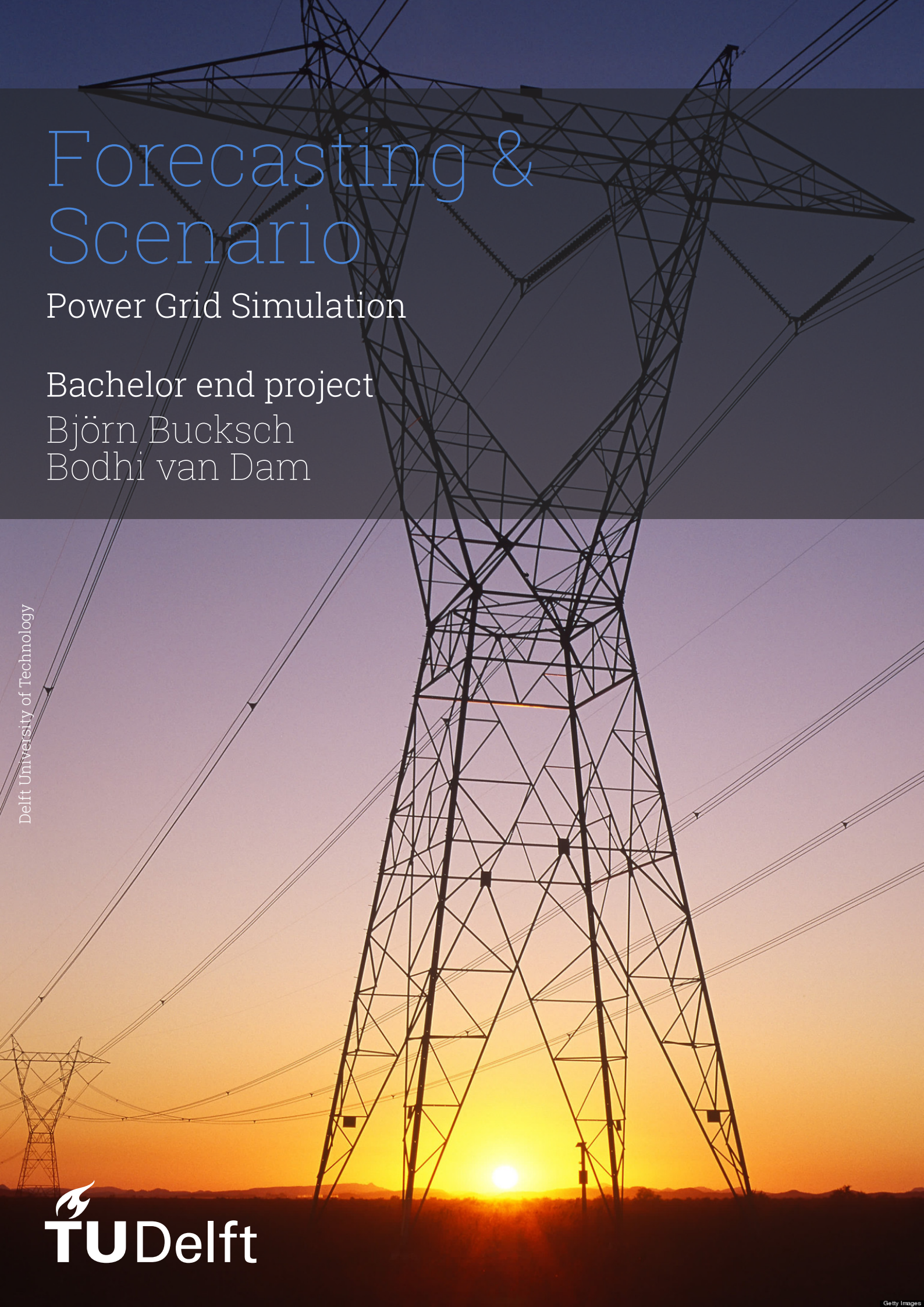
Forecasting & Scenario

Power Grid Simulation

Bachelor end project

Björn Bucksch
Bodhi van Dam

Delft University of Technology



TU Delft

Forecasting & Scenario

Power Grid Simulation

by

Björn Bucksch
Bodhi van Dam

Bachelor Graduation Thesis
Supervisor: dr. S.H. Tindemans
Defense Committee: dr. S.H. Tindemans, dr. M. Taouil, dr. C. Gao, MSc. T. van der Heijden

Friday 14 June, 2024

Delft University of Technology

Faculty of Electrical Engineering, Mathematics
and Computer Science

Electrical Engineering Programme

TU Delft

Abstract

This thesis presents the process of design and implementation of a power grid simulator, with focus on the power grid configuration and load forecasting as part of a larger project. The overarching project aims to address the increasing complexity of power grids and the occurence of grid congestion. Its primary objective is to visualize these issues for educational purposes. For this subsystem, the specific goal is to provide a grid configuration, all necessary data to simulate it, and load predictions for the grid.

Preface

This thesis is written for the Bachelor Graduation Project of the Bachelor of Electrical Engineering at the TU Delft. The project was initiated to create an interactive power grid visualisation and simulation system, which eventually will be placed in the Digital Energy Studio. Our goal is to create an engaging learning tool that will inspire future students to explore the complexity of the power grid and the challenges it face.

We would like to express our sincere gratitude to our supervisor, Simon Tindeman, as well as Ties van der Heijden, Nanda Panda and the rest of the team for their helpful guidance throughout the meetings. We are also grateful to Remko Koornneef and Martin Schumacher for allowing us to utilize the EEE project room and opening it for us every morning. Finally, we would like to express our appreciation to our colleagues, Rik Bieling, Johannes Ketelaars, Joran Kroese, and Michiel De Rop for an enjoyable work environment and constructive collaboration.

Björn Bucksch
Bodhi van Dam
Delft, July 2024

Contents

1

Introduction

The transition to a renewable energy system is driving a widespread deployment of wind power and solar PV generation, alongside the rapid electrification of energy demand. This shift presents significant challenges for electricity grids, which were not originally designed to accommodate such developments. Being able to visualize the issues in a clear manner would open up new perspectives on how to tackle these problems, as well as engage more people in the topic.

The purpose of this project is to create a power grid simulator that enables the user to easily influence the scenario through a hardware controller with interactive controls, and present realistic power grid scenario(s) together with predicted values that provide useful information about the grid. If possible, the simulator should also include scenario(s) which may pose challenges to the user. The target audience is not power grid experts, but instead interested persons or students without expertise in power grid dynamics to understand what is happening. The goal is to give the user a feeling of how the grid works, what possible problems can arise, what information can be used to fix them and how they can be fixed. Furthermore, it was stated in the project proposal that the intention is to place the simulator in the EEMCS faculty of TU Delft, with the goal of getting current and prospective students engaged in the area of power grids.

In order to accomplish the goals of the project, it has been divided into three subsystems: Hardware, Visualization & Simulation, and Forecasting & Scenario. This thesis will focus on the work of the Forecasting & Scenario subgroup.

1.1. Problem Definition

The Forecasting & Scenario subsystem has to provide suitable and sufficient realistic data to design the power grid and train a forecasting model on it. All the grid's relevant parameters have to be extracted and/or calculated, and correctly transmitted to the Visualization & Simulation module. Next to that, a forecasting model has to be trained to predict the loads and the predictions have to be transmitted together with the parameters so they can be displayed to the user. Moreover, this module should be capable of providing scenario(s) that present a problem in the grid which the user should be able to remediate by interacting with the simulator.

1.2. Thesis Structure

To address the outlined challenges, first a set of requirements will be defined. Building on the requirements, the design choices will be presented, and then the two individual components, scenario and forecasting, will be discussed in depth.

2

Programme of Requirements

In order to clearly define the scope of the project, a set of requirements have to be created that the subgroup should aim at fulfilling throughout the project, and can be used to determine its success.

2.1. Global System Requirements

Following the general project description provided in the introduction, the global system requirements are as follows:

1. The user must be able to interact with the simulated power grid and see how this affects the power grid dynamic.
2. The user must be able to see useful information about the grid, such as power generations and loads or line loadings.
3. The user must be able to access forecasted values of buses, which can be useful when taking actions in the grid.
4. The user must be able to change the portrayed time in the simulation.
5. The modules must be able to send and receive data between one another in an agreed-upon format and form.
6. The system must use a working solar and wind generator in real life that links to the scenario.

2.2. Forecasting & Scenario Requirements

From the global requirements, the Forecasting & Scenario requirements can be derived. The simulator needs to be provided with a working power grid set-up together with feasible load and generation data. The user should be encouraged to interact with the simulator in order to understand the power grid dynamics, as if it were a game. Predicted values to be used by the user when taking actions in the simulated environment must also be provided in the simulation. It is important that the data is feasible and correctly transmitted to the Visualization & Simulation module. Furthermore, if possible, the configuration should represent the Dutch power grid to increase engagement by the targeted user.

Mandatory Requirements (MR)

The mandatory requirements must be complied with for the design to be acceptable. These are as follows:

1. The Forecasting & Scenario module must provide a working power grid network configuration, including line and static bus parameters.
2. Feasible load and generator data that works with the network configuration, including power and voltage data must be obtained.
3. The load and generator data must be obtained for time intervals of maximum 24/8 = 3 hours, for the user to be able to significantly experience the changes in the power grid throughout a simulated day.
4. Forecasted values of bus-wise load with a reasonable horizon time[1] given the circumstances must be provided for every time step that the simulation is run for.
5. All the data must be correctly transmitted to the Visualization & Simulation module such that it can be read, used in the grid calculations, and portrayed correctly.
6. The models must be made open source to encourage advancement in this field, show the users that it is safe, and possibly allow for future development by other interested parties.

Trade-off Requirements (ToR)

The trade-off requirements should be an indication of the level of satisfaction of the users with the product. These are as follows:

1. The Forecasting & Scenario module should make use of a network and data that simulate the Dutch power grid, to provide the targetted user with a more realistic experience of a real power grid that they can recognize.
2. The used data should preferably be optimal given the structure of the power grid network and the provided known parameters, in order to resemble the operation of a real grid.
3. The power grid should preferably be represented in AC, not DC, to give a more realistic feel of a real power grid.
4. The data should be obtained for a time period of at least one year, so the variations throughout a year can be experienced by the user if desired.
5. At least one scenario simulating a malfunction in the grid should be created to encourage the user to interact with the simulator and get a better understanding of potential problems in the power grid.
6. The forecasted values should have a mean absolute error lower than if simply using the previous time step, previous day or previous week load value as prediction.
7. Both the scenario selection and forecast processes should easily be able to adapt to any given initial time step and total duration, so different time periods can be displayed.

[1] The amount of time into the future that is being predicted at each time step.

3

Design Choices

Following the Programme of Requirements, design choices have to be made that comply with all the requirements in it. This chapter will present those choices and describe the reasoning behind them.

3.1. Grid Set-up and Data

Grid Set-up

To make a realistic power grid simulator, the grid set-up and data should preferably simulate an existing real power grid. Since full power grid configurations would be hard to visualize, and very complicated to work with, a simplified version of it should be able to fulfill the project's purposes. The target user of this power grid simulator will be current or prospective students of the TU Delft, so it should ideally model a simplified version of the Dutch power grid.

After some discussions, it was decided that the grid set-up should preferably be that of a high-voltage (HV) transmission system, so that the windmill and solar panel inputs of the Hardware subsystem could represent an offshore wind park and a solar park respectively. For simplicity purposes, it was assumed that the simulated grid is balanced, meaning there is no mismatch between load and demand, and is hence maintained at 50 Hz [1]. This allows for accurate estimations of a three-phase HV system by using single-phase calculations. [2]

Data

In order to make an accurate simulation and an in-depth analysis of the power flows, it was decided that no DC simplification would be used for the grid, and AC calculations would be done instead.

Then, the data needed for the calculations of the AC power grid simulation by the Visualization & Simulation subsystem is:

- Bus geographical locations and their types (PV, PQ, slack or main slack).
- Corresponding data for each type of bus. Depending on the type, this may include total active power generation P and voltage magnitude V_m (for PV), total active P and reactive Q load power (for PQ), or voltage magnitude V_m and angle V_a (for slack and main slack), as well as maximum reactive power generation Q_{max} for PV and slack.
- Lines connecting the buses and their respective characteristic data. This includes total resistance R, reactance X, conductance G, susceptance B and maximum power flow P_{max}. Similarly, the same data for transformer lines.

The data needs to be accurate, measured often enough to provide a significant daily simulation (maximum 3-hour intervals), and sufficient enough to both be able to have variation and train a forecasting model on it (preferably at least a year).

During the course of the project, multiple national open-source datasets were explored, even for countries other than the Netherlands, but they usually did not contain individual bus data, fulfill the required time step interval, and/or fulfill the required total time period. The Python library OpenSTEF [3] was considered as it included a few load datasets of Dutch cities with 15-minute intervals throughout some months. The idea was to use that data to create more for all the necessary buses. However, it was decided against because there was too little data for it to be realistic enough for the project's purpose.

Optimal Power Flow

If the load data is obtained for the required specifications (time step interval and total time period), then given the power grid network configuration and adequate cost functions, the optimal values for the rest of the needed data can be calculated using an Optimal Power Flow (OPF) algorithm [4]. For simplicity and application purposes, as well as to make the process resemble a real-life case to a reasonable extent, the economic dispatch problem variant will be employed. The economic dispatch problem is the most widely used OPF variant [5], and it aims to find a minimum cost solution of power demand and supply in the grid. To make that possible, the costs of individual power generations must be defined through cost functions.

Another question was whether to use DC or AC Optimal Power Flow calculations. The DC OPF is a linearization of the original non-linear AC OPF equations, and therefore converges easily. The remaining necessary AC parameters can then be approximated using assumptions. On the other hand, the AC OPF is the most accurate representation of the power flows in a system. Compared to DC, the benefits of AC OPF are increased accuracy, inclusion of voltage, reactive power, current, and transmission losses in the network [6]. However, since the system of equations is quadratic, it is much harder to solve. The way to approach this is to ”convexify” the AC OPF by defining a convex function around the original non-convex one. Nevertheless, this might in some cases still lead to a solution not converging. Since it is important to retain accuracy in the calculations wherever possible, the decision was to attempt to use AC OPF as a first choice, and if not enough converging data could be obtained then switch to DC OPF.

Final Choice

After continuous research, the model Dutch-HV-Power-System by W. Zomerdijk [7], representing the aggregated high-voltage Dutch power grid in 2018, was found. It includes a power grid set-up with bus geographic locations, load and generator locations, interconnections, and line data. It also includes a full year of hourly time series for individual loads' active power and renewable generators' maximum active power, as well as nominal power for non-renewable generators. See Figure 3.1 for a depiction of the grid configuration.

Figure 3.1: Depiction of the Dutch aggregated grid represented by the Dutch-HV-Power-System model. Green and red represent the 220 kV and 380 kV systems respectively. Obtained from [8]

The process of obtaining the data for W. Zomerdijk's model is described in detail in his Master's Thesis [9]. A brief description follows:

- The relative load profiles per end-use sector are given by the Energy Transition Model [10] in terms of demand profiles defined in NEDU[1] (MFFBAS[2]) [11]. The total load per year per sector is comes from Klimaatmonitor [12]. The corresponding load data per sector is then found by multiplying the relative load profiles by the total load. Finally, assumptions are made to map the load time series to each of the individual loads in the grid set-up.
- The renewable generation capacity data is determined based on wind and solar power statistics by CBS[3] [13][14]. Renewable energy sources depend on weather, so their capacity varies throughout the year. The time series of capacity factors is based on weather profiles, and is obtained through the website Renewables.ninja [15][16]. Finally, the time series of maximum active power for renewable generators is obtained by multiplying the total generation capacity by the capacity factors. Similarly, assumptions are made to map the generation time series to each of the individual generators in the grid set-up.
- The nominal power or total capacity for the non-renewable generators is obtained using a combination of [17] and [18], and then divided by fuel type.
- The specific line data is obtained from the TenneT data sheet [19]. The line length of a few interconnection buses to other countries is not provided in this specification. However, since they connect interconnection buses with the Dutch grid in a very short distance, an average value of 0.1 km is assumed. Similarly, the transformer data is also obtained from TenneT[4].
- The connections to other countries are modeled as generators, with the upper and lower limits being set to the maximum capacity of their interconnection lines. The data on the capacity of the interconnection lines with Germany and Belgium is obtained from [19], while for the other countries, it is also obtained from the TenneT website.

[1] Vereniging Nederlandse Energie Data Uitwisseling
[2] Marktfaciliteringsforum en Beheerder Afspraken Stelsel
[3] Centraal Bureau voor de Statistiek
[4] https://www.tennet.eu/

- Suitable time-dependent cost functions for generators are obtained by determining the fuel cost per different production plant types. For connections to other countries (represented as slack nodes), two assumptions are made as no hourly data could be found: electricity is priced at zero when exporting, so national generators don't produce only to export, and just above the most expensive national generator when importing, so national generators are prioritized and thus electricity is only imported in times of need.

It should be noted that some of the direct data sources may not be directly accessible anymore since they were originally used three years ago. If that is the case, the general website of the source providers is cited instead.

3.2. Programming Language and Simulator

Multiple options for the power grid simulation and load forecasting were explored, initially in C++ and later in Python. It was determined that it would be most efficient to provide the grid data and predicted values at the beginning of the simulation, and there was no explicit need for real-time communication. This also ensures that there would be no compatibility issues with the Visualization & Simulation module which uses C#, meaning any programming language could be used.

The aggregated Dutch grid model was already formatted in a pandapower[5] network, with pandapower being an effective and convenient Python library for power grid calculations [20], so there would be no reason not to choose it for the power grid simulations. Moreover, most of the forecasting open-source libraries and models are written in Python, therefore also indicating a clear inclination to Python as the programming language of choice.

3.3. Forecasting

State-of-the-art Analysis

Short Term Load Forecasting (STLF) is an active field of study with significant advancements in both statistical and machine learning (ML) approaches. STLF methods can broadly be categorized into these two groups. Statistical methods, including auto-regressive integrated moving average (ARIMA), linear regression, and Kalman filtering, have been traditionally popular. However, these methods have notable limitations such as their linear nature, limited adaptability, difficulty in handling complex seasonal patterns and challenges in capturing long-term dependencies. [21]

ML models offer greater flexibility in modeling nonlinear functions. Unlike statistical methods, ML models do not require strong assumptions about the mapping function and can learn relationships between predictors and targets directly from historical data.

Neural networks (NNs) have particularly gained popularity in recent years due to their ability to address complex forecasting problems, including STLF. The diversity of architectural solutions and mechanisms to improve performance has encouraged the use of NNs in this field. Several different NNs were investigated for suitability of STLF for the hourly electrical load in the Polish power system in [22]. In this study, both General Regression NN (GRNN) and Multilayer Perceptron (MLP) stand out as performing with the highest accuracy. Another important thing to note is that this study suggests using the following features: weather conditions (temperature, wind speed, cloud cover, humidity, precipitation), time, demography, economy, electricity prices, and other factors such as geographical conditions, consumer types and their habits. However, in their work they only use univariate forecasting methods, in which only historical load time series is used as input.

In [23] a number of Deep NNs (DNNs) were compared on different datasets to a Gradient Boosting Regression Tree (GBRT). Concluding that a simpler model such as GBRT can compete and sometimes

[5] https://pandapower.readthedocs.io/en/latest/

outperform modern DNNs by efficiently feature-engineering the input and output structures of GBRT. Moreover, according to the same study, deep learning models tend to be overly complex in comparison to traditional techniques.

Random Forests (RF) is an ensemble learning algorithm based on decision trees as the base models. It is suitable for both regression and classification problems. RF overcomes the common drawbacks of single decision trees, such as unstable splits and lack of smoothness. [21]

As shown in [24], RF can compete with both classical models and NNs in STLF. It is able to deal with complex time series using appropriate data preprocessing, which produces normalized patterns of daily profiles. The fact that tree-based methods are strong competitors against NNs can also be deduced from the 2020 M5 competition. Among the top five winning models, four utilized variations of tree-based methods [25].

Another type of tree-based ensemble is named eXtreme Gradient Boosting (XGBoost). In [26] multiple variants of this algorithm were used for forecasting electricity consumption by industrial customers. Even the XGBoost without any combination of other methods performed better than other models which did not make use of a XGBoost model. XGBoost has several advantages such as dealing well with possible gaps in a dataset, dealing well with nonlinear regression [22], it hardly overfits because it incorporates a regularized model [27], it is relatively easy to use and has proven to be really effective. However the challenges of this algorithm should not be overlooked. One of them is the amount of hyper-parameters, because the algorithm relies on relatively many hyper-parameters that could drastically change the performance of the model. From this follows that finding a great balance in tuning the hyper-parameters and determine the right data characteristics is an important part.

Selected Model

There are numerous options available for STLF, and choosing the optimal model depends on several factors. Given the significant daily load fluctuations, the model should effectively handle non-linear patterns. Additionally, the model should also provide accurate results, even with datasets that cover shorter time spans. Although Neural Networks can be very promising, they typically require larger training datasets. Additionally, these models are difficult to interpret and make it difficult to enforce assumptions such as temporal smoothness [28]. As [29] suggests, NNs tends to outperform traditional models from a certain amount of observations on. In other words, the bigger the dataset, the better the performance and dominance of a NN compared to other models. However the dataset used in this study contains just one year of hourly load data, which is not a relative big dataset as other studies often use multiple years of training data. This leads to a design choice not to use a NN, a combination of the need for a relatively big dataset, difficult to interpret and the possible lack of temporal smoothness.

As a strong competitor of NNs, XGBoost excells in performance, proving its oustanding performance by winning the most solutions on the machine learning competition site Kaggle in 2015. Out of 29 solutions, 17 used XGBoost, these 17 solutions can be divided into two categories. Eight of these 17 solutions solely used XGBoost and while the other solutions combines XGBoost with NNs. [30]

Two characteristics of XGBoost were of great importance for the model selection: its ability to handle non-linear regression and its resistance to overfitting (as already explained in 3.3). Additionally, the model's ease to use was a significant advantage, particularly compared to the challenges of interpreting NN results. Given the researchers' potential lack of experience in this field, the ease of use and an interpretable model become a factor which should also be taken into account for model selection.

XGBoost's proven high performance, its ability to handle non-linear regression, its resistance to overfitting and that fact that the model was quite easy to use resulted in the final choice for this project.

3.4. Malfunction Scenario

To engage the user with the simulation, a malfunction scenario has to be represented. Initially, many suggestions were considered. From simulating an unexpectedly increased load causing a need for higher generation, to shutting off a line causing overload. Since this was the last part to be worked on, after having a working simulator and forecasting model, it was left open at the beginning. Later, it was seen that the implementation from the other subsystems that would allow for these suggestions and actions to solve them, such as time-varying line data or the possibility for the user to turn off or on lines, were not going to be feasible with the time constraints.

Balancing the amount of data that the Visualization & Simulation module could handle at a time without it affecting the visuals, while providing enough time steps to have a significant simulation and show a potential malfunction scenario, led to a 2-day simulation. Combined with the possible implementations for different malfunctions, this resulted in the decision to simulate the normal operation of the grid on the first day, and a scenario where the generator in a bus unexpectedly stops providing power for a few time steps. If this generator is in a bus connected to the solar and wind buses which the user can control, and the lines connecting it to other buses are close to their loading limit, the user can use the solar and/or wind generator to avoid overloading in the other lines. At the same time, this choice of malfunction also provides a realistic representation of a problem that might occur in real grids.

Furthermore, the forecasting module comes with an inherent malfunction scenario. Since it is not going to be perfect, it may give inaccurate predictions at times, which if acted upon by the user will create a mismatch in the network.

4

Communication

This chapter provides an overview of the choices made for communication with other subsystems, including the decided format.

4.1. Global Structure

After the simulator has been turned on, the Forecasting & Scenario module will find the parameters for the selected scenario and the corresponding load forecasts for the full duration of the scenario. The parameters and predictions will be sent to the Visualization & Simulation module, where the power flows will be calculated and the grid will thereafter be displayed. The user will then be able to interact with the simulator using the various hardware components, and the Hardware module will transmit the information to the Visualization & Simulation module so that the grid can be re-simulated and re-displayed. See a depiction of the general structure and communication in Figure 4.1.

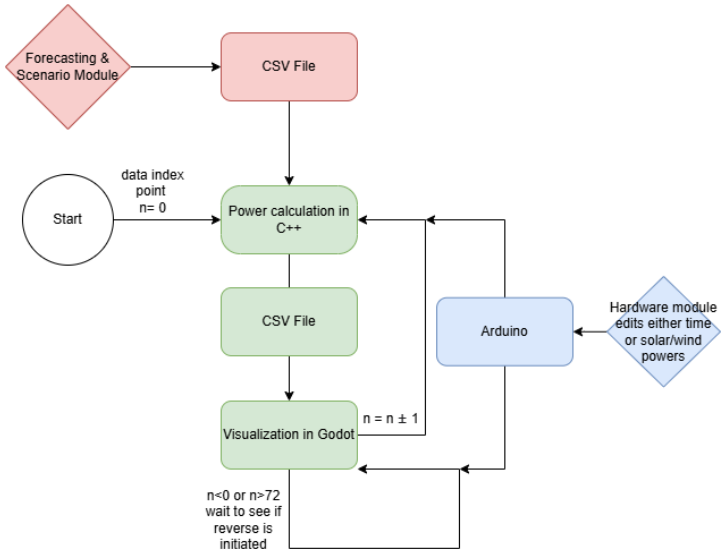


Figure 4.1: Overview of subgroups interconnections within the project

4.2. Design Choices

The Forecasting & Scenario module only needs to communicate with the Visualization & Simulation module, in order to provide it with the power grid data and forecasted values. The communication is unilateral and handled via CSV files due to their known attributes of simplicity, compatibility, and efficiency [31]. CSV files are also in a human-readable format and support excellent libraries for both C++ and Python. The data for all time steps is sent before the start of the simulation, since this is more efficient and there is no specific need for real-time communication. This also simplifies the compatibility between modules, since Godot, the game engine used by the Visualization & Simulation module, cannot handle Python code, which is essential both for obtaining the grid data and for machine learning.

The full Scenario and Forecasting modules are both automated in such a way that the data for a different set of time steps than the pre-selected ones can easily be obtained by inputting a different start time step and duration time. However, for the user to be able to select the time steps, an implementation in the Hardware module, and possibly in the Visualization module, as well as communication from the Hardware module would be required. This has not been possible to implement due to time constraints, but if it were done in a future development of the simulator, there would be no need to alter the Forecasting & Scenario module.

4.3. Scenario Data

The corresponding bus data for the pre-selected time steps, together with the line data, is sent to the Visualization & Simulation module. This is done in two separate CSV files. The first file, ”busInputData.csv”, contains the required initialization values for each bus. The file has the format of 11 (bus parameters) columns, **by** 47 (buses) $\times$ T (total time steps) rows. The bus parameters include bus type (1 = Main slack, 2 = PV, 3 = PQ), voltage magnitude and angle, generation and load active and reactive powers, maximum generation reactive power, and slack weight. All power and voltage magnitude values are given in p.u., relative to their base values. Moreover, the time step-wise data is directly concatenated. A partial overview (first time step and first 8 buses) of the agreed-upon format for the bus data can be seen in Figure 4.2.

	A	B	C	D	E	F	G	H	I	J	K
1	Type	PU Volt	Angle (Deg)	Gen MW	Gen Mvar	Load MW	Load Mvar	Gen Mvar(max)	Gen Mvar(min)	Slack yes,no	Slack Weight
2	1	1	0	0		0	0	50	-50	1	1
3	3	1.029049	-2.11144	0		0	0			0	0
4	2	1.031064	-1.972398	1.07364		1.847748	0.607325	10	-10	0	0
5	2	1.032532	-1.669874	1.43E-06		1.750798	0.637787	20	-20	0	0
6	3	1.035399	0.8264883	0		1.539603	0.506043			0	0
7	3	1.03062	-1.900822	0		0.235374	0.212207			0	0
8	2	1.038637	1.457308	7.42E-05		2.158964	0.8126	40	-40	0	0
9	3	1.035664	-0.207749	0		0	0			0	0

Figure 4.2: Format of bus data for one time step, first 8 buses

The second CSV file contains information about the high-voltage lines that connect the buses, forming the network’s topology. The file is named ”lineInputData.csv” and consists of 7 (line parameters) columns, **by** 54 (lines) rows. The line parameters include connected buses, total line resistance R, reactance X, conductance G, susceptance B, and maximum line power. All impedance and admittance values are given in p.u., relative to their base values. A partial overview (first 4 lines) can be seen in Figure 4.3.

	A	B	C	D	E	F	G
1	From Bus	To Bus	R	X	G	B	Max MVA
2	44	27	5.54E-05	0.000644	0	0.073548	19
3	3	2	0.000774	0.008534	0	0.095456	11
4	4	8	0.001068	0.011757	0	0.131024	11
5	4	6	0.000703	0.007736	0	0.08664	11

Figure 4.3: Format of line data for 4 lines

4.4. Forecasted Data

The 24-hour bus load predictions for each bus during all simulated time steps are stored in a single CSV file named ”Forecast.csv”. The agreed-upon format is structured as a matrix of 25 columns, **by** 28 (buses) $\times$ T (total time steps) rows. The first column shows the bus number, and the following 24 columns are the 24-hour load forecasts done at the pertaining time step. Moreover, like for the bus input data, the time step-wise data is directly concatenated. An example of the format can be seen in Figure 4.4.

	A	B	C	D	E	F	G	H	I	J	K
1		0	1	2	3	4	5	6	7	8	9
2	3	184.8496	185.3802	185.4144	186.3734	192.5285	201.5621	206.6579	209.3825	209.6447	209.5984
3	4	191.8669	186.0588	185.7242	188.6142	197.3084	246.6955	313.1868	388.1314	400.2298	405.4756
4	5	154.0227	154.4647	154.4932	155.2619	160.421	167.948	172.194	174.4642	174.6827	174.6441
5	6	64.58867	64.77403	64.78597	65.10832	67.27176	70.42819	72.20872	73.16073	73.25233	73.23615

Figure 4.4: Format of the forecasted load values for one time step, first 4 buses and 10 predicted hours

5 Scenario

After having decided the grid configuration, dataset, programming language and library to use for providing the scenario, there are multiple steps remaining: the correct functioning of the whole system needs to be confirmed and understood, the remaining data needs to be obtained and tested for feasibility, all data needs to be correctly transformed and formatted for the Visualization & Simulation module, and the whole process needs to be structured and parameterized so that it can easily be adapted to different sets of time steps.

The code used for this section can be found on Git Hub[1].

5.1. Pandapower

Pandapower combines pandas[2] and PYPOWER[3] into an efficient and easy-to-use library capable of maintaining complex network structures and performing optimized calculations. [32]

The different elements for modeling a power grid in pandapower are as follows:

- The base of a pandapower system is a network. This is a class[4] that will contain all the information of the system.
- Buses can be added to the network by specifying their nominal voltage, upper and lower bounds for voltage magnitude, and geolocation data.
- Loads are added by specifying which bus they are placed at, their type (wye or delta), and whether their active and reactive power is allowed to be modified when conducting Optimal Power Flow (OPF) calculations. Since existing data is in place, this should not be allowed for loads.
- Generators can be modelled in two different ways, as voltage-controlled (standard), or as non-voltage-controlled (static). Standard generators require given values for voltage and active power, while static ones require active and reactive power. This distinction is to model the real-life behaviours of different types of generators: Non-renewable generators are able to be voltage-controlled to adapt when required, and are therefore modeled as standard generators. Renewable generators, such as wind and solar plants, are modeled as static generators. Generators can be added by specifying the bus they are placed at, their minimum and maximum active and reactive power, and whether the OPF algorithm should be allowed to modify (control) its values.

[1] https://github.com/bbucksch/BacelorEndProject.git
[2] A library for analyzing data.
[3] A library for calculating power flows.
[4] A python data structure

- Slack buses, assumed to be connections to neighbouring countries, are inserted by specifying bus, minimum and maximum active and reactive powers and whether they are controllable by OPF or not.
- Lines are then added to interconnect all same-voltage buses, given their characteristic data, including length in km, impedance and admittance values per km, maximum tolerated current, and nominal voltage.
- Transformers are added to connect the buses of different voltage levels, with their characteristic data including nominal power, high- and low-voltage bus nominal voltages, and relative complex and real short circuit voltage.
- Two types of possible cost functions can be added to the network: piece-wise linear and polynomial cost functions. These are assigned to individual generators and/or slack buses, and describe their electricity production costs. These are used for OPF calculations.
- Every component (excluding buses, lines and transformers) can be given time series data by making use of controllers. A controller takes a data frame of the desired time series values, and is assigned to one (or multiple) components. The controller then ensures that the power flow simulations read the data from the appropriate time step of the time series data.

5.2. AC Optimal Power Flow

AC Optimal Power Flow (AC-OPF) is an optimization algorithm that considers the AC Power flow calculations. Compared to DC-OPF, AC-OPF offers increased accuracy by considering voltage, reactive power, currents, and transmission losses. However, AC power flow equations are quadratic, leading to non-linear, non-convex optimization problems which are more challenging to solve. Figure 5.1, as well as the equations, reasoning and steps that follow in Section 5.2 are taken from [33].

Modelling elements

The π-model is the standard representation for transmission lines in power systems which can be seen in fig.5.1. This figure shows a transmission line connected between the nodes i and j. The model consists of a series impedance $R_{ij} + jX_{ij}$ and two shunt susceptances $\frac{jB_{ij}}{2}$ at both ends of the line. The series admittance is $y_{ij} = \frac{1}{R_{ij}+jX_{ij}}$, and the shunt susceptances are $y_{sh,i} = y_{sh,j} = \frac{jB_{ij}}{2}$.

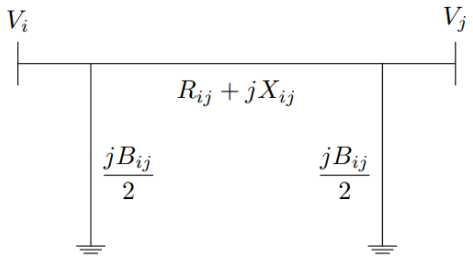


Figure 5.1: π-model of the transmission line

The current entering node i in the π-model splits between the series admittance y_{ij} and the shunt susceptance $y_{sh,i}$. The total current entering node i is:

$$I_{i\to j} = I_{sh,i} + I_{ij} = y_{sh,i}V_i + y_{ij}(V_i - V_j) \tag{5.1}$$

In matrix form:

$$I_{i\to j} = \begin{bmatrix} y_{sh,i} + y_{ij} & -y_{ij} \end{bmatrix} \begin{bmatrix} V_i \\ V_j \end{bmatrix} \tag{5.2}$$

Similarly in the case for a current entering the line at node j:

$$I_{j\to i} = I_{sh,j} + I_{ji} = y_{sh,j}V_j + y_{ij}(V_j - V_i) \tag{5.3}$$

In matrix form:

$$I_{j\to i} = \begin{bmatrix} -y_{ij} & y_{sh,j} + y_{ij} \end{bmatrix} \begin{bmatrix} V_i \\ V_j \end{bmatrix} \tag{5.4}$$

Comparing 5.1 and 5.3 a discrepancy can be observed as $I_{i\to j} \neq -I_{j\to i}$. This means that part of the current is lost during transport, this difference is due to losses.

Line Admittance Matrix

The line admittance matrix $\mathbf{Y}_{line}$ links bus voltages to current flows. This formula is defined by:

$$\mathbf{I}_{line} = \mathbf{Y}_{line}\mathbf{V} \tag{5.5}$$

Separate admittance matrices are required for currents flowing in opposite directions due to differences caused by losses. In other words, two line admittance matrices will be formulated, one for the direction $i \to j$ and one for the direction $j \to i$.

$$\begin{bmatrix} I_{1\to 2} \\ \vdots \\ I_{i\to j} \\ \vdots \\ I_{m\to n} \end{bmatrix} = \begin{bmatrix} y_{sh,1} + y_{12} & -y_{12} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & y_{sh,i} + y_{ij} & \cdots & -y_{ij} & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & y_{sh,m} + y_{mn} & \cdots & -y_{mn} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_i \\ \vdots \\ V_j \\ \vdots \\ V_n \end{bmatrix} \tag{5.6}$$

and

$$\begin{bmatrix} I_{2\to 1} \\ \vdots \\ I_{j\to i} \\ \vdots \\ I_{n\to m} \end{bmatrix} = \begin{bmatrix} -y_{12} & y_{sh,1} + y_{12} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & -y_{ij} & \cdots & y_{sh,i} + y_{ij} & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & -y_{mn} & \cdots & y_{sh,m} + y_{mn} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_i \\ \vdots \\ V_j \\ \vdots \\ V_n \end{bmatrix} \tag{5.7}$$

Creating the line admittance matrix:

1. $\mathbf{Y}_{line}$ is an $L \times N$ matrix, where L is the number of lines and N is the number of nodes.
2. If row k corresponds to line $i \to j$:
    - Start node i: $Y_{line,ki} = y_{sh,i} + y_{ij}$
    - End node j: $Y_{line,kj} = -y_{ij}$
    - Rest of the row elements are zero
3. $y_{ij} = \frac{1}{R_{ij} + jX_{ij}}$ is the admittance of line ij.

4. $y_{sh,i}$ is the shunt capacitance $\frac{jB_{ij}}{2}$ of the π-model of the line.
5. Then create two Y_{line} matrices: One for $i \to j$ and one for $j \to i$.

Bus Admittance Matrix

To form the AC-OPF constraints, it is necessary to compute the bus power injections, which show the net power entering or leaving a bus. The net apparent power at a bus i is:

$$S_i = V_i I_i^* \tag{5.8}$$

According to Kirchhoff's law, the net current injection at a bus is the sum of currents leaving the bus:

$$I_i = \sum_k I_{ik} \tag{5.9}$$

where k represents all buses connected to bus i.

Assuming bus i connects to buses m and n:

$$\begin{aligned} I_i &= I_{im} + I_{in} \\ &= (y_{sh,i} + y_{im})V_i - y_{im}V_m + (y_{sh,i} + y_{in})V_i - y_{in}V_n \\ &= (y_{sh,i} + y_{im} + y_{sh,i} + y_{in})V_i - y_{im}V_m - y_{in}V_n \end{aligned} \tag{5.10}$$

In matrix form:

$$I_i = \begin{bmatrix} y_{sh,i} + y_{im} + y_{sh,i} + y_{in} & -y_{im} & -y_{in} \end{bmatrix} \begin{bmatrix} V_i \\ V_m \\ V_n \end{bmatrix} \tag{5.11}$$

Just like the line admittance matrix, an algebraic relationship between the vectors of bus currents and bus voltages can be defined as the bus admittance matrix $\mathbf{Y}_{bus}$:

$$\mathbf{I}_{bus} = \mathbf{Y}_{bus}\mathbf{V} \tag{5.12}$$

For a generalized power system configuration, where bus i connects buses m and n, and where bus 1 is solely linked to bus 2 while bus n exclusively connects to bus i, the Bus Admittance Matrix has the following form:

$$\begin{bmatrix} I_1 \\ \vdots \\ I_i \\ \vdots \\ I_n \end{bmatrix} = \begin{bmatrix} y_{sh,1} + y_{12} & -y_{12} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & y_{sh,im} + y_{im} + y_{sh,in} + y_{in} & \cdots & y_{im} & -y_{in} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & -y_{in} & \cdots & \cdots & y_{sh,n} + y_{in} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_i \\ \vdots \\ V_j \\ \vdots \\ V_n \end{bmatrix} \tag{5.13}$$

Creating the bus admittance matrix:

1. $\mathbf{Y}_{bus}$ is an $N \times N$ matrix, where N is the number of nodes.

2. Diagonal elements:
$$Y_{bus,ii} = \sum_{t \in I} y_{sh,t} + \sum_{k} y_{ik}$$
where k are all buses connected to bus i.
3. Off-diagonal elements:
   - $Y_{bus,ij} = -y_{ij}$ if nodes i and j are connected by a line.
   - $Y_{bus,ij} = 0$ if nodes i and j are not connected.
4. $y_{ij} = \frac{1}{R_{ij}+jX_{ij}}$ is the admittance of line ij.
5. $y_{sh,i}$ includes all shunt elements t connected to bus i, including the shunt capacitance of the π-model of the line.

AC Power Flow Equations

Equation 5.13 shows that I_i, the current injection at bus i, equals $\mathbf{Y}_{bus,row-i}\mathbf{V}$, where $\mathbf{Y}_{bus,row-i}$ represents the i-th row of the bus admittance matrix $\mathbf{Y}_{bus}$. Therefore, the apparent power at bus i is given by:
$$S_i = V_i I_i^* = V_i \mathbf{Y}_{bus,row-i}^* \mathbf{V}^*$$

To express the bus apparent power in vector form, we introduce diag$(\mathbf{V})$. This notation signifies a diagonal $N \times N$ matrix, where the N diagonal elements are equal to the $N \times 1$ vector $\mathbf{V}$, and all the other elements of the matrix are zero.

Thus, the vector of apparent powers for all buses, $\mathbf{S} = [S_1, \ldots, S_N]^T$, can be formulated as:
$$\mathbf{S} = \text{diag}(\mathbf{V})\mathbf{Y}_{bus}^* \mathbf{V}^* \tag{5.14}$$

The net apparent power injection at each bus equals the total generation minus the total load connected to that bus. In vector form, this is represented as:
$$\mathbf{S} = \mathbf{S}_{gen} - \mathbf{S}_{load} \tag{5.15}$$

Combining Equations 5.14 and 5.15, we derive the AC power flow equations:
$$\mathbf{S}_{gen} - \mathbf{S}_{load} = \text{diag}(\mathbf{V})\mathbf{Y}_{bus}^* \mathbf{V}^* \tag{5.16}$$

The objective function regarding the minimization of the costs for producing electricity is defined as follows:
$$\min_{P_{G_i}} \sum_i c_{G_i} P_{G_i} \tag{5.17}$$

where c_i is the marginal cost of every generator and P_{G_i} is the amount of power it generates. P_{G_i} is bounded as a generator is not able to generate infinite energy and has a minimum defined by:
$$P_{G_i}^{min} \leq P_{G_i} \leq P_{G_i}^{max} \tag{5.18}$$

Important to note is that the generated power must be equal to the electricity demand P_D, defined by:
$$\sum_i P_{G_i} = P_D \tag{5.19}$$

Finding the minimum of the objective function is found using the gradient descent algorithm, also known as steepest descent. According to [34] this method is the simplest iteration scheme. The method involves moving from one feasible solution in the direction of the steepest descent (negative gradient) to a new feasible solution with a lower objective function value. Repeating this process will result in finding a either the global or local a minimum. This algorithm is based on the solution of the power flow by Newton's method which has proven to be efficient [35].

5.3. Optimal Power Flow Applied

Pandapower provides two functions for performing an OPF, one for DC and one for AC, both of which perform an economic dispatch. Since it was decided that AC calculations should be done, the AC OPF function runopp() is the one to use.

It is worth noting that the runopp() function makes use of the PYPOWER opf() function, and it is warned in the pandapower documentation that "the optimization with PYPOWER functionality does not have the best convergence properties". This means that even if the network configuration is appropriate for an optimization problem, runopp() may not converge.

Assumptions

Before being able to use the network for simulation, it has to be confirmed that everything is correct and works as intended, and potential missing data needs to be addressed. This may especially be true since the original network [8], was intended for DC Optimal Power Flow. Furthermore, some constraints were missing.

Load Reactive Power

The initial network included time series data for the active power P of loads, but not for their reactive power Q. After some research, it was determined that aggregated loads are generally assumed to have a power factor $pf = \cos(\phi)$ of 0.95 since this is a good approximation [36][37]. Using $P = \cos(\phi) \times S$ and $Q = \sin(\phi) \times S$, the time series of reactive powers can be found as $Q = P \times \frac{\sin(\phi)}{\cos(\phi)} = 0.3287 \times P$.

A new controller then needs to be created to ensure that this time series is used for the power flow calculations.

Constraints

The OPF algorithm requires upper and lower bounds on parameters in order to be able to find a converging solution. In the provided network, these are:

- Non-renewable generators: $0 \leq P \leq P_{nominal}$ using given nominal powers.
- Renewable generators: $0 \leq P \leq P_{max}(t)$ using given capacity time series and $0 \leq P \leq P_{max}(t)$
- Slack buses: $P_{min} \leq P \leq P_{max}$ using given interconnection line capacities.
- Bus voltages: All buses are limited at $0.95p.u. \leq V \leq 1.05p.u.$ as these are necessary for proper operating conditions in a grid.
- Line and transformer line loadings: Constrained at given maximum current and nominal voltage.

Missing Constraints

No constraints were put in place in the original network for the reactive powers of generators and slack buses. Since it is desirable to give the OPF algorithm freedom, the upper and lower bounds should not be very tight. Taking into account the active power limits, a reasonable limit seemed to be $\pm$ 1000 MVar for generators and $\pm$ 5000 MVar for slack buses, as they should be allowed more freedom than generators.

Non-Convergence

When first simulating the OPF, it was observed that as feared it often did not converge. After some digging, it was observed that the voltage for the main slack bus (automatically chosen by pandapower) was not kept at 1.0 p.u., but instead increased to almost 1.05 p.u. Since the voltages had been constrained between 0.95 and 1.05 p.u., this should mean that there was at least one bus voltage that was 0.1 p.u. lower than the main slack bus, and therefore the slack bus voltage had to be compensated. However, no voltages were observed below 1.04 p.u.

In order to solve this issue, different methods were attempted. A controller was added to keep the main slack voltage at 1.0 p.u., the controllability of the bus for OPF was turned off, but none of these solutions provided an acceptable result. Some troubleshooting sources suggested converting the rest of the slack buses into generators with permitted negative generation, both with and without a slack component, also to no avail. Finally, it was decided to constrain the main slack voltage at 1.0 p.u. by fixing the lower and upper bound, which ended up providing reasonable results and convergence.

Results

The OPF provided feasible results for generation values, with only a few of them acting a full capacity. As expected, the total power provided by neighboring countries was very close to 0. Bus voltages ranging from about 1.0 to 1.04 p.u., and line loadings generally ranging between about 0.002% - 40% and a peak of about 70%.

Testing

The results from the OPF need to be tested using a normal power flow using the function runpp(), to confirm that the values are correct. Controllers have to be created for standard generator active power and voltage (PV bus), static generator active and reactive power (PQ bus), and slack bus voltage magnitude and angle. After running the simulation, it was indeed confirmed that the results were the same.

5.4. Chosen Scenario

Taking into account the design choices, a specific scenario has to be chosen that complies with all the requirements. It should be 3 days long, and facilitate the selected malfunction scenario. One way to do so would be to find certain time steps where the load and generation in the malfunctioning bus are quite high, to cause a greater impact on the grid. Moreover, it should fall inside the testing data of the forecasting model, which as discussed later on should start at about 80% of the way through the year. Preferably, it should also be at the beginning of the testing data which would theoretically provide the most accurate forecast.

When performing the AC OPF, not all the time steps provide conversion, which leads to no results at those time steps. This reduces the possibility of scenario time steps to only those with continuous conversion for a period of at least 2 days (preferably 3 to be safe).

Accounting for all the restrictions, the period to be simulated was decided to be two days starting from the 16th of October 2018 at 00:00 (time step 6912).

To present the malfunction scenario, it was observed that the generation in the user-connected bus (the bus directly connected to the user input bus, which the user can control) was quite high on time steps 38 to 41, about 14:00 to 18:00, and the time series data for that generator was set at 0.

Bus Data Formatting

Since the grid model used by the Visualization & Simulation module is a simplified version of the one used for obtaining the parameters here, the data needs to be aggregated and reformatted where necessary. Bus-wise standard generator active and reactive power is separated from load + static generator active and reactive power, where static generator counts as negative load. Buses containing a standard generator are always voltage-controlled and therefore of type PV, while those containing a load and/or static generator will be of type PQ. The total generation and load power for each PV and PQ bus, as well as reactive power for PQ buses, needs to be aggregated from the results of the simulation. The voltage magnitude for PV and slack buses, as well as the voltage angle for slack buses, is extracted from the simulation. The data for components that are not active (do not actually contribute to the grid) is dropped, the slack weights for slack buses are read, and reactive power limits for generations and slack buses are obtained. The data for two more buses (user controlled buses) needs to be added, bus numbers need to be switched and all corresponding parameters need to be converted to p.u. using S_{base} = 100 MVA and V_{base} = 220 kV or 380 kV respectively.

Line Data

Many lines (including transformer lines) are modeled as double lines in the used network [8], however in order to simplify the simulation for the Visualization & Simulation, these are converted to equivalent single lines. Their resistance R, and reactance X values are obtained by taking the parallel values and dividing by $Z_{base} = \frac{V_{base}^2}{S_{base}}$. The conductance G is 0 for all lines, and susceptance B is calculated as $B = (2\pi f)^2 \times C_{parallel}$ divided by $Y_{base} = \frac{1}{Z_{base}}$, with f = 50 Hz. [38] The maximum line capacity is found as the parallel of $I_{max} \times V_{nominal}$.

For transformers, G and B are observed to be 0, while the maximum line capacity is the transformer rated apparent power S_{rated}. Using [32], the remaining parameters are found as:

$$R_k = \frac{vkr_{percent}}{100} \times \frac{S_{base}}{S_{rated}}, Z_k = \frac{vk_{percent}}{100} \times \frac{S_{base}}{S_{rated}}, X_k = \sqrt{Z_k^2 - R_k^2}, (R, X) = (R_k, X_k) \times \frac{V_{lv}^2 \times S_{base}}{S_{rated}} / Z_{base}$$

With $vk_{percent}$, $vkr_{percent}$ being short circuit voltage and the real component of it.

The two new user lines (lines that connect the user-controlled buses with the grid) connect the user buses with bus 12 and 3 respectively. These are given reasonable resistance, reactance and susceptance using their length (provided by Visualization & Simulation team) and similar parameters. Moreover, to allow the user for as much control over the grid as possible while keeping it realistic, they are given a capacity equal to the maximum of the other lines.

Malfunction Scenario

To be able to simulate the chosen malfunction scenario (see Section 3.4), the lines should tend to be quite loaded, especially those around the malfunctioning bus except for the user line (the one connecting to the user-controlled bus). However, when simulated, the line loadings were observed to be quite low. In order to make space for a malfunction scenario, all the line capacities were reduced by reducing the maximum current, with a significantly greater reduction for the lines connecting to the affected bus.

6

Forecasting

Some of the code for this section has been included in Appendix A, while the rest can be found on Git Hub[1].

6.1. XGBoost Theory

Tree Boosting

Basic knowledge of tree boosting algorithms is crucial to fully understanding XGBoost models.

A gradient boosting regression tree algorithm is constructed of multiple estimators or regression trees [39]. A regression tree is made up of branches and nodes. At each node, the algorithm attempts to split the target data (y) as accurately as possible into two subsets, depending on the value of one of the features ($x1$, $x2$,...). The chosen feature and value at which to split the data is the one that minimizes the sum of squares error $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ with $\hat{y}_i$ being the attempt to predict y_i. This is done successively in each node, until a certain user-imposed limit is reached (e.g. until <20 data points fall into that category) in order to avoid over-fitting[2].

When multiple trees (estimators) are stacked along each other and aim at compensating for the prediction errors of the preceding tree, it is called boosting. Regression tree boosting algorithms start by predicting the one value that minimizes the loss function for all output data points (y_{mean} for a loss function L = Mean Squared Error). The residual is then computed for each prediction as:

$$r_i = -\frac{\partial L(y_i, \hat{y}_i)}{\partial \hat{y}_i} = y_i - \hat{y}_i \tag{6.1}$$

And a regression tree is fit to predict these residuals using the available features X. These predictions are added to the initial prediction of y_{mean}, and this is done successively until a user-given limit of regression trees is reached. This ensures that each tree attempts to correct the prediction errors of the one that precedes it, and the model provides accurate aggregated results. To avoid over-fitting, a learning rate $0 \leq \epsilon \leq 1$ can be multiplied by the predictions of each tree.

Extreme Gradient Boosting

Extreme gradient boosting (XGB) is a special type of tree boosting. It makes use of regularization, a technique that aims at reducing overfitting by increasing generalizability.

[1] https://github.com/bbucksch/BacelorEndProject.git

[2] A model performing too well on the data it has been trained on, and not well enough on unseen data.

An XGB model is made up of XGB trees. XGB trees work like gradient boosting trees by aiming to predict the residuals of the predictions of their preceding tree. However, the splits in the tree are now decided by maximizing the amount of gain, aimed at clustering similar outputs together. Gain is defined as the similarity score on one side of the split + the similarity score on the other side - the original similarity score before splitting. Similarity is generally defined as:

$$\frac{(\sum_{i=1}^{n} y_i - \hat{y}_i)^2}{n + \lambda} \tag{6.2}$$

Where λ is the L2 regularization parameter. This means that if values are on opposing sides of the original prediction, they are not similar and hence the similarity score will be small. If they are close, the similarity score is big and therefore encourages clustering.

Pruning, another technique to avoid over-fitting, is also employed in XGB. By defining a minimum gain γ, the model can choose to prune, or remove, the branches that have a gain smaller than γ, ensuring the trees are not making too many splits.

When everything is put together, the optimization process of XGB is about minimizing the objective function, comprising two components: the loss function representing the prediction error and the regularization term penalizing model complexity (see Equation 6.3). Balancing the trade-off ensures that the resulting model generalizes effectively to unseen data while remaining accurate.

$$J^{(t)} = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{t} \Omega(f_k) \tag{6.3}$$

where predicted value and regularization component, respectively:

$$\hat{y_i}^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y_i}^{(t-1)} + f_t(x_i), \quad \Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2 \tag{6.4}$$

With t the index of iteration, n the number of samples, ω is the output or score of the leaf nodes, and T the total number of leaf nodes in the tree. When λ increases, ω tends to shift towards zero. When Ω is set to zero, the objective function is just a traditional gradient tree boosting function.

The goal is to determine the best split feature and split point next to ω_j. The objective function can be rewritten as below [30], writing out the loss function and moving the sum part of the regularization to the hessian, as they both are multiplied by ω_j. This results in the following equation:

$$J^{(t)} = \sum_{j=1}^{T} \left[G_j \omega_j + \frac{1}{2} (H_j + \lambda) \omega_j^2 \right] + \gamma T \tag{6.5}$$

Where G_j (sum of gradients) and H_j (sum of hessians) are sums of first and second-order derivative terms of the loss function respectively:

$$G_j = \sum_{i \in I_j}^{T} \frac{\partial L(y_i, \hat{y}_i)}{\partial \hat{y}_i}, \quad H_j = \sum_{i \in I_j}^{T} \frac{\partial^2 L(y_i, \hat{y}_i)}{\partial {\hat{y}_i}^2} \tag{6.6}$$

Where I_j represents all the data samples in leaf node j. To find the minimum value for $J^{(t)}$, the derivative can be taken with respect to ω and set to zero.

$$\frac{\delta J_j^{(t)}}{\delta \omega} = [G_j + (H_j + \lambda)\omega_j] = 0 \tag{6.7}$$

The optimal output value ω^* for a leaf can then be found as:

$$\omega_j^* = -\frac{G_j}{H_j + \lambda} = -\sum_{i \in I_j}^{T} \frac{-(y_i - \hat{y_i})}{T + \lambda} = \frac{\text{Sum of residuals}}{\text{number of residuals} + \lambda} \tag{6.8}$$

XGBoost applies a greedy algorithm[3] to traverse all the split points and finally selects the split points with the minimum value of the objective function after splitting. This results in the highest gain after the split when the optimal split point is selected. The gain is defined by:

$$Gain = \frac{1}{2}\left[\frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{\left(\sum_{i \in I} g_i\right)^2}{\sum_{i \in I} h_i + \lambda}\right] - \gamma \tag{6.9}$$

Where I_L and I_R denote the data sample sets of left and right nodes after splitting, I is the union set of I_L and I_R.

6.2. XGBoost Applied

The forecasting model used for this project is an XGBRegressor [40], ideal for regression problems. This model can be tuned in many different ways, making it quite adaptable for any desired purpose. Since the load for different buses has to be predicted, each likely to have their unique characteristics such as magnitude, the decision was to train individual models for each bus.

The intention with this model is to provide a 24-hour prediction at every time step. The machine learning term for this is number of horizon steps. However, the XGBRegressor can only provide single outputs or predictions at a time. The initial idea was to write it inside a for-loop and create 24 different models, one for each horizon time step. After some research, it became clear that this idea had already been implemented in a likely more efficient manner, under the name of MultiOutputRegressor[4].

Feature Engineering

The only information that is provided with the data to forecast is its time step and the time at which it started (01-01-2018 at 00:00). In order to make an accurate model, features have to be extracted from wherever possible.

By converting the time steps into date-time format, some periodic data can be obtained to help predict the load. This includes hour, day of the week, month, day of the month, quarter, and day of the year. By making some logical observations, some of these features can already be assumed are not going to be very useful. This especially includes the month, quarter, and day of the year. When predicting unseen data, the algorithm should try to use values for features that it has seen before. Since the data provided only includes a year, and the model is going to be trained for most of the year excluding the last few months, the day of the year, month, and quarter are not going to be repeated.

From the extracted features, there is yet another one that can be obtained: whether it's a working day or a holiday/weekend day. Reasonably, this should have a significant impact on the consumption of energy, since it distinguishes between when people are likely to be home and when not.

Another significant point is that periodic features are initially interpreted as having a numerical value (i.e. hour 0 is further from 23 than from 1), and can therefore be misinterpreted by the algorithm. One way around this is to encode them in a cyclical manner: dividing 2π over the possible values of the feature, assigning each an angle, and turning the feature into the sine and cosine of the angle. This way, the total magnitude of the feature remains the same independent of which value it has. Due to the

[3] An algorithm that aims to be as time-efficient as possible.

[4] As its name suggests, it combines multiple regression models that provide a single prediction at a time, in order to give multiple predictions. See website: https://scikit-learn.org/stable/modules/generated/sklearn.multioutput.MultiOutputRegressor.html

non-linearities of the XGBoost, cyclical encoding may not prove as effective as for other models, but it is still worth a try.

The best way to determine whether a particular feature is relevant or not is by training a model using all the respective features, and then analyze their importance. As previously described in Section 6.1, the feature importance, also referred to as gain, relates to the average gain in information that is obtained thanks to that feature. In other words, it indicates how well the feature does in splitting the data. Since it would take an unnecessarily long time to investigate the feature importances at all buses, 5 buses are chosen at random, and a model is trained for each of them using all the date-time features. The combined distribution of feature importances is displayed in a boxplot as seen in Figure 6.1. This should give a good enough representation of which features should generally be included.

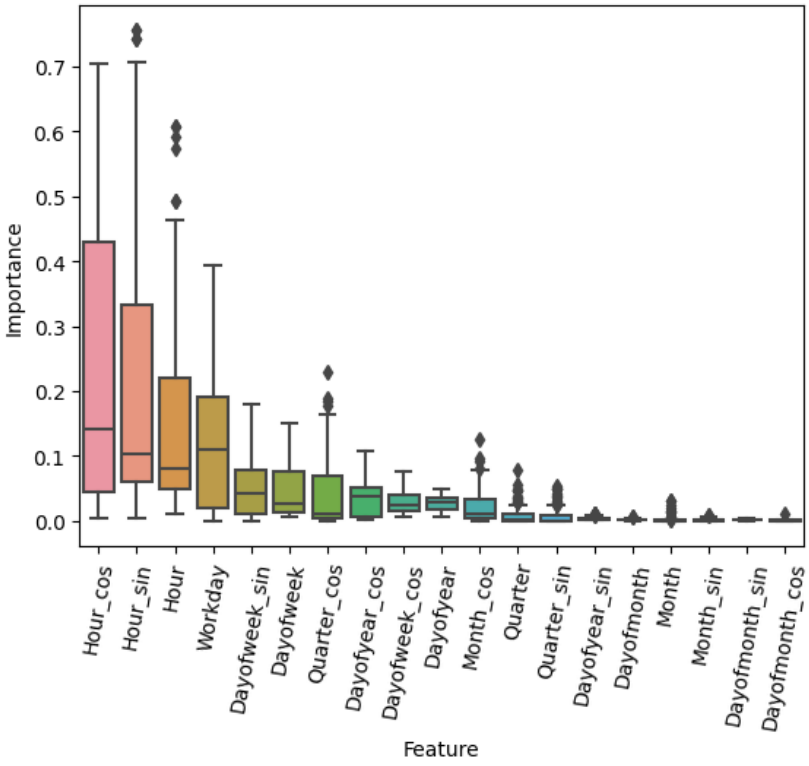


Figure 6.1: Relative date-time feature importances for buses 5, 9, 10, 14, 20.[5]

In the boxplot, it can be observed that the most important features are by far hour and working day. Also, it can be noted that the values generally lie close to the average, no abnormally big spread happens. Both the original and encoded version of the hour seem to be quite relevant. This might indicate that some information from the original format is lost when encoding it. Nevertheless, the encoded version is still more important and should be able to provide a significant representation of its original feature. The same applies to the other features.

Since a large number of features is undesired, as it takes a lot of processing time and makes the algorithm much more complicated, the aim is to only use the most important features. The lag features, explained in the following section, have been shown to have very high importance compared to date-time features (see Figure 6.2), so only the ones with average relative importance of more than 0.1 are used (excluding the non-encoded hour feature as previously discussed). This results in the sine-cosine encoding of hour, and the working day feature.

[5]Please be aware that these bus numbers are not the same as the ones in the grid. Only the buses that included loads could be used, providing a total of 28 buses, which are renamed in the forecasting scene for simplicity.

Lag Features

A quite important component for most models when aiming to forecast a time series variable is the preceding values of that same variable, as it can give a great picture of the behaviour of the data [41]. The machine learning term for this is lag features.

The downside with the use of lag features, however, is that the dataset cannot begin until the point where all the lagged features have been obtained (i.e. if a month of lagged features is used, the first data point will be one month after the start of the year). This is not a problem if a large dataset is available, but a year is not much, which limits how far back in time the lagged features can go.

In order to understand the relevance of each lagged value, the same principle is applied as for the date-time features: 5 buses are chosen at random for which different models are trained using the chosen date-time features (hour and workday), as well as all lag loads up until one week back. The combined distribution of lag feature importances is investigated. However, there seem to be too many outliers in the boxplot to make a significant plot out of it, even when reducing the amount of figures. Instead, only the average importances are plotted as seen in Figure 6.2.

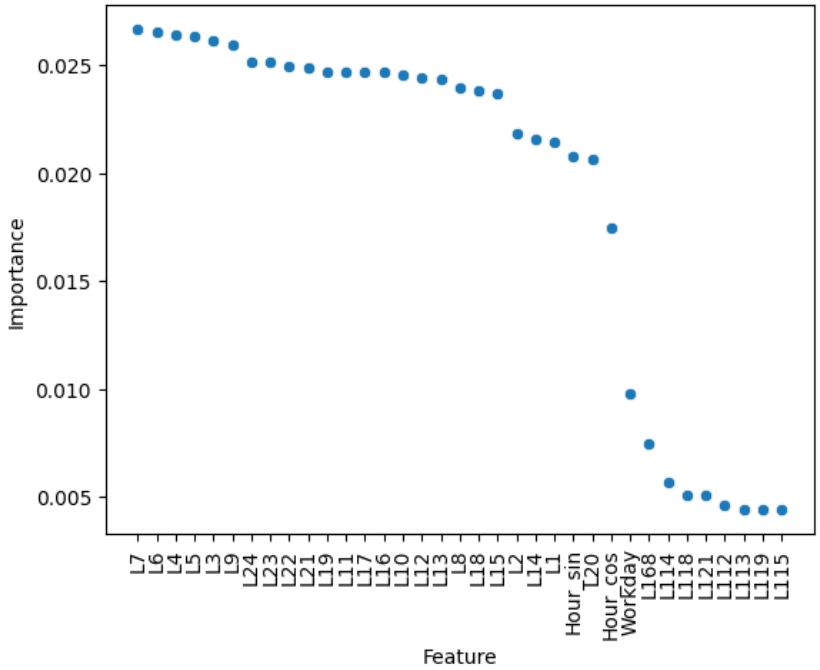


Figure 6.2: Relative lag feature importance (including the chosen date-time features) for buses 2, 4, 17, 18, 25.

Since there are many features, it is reasonable to think that the relative importance will be quite distributed. This can be observed as the maximum importance is just above 0.025 out of 1. Nevertheless, a clear distinction can be made after the cosine encoding of the hour. The importance of working day is just about 0.01, while the features that follow it can be concluded to be completely irrelevant. Furthermore, it can be observed that the previous assumption about the date-time features not being very relevant when compared to the total importance of all the lag loads is correct.

Not much can be said about the remaining lag loads, since they seem to be more or less equally important. The only thing that can be done is to use the features with the same or more importance as the lag 168 (1 week), to be on the safe side and use in a potential hyper-parameter optimization algorithm.

Hyper-parameters

Hyper-parameters are used to control the machine learning algorithms for the particular purpose required, in order to achieve the highest possible accuracy.

The important hyperparameters that this project is going to focus on are:

- Selected features
- Objective loss function: Since it is important to be relatively accurate everywhere, and hence prioritize big errors first, the squared error function is chosen for loss.
- Learning rate: The effect of each tree on the final prediction. The standard is to set it at 0.1. [42]
- Lambda: L2 regularization term. The standard is 1.
- Gamma: Controls the pruning rate of individual trees. The standard is 0.
- Max depth: How many levels the base trees are allowed to split into. The standard is 3.
- Number of estimators: Total number of base trees connected in chain. The standard is 100.
- Subsample: What fraction of the training data is used at once in each tree. Used to avoid overfitting. Standard is 1.
- Early stopping rounds: If the model is given an evaluation set, this number decides how many rounds the algorithm can keep on training without there being an improvement on the evaluation score. However, if a hyper-parameter optimization algorithm is used, this should be discarded.
- Tree method: The tree construction algorithm used in XGBoost. If a regression with the objective loss function of squared error is done, the hist method is recommended.

The default values for hyper-parameters are initially taken, in order to conduct the feature engineering. By having determined the reasonable lag loads and date-time features to use, this limits the hyper-parameter of selected features. The idea is to focus on a Tree-Structured Parzen Estimator (TPE)[6] algorithm if possible, in order to achieve maximum accuracy and depth.

Results

Since it is not possible to compare all the buses in a detailed way, a random bus is chosen to be portrayed in more detail. The bus is number 2.

Benchmarks

The benchmarks for which the algorithm had to perform better were predictions using an hour before, a day before, and a week before. On bus 2, these give the following scores:

Benchmark	Train MAE [MW]	Test MAE [MW]
1 hour	17.4579	19.0587
1 day	23.9655	26.3669
1 week	8.7388	12.6324

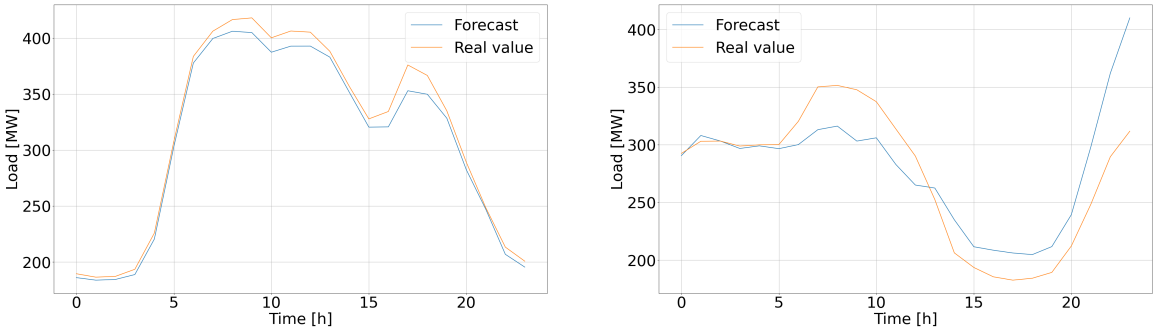
Manual Optimization

The manual optimization version of the model makes use of a total of 7-days lagged load data, and hour and workday, as discussed in Section 6.2.

The MAE scores for train and test respectively are 6.7997 MW and 11.8798 MW, which satisfies the requirement of obtaining more accuracy than the benchmarks in all fields. While the error is quite low (5% for a load of about 200 MW), a difference can still be observed between the training and testing score. Most likely, this means the model has a slight over-fit at its current version.

The resulting 24-hour prediction at the first test time step can be observed in Figure 6.3a.

[6] A hyper-parameter optimization technique that recurrently redefines its beliefs and corrects its mistakes about the effect of the hyper-parameters on the data.



(a) First 24-hour prediction and actual values for bus 2, time step 6912.

(b) Last 24-hour prediction and actual values for bus 2.

Figure 6.3: First and last time step simulation.

The picture shows a fairly accurate prediction and following of the general pattern. This prediction has a mean absolute error of 8.1726 MW, which as expected is lower than the average since it is at the beginning of the test data and the model has a better understanding due to the proximity of the training data. For contrast, the fifteenth-to-last prediction is displayed in Figure 6.3b. This prediction can directly be seen to not be as accurate as the previous one, which is logical, and with an MAE of 26.3083 MW. To depict the improvement in accuracy that is achieved when new data is obtained, the course of a full day (24/12) is shown, and every 4 hours a new prediction is made based on the known data. See figure 6.4.

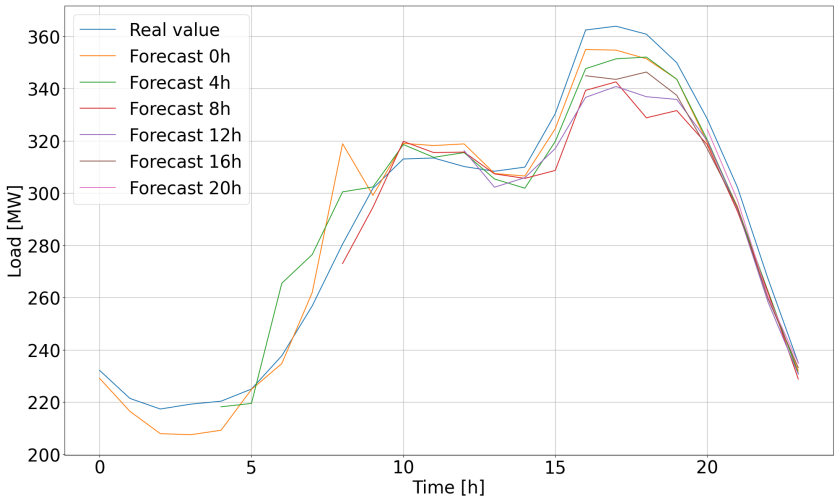


Figure 6.4: A new prediction every 4 hours.

As can be observed in the picture, the first prediction (0h) starts being less and less accurate getting replaced by predictions that have access to new data.

A final observation is that many of the predictions seem to be very accurate, almost too good to be true. This may be due to the way that the load data is compiled in the original network, as it is not actual real data but obtained based on assumptions.

6.3. Future Developments

The TPE algorithm was successfully implemented using Optuna[7] with the test loss as objective function, significantly increasing the model accuracy and generalization, but was not completed for the report.

[7] https://github.com/optuna/optuna

7

Conclusion & Discussion

The requirements that were set for the Forecasting & Scenario module have been accomplished, both those regarding the scenario and the forecasting. The mandatory requirements have been fulfilled, and even if there is room for improvement, the trade-off requirements have also all been fulfilled. The module includes a functional power grid configuration with accurate load and generation data, and provides it correctly to the Visualization & Simulation module.

The model has been made open source, encouraging advancements in the field of power engineering. Some trade-off requirements, such as simulating the Dutch power grid, have been implemented to enhance user satisfaction. Additionally, the model uses AC representation, to attempt at making the power grid experience realistic.

Overall, the module can be said to have been a success, even if time constraints have limited the amount of progress that has been made, and that can still be made in the coming time. It is worth noting that improvements to the project can still be made before the defense of the thesis.

7.1. Future work

Both the model for the power flow calculations and the forecasting section have room for improvement. However, these have not been implemented due to time constraints.

One of the main issues that could be improved is the size of the dataset. It only includes one year of limited hourly load data, which is relatively short compared to other studies of similar nature that have a training set of multiple years.

Even if the choice for the machine learning model was fully justified, and the main focus of the project was put on the XGBoost algorithm, other models could have been tried to ensure the XGB was the best model in practice too.

Moreover, additional features could be used for the model, such as weather conditions (temperature, wind speed, cloud cover, humidity, precipitation), economy, electricity prices, etc. This would however be greatly time consuming for what would likely not be a big improvement in accuracy.

More importantly, the hyper-parameters can be fine-tuned. The method of Tree-structured Parzen Estimator has been explored, and some work has been done with it already. This Bayesian optimization framework is designed to efficiently search the hyper-parameter space for the optimal set of parameters that improve the performance. This could significantly increase the accuracy of the model.

The power flow model selected in this project was aggregated, leaving room to extend this in a less aggregated model. This would result in focusing not only on high-voltage but could extend to mid-range or perhaps distribution, which would result in an even more complex model.

Additionally, another objective function could be used in the optimal power flow, as this project used the economic dispatch as the function to minimize. Other objective functions such as minimization of active and reactive power losses, maintaining a constant voltage profile or transmission investments could also be interesting to investigate.

7.2. Tips

Important notes and tips for possible students to continue the project are: Finding the right power flow model including the data was quite time-consuming and hard. When a new power flow model has been found, it needs to be imported which requires a lot of libraries one might not have already installed. Additionally, one has to get familiar with and understand the model which takes a lot of effort and time. Moreover, the model could be written in a programming language which one might not have a lot of experience with. As these models tend to be quite complex, it will take a lot of time to investigate all the parameters of all the components in the model and their relationships.

Next to importing a power flow model, one should preferably also have an intricate understanding of the power grid, there is a full world of knowledge and a lot of formulas to study. This is also true for the machine learning part, especially because this is not a mandatory part of the Electrical Engineering curriculum.

In a lot of papers, the full description of algorithms or methods, especially in the case of machine learning techniques, is not clearly described. This might make it complicated to fully understand such algorithms.

All in all, it was a very interesting project and a great amount was learned about subjects that had never been touched before.

References

[1] Miklós Gyalai-Korpos et al. “The role of electricity balancing and storage: Developing input parameters for the European Calculator for Concept Modeling”. In: *Sustainability* 12.3 (Jan. 2020), p. 811. doi: 10.3390/su12030811. url: https://www.mdpi.com/2071-1050/12/3/811.

[2] Manuel Reta-Hernández. “Transmission Line Parameters”. In: *Transmission Line Parameters*. Taylor & Francis Group, LLC, 2006, pp. 13-1–13-28.

[3] Alliander N.V. and Co. *OpenSTEF Github*. 2024. url: https://github.com/OpenSTEF/openstef.

[4] J Carpentier. “Optimal power flows”. In: *International journal of electrical power energy systems* 1.1 (Apr. 1979), pp. 3–15. doi: 10.1016/0142-0615(79)90026-7. url: https://www.sciencedirect.com/science/article/pii/0142061579900267.

[5] Imran Rahman and Junita Mohamad-Saleh. “Hybrid bio-Inspired computational intelligence techniques for solving power system optimization problems: A comprehensive survey”. In: *Applied soft computing* 69 (Aug. 2018), pp. 72–130. doi: 10.1016/j.asoc.2018.04.051. url: https://www.sciencedirect.com/science/article/pii/S1568494618302424.

[6] Arun Sukumaran Nair et al. “Computational and numerical analysis of AC optimal power flow formulations on large-scale power grids”. In: *Electric power systems research* 202 (Jan. 2022), p. 107594. doi: 10.1016/j.epsr.2021.107594. url: https://www.osti.gov/servlets/purl/1846582#:~:text=Compared%20to%20a%20DC%20optimal,power%20load%20loss)%5B6%5D..

[7] Wouter Zomerdijk et al. “Open Data Based Model of the Dutch High-Voltage Power System”. In: *Proceedings of the IEEE Innovative Smart Grid Technologies Conference Europe*. Accessed: June 10, 2024. IEEE, 2022, pp. 1–5. doi: 10.1109/ISGT-Europe46778.2022.9960703.

[8] Wouter Zomerdijk. *Dutch-HV-Power-System Github*. 2024. url: https://github.com/WZomerdijk/Dutch-HV-Power-System.

[9] Wouter Zomerdijk. “Effects of Power and Heating Sector Integration for the Future Dutch High-Voltage Electricity Grid: Development of a Representative Model of the Dutch High-Voltage Electricity Grid”. MA thesis. Delft University of Technology, 2021.

[10] Quintel Intelligence. *Energy transition model*. 2024. url: https://energytransitionmodel.com/.

[11] Commissie verbruiksprofielen. *MFFBAS Verbruiksprofielen*. 2024. url: https://www.mffbas.nl/documenten/.

[12] Rijkswaterstaat. *De Klimaatmonitor*. 2024. url: https://klimaatmonitor.databank.nl/jive.

[13] CBS. *Windenergie op land; productie en capaciteit per provincie*. 2024. url: https://opendata.cbs.nl/statline/#/CBS/nl/dataset/70960ned/table?ts=1636628111301.

[14] CBS. *Hernieuwbare energie; zonnestroom, windenergie, RES-regio*. 2024. url: https://opendata.cbs.nl/statline/#/CBS/nl/dataset/85004NED/table?ts=1630339841444.

[15] Stefan Pfenninger and Iain Staffell. “Long-term patterns of European PV output using 30 years of validated hourly reanalysis and satellite data”. In: *Energy* 114 (Nov. 2016), pp. 1251–1265. doi: 10.1016/j.energy.2016.08.060. url: https://doi.org/10.1016/j.energy.2016.08.060.

[16] Iain Staffell and Stefan Pfenninger. “Using bias-corrected reanalysis to simulate current and future wind power output”. In: *Energy* 114 (Nov. 2016), pp. 1224–1239. doi: 10.1016/j.energy.2016.08.068. url: https://doi.org/10.1016/j.energy.2016.08.068.

[17] ENTSO-E. *Installed capacity per production unit*. 2018. url: https://transparency.entsoe.eu/generation/r2/installedCapacityPerProductionUnit/show.

[18] TenneT TSO B.V. *System and transmission data: Installed capacity*. 2018. url: https://www.tennet.org/bedrijfsvoering/Systeemgegevens_voorbereiding/Aangemeld_productievermogen/Opgesteld_vermogen.aspx.

[19] TenneT TSO B.V. *Static grid model NL. Overview of 380kV and 220kV grid components inclusive of technical data*. 2020. url: https://www.tennet.org/controls/DownloadDocument.ashx?documentID=41.

[20] Leon Thurner et al. “Pandapower—An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems”. In: *IEEE Transactions on Power Systems* 33.6 (2018), pp. 6510–6521. doi: 10.1109/TPWRS.2018.2829021.

[21] Grzegorz Dudek. “A Comprehensive Study of Random Forest for Short-Term Load Forecasting”. In: *Energies* 15.20 (2022). issn: 1996-1073. doi: 10.3390/en15207547. url: https://www.mdpi.com/1996-1073/15/20/7547.

[22] Grzegorz Dudek. “Neural networks for pattern-based short-term load forecasting: A comparative study”. In: *Neurocomputing* 205 (Sept. 2016), pp. 64–74. doi: 10.1016/j.neucom.2016.04.021. url: https://www.sciencedirect.com/science/article/pii/S0925231216302764?via%3Dihub.

[23] Shereen Elsayed et al. “Do We Really Need Deep Learning Models for Time Series Forecasting?” In: *CoRR* abs/2101.02118 (2021). arXiv: 2101.02118. url: https://arxiv.org/abs/2101.02118.

[24] Grzegorz Dudek. “Short-Term Load Forecasting Using Random Forests”. In: *Intelligent Systems’2014*. Ed. by D. Filev et al. Cham: Springer International Publishing, 2015, pp. 821–828. isbn: 978-3-319-11310-4.

[25] Tim Januschowski et al. “Forecasting with trees”. In: *International journal of forecasting* 38.4 (Oct. 2022), pp. 1473–1481. doi: 10.1016/j.ijforecast.2021.10.004. url: https://www.sciencedirect.com/science/article/pii/S0169207021001679?via%3Dihub.

[26] Yuanyuan Wang et al. “Short-term load forecasting of industrial customers based on SVMD and XGBoost”. In: *International journal of electrical power energy systems* 129 (July 2021), p. 106830. doi: 10.1016/j.ijepes.2021.106830. url: https://www.sciencedirect.com/science/article/pii/S0142061521000703?via=ihub.

[27] Radu C. Lucaciu. “Time Series Forecasting And Big Data: Can An Ensemble Of Weak Learners Decrease Costs While Maintaining Accuracy?” In: *2023 17th International Conference on Engineering of Modern Electric Systems (EMES)*. 2023, pp. 1–4. doi: 10.1109/EMES58375.2023.10171639.

[28] Syama Sundar Rangapuram et al. *Deep state space models for time series Forecasting*. Tech. rep. 2018. url: https://proceedings.neurips.cc/paper/2018/file/5cf68969fb67aa6082363a6d4e6468e2-Paper.pdf.

[29] Konstantinos Benidis et al. “Deep Learning for Time Series Forecasting: Tutorial and Literature survey”. In: *ACM computing surveys* 55.6 (Dec. 2022), pp. 1–36. doi: 10.1145/3533382. url: https://doi.org/10.1145/3533382.

[30] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 785–794. isbn: 9781450342322. doi: 10.1145/2939672.2939785. url: https://doi.org/10.1145/2939672.2939785.

[31] Ayush Gupta. *Using The Right File Format For Storing Data*. Data Science Blogathon - 11. 2022. url: https://www.analyticsvidhya.com/blog/2021/09/using-the-right-file-format-for-storing-data/.

[32] Leon Thurner, Alexander Scheidler, and Co. *Pandapower documentation*. 2024. url: https://pandapower.readthedocs.io/en/latest/.

[33] Spyros Chatzivasileiadis. *Optimization in Modern Power Systems*. Sept. 2018. url: https://arxiv.org/pdf/1811.00943.

[34] Hermann W. Dommel and William F. Tinney. “Optimal Power Flow Solutions”. In: *IEEE Transactions on Power Apparatus and Systems* PAS-87.10 (1968), pp. 1866–1876. doi: 10.1109/TPAS.1968.292150.

[35] William. F. Tinney and Clifford E. Hart. “Power Flow Solution by Newton’s Method”. In: *IEEE Transactions on Power Apparatus and Systems* PAS-86.11 (1967), pp. 1449–1460. doi: 10.1109/TPAS.1967.291823.

[36] Ulf Philipp Müller et al. “Integrated Techno-Economic Power System Planning of transmission and distribution grids”. In: *Energies* 12.11 (May 2019), p. 2091. doi: 10.3390/en12112091. url: https://www.mdpi.com/1996-1073/12/11/2091.

[37] S. Bhattacharyya et al. “Analysis of power quality performance of the dutch medium and low voltage grids”. In: *2008 13th International Conference on Harmonics and Quality of Power*. 2008, pp. 1–6. doi: 10.1109/ICHQP.2008.4668800.

[38] Baoji Wang et al. “Research on the Filters for Dual-Inverter Fed Open-End Winding Transformer Topology in Photovoltaic Grid-Tied Applications”. In: *Energies* 12 (June 2019), p. 2338. doi: 10.3390/en12122338.

[39] Wei□Yin Loh. “Classification and regression trees”. In: *Wiley interdisciplinary reviews. Data mining and knowledge discovery/Wiley interdisciplinary reviews. Data mining and knowledge discovery* 1.1 (Jan. 2011), pp. 14–23. doi: 10.1002/widm.8. url: https://doi.org/10.1002/widm.8.

[40] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 785–794. isbn: 9781450342322. doi: 10.1145/2939672.2939785. url: https://doi.org/10.1145/2939672.2939785.

[41] Vitor Cerqueira, Nuno Moniz, and Carlos Soares. “VEST: automatic feature engineering for forecasting”. In: *Machine learning* (Apr. 2021). doi: 10.1007/s10994-021-05959-y. url: https://doi.org/10.1007/s10994-021-05959-y.

[42] Candice Bentéjac, Anna Csörgő, and Gonzalo Martínez-Muñoz. “A Comparative analysis of XGBoOSt”. In: *ResearchGate* (Nov. 2019). url: https://www.researchgate.net/publication/337048557_A_Comparative_Analysis_of_XGBoost.

A

XGBoost Main Functions

Some of the main functions especially made and used for the machine learning model are provided here. To see the full code, both for the machine learning model and the simulation, refer to the github link: https://github.com/bbucksch/BacelorEndProject.git

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt
import pickle
import seaborn as sns

import holidays




# GET INDEXES OF DATA SUBSEQUENCES (ONE SUBSEQUENCE INCLUDES LAGGED (X) AND HORIZON (Y) LOADS)

def get_subsequence_indexes(in_data_df, window_size, step_size):
    # in_data_df = full (or train/test) dataset, window_size = number of lagged values + predicted (horizon) values, step_size = spacing between datapoints

    last_index = len(in_data_df)

    subseq_first_idx = 0 # Subsequence start and end index
    subseq_last_idx = window_size

    subseq_indexes = []

    while subseq_last_idx <= last_index: # Divide all data into subsequences (and get their indexes)

        subseq_indexes.append((subseq_first_idx, subseq_last_idx))

        subseq_first_idx += step_size
        subseq_last_idx += step_size

    return subseq_indexes




# GET X,Y DATA SPLIT (EVERY DATAPOINT IS MADE UP OF SUB-SEQUENCE)

def get_xy_lagged(subseq_indexes, load_data, horizon_size, lag_size):
```

```
43
44      for i, idx in enumerate(subseq_indexes):
45
46          # Create subsequences
47          subsequence = load_data[idx[0]:idx[1]] # Flat np array
48
49          xi = subsequence[0: lag_size]
50          yi = subsequence[lag_size: lag_size + horizon_size]
51
52          if i == 0: # No existing array to append to
53              y = np.array([yi]) # Turn y and x into rows, to make an array of arrays
54              x = np.array([xi])
55
56          else:
57              y = np.concatenate((y, np.array([yi])), axis=0) # shape (datapoints, horizon)
58              x = np.concatenate((x, np.array([xi])), axis=0) # shape (datapoints, input
                    features)
59
60      return x, y
61
62
63
64
65  # GET DATETIME FEATURES
66
67  def create_dt_features(df):
68      df_c = df.copy()
69      df_c['Hour'] = df_c.index.hour
70      df_c['Workday'] = df_c.index.map(lambda x: 0 if (x in holidays.Netherlands() or x.
            dayofweek in (5,6)) else 1) # 1 if workday, 0 if holiday or weekend
71      df_c['Dayofweek'] = df_c.index.dayofweek
72      df_c['Quarter'] = df_c.index.quarter
73      df_c['Month'] = df_c.index.month
74      df_c['Dayofyear'] = df_c.index.dayofyear
75      df_c['Dayofmonth'] = df_c.index.day
76      return df_c
77
78
79
80
81  # GET CYCLICAL ENCODING
82
83  def cyclical_encoding(df, features):
84      df_c = df.copy()
85
86      for f in features:
87          total_values = df_c[f].max() # E.g. total months = 12, starting at 1
88
89          if df_c[f].min() == 0: # If first value is 0, total values is 1 more e.g. 24 hours
90              total_values += 1
91
92          df_c[f + '_cos'] = np.cos(2*np.pi* df_c[f]/ total_values) # Encode into cos and sin
                values, that way end value is close to start value
93          df_c[f + '_sin'] = np.sin(2*np.pi* df_c[f]/ total_values)
94
95      return df_c
96
97
98
99
100 # GET FORMATTED DATA (IN DATE-TIME) AND TRAIN-TEST SPLIT
101
102 def date_formatting(bus):
103
104     df = pd.read_csv(f"./data/bus_{bus}_load.csv")
105     df.index = pd.to_datetime(df.index, unit='h', origin=pd.Timestamp("2018-01-01"))
106     df.index.name = "Time"
107     df = create_dt_features(df)
108     df = cyclical_encoding(df, features=["Hour", "Dayofweek", "Quarter", "Month", "Dayofyear"
            , "Dayofmonth"])
109
```

```
110     # training_data = df[df.index < split_date] # Splitting data now will cause it to lose 24
            training datapoints and 24*7 test datapoints
111     # test_data = df[df.index >= split_date]
112 
113     return df
114 
115 def allbus_date_formatting(buses=list(range(1, 29))):
116 
117     allbus_df = {}
118 
119     for b in buses:
120         allbus_df[b] = date_formatting(b)
121 
122     return allbus_df
123 
124 
125 
126 
127 # GET X,Y DATA
128 
129 def get_xy(in_data_df, step_size, horizon_size, hyperparameters):
130 
131     subseq_indexes = get_subsequence_indexes(in_data_df = in_data_df, window_size =
            hyperparameters["lag_size"] + horizon_size, step_size = step_size)
132 
133     lagged_x, y = get_xy_lagged(subseq_indexes=subseq_indexes, load_data=in_data_df[
            hyperparameters["selected_features"][0]].to_numpy(),
134                                 horizon_size=horizon_size, lag_size=hyperparameters["lag_size
                                    "])
135 
136     no_lag_features = in_data_df[hyperparameters["selected_features"][1:]].to_numpy() # Array
            of features that are not lagged, by rows (each row a timestep)
137 
138     first_timestep = hyperparameters["lag_size"] # Datapoints start after all lagged values
            can be obtained
139     last_timestep = len(in_data_df) - (horizon_size - 1) # Last datapoint until it is
            possible to obtain all horizon values (horizon_size - 1)
140 
141     x = np.append(lagged_x, no_lag_features[first_timestep: last_timestep], axis=1) # Append
            no-lag features to lagged load features
142 
143     return x, y
144 
145 
146 def get_allbus_xy(allbus_in_data, hyperparameters, buses=list(range(1, 29)), step_size=1,
        horizon_size=24):
147 
148     allbus_x, allbus_y = {}, {}
149 
150     for b in buses:
151         allbus_x[b], allbus_y[b] = get_xy(allbus_in_data[b], step_size, horizon_size,
                hyperparameters)
152 
153     return allbus_x, allbus_y
154 
155 
156 
157 
158 # GET X,Y SPLIT IN TRAIN, TEST
159 
160 def split_train_test(x, y, lag_size, split_date="2018-10-16", train_val_split=0.8): #
        Timestep 6912
161 
162     original_timestep = (pd.Timestamp(split_date) - pd.Timestamp("2018-01-01␣00:00:00")).
            total_seconds()/3600 # Get original timestep (hour) from split date
163     split_timestep = int(original_timestep - lag_size) # Split timestep in the new dataframe
            (starts at timestep lag_size)
164 
165     x_tr = x[:split_timestep]
166     y_tr = y[:split_timestep]
167 
```

```
    x_train, x_val = x_tr[:int(train_val_split * len(x_tr))], x_tr[int(train_val_split * len(x_tr)):] # Split in train and validation
    y_train, y_val = y_tr[:int(train_val_split * len(y_tr))], y_tr[int(train_val_split * len(y_tr)):]

    x_test = x[split_timestep:]
    y_test = y[split_timestep:]

    return x_train, y_train, x_val, y_val, x_test, y_test


def allbus_split_train_test(allbus_x, allbus_y, lag_size, buses=list(range(1, 29)), split_date="2018-10-16", train_val_split=0.8):

    allbus_x_train, allbus_y_train, allbus_x_val, allbus_y_val, allbus_x_test, allbus_y_test = {}, {}, {}, {}, {}, {}

    for b in buses:
        allbus_x_train[b], allbus_y_train[b], allbus_x_val[b], allbus_y_val[b], allbus_x_test[b], allbus_y_test[b] = \
            split_train_test(allbus_x[b], allbus_y[b], lag_size, split_date, train_val_split)

    return allbus_x_train, allbus_y_train, allbus_x_val, allbus_y_val, allbus_x_test, allbus_y_test




# GET AND STORE ALL MODELS FOR ALL BUSSES AND THEIR TRAIN/TEST SCORES

def get_model(x_train, y_train, x_val, y_val, x_test, y_test, hyperparameters, score_function):

    if x_val == "None" or y_val == "None":
        model = XGBRegressor(n_estimators=hyperparameters["n_estimators"], max_depth=hyperparameters["max_depth"], subsample=hyperparameters["subsample"],
                             gamma=hyperparameters["gamma"], reg_lambda=hyperparameters["lambda"], objective="reg:squarederror", tree_method="hist",
                             verbosity=3, learning_rate=hyperparameters["learning_rate"])

        trained_model = MultiOutputRegressor(model).fit(x_train, y_train, verbose=True) # Evaluate on validation data

        valid_score = None



    else:
        model = XGBRegressor(n_estimators=hyperparameters["n_estimators"], max_depth=hyperparameters["max_depth"], subsample=hyperparameters["subsample"],
                             gamma=hyperparameters["gamma"], reg_lambda=hyperparameters["lambda"], objective="reg:squarederror", tree_method="hist",
                             verbosity=3, learning_rate=hyperparameters["learning_rate"], early_stopping_rounds=hyperparameters["early_stopping_rounds"])

        trained_model = MultiOutputRegressor(model).fit(x_train, y_train, fit_params={"eval_set": [(x_val, y_val)]}, verbose=True)
        # Evaluate on validation data

        valid_forecasts = trained_model.predict(x_val)
        valid_score = score_function(y_val, valid_forecasts)

    train_forecasts = trained_model.predict(x_train)
    train_score = score_function(y_train, train_forecasts)

    test_forecasts = trained_model.predict(x_test)
    test_score = score_function(y_test, test_forecasts)

    model_score = {"Train score": train_score, "Validation score": valid_score, "Test score": test_score}
```

```
223
224     return trained_model, model_score
225
226
227 def get_models(models_datapath, hyperparameters, score_function, buses=list(range(1, 29)),
        horizon_size=24, step_size=1,
228                allbus_x_train="None", allbus_y_train="None", allbus_x_val="None",
                       allbus_y_val="None", allbus_x_test="None", allbus_y_test="None"):
229
230     trained_models = {}
231     model_scores = {}
232
233     for bus in buses:
234
235         if (allbus_x_train != "None") and (allbus_y_train != "None") and (allbus_x_test != "
                None") and (allbus_y_test != "None"): # If they're all defined
236             x_train, y_train, x_val, y_val, x_test, y_test = allbus_x_train[bus],
                    allbus_y_train[bus], allbus_x_val[bus] if allbus_x_val != "None" else "None",
                     \
237             allbus_y_val[bus] if allbus_y_val != "None" else "None", allbus_x_test[bus],
                    allbus_y_test[bus]
238
239         else:
240             print("Missing␣input␣data␣to␣get_models()")
241
242             df = date_formatting(bus)
243
244             x, y = get_xy(df, step_size, horizon_size, hyperparameters)
245
246             x_train, y_train, x_val, y_val, x_test, y_test = split_train_test(x, y,
                    hyperparameters["lag_size"])
247
248
249         trained_model, model_scores[bus] = get_model(x_train, y_train, x_val, y_val, x_test,
                y_test, hyperparameters, score_function)
250
251         trained_models[bus] = trained_model
252
253         # Store models
254         with open(f"{models_datapath}/MOR_bus{bus}.pkl", "wb") as f1:
255             pickle.dump(trained_model, f1)
256
257     # Store model scores
258     with open(f"{models_datapath}/MOR_scores.pkl", "wb") as f2:
259                 pickle.dump(model_scores, f2)
260
261     return trained_models, model_scores
262
263
264
265
266
267 # ------------------------------------------- FEATURE IMPORTANCES
        -----------------------------------------------------
268
269
270 def get_boxplot(chosen_buses, trained_models, feature_names):
271     allbus_importances = []
272     allbus_importance_names = []
273
274     for bus in chosen_buses:
275         allbus_importances.append(np.concatenate([trained_models[bus].estimators_[t].
                feature_importances_ for t in range(24)]))
276         allbus_importance_names.append(np.concatenate([feature_names for _ in range(24)]))
277
278     importance = np.concatenate(allbus_importances)
279     ft_name = np.concatenate(allbus_importance_names)
280
281     feature_importance_df = pd.DataFrame({"Feature": ft_name, "Importance": importance})
282
```

```
283     sorting = feature_importance_df.groupby("Feature").mean().sort_values(by="Importance",
            ascending=False)
284
285     plot = sns.boxplot(data=feature_importance_df, x="Feature", y="Importance", order=sorting
            .index).set_xticklabels(sorting.index, rotation=80)
286
287     return feature_importance_df, sorting, plot
```