))
w_rolling = np.zeros((N, K))
ones = np.ones(K)
for i_N in range(n_init, N):
sigma_inv_expanding[i_N] = np.linalg.inv(np.cov(error[:i_N], rowvar=False))
sigma_inv_rolling[i_N] = np.linalg.inv(
np.cov(error[(i_N-n_init):i_N], rowvar=False))
tmp = sigma_inv_expanding[i_N] @ ones
w_expanding[i_N] = tmp/tmp.sum()
tmp = sigma_inv_rolling[i_N] @ ones
w_rolling[i_N] = tmp/tmp.sum()
# replace initial part by naive
w_expanding[:n_init] = w_rolling[:n_init] = 1/K
```

Then we can calculate forecasts as before, and also plot the weights over time.

```
# evolution of weights expanding
fig = plt.figure(figsize=(10, 5))
plt.title("Expanding window")
plt.plot(w_expanding)
plt.grid()
plt.ylabel("Weight")
plt.xlabel("Time")
plt.legend(model_names[1:])
plt.tight_layout()
plt.show()

# evolution of weights rolling
fig = plt.figure(figsize=(10, 5))
plt.title("Rolling window")
plt.plot(w_rolling)
plt.grid()
plt.ylabel("Weight")
plt.xlabel("Time")
plt.legend(model_names[1:])
plt.tight_layout()
plt.show()

# calculate forecasts & errors
comb_expanding = (w_expanding * forecasts[:, s, 1:]).sum(1)
comb_rolling = (w_rolling * forecasts[:, s, 1:]).sum(1)

print(np.sqrt(((forecasts[:, s, 0] - comb_expanding)**2).mean()))
print(np.sqrt(((forecasts[:, s, 0] - comb_rolling)**2).mean()))
```

Mixing this approach and the one based on the past performance, we end up with another method.

```
n_init = int(D / 2)  # min K required for evaluate past performance
rmse_expanding = np.zeros((N, K))
rmse_rolling = np.zeros((N, K))
w_expanding = np.zeros((N, K))
w_rolling = np.zeros((N, K))
for i_N in range(n_init, N):
rmse_expanding[i_N] = np.sqrt((error[:i_N]**2).mean(0))
rmse_rolling[i_N] = np.sqrt((error[(i_N-n_init):i_N]**2).mean(0))
tmp = 1 / rmse_expanding[i_N]
w_expanding[i_N] = tmp/tmp.sum()
tmp = 1 / rmse_rolling[i_N]
w_rolling[i_N] = tmp/tmp.sum()
# replace initial part by naive
w_expanding[:n_init] = w_rolling[:n_init] = 1/K
```

We see that this method is better than the others, but still not better than the second naive. Another, more general combining method is the one described by equation (204). Below we implement it similarly as before.

```
n_init = int(D / 2)  # min K required for estimation of (inverse) covariance
sigma_expanding = np.zeros((N, K+1, K+1))  # of FORECAST incl. true
sigma_rolling = np.zeros((N, K+1, K+1))
w_expanding = np.zeros((N, K))
w_rolling = np.zeros((N, K))
for i_N in range(n_init, N):
sigma_expanding[i_N] = np.cov(forecasts[:i_N, s], rowvar=False)
sigma_rolling[i_N] = np.cov(forecasts[(i_N-n_init):i_N, s], rowvar=False)
w_expanding[i_N] = np.linalg.inv(
sigma_expanding[i_N, 1:, 1:]) @ sigma_expanding[i_N, 1:, 0]
w_rolling[i_N] = np.linalg.inv(
sigma_rolling[i_N, 1:, 1:]) @ sigma_rolling[i_N, 1:, 0]
# replace initial part by naive
w_expanding[:n_init] = w_rolling[:n_init] = 1/K
```

We see that there are many possibilities of creating forecasting combinations.

py.9 Univariate models in python

In this last chapter, we look at univariate models. That is, our models do not depend on S. The first model that we are going to implement is (208). We first create the univariate time series.

```
Y_temp = Y.copy()
Y_univariate = Y_temp.values.flatten()
```

Now we can estimate the model using the modification of the yule-walker function from statsmodels. It will choose the order based on the AIC, we just have to provide the maximum order that we want to consider. We chose 196 which corresponds to one week plus one day.

```
r = np.zeros(order_max+1, np.float64)
r[0] = (Y_univariate ** 2).sum() / n

coefs_df = list()
for k in range(1, order_max+1):
r[k] = (Y_univariate[0:-k] * Y_univariate[k:]).sum() / n
R = toeplitz(r[:k])
rho = np.linalg.solve(R, r[1:(k+1)])
coefs_df.append(rho)

def get_lagged(lag, Z):
return np.concatenate((np.repeat(np.nan, lag), Z[:(len(Z)+1 - lag)]))
XREG = np.transpose(np.array(
[get_lagged(lag=lag, Z=Y_univariate)
for lag in range(1, order_max+1)]))
act_index = ~ np.isnan(XREG).any(axis=1)
act_index[-1] = False
pred = np.array([XREG[act_index, :(i+1)] @ coefs_df[i]
for i in range(order_max)])
RSS = ((pred - Y_univariate[act_index[:-1]])**2).sum(axis=1)
nobs = XREG.shape[0]
AIC = np.log(RSS) + 2*np.arange(1, order_max+1)/nobs
order = np.argmin(AIC)
```

Predicting the desired 24 prices requires recursive approach.

```
last_obs = XREG[-1, :order+1]
forecast = np.repeat(np.nan, S)
for s in range(S):
forecast[s] = last_obs @ coefs_df[order]
last_obs = np.insert(last_obs[:-1], 0, forecast[s])
```

Now, we implement model (210). We will reuse some code defined above. However, since our forecast depends on the weekday, we have to extract it from the data. Remember that we want to demean the time-series by subtracting the average values at the respective hour of the week. That is, we have to calculate those 196 means.

```
  Y_temp = Y.copy()
HoW_mean = Y_temp.groupby(days.dt.weekday.values).transform("mean")
HoW_mean.index = days
Y_demeaned = Y_temp.sub(HoW_mean)
Y_univariate = Y_demeaned.values.flatten()
```

The model estimation is as above. The prediction requires again a recursive approach. Additionally we need to add the 24 means which correspond to the day we want to forecast.

```
  last_obs = XREG[-1, :order+1]
forecast = np.repeat(np.nan, S)
for s in range(S):
forecast[s] = last_obs @ coefs_df[order]
last_obs = np.insert(last_obs[:-1], 0, forecast[s])
next_day_w = (days.iloc[-1] + pd.DateOffset(1)).weekday()
forecast += HoW_mean.iloc[np.argmax(
HoW_mean.index.weekday == next_day_w), :]
```

In the tutorial, you will implement both models as functions and use them to conduct a forecasting study. Refer to the code files for details about this.