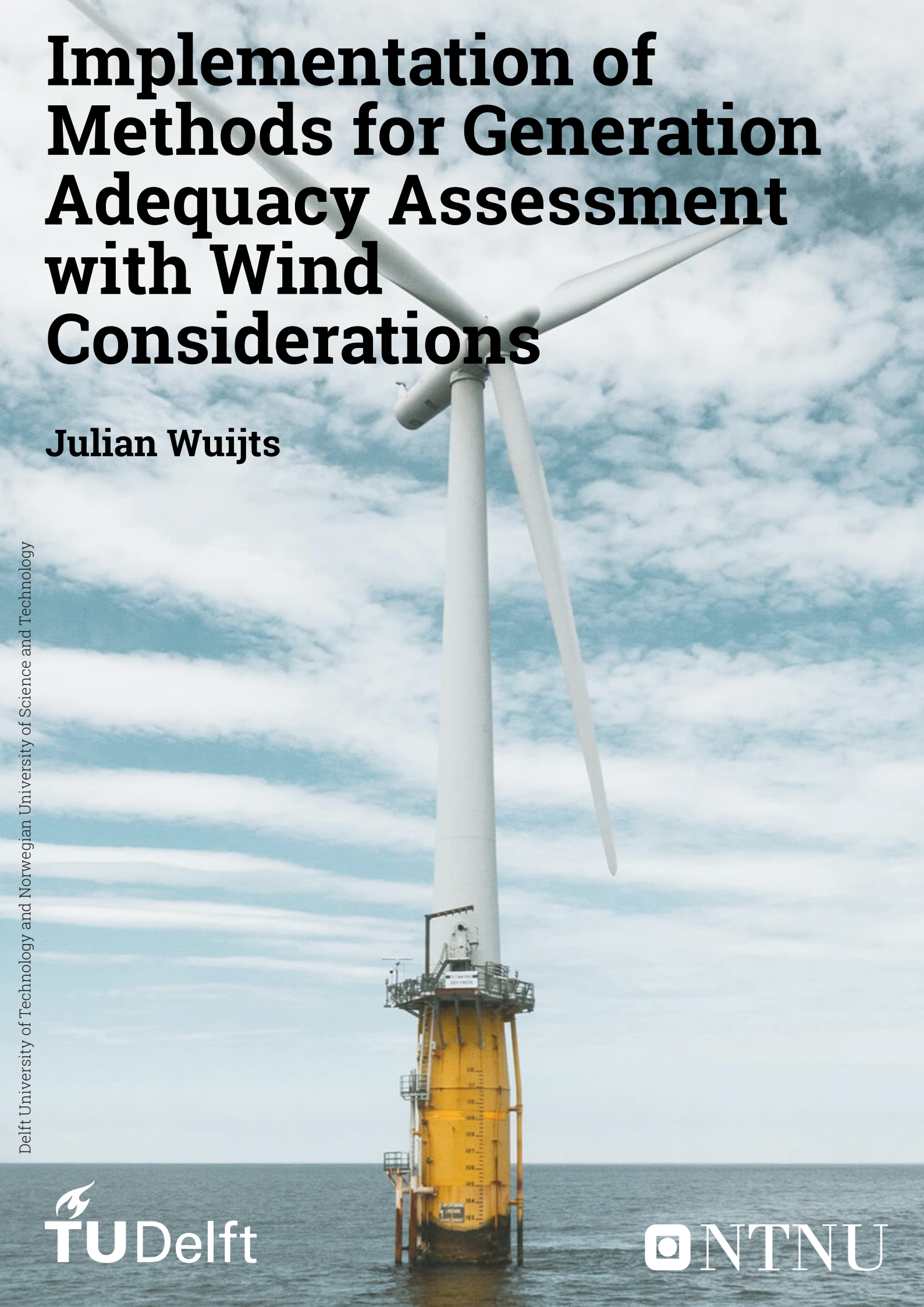
Implementation of Methods for Generation Adequacy Assessment with Wind Considerations

Julian Wuijts



Delft University of Technology and Norwegian University of Science and Technology

TU Delft

NTNU

Implementation of Methods for Generation Adequacy Assessment with Wind Considerations

by

Julian Wuijts

to obtain the degree of MSc. Electrical Engineering
at the Delft University of Technology,
and the degree of MSc. Wind Energy,
at the Norwegian University of Science and Technology,
to be defended publicly on the 5th of July, 2024 at 09:00 AM

Student number:	5168538 (TU Delft) and 103162 (NTNU)	
Project duration:	November 1, 2023 – July 5, 2024	
Supervision Team:	Dr. ir. Vijay Venu Vadlamudi,	NTNU
	Dr. ir. Iver Bakken Sperstad,	SINTEF Energy
	Ir. Ivar Bjerkebæk,	SINTEF Energy
	Dr. ir. José Rueda Torres,	TU Delft
Thesis committee:	Dr. ir. José Rueda Torres,	TU Delft, committee chair & supervisor
	Dr. ir. Vijay Venu Vadlamudi,	NTNU, supervisor
	Dr. ir. Aditya Shekhar,	TU Delft, committee member

Cover:	Unitech Zefyros (Hywind Demo) [1]
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.
The scripts presented in this thesis are available at
`https://github.com/Julian-Wuijts/Generation-Adequacy-Scripts`

TU Delft EWEM NTNU SINTEF

Acknowledgements

With this Master's thesis, my academic journey comes to an end for now. The European Wind Energy Master program has helped me develop both on a personal and academic level. I want to express my gratitude to a few people who helped me during this period.

First, I would like to thank the supervision team at NTNU: Vijay Venu Vadlamudi, Ivar Bjerkebæk and Iver Bakken Sperstad. They have encouraged me throughout the thesis process and helped me deepen my knowledge in the wonderful area of power system reliability studies. Their step-by-step approach to creating this thesis work has made the whole process clear and enjoyable.

I also want to thank my TU Delft supervisor, Jose Rueda Torres. He has provided me with valuable insights to improve the quality of my thesis. His guidance throughout the thesis work has allowed me to finish my thesis within the given timeframe.

Finally, I am eternally thankful to my parents and sister for their support throughout my studies and thesis work.

Julian Wuijts
Trondheim, July 2024

Abstract

This thesis creates open-source Python scripts for conducting generation adequacy analysis studies (available on GitHub). First, scripts without wind considerations are constructed, and then the scripts have been modified to include wind considerations. Generation and load models have been implemented to obtain the reliability (adequacy) indices Loss of Load Expectation (LOLE) and Expected Energy Not Served (EENS). Two probabilistic methods have been implemented: an analytical method and a non-sequential Monte Carlo Simulation (MCS) method. Both methods use the IEEE load curve for their load model, but different generation models are used. The analytical model uses a recursive algorithm, where generation units are added one by one to obtain a Capacity Outage Probability Table (COPT). The non-sequential MCS method uses state sampling, where a uniform random number is generated for each hourly increment and a state is selected based on the state probabilities of a generator.

Wind considerations, which can be any number of turbines located on- or offshore, have been added to the scripts by modelling the power output curve of a wind turbine. The analytical method models a wind turbine as a conventional generation with multiple states, where a state probability table is based on the wind power distribution and the mechanical availability. The power distribution is obtained by converting wind speeds to power outputs using the power output curve. The MCS method, on the other hand, simulates the wind speed with a Weibull distribution and a uniform random number generator. It also uses the power output curve, and the result is combined with the mechanical availability.

The proposed scripts have been tested on two test systems - the Roy Billinton Test System and the IEEE Reliability Test System. The resulting reliability metrics have been compared with benchmark values in the literature and found to be closely matching. Furthermore, methodological clarity on how to obtain the presented scripts is given. Even though all elements of the implemented methods are present in the literature, not all are clearly explained or combined in one place. This work aims to overcome this limitation.

Additionally, the effect of turbine outage rates on the output of an offshore wind farm has been assessed. The cumulative distribution showed that the portion of the graph operating between no output and the rated output is close to linear for turbines with perfect reliability. Introducing non-zero Forced Outage Rates for the turbines results in a stepwise probability behaviour towards the upper end of the wind farm output. These steps occur around increments of the rated capacity of a singular turbine and lower the probability of operating in the intervals at the upper end of the wind farm.

An approximately linear relation between a turbine's Forced Outage Rate (FOR) and the yearly energy production can be observed. An increase in FOR, for FORs above 0.10, seems to have an increasingly bigger impact on the yearly energy production. The same behaviour can be observed in the case study that is carried out. The study looks at how many turbines are required to supply energy to the city of Trondheim, assuming that a storage solution takes care of all the power fluctuations. A constant power demand of 688 MW can be covered by a 1.43 GW wind farm with perfect reliability. The farm uses 10 MW turbines, and 3 additional turbines are needed for a 0.02 increase in FOR. From a FOR of 0.10, this value seems to increase to 4 turbines to satisfy the demand.

The results for the non-sequential MCS method with wind considerations are off by up to 15% due to a poorly fitting Weibull distribution for the wind speeds. Furthermore, a third probabilistic method has also been implemented, a sequential MCS without wind considerations in the form of the state duration method. These results are off by 5 to 10%, but there are two main reasons for this mismatch. One of them is rounding the simulated state duration to an integer number of hours to match the integer hourly load profile. The second limitation is due to the way that the generation profile is initialised. The generator is currently assumed to be operational at the start of each simulation year. However, carrying over the state and state duration from the previous simulation year would yield better results. These two limitations can be addressed in future work.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
ARMA	Autoregressive Moving-Average
CDF	Cumulative Distribution Function
COPT	Capacity Outage Probability Table
CoV	Coefficient of Variation
CYPL	Constant Yearly Peak Load
DPL	Daily Peak Load
EENS	Expected Energy Not Served
EDNS	Expected Demand Not Served (derived from EENS)
FOR	Forced Outage Rate
HL	Hierarchical Level
HPL	Hourly Peak Load
IEEE	Institute of Electrical and Electronics Engineers
IEEE RTS	IEEE Reliability Test System
LDC	Load Duration Curve
LOEE	Loss of Energy Expectation
LOLE	Loss of Load Expectation
LOLP	Loss of Load Probability
LOLP′	Loss of Load Probability (derived from LOLE)
MCS	Monte Carlo Simulation
MTTF	Mean Time to Failure
MTTR	Mean Time to Repair
PDF	Probability Distribution Function
PRNG	Pseudo-Random Number Generator
pu	Per-Unit
RBTS	Roy Billinton Test System
RES	Renewable Energy Sources
SCOPT	State Capacity Outage Probability Table
TTF	Time to Failure
TTR	Time to Repair
WECS	Wind Energy Conversion System Model
WPL	Weekly Peak Load

Symbols

Symbol	Definition	Unit
A_i	Availability of state i	[-]
A_{mi}	Availability of the upper apportioned level	[-]
A_{ni}	Availability of the lower apportioned level	[-]
C	Capacity of the added unit	[MW]
C_i	Outage capacity of state i	[MW]
C_m	Outage capacity of the upper apportioned level	[MW]
C_n	Outage capacity of the lower apportioned level	[MW]
c_j	Turbine output in state j	[-]
E(x)	Expected value	[-]
f(**x**)	Probability distribution function	[-]
F(**x**)	Cumulative distribution function	[-]
H(x)	Heavyside step function	[-]
L_t	Load at time instance t	[MW]
N	Number of iterations	[-]
N/A	Not Applicable	[-]
P[X]	Cumulative probability of the capacity outage state X after adding a unit	[-]
P'[X]	Cumulative probability of the capacity outage state X before adding a unit	[-]
p_i	Probability of i turbines being available	[-]
P_{output}	Output power	[MW]
P_r	Rated power	[MW]
q_j	Probability of operating in output state c_j	[-]
u	Unavailability of a generator	[-]
U	Uniform distribution	[-]
V_{ci}	Cut-in wind speed	[m/s]
V_{co}	Cut-out wind speed	[m/s]
V_r	Rated wind speed	[m/s]
X	Capacity outage state	[MW]
X_{output}	Wind farm output	[MW]
x	Random variate	[-]
x_j	Outage state	[MW]
α	Weibull shape parameter	[-]
β	Weibull scale parameter	[-]
λ	Failure rate parameter	[-]
μ	Repair rate parameter	[-]
σ	Standard deviation	[-]
Δ	Procentual difference	[%]
ΔT	duration of the time increment	[1 day/ 1 hour/ 1 week/ 1 year]

List of Figures

List of Tables

1 Introduction

The world is moving away from fossil fuels as an energy source in order to limit the temperature increase in compliance with the Paris Agreement [2]. Electricity accounts for a large share of the total energy mix, which will only increase due to the electrification of the industry and mobility. A way to reduce the carbon emissions of electricity production is to phase out fossil fuels and thus increase the share of renewable energy sources (RES), such as wind, hydro and solar. One major difference between RES and the fossil-fuel-based generation they are replacing is the variable, weather-dependent power output over time. This creates challenges regarding the adequacy of the grid, which needs to be considered carefully to retain a reliable system.

1.1. Motivation

In Norway, the grid already has one of the lowest CO_2 emissions per kWh worldwide [3], due to its suitable geography for hydro power [4]. This is a controllable generation source, but its year-by-year power output varies based on the amount of precipitation. With limited solar resources and great wind resources off the coast, offshore wind farms seem suitable to complement the existing hydro-based power grid and prepare for the increasing electricity demand. One of the projects that looks into this is the OceanGrid project [5], which considers the role that offshore wind farms could play in the future power grid, more specifically from 2030 onwards.

The research work in this thesis is carried out in collaboration with SINTEF Energy as part of work package 1 of Sub Project 5 [6] of the OceanGrid project. The OceanGrid project focuses on building knowledge, conducting open research and publishing the results in four key areas: Grid expansion optimisation, Energy market design, System interoperability, and Wet design cables. This thesis work is part of the grid expansion optimisation package; the package aims to develop optimisation models to identify a step-by-step optimal offshore energy infrastructure buildout, also considering the reliability (adequacy) perspective. The work done in this thesis is a first step towards addressing the reliability perspective of the grid expansion optimisation package.

Even though the algorithms used for reliability assessment of traditional power systems are present in the literature [7], [8], [9], and [10], and the ones dealing with (onshore) wind power considerations are present in the literature [11], [12], and [13], not all are clearly explained or synthesised in one place. For example, the apportioning method is mentioned in [13], and a reference to [7] is provided. Eventually, the method was found in this textbook, but the word 'apportioning' is not mentioned in the explanation. A reference that does clearly mention the apportioning method and provides an explanation is [14].

This work aims to overcome this limitation by aiming for methodological clarity and algorithmic transparency in obtaining the scripts for reliability metrics.

1.2. Scope and Contributions

This thesis builds towards developing a framework to assess the adequacy of a future grid with the addition of an offshore energy infrastructure. However, the scope of this thesis is limited to the development and validation of open-source Python scripts for generation adequacy analysis with the possibility to include wind considerations and examining the impact of outages on the output of a wind farm. With wind considerations, any number of turbines located on- or offshore is meant. Transmission system considerations are not taken into account in this work.

Probabilistic methods for reliability assessment are well-defined within the field of power system reliability, and two main methods can be identified: Analytical methods and Monte Carlo Simulation (MCS) methods. Both methods will be implemented to obtain the reliability metrics Loss-of-Load Expectancy (LOLE) and Expected Energy Not Served (EENS) for systems with and without wind generation. Methodological clarity for creating the scripts will be provided, along with a comparison of the results with benchmark values in the literature and between the two methods to validate the scripts.

Some time will also be spent on understanding the impact that turbine outages have on the output of wind farms. The impact of turbine outage rates on the power output probability distribution will be studied, as well as the yearly energy output of a wind farm for different outage rates. A case study will be carried out to assess the impact of turbine outage rates on the required number of turbines.

The main focus of this thesis is to develop open-source Python scripts for generation adequacy and validate them based on test systems found in the literature. Therefore, this work will not consider the transmission system and actual power system data.

The presented open-source scripts are licensed using the GNU General Public License (GPL) Version 3.0. The scripts can be freely used, modified and distributed, as long as any new distributed versions of the scripts are also published as open-source scripts and licensed under the GPL Version 3.0 license. The complete description of the GNU GPL v3.0 can be found in [15]. The scripts are available on GitHub at https://github.com/Julian-Wuijts/Generation-Adequacy-Scripts.

1.3. Thesis Outline

First, the basics of reliability assessment and the theory for implementing the required scripts are covered in Chapter 2. The test systems that will be used to test the script are introduced in Chapter 3, after which the methodology for implementing the Analytical method and Monte Carlo Simulation method scripts, along with the results, is presented in Chapters 4 and 5. A chapter that focuses on the impact of outages on the output of a wind farm can be found in Chapter 6, with a case study on a wind farm to supply energy to the city of Trondheim. Finally, Chapter 7 provides conclusions and presents recommendations for future work. A flowchart of the thesis outline is presented in Figure 1.1.

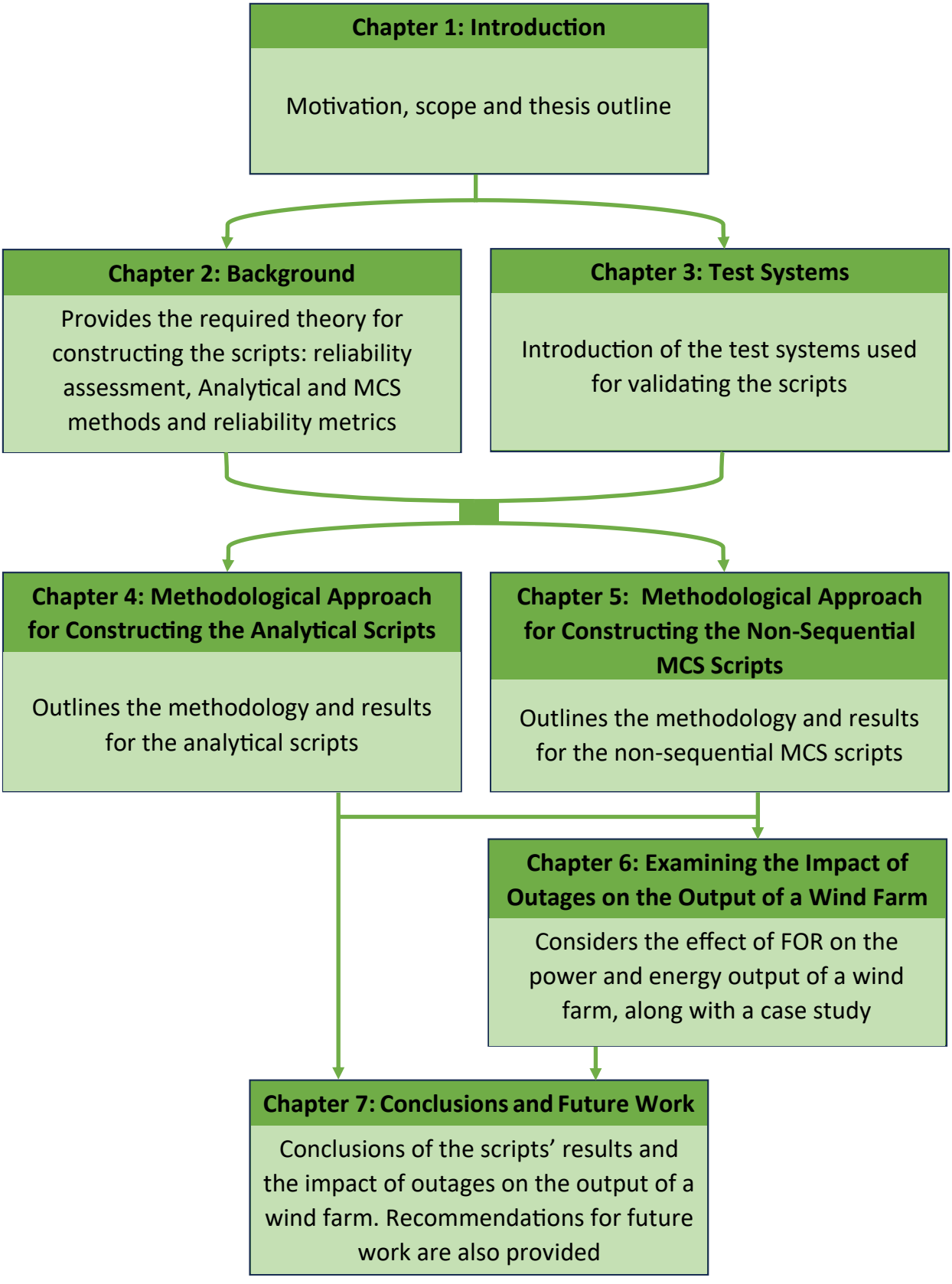


Figure 1.1: Schematic outline of the thesis structure

2
Background

This Chapter will give an overview of the terminology used in power system reliability assessment. Furthermore, the metrics used to quantify reliability and the methods for obtaining these metrics will be presented.

2.1. Reliability Assessment of Power Systems

Power system reliability studies cover many aspects, and various definitions are available online. Reliability in itself can be defined as:

> "Reliability is the ability of the system to fulfil its intended function" [16]

This general definition can be extended to power systems by including its function: supply power to customers as economically as possible and with acceptable reliability and quality [17].

Within the definition of a reliable power system, there is still room for different concepts. They can be defined as system security and system adequacy. System security considers the system's ability to respond to disturbances [7]. It can be used to ensure that the grid operates within its limits and the system can fulfil its function. Limit violations that are observed during reliability studies can be used to make changes to the system configuration.

System adequacy, on the other hand, doesn't consider transient disturbances, i.e. static conditions are assumed. It checks whether the components in the system are sufficient to supply energy to the customers [7]. It can be used for long-term system planning to identify weaknesses and shortcomings in the current system and/or possible future system expansions. A schematic indicating the relations between the different power system reliability concepts is given in Figure 2.1.

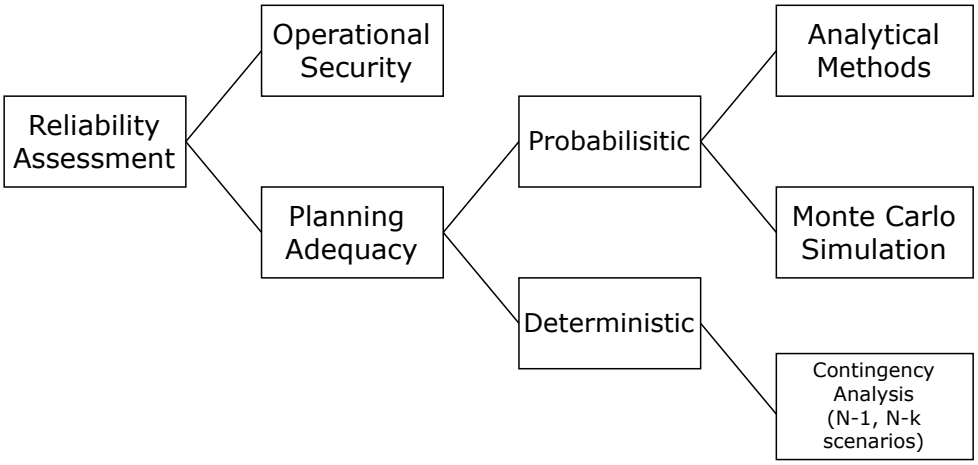


Figure 2.1: An overview of the methods used in power system reliability studies, adapted from [11]

Both concepts should be considered to create a stable system that is resilient to disturbances in the short term and can meet the consumer's demand in the long term. Only system adequacy will be considered for this thesis, as the focus is on long-term offshore wind farm planning.

Power system reliability assessment can also be split into functional zones: generation, transmission and distribution facilities [7]. Doing so allows for considering smaller, more relevant parts of a system to improve the computational time and the interpretability of the results.

Based on these functional zones, hierarchical levels can be defined. They classify the considered functional zones [17]. Hierarchical level 1 (HL1) only considers generation. Level 2 adds the transmission system to the scope. Hierarchical level 3 considers all functional zones and thus looks at the complete system. An overview of the different hierarchical levels is given in Figure 2.2.

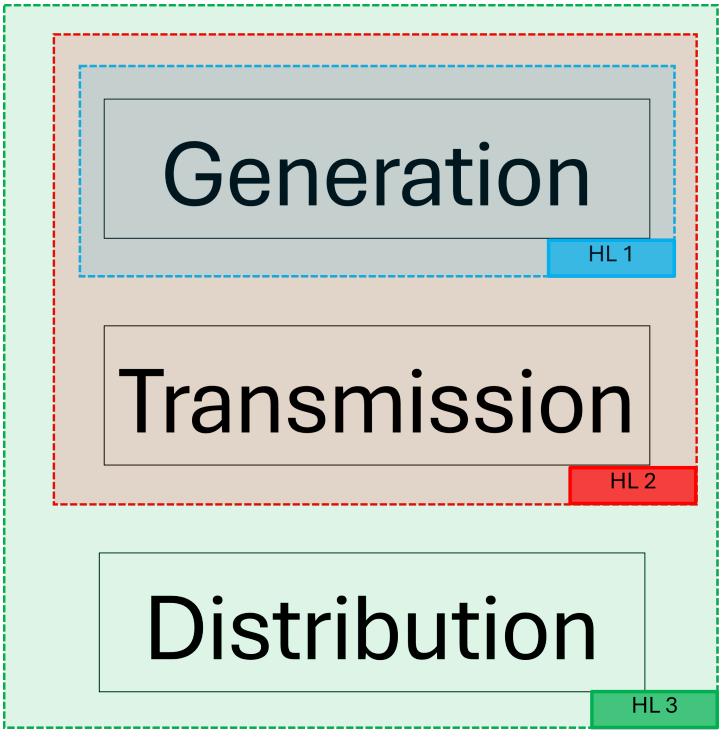


Figure 2.2: Hierarchical levels within power system reliability assessment, based on [7].

2.1.1. Deterministic Methods

With deterministic methods, the effect of an event is studied, and it is checked whether the system is still within an acceptable range. This method, thus, doesn't consider the likelihood of an event and will not be considered in this thesis.

A popular type of deterministic reliability study is the N-1 study, where one checks if the system is still operational if any singular component fails at a particular time.

2.1.2. Probabilistic Methods

Some system states are more likely to occur than others, so it makes sense to consider the probability of being in that state. Probabilistic methods do exactly this, and two commonly used methods, analytical and Monte Carlo Simulation methods, will be elaborated on below.

2.2. Analytical Methods

With analytical reliability studies, a mathematical model of a system is used to calculate the reliability metrics. Generation and load models are required to assess the system's adequacy for a given time instance to obtain these metrics. The chosen models are elaborated on in the forthcoming sections.

2.2.1. Generation Model

A way to model the generation units is by using a Capacity Outage Probability Table (COPT), presented in [7]. It gives an overview of the states the generators can be in and their corresponding output. In its basic form, it takes the capacity and (un)availability as inputs, with which the probability of being in a specific state can be found using a recursive algorithm. The formula for finding the cumulative probability for a given capacity outage state is shown in Equation 2.1.

$$P[X] = (1 - u) \cdot P'[X] + u \cdot P'[X - C] \tag{2.1}$$

where: X = Capacity outage state [MW]
u = Unavailability of the generator
C = Capacity of the generation unit being added
p_i = Probability of being in state i
$P[X]$ = Cumulative probability after the new generation unit is added
$P'[X]$ = Cumulative probability before the new generation unit is added

In Equation 2.1, P'[X] is initialised to 1 for X $\leq$ 0 and the probabilities that are not defined yet are set to zero.

A possible extension to this method is to include additional states for a generation unit. In its simplest form, a generation unit has two states: available at full capacity or unavailable. It could, however, also be possible for the generation unit to unintentionally be forced to operate below the rated capacity in a so-called derated state. Adding a generation unit with derated states can be done by summing over all states, as shown in Equation 2.2. In the case that n=2, i.e. no derated states, Equation 2.2 reduces to Equation 2.1.

$$P[X] = \sum_{i=1}^{n} p_i \cdot P'[X - C_i] \tag{2.2}$$

where: X = Capacity outage state [MW]
n = Number of unit states
C_i = Capacity outage of state i for the generation unit being added
p_i = Probability of being in state i
$P[X]$ = Cumulative probability after the new generation unit is added
$P'[X]$ = Cumulative probability before the new generation unit is added

A short example of how to construct a COPT with a few generation units, where one has a derated state, is given below, with the generator data given in Tables 2.1 and 2.2.

Table 2.1: Generator data for an example of how to construct a COPT

Generator number	Capacity [MW]	Unavailability
1	10	0.02
2	25	0.03
3	50	0.01
4	25	see Table 2.2

Table 2.2: Generator states for **Generator 4** for an example on how to construct a COPT

State number	Capacity outage [MW]	State probability
1	0	0.97
2	15	0.02
3	25	0.01

Adding the first generation unit:

$$P(0) = (1 - 0.02) \cdot 1.0 + 0.02 \cdot 1.0 = 1.0$$
$$P(10) = (1 - 0.02) \cdot 0 + 0.02 \cdot 1.0 = 0.02$$

Adding the second generation unit:

$$P(0) = (1 - 0.03) \cdot 1.0 + 0.03 \cdot 1.0 = 1.0$$
$$P(10) = (1 - 0.03) \cdot 0.02 + 0.03 \cdot 1.0 = 0.0494$$
$$P(25) = (1 - 0.03) \cdot 0 + 0.03 \cdot 1.0 = 0.03$$
$$P(35) = (1 - 0.03) \cdot 0 + 0.03 \cdot 0.02 = 0.0006$$

Adding the third generation unit:

$$P(0) = (1 - 0.01) \cdot 1.0 + 0.01 \cdot 1.0 = 1.0$$
$$P(10) = (1 - 0.01) \cdot 0.0494 + 0.01 \cdot 1.0 = 0.058906$$
$$P(25) = (1 - 0.01) \cdot 0.03 + 0.01 \cdot 1.0 = 0.0397$$
$$P(35) = (1 - 0.01) \cdot 0.0006 + 0.01 \cdot 1.0 = 0.010594$$
$$P(50) = (1 - 0.01) \cdot 0 + 0.01 \cdot 1.0 = 0.01$$
$$P(60) = (1 - 0.01) \cdot 0 + 0.01 \cdot 0.0494 = 0.000494$$
$$P(75) = (1 - 0.01) \cdot 0 + 0.01 \cdot 0.03 = 0.0003$$
$$P(85) = (1 - 0.01) \cdot 0 + 0.01 \cdot 0.0006 = 0.000006$$

Adding the fourth generation unit:

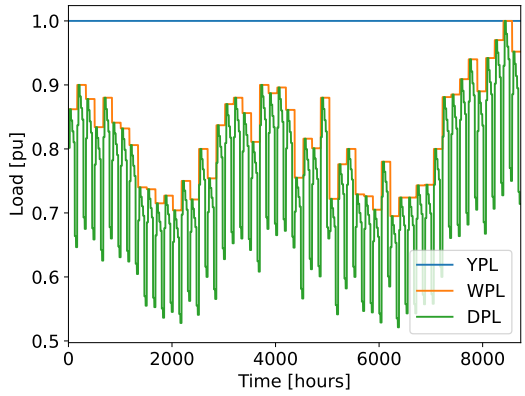
$$P(0) = 0.97 \cdot 1.0 + 0.02 \cdot 1.0 + 0.01 \cdot 1.0 = 1.0$$
$$P(10) = 0.97 \cdot 0.058906 + 0.02 \cdot 1.0 + 0.01 \cdot 1.0 = 0.08713882$$
$$P(15) = 0.97 \cdot 0.0397 + 0.02 \cdot 1.0 + 0.01 \cdot 1.0 = 0.068509$$
$$P(25) = 0.97 \cdot 0.0397 + 0.02 \cdot 0.058906 + 0.01 \cdot 1.0 = 0.04968712$$
$$P(35) = 0.97 \cdot 0.010594 + 0.02 \cdot 0.0397 + 0.01 \cdot 0.058906 = 0.01165924$$
$$P(40) = 0.97 \cdot 0.01 + 0.02 \cdot 0.0397 + 0.01 \cdot 0.0397 = 0.010891$$
$$P(50) = 0.97 \cdot 0.01 + 0.02 \cdot 0.010594 + 0.01 \cdot 0.0397 = 0.01030888$$
$$P(60) = 0.97 \cdot 0.000494 + 0.02 \cdot 0.01 + 0.01 \cdot 0.010594 = 0.00078512$$
$$P(65) = 0.97 \cdot 0.0003 + 0.02 \cdot 0.01 + 0.01 \cdot 0.01 = 0.000591$$
$$P(75) = 0.97 \cdot 0.0003 + 0.02 \cdot 0.000494 + 0.01 \cdot 0.01 = 0.00040088$$
$$P(85) = 0.97 \cdot 0.000006 + 0.02 \cdot 0.0003 + 0.01 \cdot 0.000494 = 0.00001676$$
$$P(90) = 0.97 \cdot 0 + 0.02 \cdot 0.0003 + 0.01 \cdot 0.0003 = 8.99 \cdot 10^{-6}$$
$$P(100) = 0.97 \cdot 0 + 0.02 \cdot 0.000006 + 0.01 \cdot 0.0003 = 3.12 \cdot 10^{-6}$$
$$P(110) = 0.97 \cdot 0 + 0.02 \cdot 0 + 0.01 \cdot 0.000006 = 6 \cdot 10^{-8}$$

2.2.2. Load Model

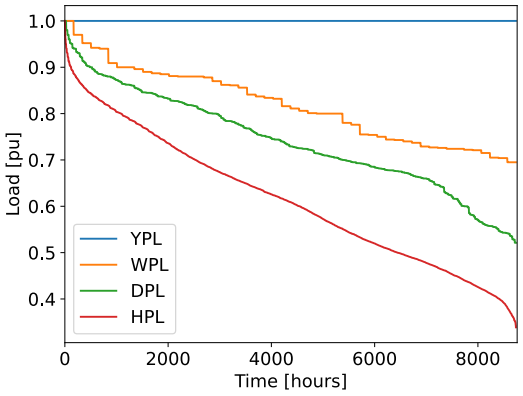
A load model describes the loading in a system over a given period, which is usually chosen to be a year. Within the specified period, there are several time increments of equal size. Commonly used increments are yearly, weekly, daily and hourly loads. The profiles can be created with real-world data or be based on it.

The chosen profile for this application is the IEEE load model due to the availability of reference reliability metrics in literature, which will be elaborated on in 4.1.3. It has multiple timescales, such as a yearly, weekly, daily and hourly peak load. Only the yearly peak load has to be specified to create these different timescales. The other profiles are a percentage of the peak load one timescale layer above it. For example, the Weekly Peak Load (WPL) is a percentage of the Yearly Peak Load (YPL) for 52 weeks. To obtain the Daily Peak Load (DPL) from the WPL, a percentage of the WPL is specified for all seven days of the week. Finally, the Hourly Peak Load (HPL) is calculated as a percentage of the DPL for the 24 hours in a day. The IEEE load profile even distinguishes between a weekday and the weekend, as well as between winter, summer and spring/fall. The percentages to obtain these profiles are given in Tables F.1, F.2 and F.3 in Appendix F.

The load curves can be plotted in chronological or descending order, as shown in Figure 2.3. The latter is called a Load Duration Curve (LDC) and gives an overview of the distribution of the load levels for a year in per-unit (pu). From Figure 2.3b, it can be noted that increasing the resolution of the load profiles results in lower peak load values. As the profiles with fewer time increments consider the peak load value during that period for the entire duration, the actual loading is overestimated.



(a) IEEE load curve for the various peak loads in per-unit



(b) IEEE load duration curve for the various peak loads in per-unit

Figure 2.3: IEEE load curve and load duration curve

2.3. Monte Carlo Simulation Methods

If no mathematical system model is available or when an analytical method's computational time becomes too large, Monte Carlo Simulation (MCS) methods become a viable alternative. Instead of using predefined mathematical models and data, a number is generated from a distribution to indicate the system's state. Uniform and exponential distributions are commonly used to draw random variables. The former has an equal probability for every value between 0 and 1, while the latter has a probability that decreases exponentially the closer it gets to 1. A graph illustrating the probabilities within a uniform distribution is shown in Figure 2.4.

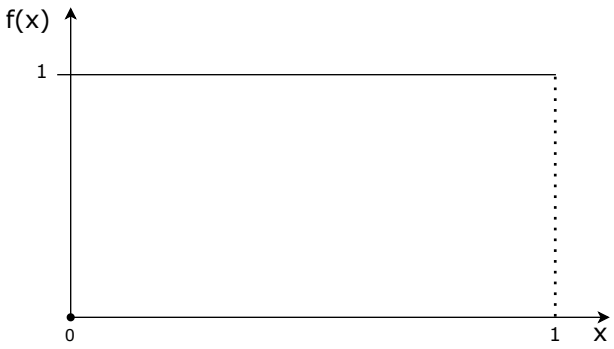


Figure 2.4: Probability density function f(x) for a uniform distribution, adapted from [18]

In Figure 2.4, the entire interval between 0 and 1 has the same probability, and integrating the area underneath the curve results in a cumulative probability of 1.
A random variate from a non-uniform distribution cannot be obtained directly from a random number generator. Therefore, another method is used: the inverse transform method. The exponential distribution will be used as an example here, with a probability distribution function (PDF) as shown in Equation 2.3, where λ represents the rate parameter of the distribution.

$$f(\mathbf{x}) = \lambda \cdot e^{-\lambda \cdot \mathbf{x}} \tag{2.3}$$

The cumulative distribution function (CDF) can be obtained by integrating the PDF, resulting in a distribution between 0 and 1. The formula to obtain the CDF is shown in Equation 2.4.

$$F(\mathbf{x}) = 1 - e^{-\lambda \cdot \mathbf{x}} \tag{2.4}$$

Now that the CDF has been obtained as a distribution between 0 and 1, it can be set equal to a uniform distribution U, resulting in F(**x**)=U. The inverse transform method now takes the inverse of the CDF to obtain Equation 2.5. Figure 2.5 gives a graphical representation of the inverse transform method.

$$X = F^{-1}(\mathrm{U}) = -\frac{1}{\lambda} \cdot \ln(1 - \mathrm{U}) = -\frac{1}{\lambda} \cdot \ln(\mathrm{U}) \tag{2.5}$$

Simplifying 1-U to U in Equation 2.5 is possible due to the uniform probabilities between 0 and 1 for U. 1-U results in a mirrored distribution, but this is the same as the distribution U because of the equal probability for all values.

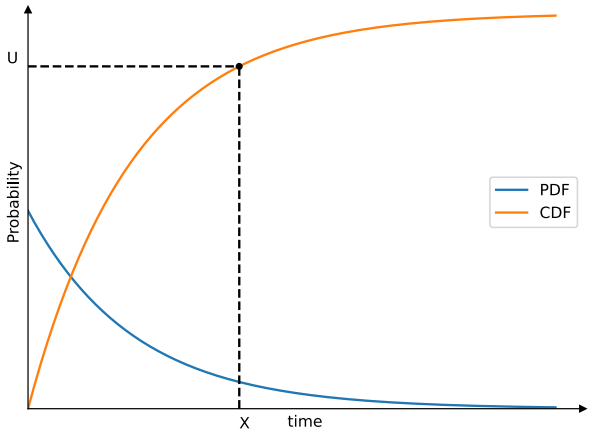


Figure 2.5: Graphical representation of the inverse transform method, adapted from [9]

From Figure 2.5, it can be observed that integrating the PDF results in a CDF. A random number from the uniform distribution U is found, and the intersection with the CDF curve results in the value of **x**, called the random variate. A random number is drawn from a certain distribution with a value between 0 and 1. It can indicate the state of a system or component, which will be elaborated on in the three MCS methods below.

The MCS methods can be introduced now that the different ways to obtain a random variate have been discussed. This thesis will focus on how to apply the MCS methods to obtain a generation model. A more general description of MCS methods can be found in [19].

There are a few differences between the methods, but one significant distinction is whether a sequential or non-sequential sampling method is used. The sequential method considers the previous state in determining the next state, while the non-sequential method assumes that all sampled states are independent. The available MCS methods will be elaborated on in the sections below.

2.3.1. State Sampling Method

State sampling is a non-sequential method. This means that the current state is not dependent on previous or future states. A random number is generated in the interval [0,1] for each component in the system and combined to determine the state of the complete system. Taking a generator with an UP and DOWN state and a specific forced outage rate (FOR) as an example, the generator state can be determined by generating a random number and comparing it with the FOR. A random number smaller than the FOR means that the generator is in the DOWN state, while a random number that is greater than the FOR results in the generator being in the UP state. This modelling approach can also be used for a generator with derated states by adding additional probability ranges for each state. An example of how to transform a generator model to a component state probability table is given in Table 2.3.

Table 2.3: A generator state probability table and its corresponding state sampling table

Table 2.4: Generator state probability table

Generator State		Probability
Up	(100%)	0.85
Derated 1	(50%)	0.07
Derated 2	(25%)	0.05
Down	(0%)	0.03

Table 2.5: Generator state sampling table

Generator State		Probability Range
Up	(100%)	$U \geq 0.15$
Derated 1	(50%)	$0.08 \leq U < 0.15$
Derated 2	(25%)	$0.03 \leq U < 0.08$
Down	(0%)	$U < 0.03$

One of the advantages of state sampling is the simplicity of sampling a random number from a uniform distribution, while a drawback is that the frequency and duration of individual events cannot be represented with this non-sequential method [10].

2.3.2. State Duration Method

A sequential approach to MCS could be the state duration method. The sequential nature means that the next component state depends on the previous one. A chronological state profile is created for each component, where the component alternates between an UP and DOWN state. Furthermore, the assumption is made that the generator is in the UP state at the start of an iteration. The profiles are obtained by considering a component's time to failure (TTF) and time to repair (TTR) distributions. Exponential distributions are commonly chosen to represent the TTF and TTR distributions, but other distributions could also be used. The formulas for obtaining TTF and TTR values can be found in Equation 2.6.

$$\begin{aligned} \text{TTF} &= -\frac{1}{\lambda} \cdot \ln(U) \\ \text{TTR} &= -\frac{1}{\mu} \cdot \ln(U) \end{aligned} \tag{2.6}$$

An example of constructing a state profile for a two-generator system will be given below. The two generators in this example use imaginary and unrealistic data with a high MTTR value. This is done to clearly demonstrate the sequence of states, although a more realistic system would have a much shorter repair time in relation to the MTTF. The generator data that is used in this example is presented in Table 2.6, while the resulting state duration profiles are given in Figure 2.6. The summed generation profile is obtained by summing the two generation profiles together and is shown in Figure 2.7.

Table 2.6: Characteristics of the generators used in the summed generation profile example in Figures 2.6 and 2.7

Generator type	Hydro	Thermal
Capacity [MW]	20	40
MTTF [hours]	3650	1460
MTTR [hours]	825	675

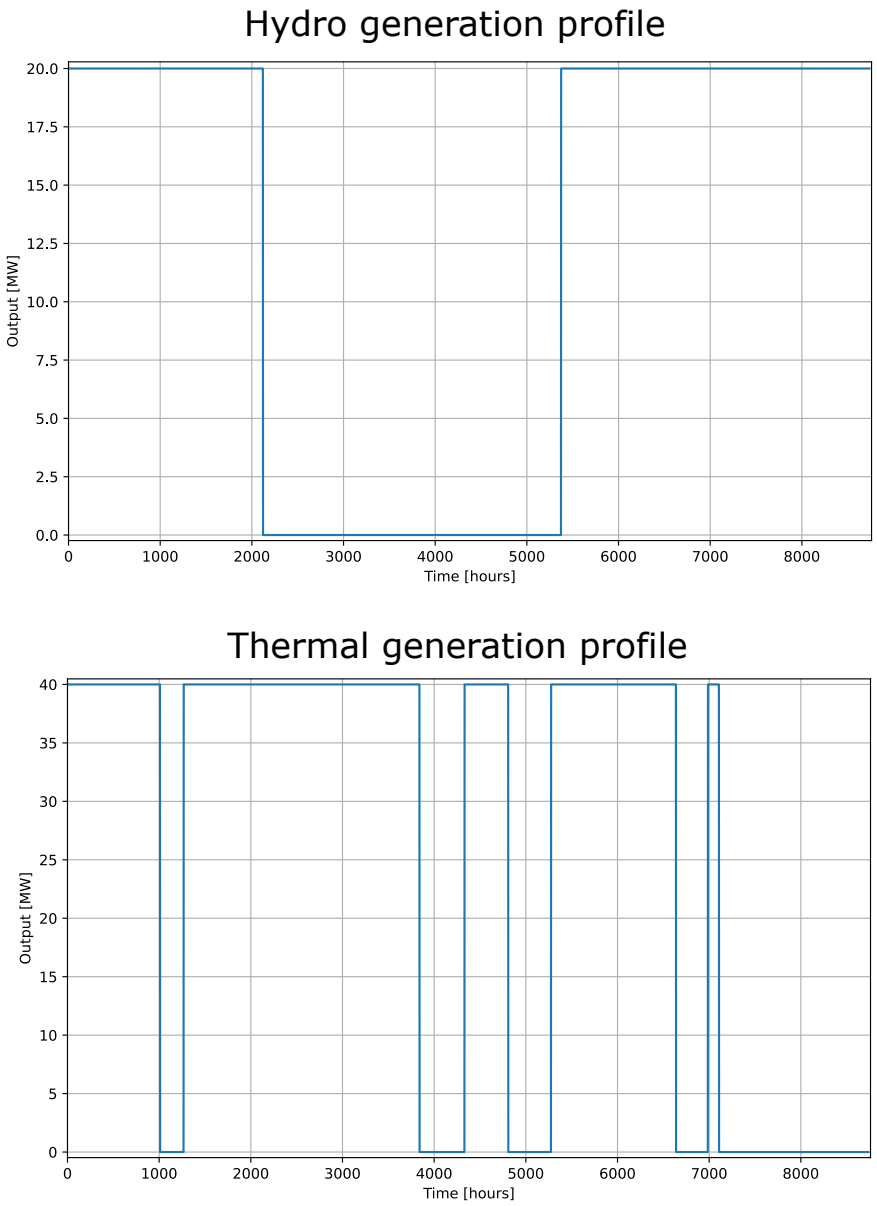


Figure 2.6: Individual generation profiles for the generators

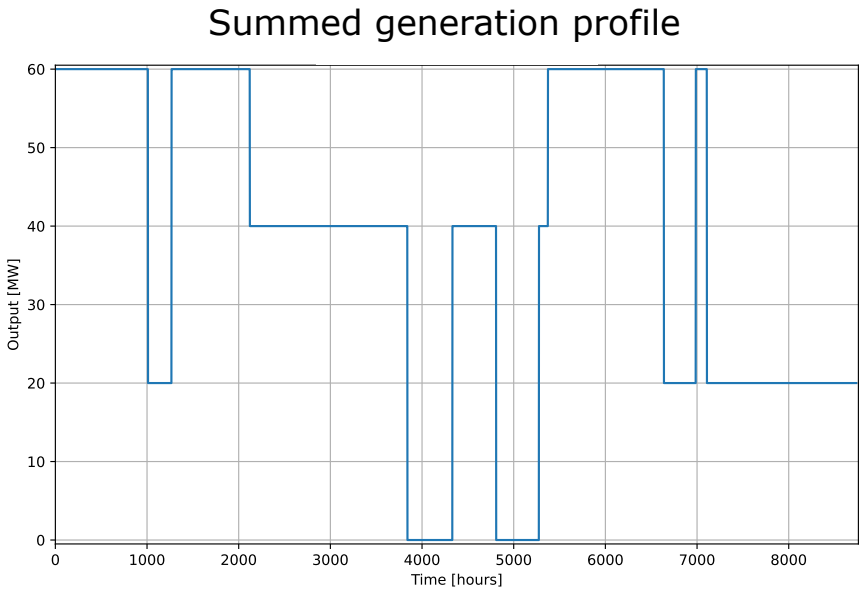


Figure 2.7: System state profile for the generators presented in Table 2.6

A similar procedure is followed when adding derated states. The assumption that the generator is operational at the start of the generator profile still holds. After that, the failure rates that connect the operational state to all the other states are used to calculate a TTF for each transition. The shortest duration out of these TTFs will be chosen as the state to transition to. An example of a generator model with a derated state is given in Figure 2.8.

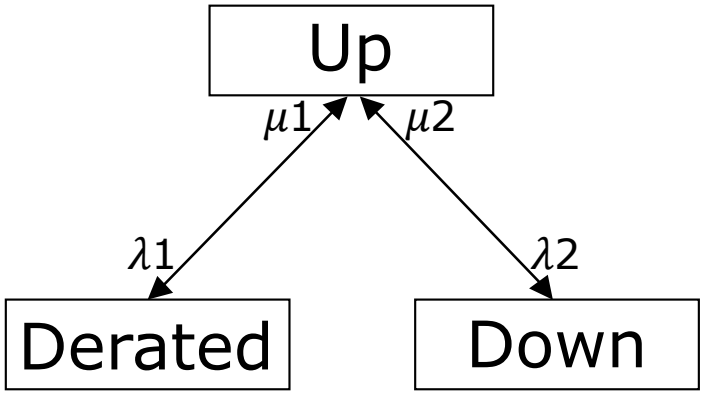


Figure 2.8: Schematic of the states of a generation unit with a derated state, based on [20]

The λ and μ parameters shown in Figure 2.8 represent the transition rates between the states. It can be observed that there are no transitions between the derated state and down state. This is because the assumption is made that the generator will be repaired immediately once it leaves the operational state. The shortest TTF is thus followed by the TTR that is simulated using the repair rate from the state it is departing. Implementations without this assumption are described in [9] and [20].

An advantage of this method is the ease of reliability metric calculations by overlaying the system state profile with the load profile. Disadvantages include the increased computational time compared to the state sampling method and the potential lack of available metrics like the TTF and TTR distributions [10].

2.3.3. State Transition Method

The state transition method is also sequential but considers the transitions between system states instead of individual components. An important criterion is that this method can only be used if the TTF and TTR distributions are exponential. This allows for the summation of the individual component failure rates, resulting in a system transition rate λ that can be used to determine the duration of the current state. The system transition rate can be calculated using Equation 2.7.

$$\lambda = \sum_{i=1}^{n} \lambda_i \tag{2.7}$$

To obtain the next state that the system will transition to, the probability of transitioning to each state has to be calculated using Equation 2.8. In this equation, the transition rate for state j is divided by the sum of all state transition rates to get the probability of transitioning to state j, P_j.

$$P_j = \frac{\lambda_j}{\sum_{i=1}^{n} \lambda_i} \tag{2.8}$$

The sum of the individual state probabilities is one, as the system will transition to another state at some point. A list with the cumulative state probabilities is created to find the next state. Then, a uniform random number is generated and projected on the cumulative probability distribution. The interval in which the random number falls will be selected as the next state. If this means that a component fails, its failure rate will be replaced by its repair rate, and the procedure of obtaining a new state will be repeated.
A deeper understanding of the state transition method, including the mathematical proof, can be obtained from [21].

2.3.4. Stopping Criteria

The principle behind MCS methods is built upon two fundamental mathematical theorems: the law of large numbers and the central limit theorem [22]. An observation X_i can be made, but one observation will not give an accurate result. However, by repeating this process for N simulation years, a sample mean $E(X)$ can be obtained. The formula for obtaining the sample mean can be found in Equation 2.9.

$$E(X) = \frac{1}{N} \sum_{i=1}^{N} X_i \tag{2.9}$$

According to the law of large numbers, the sample mean will converge to the true mean if a large number of simulation years is considered. This law can be combined with the central limit theorem, that states that any distribution with a large number of samples will follow a Gaussian distribution. In this case, the sampled mean can be approximated by a mean of μ_x and a variance of $\frac{{\sigma_x}^2}{N}$, when N is sufficiently large, as shown in Equation 2.10.

$$\lim_{N \to \infty} E(X) = \mu_x \tag{2.10}$$

Increasing the number of simulation years will reduce the variance of the sampled mean. This will increase the accuracy of the sampled mean and get it closer to the true mean. An important distinction can be made between accuracy and precision. The accuracy of an index refers to the difference between the estimated value and the true value, while the precision considers the variance from the obtained sample mean. An example illustrating the differences between accuracy and precision can be found in [9].

Increasing the number of simulation years will increase computational time, so a trade-off has to be made between accuracy and computational time. An index that can aid in making that decision is the coefficient of variation (CoV). It is a unitless quantity that measures the convergence of a simulation and is given as the ratio between the sampled standard deviation and the mean. The formula for the standard deviation is shown in Equation 2.11, while the formula for the CoV can be found in Equation 2.12.

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(X_i - E(X))^2} \tag{2.11}$$

$$CoV = \frac{\sigma}{\sqrt{N}\cdot E(X)} \tag{2.12}$$

The CoV can be used in two ways to determine when to stop the simulation. The first option is to calculate the CoV during every iteration and stop when it is less than a predefined value. Although this approach ensures that no redundant simulations are done, the computational time could still be higher due to calculating the CoV, which includes a computationally intensive summation, during each iteration. A second option would be to calculate the CoV once after a specified number of iterations has been carried out. If the CoV has not gotten below a predefined value, the number of samples has to be increased. This method has reduced computational requirements because the CoV is only calculated once, but finding the correct number of iterations takes some trial and error. Not choosing the correct number of iterations could lead to an unnecessarily longer computation time.

2.4. Reliability Metrics

Now that a framework for classifying reliability studies has been defined, a closer look will be taken at the output of these studies: How can reliability be quantified? When specifically looking at generation adequacy, the moments when the load is greater than the generation are given as a fraction of the total timescale. This can be a single number, such as the total time or total lost energy, or a series of numbers with the frequency and duration of insufficient generation. The first metric gives a quick overview of a system's adequacy, while the latter provides more insight into the severity of individual generation deficits over time. Both metrics are used for reliability studies, depending on the type of study.

2.4.1. LOLP

The simplest Loss of Load based index is the Loss of Load Probability (LOLP). It combines the load profile and installed capacity to compute the maximum allowable capacity outage at a given instance, t. The probability of having a capacity outage larger than the maximum allowable capacity outage yields LOLP, using Equation 2.13.

$$LOLP_t = P(X > C - L_t) \tag{2.13}$$

LOLP is a given as a unitless probability but can be interpreted as the fraction of time that the load demand is not met during the time interval considered for calculating LOLP (i.e. x hour(s) or x year(s), with x $\leq$ 1). The conversion from a unitless probability to a fraction with time units can **only** be done if a singular load increment is used, like a CYPL model. Furthermore, with a singular load increment, it is possible to change the fraction to a different time increment, like hours or days/year for the CYPL model, by multiplying the fraction by the number of hours (8760) or days (365) in a year, respectively.

2.4.2. LOLE

The most used metric for power system reliability studies is Loss of Load Expectation (LOLE) [8], which builds on the calculation for LOLP by multiplying the probability of each time increment with the duration of that time increment. Equation 2.14 gives the formula to calculate LOLE.

$$LOLE = \sum_{t=1}^{365 \text{ or } 8760} LOLP_t \cdot \Delta T \left[\frac{\text{days}}{\text{year}} \text{ or } \frac{\text{hours}}{\text{year}}\right] \tag{2.14}$$

The unit for LOLE depends on the time increment used. Any of the load profiles mentioned in 2.2.2 can be used, but the two most common ones are the DPL, where ΔT is one day, and the HPL, where one hour is used for ΔT. There are some important things to be considered when converting between LOLE, LOLP and the time units. It is possible to convert an LOLE value to an LOLP value by multiplying by the number of time increments in a year, which can only be done the other way around by considering a single load increment.

Converting between units for LOLE is not possible, as the load profiles used are different. For example, if a LOLE value in days/year would be converted to hours/year by multiplying by a factor of 24, it is assumed that the loading during each hour stays the same. This is rarely the case and does not occur for this thesis's daily peak load model.

2.4.3. EENS

Another metric used is the Expected Energy Not Served (EENS), also known as Loss of Energy Expectation (LOEE). The benefit of this metric is that the severity of an outage is now taken into account instead of only indicating the fraction of time when generation is insufficient for the load. The first step in obtaining EENS is to find the expected unserved energy by multiplying the amount of unserved energy by the probability of being in that state, as shown in Equation 2.15.

$$\text{Expected Unserved Energy} = [x_j - (C - L_t)] \cdot p(X = x_j) \tag{2.15}$$

It should be noted that the expected unserved energy cannot be negative. If the outage state x_j is smaller than the allowed outage, $[x_j - (C - L_t)]$ will be a negative number. In this case, there is no unserved energy, and its value should be discarded when calculating EENS. The expected unserved energy for a singular time increment can now be found using Equation 2.15 for all outage states, starting from $x_j = C - L_t$ to avoid the negative capacity deficits. The resulting formula is given in Equation 2.16

$$EENS_t = \sum_{x_j = C - L_t}^{C} [x_j - (C - L_t)] \cdot p(X = x_j) \text{ [MWh]} \tag{2.16}$$

Finally, EENS can be obtained by iterating over all time increments with Equation 2.17.

$$EENS = \sum_{t=1}^{8760} \sum_{x_j = C - L_t}^{C} [x_j - (C - L_t)] \cdot p(X = x_j) \left[\frac{\text{MWh}}{\text{year}}\right] \tag{2.17}$$

2.4.4. ENDS and LOLP'

Furthermore, complementary metrics exist for LOLE and EENS, where the metrics are given as a percentage of a year instead of the hours or days per year. These are called LOLP' and Expected Demand Not Served (EDNS), respectively, and can be obtained by dividing LOLE or EENS by the number of hours or days in a year, depending on the unit used. LOLP' is used instead of LOLP to indicate that the quantity could have been derived from a LOLE value that was constructed with more than one time increment. However, when a CYPL model is used, the values of LOLP and LOLP' are the same.

3
Test Systems

This chapter presents the creation and validation of scripts for HL1 adequacy analysis for the analytical and Monte Carlo simulation methods. The scripts will be tested using two well-known test systems: the IEEE Reliability Test System (IEEE RTS) [23] and the Roy Billinton Test System (RBTS) [24]. These will be elaborated on in the sections below.

3.1. IEEE Reliability Test System

The IEEE RTS was published in 1979 by the IEEE Subcommittee on the Application of Probability Methods to compare reliability evaluation techniques. The reliability test system has been updated over the years to incorporate new technologies and additional component data into the system. In the literature, these revisions are denoted by including the year of the revision behind the name of the test system. The original system from 1979 would thus be denoted as "RTS-79". Revisions were made in 1986 and 1996, but the benchmark data used for validating the scripts in this thesis use the original system from 1979. Some key figures of the IEEE RTS-79 can be found in Table 3.1

Table 3.1: Key figures for the IEEE RTS

Number of buses	24
Number of transmission lines	38
Number of generators	32
Voltages [kV]	230, 138
System peak load [MW]	2850
Total installed capacity [MW]	3405

To create a COPT for the analytical method, the capacities of individual generators must be known. The type of generator, along with their capacity and FOR, is presented in Table 3.2

Table 3.2: Generator data for the IEEE RTS

Generator type	Capacity [MW]	Number of units	Forced Outage Rate (FOR)
Oil/Steam	12	5	0.02
Oil/Combustion Turbine	20	4	0.1
Hydro	50	6	0.01
Coal/Steam	76	4	0.02
Oil/Steam	100	3	0.04
Coal/Steam	155	4	0.04
Oil/Steam	197	3	0.05
Coal/Steam	350	1	0.08
Nuclear	400	2	0.12

A one-line diagram of the IEEE RTS-79 can be found below in Figure 3.1. Additional system data can be found in [23].

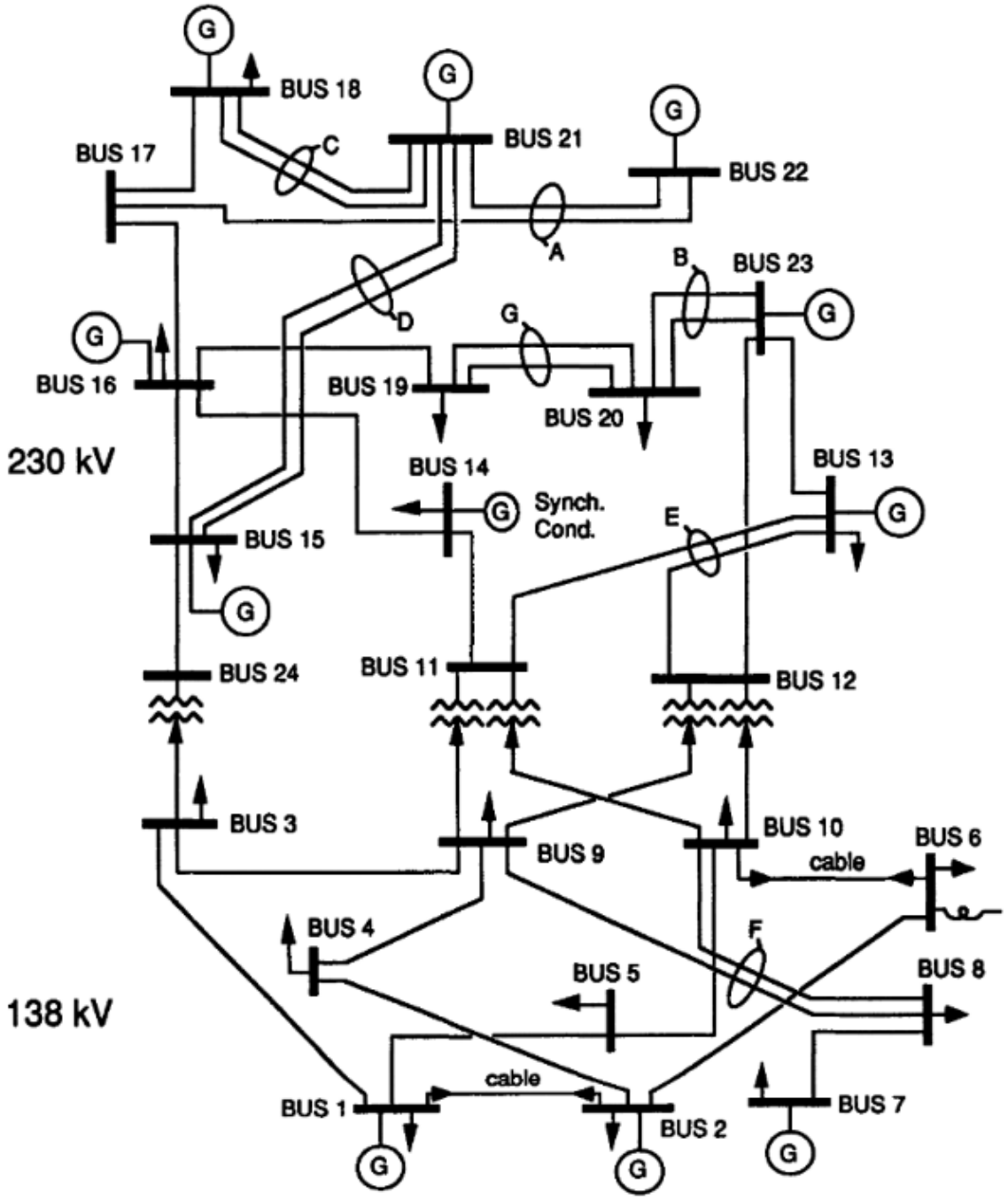


Figure 3.1: Schematic of the IEEE RTS, reproduced from [10]

3.2. Roy Billinton Test System

The RBTS is a test system developed by the Power System Research Group at the University of Saskatchewan, Canada, as part of their research programs and education on reliability. As the IEEE RTS already existed, the reason for proposing a new test system is that the IEEE RTS needs a computer to be solved. The RBTS aims to be more suitable for understanding basic reliability concepts. This is mainly done through reducing the size of the network. Derated states can be included for the two 40 MW thermal units if desired. They are operation states of a generation unit in which the output is somewhere in between the fully operational, UP, state and the outage, DOWN, state. The 40 MW thermal generators in the RBTS have been assigned one derated state, which operates at half of the maximum capacity. Including an extra state means that the availability and FOR probabilities also change. Each output has been assigned a new probability to be in that particular state, with the sum of all state probabilities still equal to one.

The original system [24], which was created in 1989, will be used in this thesis. A summary of some key characteristics of the RBTS is given in Table 3.3.

Table 3.3: Key figures for the RBTS

Number of buses	6
Number of transmission lines	9
Number of generators	11
Voltage [kV]	230
Bus voltage limits V_{min}, V_{max} [pu]	0.97, 1.05
System peak load [MW]	185
Total installed capacity [MW]	240

The generator characteristics for constructing a COPT for the analytical method are given in Table 3.4.

Table 3.4: Generator data for the RBTS

Generator type	Capacity [MW]	Number of units	Forced Outage Rate (FOR)
Hydro	5	2	0.01
Thermal	10	1	0.02
Hydro	20	4	0.015
Thermal	20	1	0.025
Hydro	40	1	0.02
Thermal	40	2	0.03

A one-line diagram of the RBTS can be found below in Figure 3.2. For additional data about the system, such as the bus and line data, the reader is referred to [24].

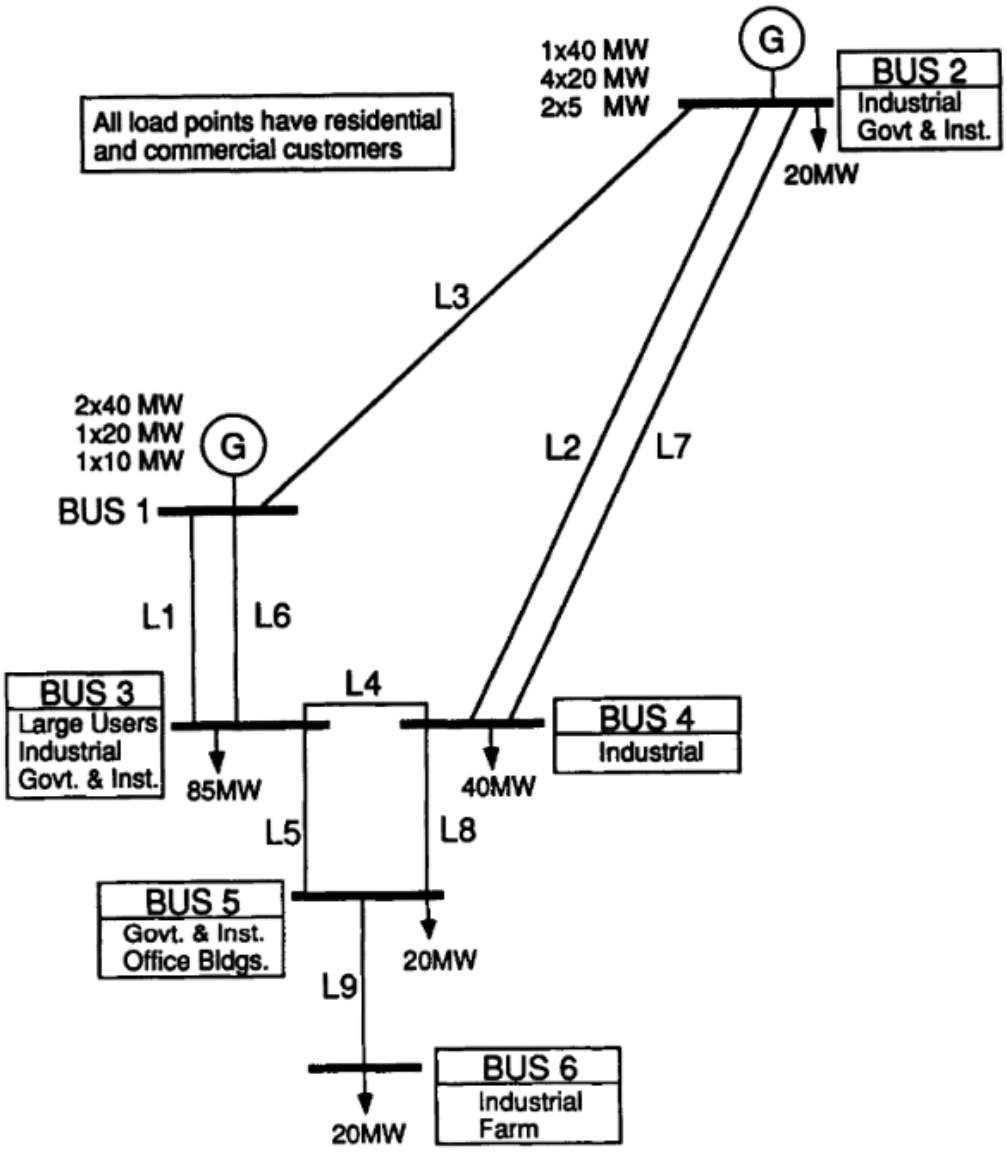


Figure 3.2: Schematic of the RBTS, reproduced from [10]

4
Methodological Approach for Constructing the Analytical Scripts

Using the theory in Sections 2.2 and 2.4, scripts are created to obtain the load curves and the COPT and calculate the reliability metrics. A distinction will be made between the analytical method with and without wind, as there are differences between the procedure for constructing the COPT and the available benchmark data for validating the results. As the scripts for obtaining the load curves and the reliability indices are only used for validating the results for the method without wind, they will only be elaborated on in Section 4.1. The inclusion of wind generation into the scripts will be covered in Section 4.2.

4.1. Analytical Method without Wind Considerations

The scripts used to obtain reliability indices using the analytical method without wind considerations will be elaborated on in the upcoming sections. The complete codes for the COPT, load curves and reliability indices scripts can be found in Appendices B.1, B.2 and B.3.

4.1.1. COPT

The procedure for obtaining the Capacity Outage Probability Table is based on the description given in Section 2.2.1. All the computations for obtaining the COPT are done within the **calc_COPT** function, which takes the generator data as input. It contains the capacity outage levels and accompanying probabilities of having that much capacity outage for all generators. This data is defined in an Excel file and imported to the script, after which the capacity outage levels and probabilities are stored in separate variables. These are either lists or matrices, depending on whether or not derated states are included.

A COPT is created using a recursive algorithm. The same procedure is used for each iteration in which a new generator is added. The first step of an iteration is to create a list with the new generator capacity outage levels. This is done by adding all the capacity outage states of the generator that are added during that iteration individually to the existing list of outage capacities, resulting in an updated list. There could be duplicate outage levels in the list, which will be checked by the **remove_duplicates** function, and duplicate levels will be removed if applicable.

The cumulative probability associated with every capacity outage level can now be calculated using Equation 2.1. The cumulative probabilities from the previous iteration are found by subtracting each outage capacity level of the added generator from the capacity outage list. The result of this subtraction is then checked in the COPT from the previous iteration. If the result is smaller than zero, the cumulative probability equals 0. A result larger than the largest capacity outage in the previous COPT also yields a zero probability. Another possibility is an exact match between the COPT and the result, in which case the probability of that outage level in the COPT is assigned. For the final case, when the result is between the minimum and maximum capacity outage level of the COPT, and there is no exact match, the capacity outage levels in which the result lies are identified. The probability of being in the upper capacity outage level out of the two neighbouring outage levels is chosen. These probabilities are obtained for all outage states for the generator that is added to the COPT and appended to a list.

The list of cumulative probabilities is then multiplied by the probability of being in that generator's capacity outage state. A generator without derated states would only have an available and unavailable state. Summing the multiplication results together yields the cumulative probabilities for the updated COPT. This process is repeated until all generators are added to the COPT. Pseudo-code for creating a COPT without wind considerations can be found below in Algorithm 1.

Algorithm 1: Pseudo-code for obtaining a COPT without wind considerations

Input: Generator data is imported from a datafile, containing the capacity outage levels for each generator *Gen_outage_capacity* and the associated probability of being in that state *Gen_outage_probability*.

Output: A list with the combined generator outage capacities *System_Outage_Capacity* and a list with the cumulative probabilities for the combined generator outage capacities *System_Outage_Probability*.

```
Function calc_COPT(Generator_data):
    Initialise lists for the System_Outage_Probability for the next and previous iteration;
    for Each generator do
        for The selected generator do
            Add each capacity outage state of the specific generator to each value in the
             System_Outage_Capacity list and remove duplicate outage levels;
            Initialise the new_System_Outage_Probability list;
        for Each System Outage Capacity do
            Initialise the new and old probability lists;
            for Each Generator Outage capacity state do
                Calculate X - C for each System Outage Capacity and store the results in System
                 Outage Capacity minus New Generator;
                if X-C is smaller than 0 then
                    P_old equals 1;
                else if X-C is bigger than the maximum old system outage then
                    P_old equals 0;
                else if X-C is equal to an old system outage value then
                    P_old equals the old system outage probability value;
                else
                    X-C falls between one of the states. The probability associated with the upper
                     state boundary is assigned to P_old;
                Add the value of P_old to a list
                 Calculate P_new,temp using EQUATION
                 Add the value of P_new,temp to a list
            Sum the probability contributions of one System Outage Capacity and add them to
             the list of all System Outage Capacities.
        Assign the new system outage capacity and probability to the old lists in preparation for
         the next iteration.
    return System_Outage_Capacity, System_Outage_Probability
```

4.1.2. Load Curves

The load curves for the yearly, weekly, daily and hourly peak load are made with the function **CreateLoadCurve**, which takes the YPL as an input. This thesis uses the YPL from either the IEEE RTS-79 or RBTS. The percentages for obtaining WPL, DPL and HPL, given in Appendix F, are imported from an Excel file. All load profiles are then modified to have a load increment for every hour, which allows for plotting all load profiles in one graph.

Obtaining lists with hourly load increments is straightforward for the YPL, WPL and DPL. Multiply the load with a list of ones with the number of hours in the load profile's time increment as its length. The YPL would thus be multiplied by a list of ones with a length equal to the number of hours in a year. The IEEE load profile uses 8736 for the number of hours in a year instead of the usual 8760. This is because the model uses 52 weeks to represent a year, seven days for a week and 24 hours for a day. This results in $52 \cdot 7 \cdot 24 = 8736$ hours in a year. The YPL variable becomes a list with 8736 times the YPL as an entry.

The WPL and DPL are constructed using the same method by multiplying all load increments within the load profile with a list of ones that have a length of either 168 (the number of hours in a week) or 24 (hours in a day). The HPL, on the other hand, is obtained using a different procedure. Its time increment is already hours, so there is no need to use lists with ones to change the unit of time. The hourly data is given as 24 data points during a day, but a distinction is made between a weekday and a weekend. Furthermore, different load profiles are given depending on the season. Profiles are specified for the summer, winter and spring/fall, as can be seen in Table F.3 in Appendix F. The first step in constructing the HPL variable is to create lists for the three given seasons, with five weekdays and two weekenddays. Based on the indicated week numbers for each season, i.e. weeks 1-8 correspond to the winter, the corresponding data for a whole week during that season can be added to a list. The resulting load curves are shown in Figure 2.3a, while load duration curves can be obtained by sorting the load curves in descending order. These can be found in Figure 2.3b. The Python code used for constructing and plotting the load (duration) curves can be found in Appendix B.2. Pseudo-code for the load curves is shown below in Algorithm 2.

Algorithm 2: Pseudo-code for obtaining the IEEE load curves

Input: The Yearly Peak Load value of the test system is defined outside of the function and used as an input for the load profiles

Output: Lists for the YPL, WPL, DPL and HPL in chronological and descending order for each time increment specified in the *time* list. The descending lists are obtained by sorting the chronological lists

1 **Function** `createLoadCurve`(*YPL_value*)**:**

2 Define the number of time increments, which is 8736 for the IEEE profile
Multiply the *YPL_value* with a list of ones with the length of the time to obtain *YPL_plot*. As all the values are the same, this is both the chronological and descending order list

3 *WPL_chrono* can be obtained by multiplying each weekly percentage by a list of 168 increments for the 7 x 24 = 168 hours in a week.

4 *DPL_chrono* is created by multiplying all the daily percentages by a list of 24 ones to create a profile of a week with hourly increments. The profile is then duplicated to create a profile with 52 weeks.

5 The *HPL_chrono* has different values for weekdays and weekends, as well as seasons. First, a profile for a week is created for all seasons by adding 5 weekdays and 2 weekend days to obtain week profiles.
The week profiles are then added to the year profile by selecting the right season based on the week number ranges for the seasons.

6 **return** *time, YPL_plot, WPL_chrono,WPL_descend,DPL_chrono,DPL_descendHPL_chrono,HPL_descend*

4.1.3. Reliability Metrics

The script for calculating LOLP, LOLE, EENS, LOLP' and ENDS is based on the theory presented in Section 2.4. The three main metrics, LOLP, LOLE and EENS, are calculated using the following functions: **calc_LOLP**, **calc_LOLE** and **calc_EENS**. The derived metrics, LOLP' and ENDS, are obtained by dividing LOLE and EENS by the number of time units in a year for the used unit. For example, if the unit of LOLE is hours/year, the unitless metric LOLP' can be found by dividing by the number of hours in a year, which is 8736 for the used load profile. The Python code used for calculating the reliability metrics can be found in Appendix B.3.

LOLP is a unitless metric that gives the probability of having a capacity deficit at one particular time instant. The **calc_LOLP** function takes the capacity outage and cumulative probability of the COPT and the load value as inputs. The load is subtracted from the installed capacity, and the COPT is used to find the cumulative probability corresponding to the resulting value. For values below zero, the cumulative probability is 1. A value higher than the highest outage level in the COPT has a cumulative probability of 0. An exact match between the COPT outage level and value means that the probability corresponding to the COPT outage level is assigned. Finally, if the value is between two outage levels, the probability of the higher outage level is used.

The function for calculating LOLE, **calc_LOLE**, uses the same logic as the LOLP function. The only difference this time is that multiple load increments are considered. The time increment is usually a day or hour, and the considered period is often a year. The LOLP procedure is carried out for each time increment, and the probabilities from each iteration are multiplied by the time increment used and summed together. The Pseudo-code for obtaining LOLP and LOLE is given in Algorithm 27.

Algorithm 3: Pseudo-code for obtaining the reliability indices LOLP and LOLE using an analytical method

Input: The capacity outage levels for each generator *Gen_outage_capacity* and the associated cumulative probability of being in that state *Gen_outage_probability* are used for obtaining the reliability metrics, as well as the load profile
Output: A singular value for either *LOLP* or *LOLE*

```
Function calc_LOLP(Generator_Outage_Capacity, Cumulative_Outage_Probability, Load_Profile):
    Find the maximum installed capacity Max_Capacity
     The amount of outage that would cause a deficit Max_Outage = Max_Capacity - Load
    if Max_Outage is smaller than 0 then
        LOLP equals 1;
    else if Max_Outage is equal to an outage state in Generator_Outage_Capacity then
        LOLP equals the Cumulative_Outage_Probability associated with
         Generator_Outage_Capacity;
    else if Max_Outage is larger than Generator_Outage_Capacity then
        LOLP equals 0;
    else
        Max_Outage falls between one of the states. for every outage state i in
         Generator_Outage_Capacity do
            if Max_Outage is between Generator_Outage_Capacity[i] and
             Generator_Outage_Capacity[i+1] then
                The probability associated with the upper state boundary is assigned to LOLP;
    return LOLP
Function calc_LOLE(Generator_Outage_Capacity, Cumulative_Outage_Probability, Load_Profile):
    Initialise an empty list P_list to store all probabilities
     Find the maximum installed capacity Max_Capacity
    for each time increment do
        The amount of outage that would cause a deficit Max_Outage = Max_Capacity - Load if
         Max_Outage is smaller than 0 then
            P for the time increment equals 1;
        else if Max_Outage is equal to an outage state in Generator_Outage_Capacity then
            P for the time increment equals the Cumulative_Outage_Probability associated with one
             state above the matching Generator_Outage_Capacity;
        else if Max_Outage is larger than Generator_Outage_Capacity then
            P for the time increment equals 0;
        else
            Max_Outage falls between one of the states. for every outage state i in
             Generator_Outage_Capacity do
                if Max_Outage is between Generator_Outage_Capacity[i] and
                 Generator_Outage_Capacity[i+1] then
                    The probability associated with the upper state boundary is assigned to LOLP;
    return LOLE
```

The **calc_EENS** function builds upon the LOLE function by including the severity of a capacity deficit. The difference between the installed capacity and load is subtracted from all capacity outage states to find possible capacity deficits. If there is a capacity deficit, it is multiplied by the probability of having that amount of capacity outage. The energy computed for a single iteration is then summed together for all time increments to obtain EENS over the specified period. The Pseudo-code for obtaining EENS is given in Algorithm 4.

Algorithm 4: Pseudo-code for obtaining the reliability index EENS using an analytical method

Input: The capacity outage levels for each generator *Gen_outage_capacity* and the associated individual probability of being in that state *Gen_outage_probability* are used for obtaining the reliability metrics, as well as the load profile

Output: A singular value for*EENS*

```
Function calc_EENS(Generator_Outage_Capacity, Individual_Outage_Probability, Load_Profile):
    Initialise the EENS value to 0
     Find the maximum installed capacity Max_Capacity
    for each time increment do
        Initialise the energy sum E_Sum which stores the value of the energy not delivered
         during one time increment for each outage state in Generator_Outage_Capacity do
            The capacity outage Cap_Outage is equal to
             (Generator_Outage_Capacity-(Max_Capacity-Load) if Cap_Outage is larger than 0 then
                Energy not served contribution = Cap_Outage · Individual_Outage_Probability
                 associated with the outage state in Generator_Outage_Capacity
                 The individual Energy not served contributions are summed together in E_Sum ;
                else
                    pass
            The E_Sum of each time increment is added to EENS
    return EENS
```

The reliability indices that are obtained using the script described in this section are verified using the two test systems introduced in Sections 3.1 and 3.2: the IEEE RTS and RBTS. Benchmark values for the IEEE RTS-79 are presented in [25] for an HPL model. It contains results for LOLE, EENS and their derived metrics LOLP' and EDNS.

The RBTS benchmark values are obtained from [26], which presents results for both an HPL and YPL model. It doesn't provide the derived quantities LOLP' and EDNS, but these could be obtained using the method presented in Section 2.4.4 if desired.

Another type of benchmark data that will be used are the values obtained from the scripts in [8]. The scripts developed in Kjetil's Master Thesis also calculate reliability indices for the two test systems mentioned before. A comparison between the available benchmark data and the results from the scripts described in this thesis is given in Table 4.1 for the HPL model and Table 4.2 for the YPL model. The absolute percentage difference between the benchmark value and the script is denoted with Δ. Benchmark data that isn't available has been denoted with 'N/A' in the tables.

Table 4.1: Comparison between benchmark data and the scripts presented in this thesis for the HPL model

	LOLE [hours/year]				LOLP' [unitless]			
Test System	Benchmark	Kjetil's Script [8]	Script	Δ [%]	Benchmark	Kjetil's Sscript [8]	Script	Δ [%]
IEEE RTS-79	9.36881	9.39389	9.39390	0.268	0.001069	0.001075	0.001075	0.561
RBTS	1.0919	1.0914	1.0914	0.046	N/A	0.000125	0.000125	N/A

	EENS [MWh/year]				EDNS [MW/year]			
Test System	Benchmark	Kjetil's Script [8]	Script	Δ [%]	Benchmark	Kjetil's Script [8]	Script	Δ [%]
IEEE RTS-79	1181.195	1176.278	1176.278	0.416	0.1348396	0.13464710	0.13464716	0.143
RBTS	9.8613	9.8603	9.8603	0.010	N/A	0.00112869	0.00112869	N/A

Table 4.2: Comparison between benchmark data and the scripts presented in this thesis for the YPL model

	LOLE [hours/year]				LOLP' [unitless]			
Test System	Benchmark	Kjetil's Script [8]	Script	Δ [%]	Benchmark	Kjetil's Script [8]	Script	Δ [%]
IEEE RTS-79	N/A	738.874	738.874	N/A	N/A	0.08458	0.08458	N/A
RBTS	73.0728	72.8722	72.8723	0.274	N/A	0.00834	0.00834	N/A

	EENS [MWh/year]				EDNS [MW/year]			
Test System	Benchmark	Kjetil's Script [8]	Script	Δ [%]	Benchmark	Kjetil's Script [8]	Script	Δ [%]
IEEE RTS-79	N/A	128364.0	128364.0	N/A	N/A	14.69368	14.69368	N/A
RBTS	823.2555	821.0000	821.0000	0.274	N/A	0.093979	0.093979	N/A

The results in Tables 4.1 and 4.2 show that the script developed in this thesis exactly matches the values obtained in [8] apart from a one-digit difference on the last significant digit of some metrics. There are minor differences compared to the other benchmark data, ranging from 0.1 to 0.5 %. A possible factor contributing to the differences could be how different programming languages round intermediate results. Overall, it can be concluded that the script used in this thesis is accurate enough to obtain reliability metrics for generation adequacy analysis.

4.2. Analytical Method with Wind Considerations

The COPT script used for creating a COPT table for a wind farm, and the procedure for validating the obtained COPT will be elaborated on below.

4.2.1. COPT

A COPT can be constructed for a wind farm, as it consists of individual generators. One of the major differences, however, is the absence of a limited number of power output states. A wind turbine starts producing power from a wind speed referred to as the cut-in wind speed. With increasing wind speeds, the power increases to its rated power and accompanying rated wind speed. From this point onwards, the power is kept at the rated power level to protect the wind turbine's components and limit the noise production if there are limits at the turbine site, but this won't be an issue for offshore wind farms. The angle of the turbine blades with respect to the rotor hub, the so-called pitch, is altered for wind speeds above the rated speed. If the pitch cannot be altered sufficiently anymore for higher wind speeds to keep the stresses on the wind turbine within limits, the rotor will be positioned out of the wind direction, and a brake will be applied. The wind speed at which this happens is called the cut-out wind speed. A method to model the power profile of a wind turbine based on the rated power and wind speed characteristics like the cut-in, rated, and cut-out wind speed is proposed in [27]. The power curve can be plotted based on Equation 4.1:

$$\begin{aligned} P_{\text{output}} &= 0 && \text{if } 0 \leq V < V_{ci} \\ &= (A + Bv + Cv^2)P_r && \text{if } V_{ci} \leq V < V_r \\ &= P_r && \text{if } V_r \leq V < V_{co} \\ &= 0 && \text{if } V \geq V_{co} \end{aligned} \tag{4.1}$$

where: P_r = rated power
V_{ci} = cut-in wind speed
V_r = rated wind speed
V_{co} = cut-out wind speed

The variables A, B and C, which are used to find the power output between the cut-in and rated wind speed, can be calculated using Equations 4.2, 4.3 and 4.4.

$$A = \frac{1}{(V_{ci} - V_r)^2} \cdot \left(V_{ci}(V_{ci} + V_r) - 4V_{ci}V_r \cdot \left(\frac{V_{ci} + V_r}{2V_r} \right)^3 \right) \tag{4.2}$$

$$B = \frac{1}{(V_{ci} - V_r)^2} \cdot \left(4(V_{ci} + V_r) \cdot \left(\frac{V_{ci} + V_r}{2V_r} \right)^3 - (3V_{ci} + V_r) \right) \tag{4.3}$$

$$C = \frac{1}{(V_{ci} - V_r)^2} \cdot \left(2 - 4 \cdot \left(\frac{V_{ci} + V_r}{2V_r} \right)^3 \right) \tag{4.4}$$

By inputting a wind speed profile ranging from 0 to 30 m/s into Equation 4.1, the power profile of a wind turbine can be obtained by plotting the wind speed against the power output. This is done for a typical wind turbine, the DTU 10 MW reference wind turbine [28], in Figure 4.1.

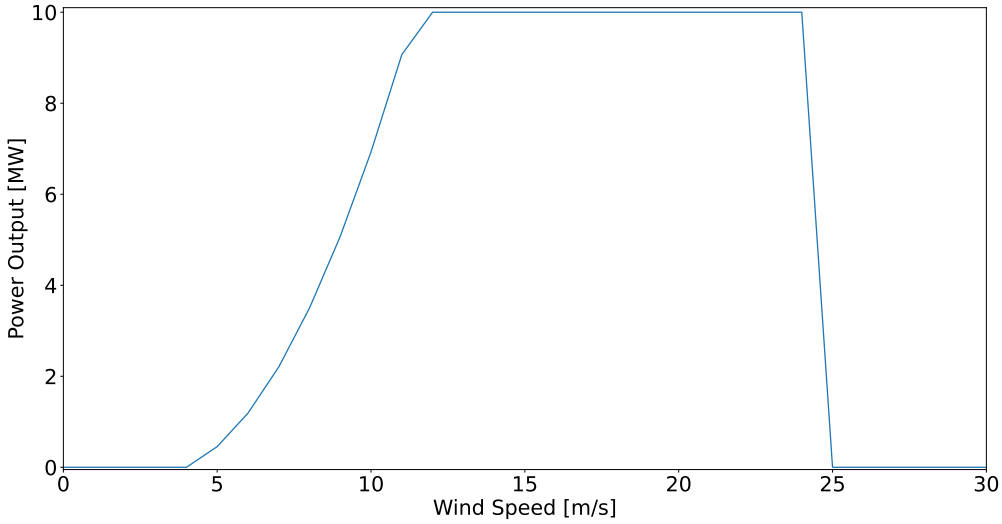
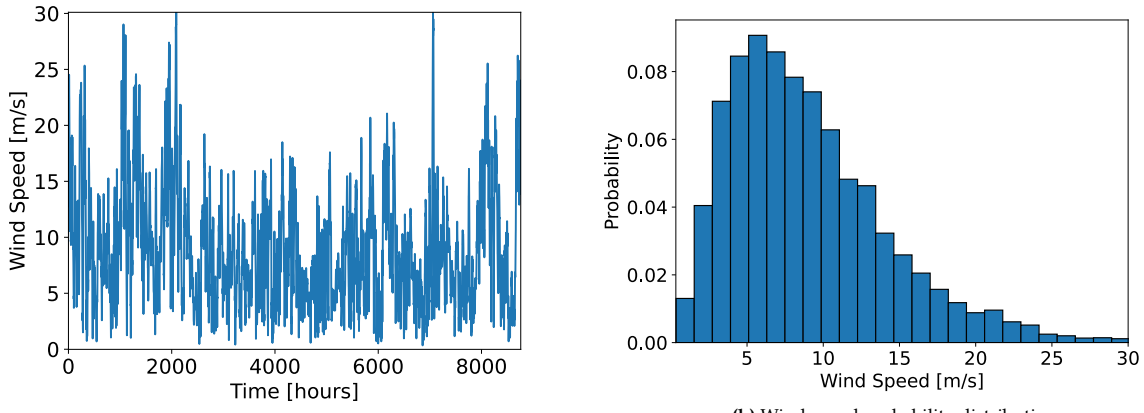


Figure 4.1: Power output profile for the DTU 10 MW reference turbine based on the method in [27]

The next step in obtaining a wind turbine COPT would be creating a state probability table for the power output. The power output of a wind turbine depends on the encountered wind speeds. For this example, wind speeds of the coast of Trondheim (latitude 64°N, longitude 8°E) for 2019 have been used from the xyz windatlas [29]. A chronological plot of the wind speeds is shown in Figure 4.2a, while Figure 4.2b gives a probability distribution of the same data.



(a) Chronological wind speeds

(b) Wind speed probability distribution

Figure 4.2: Wind speed data of the coast of Trondheim for 2019 [29]

By combining the power profile of a turbine and wind speed data, a probability distribution for the power output can be made. This distribution can be split into several intervals, representing the output states of a turbine. These states could be compared to the derated states of a conventional generation, but the mechanical availability has not yet been considered. An example of a power output probability distribution for the DTU 10 MW reference turbine with wind speeds from [29] is presented in Figure 4.3.

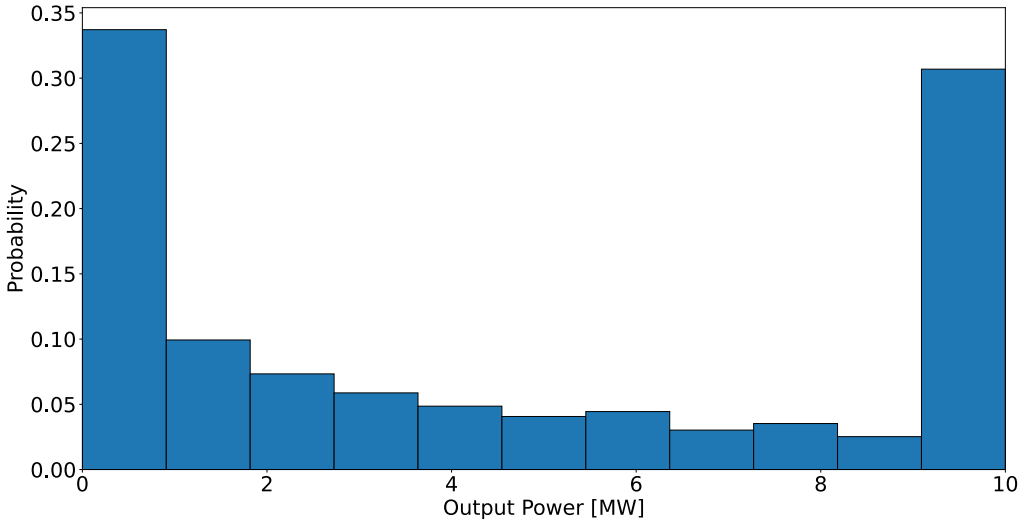


Figure 4.3: Power output probability distribution for a turbine of the coast of Trondheim in 2019

With a probability distribution for the power output of a wind turbine now in place, the mechanical failures of the turbine should be added to create a COPT. This can be done using Equation 4.5, presented in [27]. It should be noted, however, that this equation returns the cumulative **output** instead of the cumulative **outage**. Another example to aid the reader's understanding can be found in [11].

$$\text{Cumulative output probability for } X_{\text{output}} = \sum_{k=0}^{X_{max}} \left(\sum_{i=0}^{t} \left(\sum_{j=0}^{s} q_j H\left(\frac{X_{\text{output}}}{i} - c_j \right) \right) \cdot p_i \right) \tag{4.5}$$

where: X_{max} = Maximum wind farm output
k = Number of possible X_{output} states
t = Number of turbines in the wind farm
s = Number of capacity states for a wind turbine
q_j = Probability of operating in output state c_j
H = Heaviside step function to check for valid states
X_{output} = Wind farm output [MW]
i = Available wind turbines
c_j = Turbine output in state j
p_i = Probability of i turbines being available

With Equation 4.5, the probability of having a particular output due to the wind is multiplied by the probability of having a certain number of turbines available due to mechanical outages. The Heaviside step function, $H\left(\frac{X_{output}}{i} - c_j\right)$, in Equation 4.5 checks if a combination of the number of available turbines i and the power output state j is a valid result. If the argument is smaller than zero, H will be 0. Otherwise, H is equal to 1.

If the result of the check is smaller than zero, the combination of i and j will result in an X_{output} larger than the specified output. It should, therefore, not be included in the cumulative probability ranging from 0 to X_{output}.

Going through all positive output levels yields a Capacity **Output** Probability Table. A Capacity **Outage** Probability table can now be obtained by reversing the order of the probabilities.
One optional step could be to reduce the number of states in the COPT. The number of states could be significant, depending on the number of turbines and output states per turbine. The apportioning method [14] can reduce the number of states to any desired number. The probabilities from the states that fall between two of the new state levels are split up and assigned/apportioned to the probability of the new state levels. The following equations, 4.6 and 4.7, can reduce the number of states in a COPT. It uses the availability A and the outage capacity C. The subscripts m and n are used for the upper and lower levels of the apportioned capacity outage levels, while the subscript i can be used for both the C and A of the state that is being apportioned.

$$A_{ni} = A_i \cdot \frac{C_m - C_i}{C_m - C_n} \tag{4.6}$$

$$A_{mi} = A_i \cdot \frac{C_i - C_n}{C_m - C_n} \tag{4.7}$$

The described methods have been implemented in Python and will be validated in Section 4.2.2. The script can be found in Appendix B.4, while the pseudo-code can be found in Algorithm 5.

Algorithm 5: Pseudo-code for obtaining a COPT with wind considerations

Input: Turbine data is imported from a datafile, containing the rated power, cut-in wind speed, rated wind speed and cut-out wind speed. The FOR is specified in the file itself

Output: A list with the combined wind farm outage capacities *Farm_Outage_Capacity* and a list with the cumulative probabilities for the combined wind farm outage capacities *Farm_Outage_Probability*.

1 **Function** `createPowerProfile`(*wind_speed,turbine_power_profile*)**:**

2 Import the turbine characteristics
Put the wind speed list through Equation 4.1 to find the power output profile. **return** *P_output, P_rated,wind_speed*

3 A list of available power outputs states of the wind farm is created. Based on the power output profile, the probability of being in any of the states can be determined.
The mechanical outage probability of the turbines can be found using a binomial distribution for the (un)availability. The mechanical outage and wind power **output** are combined using Equation 4.5.
The **output** probability table can be converted to an **outage** probability table by inverting the order of the probability list.

4 **Function** `Apportioning`(*Apportioned_States,State_Outage,State_Outage_Probability*)**:**

5 The desired *Apportioned_States* can be specified
Go through all states and apportion the probability of the states according to Equations 4.6 and 4.7.

6 **return** *Apportioned_State_Outage, Apportioned_State_Outage_Probability*

4.2.2. Validation of the Results

Not many references are available that clearly describe the complete process from wind speeds to a COPT. Therefore, the validation of the analytical method with wind considerations will be limited to creating a COPT from the power output probability distribution.

An example of obtaining a reduced COPT using the apportioning method from a power output probability distribution is presented in [13]. This paper uses a wind farm consisting of 10 turbines with a power rating of 2 MW each. The number of states used to describe the wind conditions can be chosen based on the desired accuracy and possible limitations in computational power. These are presented in [13] in so-called State Capacity Outage Probability Tables (SCOPT), ranging from 2 to 11 states between 0 and 100 % capacity outage. A 5-state SCOPT for the wind farm has been chosen to validate the results, as it is later given as an example in [13] when the wind conditions and mechanical reliability are combined. The 5-state SCOPT obtained from the script is shown below in Table 4.3.

Table 4.3: 5 state COPT for the wind conditions

5 SCOPT	
Capacity Outage [%]	Individual Probability
0	0.07021
25	0.05944
50	0.11688
75	0.24450
100	0.50897

The unavailability of the wind turbines is assumed to be four percent. The probability distribution in Table 4.4 is obtained for the wind farm using a binomial distribution.

Table 4.4: Capacity outage probability due to the mechanical reliability of the turbines for a FOR of 4%

Capacity Outage [MW]	Individual Probability
0	0.66486
2	0.27701
4	0.05194
6	0.00577
8	0.00042
10	0.00002
12	0.00000
14	0.00000
16	0.00000
18	0.00000
20	0.00000

The outage capacity due to the wind conditions is converted to a **output** capacity table. It can now be combined with the mechanical outage data using Equation 4.5, resulting in a **capacity output table**. A so-called Wind Energy Conversion System Model (WECS) can be obtained with the outage probabilities by reversing the order of the probabilities list. The WECS COPT for a 5-state model is shown in Table 4.5.

Table 4.5: COPT for the 20 MW wind farm presented in [13]

COPT for 5 SCOPT			
Capacity Outage [MW]	Individual Probability	Capacity Outage [MW]	Individual Probability
0	0.04668	14	0.00005
2	0.01945	15	0.16255
4	0.00365	15.5	0.06773
5	0.03952	16	0.01270
6	0.00041	16.5	0.00141
6.5	0.01647	17	0.00010
8	0.00312	17.5	0.00001
9.5	0.00034	18	0.00000
10	0.07771	18.5	0.00000
11	0.03240	19	0.00000
12	0.00607	19.5	0.00000
12.5	0.00000	20	0.50897
13	0.00067		

The reduced 5-state COPT for a 20 MW wind farm is obtained by applying the apportioning method and can be found in Table 4.6.

Table 4.6: Reduced 5-state COPT for the 20 MW wind farm presented in [13]

Capacity Outage [MW]	Individual Probability
0	0.05908
5	0.06335
10	0.11475
15	0.24408
20	0.51875

These results match the ones presented in [13], and the script for obtaining the COPT of a wind farm thus works correctly.

5
Methodological Approach for Constructing the Non-Sequential MCS Scripts

Using the theory in Sections 2.3, scripts are created for obtaining a generation profile and calculating the reliability metrics. The load curves script is the same as the one used for the analytical method, presented in Section 4.1.2. An overview of the methodology used to create the MCS scripts and a comparison of the results with analytical and MCS benchmark values will be given.

5.1. Monte Carlo Simulation Method without Wind Considerations

The scripts that are used to obtain reliability indices using an MCS method without wind considerations will be elaborated on below. There are four scripts for the MCS method: two for the generation profile, one for the load curves and one for calculating the reliability metrics. An additional script for the generation profile has been created when there are no derated states in the system, as the script's execution time can be significantly sped up in this case. The load curves script can be found in Appendix B.2, while the scripts for the two generation profiles and the reliability metrics can be found in Appendices C.1, C.2 and C.3.

5.1.1. Generator Profile

Three methods can be used to implement the generator profile for an MCS, which are mentioned in Section 2.3: the state sampling, state duration and state transition method. Out of these options, the state sampling and state duration method will be implemented, but the focus in this section will be on the non-sequential state sampling method. The scope of this thesis is limited to obtaining the reliability metrics LOLE and EENS, for which the chronology is not important. An initial implementation of the sequential state duration method can be found in Appendix A.

The state sampling method uses the state probabilities of a generator, and the generator state profile matches the duration of the IEEE load curve with 8736 hourly generation increments. The state probabilities are either based on previous reliability data of generators or an educated guess is made based on similar generators. The generated random numbers follow a uniform random number generator from the NumPy library called **'numpy.random.rand()'**. Although computers cannot generate truly random numbers due to their deterministic nature, the mentioned function uses pseudo-random number generators (PRNGs). The generated sequences can be assumed to be independent for all practical purposes and thus for this implementation as well [30].

For every generator in the system, a state is obtained for each hourly increment by comparing a random number from a uniform distribution to some specified state probability boundaries. The generator state profile is now an array consisting of zeros and ones to indicate whether the generator is down or operational for each hourly increment. If derated states are present, the state probability ranges can be adjusted as shown in Table 2.3. When a generator operates in a derated state, the percentage of the operational installed capacity will be added to the generator state profile. For example, if a derated state operates at 40% of the rated capacity, 0.4 will be added to the generator state profile. By multiplying the generator state profile with the generator capacity, a new array is created that has the output profile of a

generator. Summing the contributions from all generators together results in a summed generation profile for the entire system. Pseudo-code for creating the generator profile can be found below in Algorithm 6.

Algorithm 6: Pseudo-code for obtaining a generation profile for the non-sequential MCS method

Input: Generator data is imported from a datafile, containing the capacity outage levels for each generator *Gen_outage_capacity* and the associated probability of being in that state *Gen_outage_probability*. If wind generation is present in the system, the characteristics of the turbines are loaded from a datafile. The wind speed parameters α and β can be specified in the code

Output: A list with the available generation capacity for each time increment *Summed_Generation_Profile*

1 **Function** `generate_random_values(`*number of time increments*`):`
2 Pre-compute a list of uniformly distributed numbers for each time increment and store in a list U
return U
3 **Function** `generate_wind_speeds(`*number of time increments*, α, β`):`
4 Pre-compute a list of wind speeds by generating a list of uniformly distributed numbers for each time increment and plugging those into Equation 5.1 to obtain *Wind_Speeds_List*
return *Wind_Speeds_List*
5 **Function** `generate_generator_state(`*number of time increments, Generator_Outage_Probability_List ,Generator_Outage_List*`):`
6 Use the function *generate_random_values* to generate a list of random numbers
Create a list with the cumulative probabilities of all generator outage states using *Generator_Outage_Probability_List*
Check for each random number in which cumulative probability interval it belongs to determine the *Generator_Outage* for that time instance
Store the generator **output** as a percentage by using 1 - $(\frac{Generator\_Outage}{max(Generator\_Outage\_List}$)
return *generator_profile_state*
7 **Function** `generation_profile(`*Generator_Data*`)`,α, β, *Turbine_Data*`:`
8 Use the *generate_wind_speeds* and *createPowerProfile* (from Algorithm 5) functions to create a *wind_power_output_profile* for a wind turbine if present in the system.
9 Initialise a list with zeros for each time increment called *Summed_Generation_Profile*
for *each generator* **do**
10 Use the *generate_generator_state* function to generate a list of generator states **if** *The generator is a conventional generator* **then**
11 The *generator_profile_state* is multiplied by the rated generator capacity;
12 **else if** *The generator is a wind turbine* **then**
13 The *generator_profile_state* is multiplied by the *wind_power_output_profile* ;
14 **return** *Summed_Generation_Profile*

A benefit of the state sampling implementation over the state duration method is the fact that the state sampling method can deal with more types of input data. It uses the state probabilities, which can be obtained from the failure and repair rates. It is, however, not possible to convert from state probabilities to failure and repair rates. The relation between the two can be derived, but not their exact values. The state sampling is, therefore, a more versatile method if failure and repair rate data are missing.

5.1.2. Load Curves

The load curves used for the MCS scripts are the same as those used for the analytical method. The chosen load profiles depend on the presented benchmark values, but this again leads to the use of the CYPL and HPL. One benchmark value uses the DPL, but the WPL is still unused. The load increments are taken in chronological order, but this is not required due to the non-sequential nature of the state sampling method. The benchmark values found in the literature [20], [26] do not sample the load curve either, which is why a sequential load curve is used in a non-sequential MCS method.

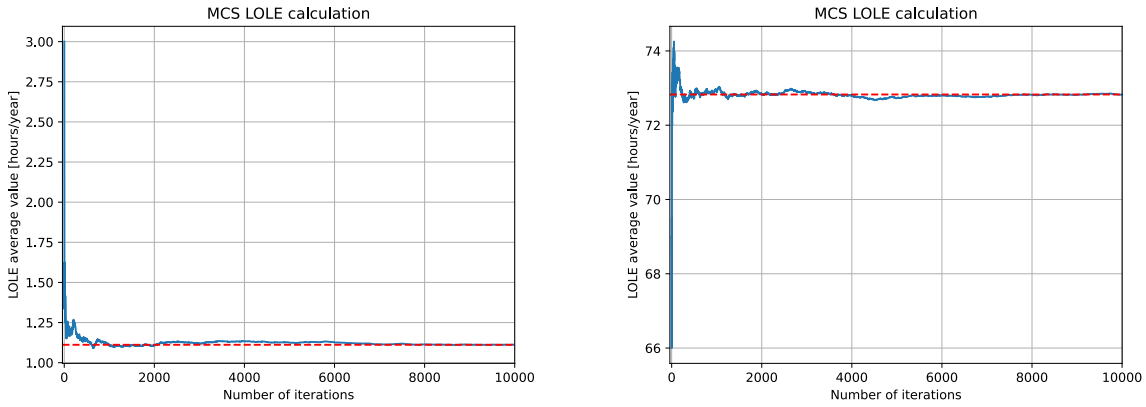
Furthermore, the assumption is made that the load profile doesn't change yearly. One could argue that the load profile changes from year to year, for example due to a change in the temperature profile and the subsequent heating and cooling requirements. Still, the year-to-year variation will be averaged out by taking a large number of simulation years.

5.1.3. Reliability Metrics

The scripts for calculating LOLE and EENS are again based on the theory presented in Section 2.4 and are computed with the functions **calc_LOLE_MCS** and **calc_EENS_MCS**. The derived quantities LOLP' and ENDS could also be computed, but they are not presented here as they provide no additional insight. They are obtained by dividing LOLE and EENS by a constant, such that the relative difference with benchmark values stays the same.

The **calc_LOLE_MCS** function has the generator profile, load profile and stopping criteria as inputs. The load is subtracted from the generation, and the instances with an energy deficit are counted. This process is repeated until the stopping criteria is reached. As mentioned in Section 2.3.4, there are two ways to implement the stopping criteria. For this script, the number of iterations is used as the stopping criterion, as it removes the computational power of calculating the CoV during every iteration. Once the specified number of iterations is reached, the CoV is calculated, and it can be checked whether this value is acceptable.

Since the EENS metric has two variables, energy deficit instances and energy deficit magnitude, compared to the single variable of energy deficit instances for LOLE, the EENS metric takes more iterations to converge. It will, therefore, determine the number of simulation years for an acceptable CoV. After trying multiple simulation years, it could be observed that the EENS value stabilises after 10000 simulation years, with a CoV of 0.0002 for the HPL model. The CoV for LOLE using the HPL model is even lower, with a value of 10^{-6} using 10000 simulation years.
The convergence process for the LOLE value of the RBTS for both an HPL and CYPL model is shown in Figure 5.1, while the LOLE values are shown in Table 5.1 along with MCS benchmark values [26]. The average LOLE value convergence graphs for the IEEE-RTS can be found in Appendix G.1.



(a) average LOLE value for the HPL model

(b) average LOLE value for the CYPL model

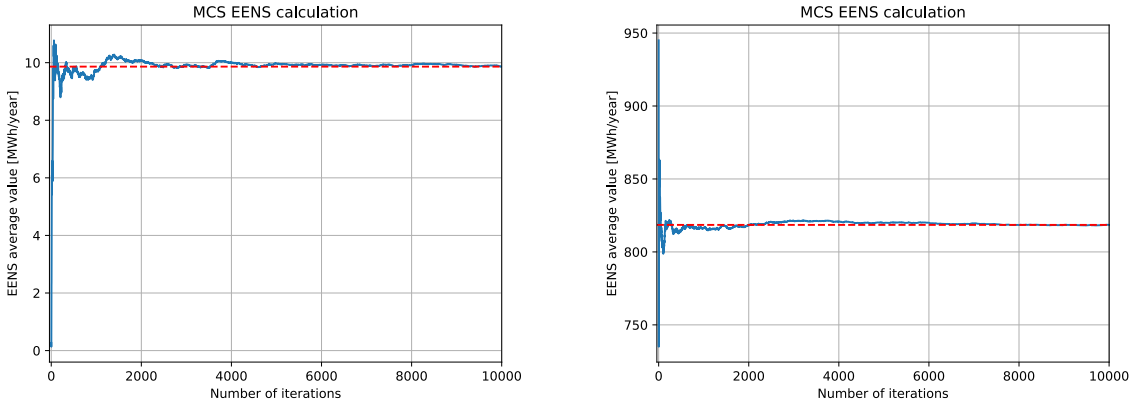
Figure 5.1: Converging process for the average LOLE value

Table 5.1: Comparison between an MCS benchmark [26] and the MCS scripts for LOLE

	LOLE HPL [hours/year]			LOLE CYPL [hours/year]		
Test System	MCS benchmark	MCS script	Δ [%]	MCS benchmark	MCS script	Δ [%]
IEEE RTS-79	9.3868	9.3889	0.022	-	738.4945	-
RBTS	1.0901	1.0945	0.40	72.6183	72.8445	0.31

The **calc_EENS_MCS** function also calculates the net energy, but instead of only counting the energy deficit instances, it sums the magnitudes of these energy deficits together. The resulting "energy not served" value is then stored in a list for each simulation year, which is used for the average EENS value and the CoV.

A graphical representation of how the average EENS value for the RBTS converges can be found in Figure 5.2. Performance metrics of the EENS calculation for the HPL and CYPL models are presented in Table 5.2. The average EENS value convergence graphs for the IEEE-RTS are given in Appendix G.2.



(a) average EENS value for the HPL model

(b) average EENS value for the CYPL model

Figure 5.2: Converging process for the average EENS value

Table 5.2: Comparison between an MCS benchmark [26] and the MCS scripts for EENS

	EENS HPL [MWh/year]			EENS CYPL [MWh/year]		
Test System	MCS benchmark	MCS script	Δ [%]	MCS benchmark	MCS script	Δ [%]
IEEE RTS-79	1192.5072	1179.0790	1.1	-	128402.0322	-
RBTS	9.9268	9.8535	0.74	816.8147	820.983	0.51

To test the script with derated states, a modified version of the RBTS is used [24], where the two 40 MW thermal power plants have a derated state of 20 MW. Reference values for HL1 adequacy for both the RBTS and IEEE RTS are presented in [20]. A comparison with the values obtained from the script can be found in Table 5.3. A sample size of 10000 simulation years has been chosen, as it again gives an acceptable CoV below $2 \cdot 10^{-5}$ for EENS.

Table 5.3: Comparison between the benchmark and the script reliability metric values for an MCS with derated states using the RBTS

Reliability metric	HPL			CYPL		
	MCS benchmark [20]	MCS script	Δ [%]	MCS benchmark [20]	MCS script	Δ [%]
LOLE [hours/year]	0.68	0.7279	6.58	66.97	65.771	1.82
EENS [MWh/year]	6.58	6.3345	3.88	635.68	633.6075	0.33

A modified version of the IEEE RTS is given in the same paper [20]. The 350 and 400 MW generator units are modified to include a 50 % derated state. Making these adjustments results in the LOLE and EENS values in Table 5.5, which are compared to the values in [20].

Table 5.4: Comparison between the benchmark and the script reliability metric values for an MCS with derated states using the IEEE RTS

Reliability metric	HPL			CYPL		
	MCS benchmark [20]	MCS script	Δ [%]	MCS benchmark [20]	MCS script	Δ [%]
LOLE [hours/year]	5.78	5.6773	1.81	663.57	652.9985	1.62
EENS [MWh/year]	648.65	651.7428	0.47	93088.28	93222.141	0.14

In another paper [31], an LOLE value using a DPL profile is given. A comparison of this reference LOLE value and the one from the script is shown in Table 5.5.

Table 5.5: Comparison between the benchmark and the script LOLE value for an MCS with derated states using the IEEE RTS

	MCS benchmark [31]	MCS script	Δ [%]
LOLE [days/year]	0.88258	0.8833	0.08

From Tables 5.1, 5.2, 5.3, 5.4 and 5.5, it can be observed that the reliability metrics from the script match the reference values within a few percent. Due to the nature of MCS, a slightly different value will be obtained each time an MCS is carried out. Repeating the simulations a few times could result in a value closer to the benchmark values, but the obtained values are always within a few percent of the benchmark values. It can be concluded that the scripts work as intended and provide accurate results. The script for computing the reliability metrcis with an MCS method can be found in Appendix C.3. The pseudo-code for the script is given in Algorithm 7.

Algorithm 7: Pseudo-code for obtaining the reliability indices LOLE and EENS for an MCS method

Input: The capacity outage levels for each generator *Gen_outage_capacity* and the associated cumulative probability of being in that state *Gen_outage_probability* are used for obtaining the reliability metrics, as well as the load profile and a stopping criterion

Output: A singular value for either *LOLE* or *EENS*

Function `calc_LOLE`(*Generator_Outage_Capacity, Cumulative_Outage_Probability, Load_Profile, Stopping_Criterion*)**:**
- Initialise empty lists *LOLE_List* and *LOLE_Average_Value_List* to store the LOLE contributions for each iteration and the mean LOLE value after each iteration
  Find the maximum installed capacity *Max_Capacity*
- **while** *number of iterations is smaller than the stopping criterion* **do**
  - Obtain the *Summed_Generation_Profile* from the *Generation_Profile* function
    Initialise the LOLE value *LOLE_value*
    Calculate the net energy by subtracting the *Load* from the *Summed_Generation_Profile*
    Sum together all the instances where the net energy is smaller than 0 and store them in *LOLE_value*
    Store *LOLE_value* in the *LOLE_List* and add the mean value of *LOLE_List* to *LOLE_Average_Value_List*
- Calculate the Coefficient of Variation using Equation 2.12 to check if the chosen number of simulation years is sufficient
- **return** *LOLE*

Function `calc_EENS`(*Generator_Outage_Capacity, Individual_Outage_Probability, Load_Profile, Stopping_Criterion*)**:**
- Initialise empty lists *EENS_List* and *EENS_Average_Value_List* to store the EENS contributions for each iteration and the mean EENS value after each iteration
- **while** *number of iterations is smaller than the stopping criterion* **do**
  - Obtain the *Summed_Generation_Profile* from the *Generation_Profile* function
    Initialise the EENS value *LOLE_value*
    Calculate the net energy by subtracting the *Load* from the *Summed_Generation_Profile*
    Keep all the instances where the net energy is smaller than 0 and sum the remaining *Energy_Not_Served* instances together in *Sum_Energy_Not_Served*
    Store *Sum_Energy_Not_Served* in the *EENS_List* and add the mean value of *EENS_List* to *EENS_Average_Value_List*
- Calculate the Coefficient of Variation using Equation 2.12 to check if the chosen number of simulation years is sufficient
- **return** *EENS*

5.1.4. Comparing the Reliability Metrics for the MCS Method with the Analytical Method

Now that reliability metrics have been obtained for the RBTS and IEEE RTS using the analytical and MCS methods, a comparison can be made between the two. The results for both methods are presented in Tables 5.6 and 5.7 below,

Table 5.6: Comparison between the analytical and MCS scripts for LOLE

	LOLE HPL [hours/year]			LOLE CYPL [hours/year]		
Test System	MCS benchmark	MCS script	Δ [%]	Analytical script	MCS script	Δ [%]
IEEE RTS-79	9.39390	9.3889	0.05	738.874	738.4945	0.05
RBTS	1.0914	1.0945	0.28	72.8723	72.8445	0.04

Table 5.7: Comparison between the analytical and MCS scripts for EENS

	EENS HPL [MWh/year]			EENS CYPL [MWh/year]		
Test System	Analytical script	MCS script	Δ [%]	MCS benchmark	MCS script	Δ [%]
IEEE RTS-79	1176.278	1179.0790	0.24	128364.0	128402.0322	0.03
RBTS	9.8603	9.8535	0.07	821.0000	820.983	0.002

The results show that the analytical script matches the reliability metrics of the non-sequential state sampling MCS script with negligible differences of less than 0.3%. If the test system is well-defined, the two scripts can thus be used interchangeably.

5.2. Monte Carlo Simulation Method with Wind Considerations

Including wind farms in the MCS generation adequacy is not much different from the method without wind farms. The mechanical availability of the wind farm, so not considering a loss of output due to the wind conditions, follows the same procedure. This can be with or without derated states, as Section 5.1.1 explains. The scripts for the load model and the reliability metrics calculations are the same, with the difference being in the power profile of the generators. The generator state profile, which indicates the mechanical availability of the generator, is now multiplied by the power output profile based on the wind speeds.

The first step in creating a power output profile is obtaining 8736 wind speeds to match the IEEE load curve. If the wind characteristics of a specific site can be expressed as a Weibull distribution, Equation 5.1 can be used to find a wind speed. This process can be repeated to create a wind speed array for a year.

$$\text{Wind speed} = \alpha \cdot -ln(\text{U})^{(\frac{1}{\beta})} \tag{5.1}$$

In Equation 5.1, α and β are the Weibull shape and scale parameters, which determine the characteristics of the distribution. The built-in Python function **'weibull_min.fit'** is used to fit a Weibull curve with parameters α and β to a probability distribution of wind speeds. An example of a wind speed probability distribution curve with a Weibull curve estimate can be found in Figure 5.3. It uses data from the website "windatlas.xyz" [29] for a period of 40 years (1980 - 2019) of the coast of Trondheim (latitude: 64°, longitude: 8°).

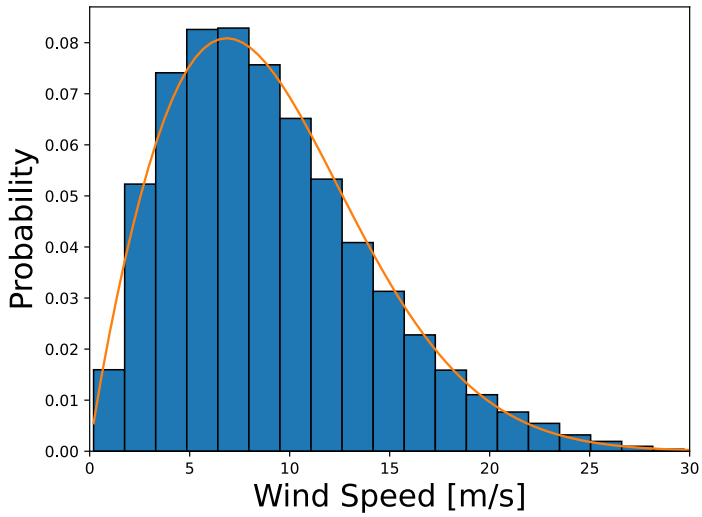


Figure 5.3: Weibull distribution fitting to a wind speed probability distribution with data from [29]

The conversion from wind speed to power output is again done using the **CreatePowerProfile** function. In the case of a singular wind farm, the wind conditions are the same for each turbine if wake effects are neglected. Therefore, the same sampled wind profile will be used for all turbines. The system will be benchmarked against a system with known reliability metrics to verify that the script works as intended.

The system of choice will be the RBTS with an additional 22.5 MW of wind turbines, presented in [12]. It uses wind data from a weather station near North Battleford. The exact station is not specified, but the "North Battleford A" station data from Environment Canada is used in this thesis [32]. The paper specifies that hourly wind data from 1991 to 1993 is used, as well as the average and standard deviation of the wind speed over a longer period. The wind speeds in the paper are modelled using an Auto-regressive Moving-Average (ARMA) model, but the Weibull distribution used in this thesis should yield similar results. The added wind generation capacity consists of 100 turbines with a rating of 225 kW each. The reliability data for the wind turbines in [32] is only given as the FOR, so the state sampling method is used. Doing so results in the reliability metrics shown in Table 5.8, where they are compared to the reference values presented in [12].

Table 5.8: Comparison between the reliability metrics in [12] and the script for the RBTS with added wind generation

	MCS benchmark [12]	MCS script	Δ [%]
LOLE [days/year]	0.7895	0.9076	15.0
EENS [MWh/year]	7.3572	7.7681	5.6

From Table 5.8, it can be observed that there is some difference between the reliability metrics. The most likely reason for this is the wind profile that is used for each respective method. The average wind speed of the data used in this thesis is 13.94 km/h, while the exact average wind speed value in [12] is not given. However, a graph is presented in the paper with the observed and simulated wind speeds for each month in a year. It isn't easy to obtain an exact average from Figure 5.4, but it seems to be slightly above 14 km/h. Higher wind speeds result in larger power outputs and, thus, potentially fewer loss of load instances, with lower deficits for each instance.

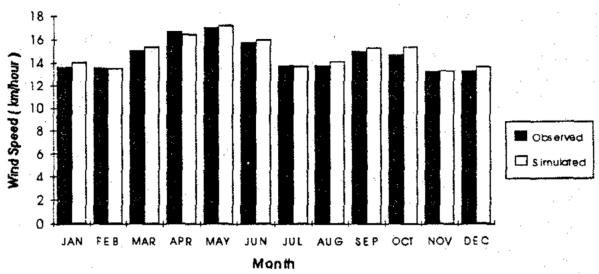


Figure 5.4: Observed and simulated wind speed at North Battleford, reproduced from [12]

Furthermore, the assumption that the wind speeds follow a Weibull distribution doesn't fully hold in this case. The Weibull fitting shown in Figure 5.5 has some outliers in the wind speeds, contributing to the differences in the reliability metrics between the benchmark values and the script.

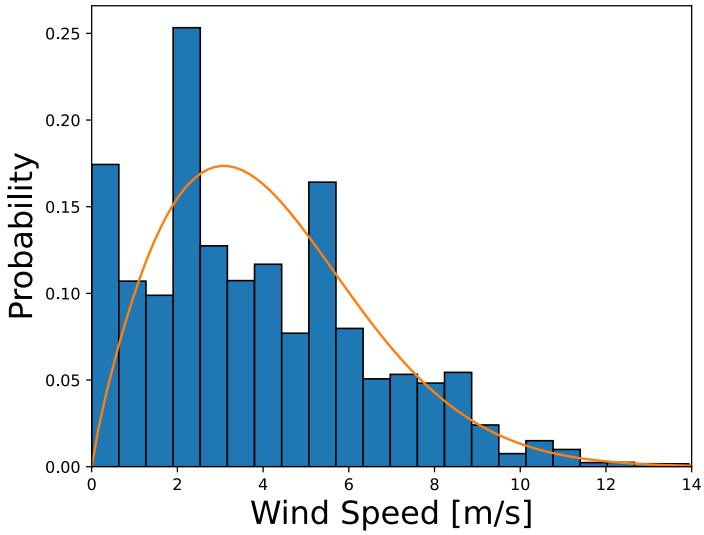


Figure 5.5: Weibull distribution fitting to a wind speed probability distribution with data from [32]

Apart from reliability metrics for the RBTS with wind generation, the metrics for the RBTS with an additional 22.5 MW of conventional generation are given, as well as the RBTS metrics. These do not depend on the wind speed and should thus provide the same results. An overview of the results from [12], the scripts presented in this thesis and a recent master thesis with benchmark values [9] is given in Table 5.9.

Table 5.9: Comparison between the reliability metrics in [12], [9] and the scripts for the RBTS and RBTS with additional generation

		MCS [12]	MCS [9]	MCS script	Δ [12] [%]	Δ [9] [%]
RBTS	LOLE [days/year]	1.1282	1.0899	1.1157	1.12	2.31
	EENS [MWh/year]	10.3109	9.8260	10.0392	2.71	2.12
RBTS + 22.5MW convential generation	LOLE [days/year]	0.099	-	0.1007	1.69	-
	EENS [MWh/year]	0.8251	-	0.8342	1.09	-

The results in Table 5.9 show that the script gives correct results. The differences in Table 5.8 are thus due to the addition of wind generation to the system. As the procedure for converting the wind speed to a power output has been verified in Section 4.2.2, it seems like the difference in wind speed data is responsible for the observed differences.

6
Examining the Impact of Outages on the Output of a Wind Farm

A useful insight for the energy sector could be looking into the effect a turbine's FOR has on a wind farm's power output. It can help wind farm planners calculate the expected energy production, and wind turbine manufacturers can decide whether it makes sense to focus time and money on improving the FOR. To gain more insight into the relation between the turbine FOR and wind farm power output, a closer look will be taken at the distribution of the power output of a wind farm and the yearly energy production. Both of these metrics will be obtained and analysed in Sections 6.1.1 and 6.1.2 below. A case study will also be carried out for the city of Trondheim to see the impact of outages on the yearly energy production.

6.1. Investigating the Effect of Different FORs on the Output of Wind Farms CDFs

The power output CDF and yearly energy production will be obtained using a modified version of the MCS generation model, presented in Section 5.1.1. The state sampling method will be used to obtain a summed generation model. The data presented in Table 6.1 will be used in the analysis in this section.

Table 6.1: Data used for the analysis on the effect of FOR on the power output of wind farms

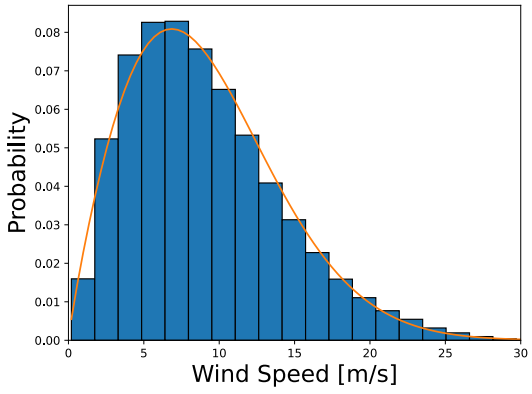
Number of simulation years	25 years
Number of turbines	10
Rated power turbine	10 MW
Cut-in wind speed	4 m/s
Rated wind speed	11.4 m/s
Cut-out wind speed	25 m/s
Wind farm location	latitude 64° longitude 8°
Wind speed parameters	α = 10.2239
	β = 1.8865

Some further explanation about the data in Table 6.1 will be given below. The number of simulation years, 25, is selected because modern wind farms are designed to operate for this period.[33] The wind turbine data is from the DTU 10 MW reference turbine [28], and the wind speeds at the specified coordinates are from the xyz windatlas [29], based on 40 years of data between 1980 and 2019.

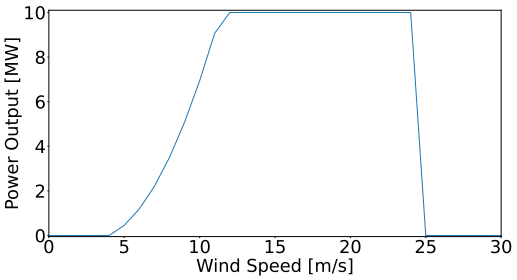
6.1.1. Power Output CDF

One way to look at the difference that a turbine's FOR can make on a wind farm's power output is to consider the distribution of the power outputs within a year. There are two main factors influencing the power output of a wind farm: the fluctuations in the output due to varying wind speed and the 'mechanical' reliability of wind turbines, causing a loss of power if turbines are unavailable.

When perfect reliability is considered, the variation in wind speed becomes the only variable influencing the power output. The exact distribution of wind speeds can vary between sites, but a Weibull distribution can generally characterise wind speeds [34]. This distribution can also be observed in Figure 6.1a, where a Weibull distribution has been fitted to the wind speed distribution. The given wind speed distribution is used as an input for the power curve formula in Equation 4.1, and the resulting power output probability for a singular turbine is presented in Figure 6.1b.



(a) Weibull distribution fitting to a wind speed probability distribution with data from [29]



(b) Power output probability distribution

Figure 6.1: Wind speed and power output data for the DTU 10 MW reference turbine of the coast of Trondheim

It can be observed that the two largest probabilities are at the minimum and maximum power output of the wind turbine, with an almost uniform distribution of the power between 0 MW and the rated power. Within this range, the probability of being in the lower half of the 0 to 10 MW is higher than the upper half because the wind speed is often closer to the cut-in wind speed than the rated wind speed. A power distribution function with perfect reliability would thus have a vertical probability increase at 0 MW and 10 MW and a continuous but slowly decreasing probability between these two values.

Introducing mechanical failures to the turbines will reduce the probability of higher power outputs, as the probability of having none or a few turbines available is way more significant than more than 50% of the turbines. The lower power outputs are thus less affected than the higher outputs, and the CDF curves will shift up for bigger FORs. On top of that, vertical jumps in the CDF plot will appear at the turbine power increments. Previously, all wind speeds above the rated value would result in the wind farm operating at rated power. With unavailable turbines, fractions of this probability will be allocated to power levels at intervals equal to the rated power of a singular turbine. In this case, the wind speed is still high enough for the turbines to operate at rated power, but the wind farm production would be capped at 80 or 90 MW if one or two turbines are unavailable. This effect does not significantly impact the lower power output range of the wind farm, as the probability of having a lot of turbines unavailable with realistic FORs is small. An example of the probability associated with having some turbines unavailable for different FORs is given in Table 6.2, while the CDF power output plot for various turbine FORs can be found below in Figure 6.2.

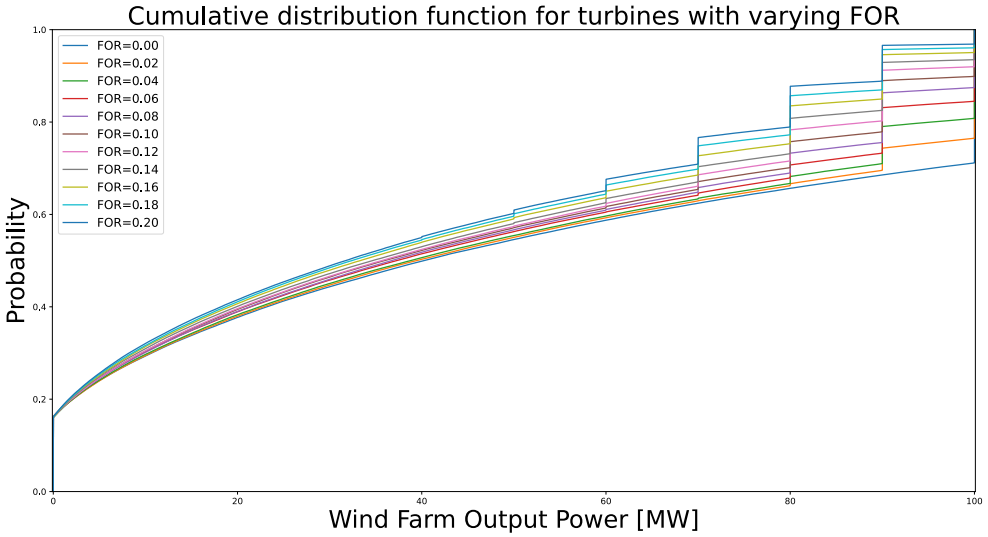


Figure 6.2: CDF for a 100 MW wind farm with 10x 10 MW turbines with different FORs

The graph shows that the probability distributions for the different FORs go down in steps from the rated wind farm output of 100 MW before they converge the closer they get to zero. When looking at the distribution for a wind farm with perfect reliability, i.e. FOR = 0.00, it can be seen that the curve is smooth between 0 and 10 MW, with jumps at 0 and 10 MW if the wind speed is below the cut-in or above the rated wind speed. This curve thus follows the wind speed distribution.

Any non-zero FOR will lead to jumps in the CDF graphs between 0 and 10 MW. This stepwise behaviour can be explained by looking at the probability tables for the number of turbines out of service. The probabilities of having a certain number of turbines unavailable for a FOR of both 0.02 and 0.10 are given in Table 6.2. Probabilities smaller than 10^{-5} have been rounded to zero.

Table 6.2: Individual probabilities of having a certain number of turbines available for a FOR of 0.02 and 0.10

Number of unavailable turbines	Individual probability	
	FOR = 0.02	FOR = 0.10
0	0.8171	0.3487
1	0.1668	0.3874
2	0.0153	0.1937
3	0.0008	0.0574
4	$2.98 \cdot 10^{-5}$	0.0112
5	0	0.0015
6	0	0.0001
7	0	0
8	0	0
9	0	0
10	0	0

From Table 6.2, it can be concluded that the probability of having none or a few turbines out of service is way higher than having more than half of them unavailable. The jumps in the CDFs in Figure 6.2 happen at every 10 MW increment, which is the capacity of one turbine. A higher FOR means having a lower probability of all turbines being available, reducing the probability of having a 100 MW output. It also results in the curves being above the CDF of a wind farm with lower FORs, as the probability of having a higher output is smaller. Another observation is that a higher FOR results in more steps in the CDF. This is because a higher FOR results in a higher probability of having more unavailable turbines.

Table 6.2 illustrates this phenomenon, where the probability of having x turbines available for a FOR of 0.10 is significant enough to show up in the curve. At the same time, a step will barely be visible in the CDF for a FOR of 0.02.

Furthermore, it can be noted that the distributions all overlap at a power output of 0 MW. This is because wind speeds below the cut-in wind speed are responsible for an output of 0 MW, irrespective of the FOR of the turbines in the wind farm.

6.1.2. Yearly Energy Production

Another metric that can contribute to gaining insight into the difference that the FOR of a turbine can make is the yearly energy output. It can be calculated by summing together all the hourly power increments in the summed generation profile. Repeating this process 25 times for the lifespan of a wind farm and taking the average value results in the yearly energy production. The absolute value of the annual energy production of the wind farm and the energy production as a percentage of the production with perfect reliability are presented in Table 6.3 below.

Table 6.3: Yearly energy production of a 100 MW wind farm with 10x 10 MW turbines for different FORs

FOR	Yearly Energy Production	
	[MWh]	[%]
0.00	424204	100.00
0.02	416281	98.13
0.04	407283	96.01
0.06	400132	94.33
0.08	389887	91.91
0.10	381420	89.91
0.12	372538	87.82
0.14	365036	86.05
0.16	355658	83.84
0.18	348098	82.06
0.20	338210	79.73

From Table 6.3, it is clear that there is a relation between the yearly energy production of the wind farm and the FOR of the individual turbines. The FOR of the turbine corresponds to the percentage of energy lost compared to the case with perfect reliability. It does seem to be the case that the percentage of lost energy becomes marginally larger than the FOR for the higher FORs, which could be due to the rapid decline in the probability of having all turbines available for higher FORs. This means that a larger fraction of the rated wind farm power output is taken, resulting in an energy production percentage decrease greater than the FOR percentage.

6.2. Case Study - Supplying the City of Trondheim with Power from an Offshore Wind Farm

One of the ways to model a wind farm similarly to a conventional, controllable generation unit would be to assume that a wind farm can deliver a constant power output and that a storage solution would take care of the fluctuations in the power production. The correct sizing and feasibility of such a storage solution are out of the scope of this thesis, so for now, the assumption is made that the storage is adequate.

The script introduced in Section 6.1 will now be used to find the required amount of wind turbines in a wind farm to power an area. This will be done by matching the yearly energy production of the wind farm to the energy consumption of the selected area for various turbine FORs. The area of choice is the city of Trondheim, where the main NTNU university campuses are located. The wind farm will be located off the coast of Trondheim, as shown on the map in Figure 6.3.

Figure 6.3: Location of the wind farm off the coast of Trondheim at 64°N, 8°E

The wind speeds at this location are obtained from the website "windatlas.xyz" [29]. As for the turbines, the 10 MW DTU reference turbine will be chosen [28]. In contrast to actual turbines from manufacturers, the hypothetical DTU reference turbine is well defined.

According to Our World in Data [35], Norway's electricity consumption per capita in 2022 was equal to 28,095 kWh. The city of Trondheim has a population of 214,565 [36], resulting in an energy consumption of 6.028 TWh. This would equate to a constant power requirement of 6.028TWh/8760 = 688MW. This is again under the assumption that a storage solution, which could be batteries, hydrogen or hydro reservoirs, filters out the fluctuations in the power production. If the wind farm would operate at rated capacity with perfect reliability throughout the year, 688/10 = 69 turbines are needed. However, the wind speeds are not always high enough to operate at rated capacity, and actual turbines have failures. The actual energy production as a percentage of the rated energy production is defined as the capacity factor, which can range between 40 and 60% for offshore wind farms. This percentage depends on the wind site conditions and turbine characteristics, but the number of turbines needed to supply Trondheim, assuming perfect reliability, would at least be double. An overview of the minimum amount of turbines needed to match the yearly energy demand of the city of Trondheim for different turbine FORs can be found below in Table 6.4.

Table 6.4: Minimum number of turbines needed to match the yearly energy demand of the city of Trondheim for different turbine FORs

FOR	Yearly Energy Production [TWh]	Number of Turbines
0.00	6.063	143
0.02	6.073	146
0.04	6.064	149
0.06	6.048	152
0.08	6.072	155
0.10	6.078	159
0.12	6.063	162
0.14	6.078	167
0.16	6.049	170
0.18	6.039	174
0.20	6.034	178

From Table 6.4, an approximately linear relation between the FOR and the number of turbines can be observed, especially for a FOR below 0.10, where a 0.02 increment in the FOR requires three additional turbines to satisfy Trondheim's energy consumption. There is still a linear relation for higher FOR values, but the number of additional turbines for a 0.02 FOR increase seems to move closer to 4. An increase in the FOR leads to a more than linear decrease in energy production. This behaviour can be observed in Figure 6.4 and is responsible for increasing the additional required turbines for bigger FOR values.

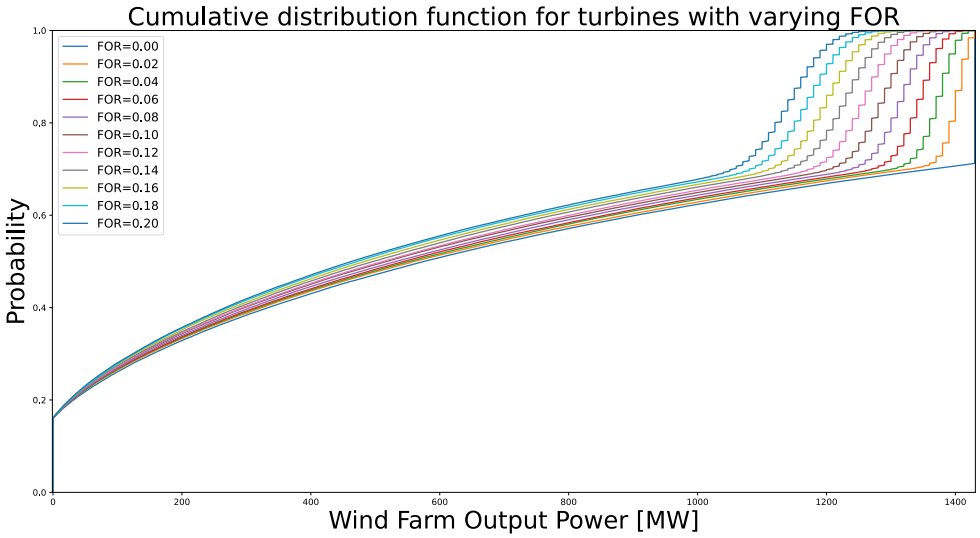


Figure 6.4: CDF for a 1.43 GW wind farm with 143x 10 MW turbines with different FORs

7
Conclusions and Future Work

7.1. Conclusions

This work contributes to building a framework to assess the adequacy of a future grid with offshore wind farms. The scope of the thesis has been limited to developing and validating open-source scripts for generation adequacy analysis and examining the impact of outages on the output of a wind farm. Two methods have been implemented: an analytical method and an MCS method. Both of these methods can analyse a conventional generation-based system, a wind generation-based system or a combination of the two systems. The outputs of these scripts are the reliability indices LOLE and EENS.

The validation of the scripts has been done using two test systems: the RBTS [24] and IEEE RTS [31], including their variants with derated states. Other test systems that were used are a standalone 20 MW wind farm [13] and the RBTS system with an additional 22.5 MW MW of wind generation, consisting of 100 turbines with a 225 kW rating [12].

The analytical method script is constructed using a recursive algorithm [7] to obtain a COPT as the generation model. The IEEE load curve is chosen for the load model. These two models are combined to obtain LOLE and EENS for both test systems using a CYPL and HPL. The resulting metrics are presented in Tables 4.1 and 4.2, and are compared to two benchmark values: a paper from Roy Billinton [26] and a recent master thesis [8]. It can be observed that the values from the scripts developed in this thesis match the results in [8] one to one, while there are marginal differences of at most 0.5% when comparing the results to the ones in [26]. A possible reason for the difference could be the rounding of intermediate results and the software used back in 2006.

Wind can be added to the analytical method by converting a wind speed profile to a power output profile using Equation 4.1. A state probability table with a select number of states can be constructed by obtaining the PDF of the power output. The state probabilities due to fluctuations in the wind speed are then combined with the state probabilities due to the mechanical reliability with Equation 4.5. The resulting COPT could have many states, which can be reduced using the apportioning method. The resulting COPT can be used to assess a system consisting solely of wind generation or to create a generation unit with derated states to be added to a system with conventional generation units. The script has been tested using data from [13], and the resulting apportioned COPT is identical to the one presented in the paper.

The chosen MCS method uses non-sequential state sampling to obtain a generation profile. A uniform random number is generated for each hourly time increment, and the state associated with the probability interval where the number falls between is selected. The resulting array contains values between 0 and 1 to indicate the percentage of installed capacity available during that period. Combining the generation and load profiles again results in reliability indices. With an MCS, this procedure has to be repeated for multiple years for the simulation to converge towards the true mean value. A stopping criteria, given in Equation 2.12, can be used to determine whether the chosen number of simulations has an acceptable variance. The number of years used for these scripts is 10000, and the resulting reliability metrics can be found in Tables 5.1 and 5.2.

The results are compared to MCS benchmark values [26], and it can be observed that the scripts closely match the benchmark values, with less than 0.5% of a difference for the LOLE values. The EENS values are also close to the benchmark values, with the differences ranging between 0.5% and 1.1%. The larger error margin for EENS compared to LOLE is due to the difference in the degrees of freedom, as the number of energy deficit instances is combined with the magnitude of the energy deficit for EENS. LOLE, on the other hand, only considers the number of energy deficit instances and has one source of discrepancy compared to the two for EENS.

Derated states are added to the scripts by including additional state probability ranges for each generation unit. From the results in Tables 5.3 and 5.5, it can be seen that the values match closely, except for the HPL model for the RBTS.
Wind considerations are added to the MCS by sampling wind speeds from a Weibull distribution using Equation 5.1. The Weibull shape and scale parameters, α and β, are obtained by fitting a Weibull distribution to a wind speed probability distribution. Converting the wind speeds to a power output is again done with Equation 4.1, and the resulting output power is combined with a sampled state based on the state probability profile of the turbine. The result is added to the generation profile, and reliability metrics can be obtained. These can be found in Table 5.8, where the results of the scripts are compared to MCS benchmark values [12]. It can be seen that the metrics are off by quite a bit, with a 15% difference for the LOLE value. It can, however, be noted that both metrics show the same pattern of being less reliable than the benchmark. The most probable factor contributing to the differences is the uncertainty of the exact wind data used in [12]. Based on Figure 5.4, it seems like the average wind speed for the paper is higher, resulting in higher power outputs and smaller values for LOLE and EENS. Another reason could be the implemented wind speed model. An ARMA model is used for the benchmark results, while this thesis uses a Weibull distribution.

Additional results from [9] and [12] are used to verify that the discrepancy in the results is solely due to a difference in the wind data. Reliability metrics for the RBTS are given, as well as the RBTS system with an additional 22.5 MW of conventional generation. The results are within 1% to 3% of the benchmark values, so the differences are due to the inclusion of wind generation. As the procedure to convert from wind speeds to power output has been tested in Section 4.2.2, the difference in the wind speed data seems responsible for the mismatch in results.

The final test for the adequacy scripts compares the output of the analytical and MCS scripts. The results are given in Tables 5.6 and 5.7. It can be observed that the difference in results between the two methods is negligible, with most differences under 0.05% and the largest differences being 0.28%.

When considering the effect of outage on the output of a wind farm, a closer look has been taken at the impact of the FOR of a turbine on the wind farm power output by plotting the CDF of the power output for various FORs in Figure 6.2. A stepwise pattern has been observed towards the upper end of the power output due to the mechanical reliability probabilities superimposed on the wind power output.

Furthermore, the yearly energy production averaged over the 25-year lifespan of a wind farm has been computed in Table 6.3, from which it can be seen that the percentage of lost energy production compared to the production with perfect reliability closely follows the FOR of the individual turbines. The production loss percentage seems to increase faster than the increase in FOR, as a higher FOR takes away a larger fraction of the highest power output range.

The script for calculating the yearly energy production was used in a case study to see how many turbines would be needed to power the city of Trondheim. The wind farm is assumed to behave like a conventional generating unit with a constant power output throughout the year. For this simplification to hold, the assumption is made that a storage solution deals with fluctuations in the power output. From Table 6.4, it can be concluded that a linear relation exists between the FOR and the number of turbines needed to supply energy to Trondheim. For turbine outage rates above 0.10, it can be observed that the linear coefficient slowly increases. This is due to a more-than-linear decrease in probability for the highest wind farm power output for higher FORs.

7.2. Future Work

The scope of this thesis only represents a small part of the previously mentioned framework. Multiple aspects can be expanded on in future work. These will be elaborated on in the sections below.

7.2.1. Extending the Generation Adequacy to Composite Adequacy Analysis

A logical next step towards building a framework for adequacy analysis would be to include the transmission system in the scripts. The assumption that there are no limitations to the transmission network with perfect reliability, the so-called copper plate assumption, is not applicable in the real world. Including the reliability and limitations of the transmission network would consider the load flows and failure rates of lines and transformers.

7.2.2. Improving the Sequential State Duration MCS Method

A sequential MCS allows gathering information about the frequency and duration of events within the system that cannot be obtained from a non-sequential method like the state sampling method. A start has been made with implementing the sequential MCS method, but two main limitations prevent the scripts from getting results that match the benchmark values.

One of the main reasons is that the generation profile of each simulation year starts in the **UP** state, and the time overshoot from the previous year is currently not being carried over. This could lead to a system that appears to be more reliable than it is, which is observed in the simulation results in Tables A.1 and A.2. This can be solved by keeping track of the amount of overshoot at the end of a year and the state of this overshoot.

Another factor that could be addressed is the rounding of TTF and TTR durations. The generation profile array is created using 8736 hourly increments. The calculated state durations contain a decimal number of hours behind the dot, but these are omitted to fit within the integer number of hours in a year. This shortens the duration of both the TTF and the TTR. In the test systems used in this thesis, the MTTF is in the order of 10^3, while that of the MTTR is in the order of 10^1, so the rounding will have a bigger effect on the TTR. Getting rid of up to 1 hour on that scale for the TTR will reduce the repair time, and the system will thus be down for a shorter period. Changing this would lead to longer operational and repair times but would influence the repair rate more, resulting in worse reliability. This would bring the results closer to the benchmark values as well.

7.2.3. Compare Methods to Integrate Offshore Wind Farms into the Existing Grid

The optimal way to integrate wind farms into the existing energy mix could vary based on its purpose. Most wind farms are currently connected to shore by a single connection. It is also possible to interconnect countries through a wind farm, with an example being the planned Lionlink connection between the Netherlands and the UK [37].

It is also possible to interconnect multiple wind farms with each other and even connect them to multiple points onshore. Doing so would create a grid at sea that extends the existing grid on land. Plans for such an offshore grid have already been made, with an example of an offshore grid for the North Sea presented below in Figure 7.1.

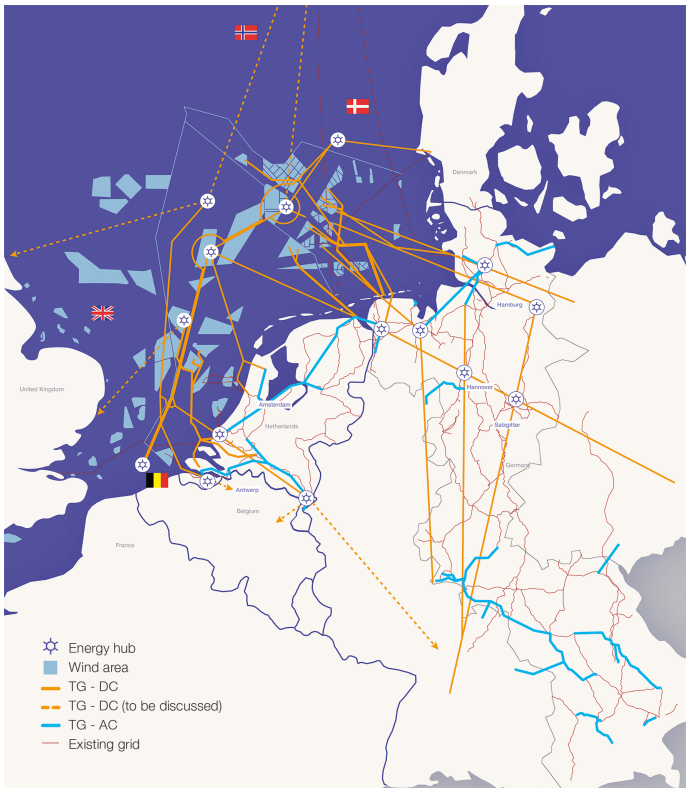


Figure 7.1: Tennet's target grid map for the North Sea in 2045, reproduced from [38]

Another possibility would be to add storage options to the offshore wind farms. The main options for storage are batteries or using electrolysis to produce hydrogen. Both options could be used to control and smooth out the power sent to the onshore grid, but the conversion from electricity to hydrogen and back will have significant losses. That's why including pipelines to shore for hydrogen could be a good idea, which eliminates the electrical losses for transporting the electricity to an onshore electrolyser and can be used in industry processes that cannot be electrified.

An example of integrating storage with an offshore wind farm is the energy islands Denmark plans to build [39], where hydrogen is produced offshore.
The mentioned possibilities could be considered, and the optimal one can be selected for each country or continent.

References

[1] Zephyros, *Unitech Zefyros (Hywind Demo) [Online]*, Available: https://zephyros.no/sites/, 2024.

[2] United Nations Framework Convention on Climate Change (UNFCCC), *Paris Agreement*, Available: https://unfccc.int/sites/default/files/english_paris_agreement.pdf, 2015.

[3] Electricity Maps, *Live electricity map*, Available: https://app.electricitymaps.com/map, 2024.

[4] Statkraft, *Hydropower*, Available: https://www.statkraft.com/what-we-do/hydropower/, 2024.

[5] Ocean Grid Project, *New grid solutions for profitable offshore wind development*, Available: https://oceangridproject.no/, 2024.

[6] Ocean Grid Project, *Ocean Grid Research (SP5)*, Available: https://oceangridproject.no/research/ocean-grid-research, 2024.

[7] R. Billinton and R. N. Allan, *Reliability Evaluation of Power Systems*, 2nd ed. New York and London: Plenum Press, 1996.

[8] K. Koldingsnes, "Reliability-based Derating Approach for Interconnectors," Department of Electric Power Engineering, IEL, M.S. thesis, NTNU, Trondheim, Norway, Jun. 2017.

[9] Ø. S. Laengen, "Application of Monte Carlo Simulation to Power System Adequacy Assessment," Department of Electric Power Engineering, IEL, M.S. thesis, NTNU, Trondheim, Norway, Jun. 2018.

[10] R. Billinton and W. Li, *Reliability Assessment of Electric Power Systems Using Monte Carlo Methods*. Boston, MA: Springer US, 1994. [Online]. Available: http://link.springer.com/10.1007/978-1-4899-1346-3.

[11] M. Bjørkeland, "Generation System Adequacy Studies in the Presence of Wind Energy Resources," Department of Electric Power Engineering, IEL, M.S. thesis, NTNU, Trondheim, Norway, Feb. 2018.

[12] R. Billinton, H. Chen, and R. Ghajar, "A sequential simulation technique for adequacy evaluation of generating systems including wind energy," *IEEE Transactions on Energy Conversion*, vol. 11, no. 4, pp. 728–734, Dec. 1996. [Online]. Available: http://ieeexplore.ieee.org/document/556371/.

[13] R. Billinton and Y. Gao, "Multistate Wind Energy Conversion System Models for Adequacy Assessment of Generating Systems Incorporating Wind Energy," *IEEE Transactions on Energy Conversion*, vol. 23, no. 1, pp. 163–170, Mar. 2008. [Online]. Available: http://ieeexplore.ieee.org/document/4453976/.

[14] R. Allan and F. Takieddine, "Generator-maintenance scheduling using simplified frequency-and duration-reliability criteria," *Proceedings of the Institution of Electrical Engineers*, vol. 124, no. 10, p. 873, 1977. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/piee.1977.0191.

[15] Free Software Foundation, *GNU General Public License, Version 3*, Available: https://www.gnu.org/licenses/gpl-3.0.txt, Jun. 2007.

[16] B. W. Tuinema, J. L. Rueda Torres, A. I. Stefanov, F. M. Gonzalez-Longatt, and M. A. M. M. van der Meijden, *Probabilistic Reliability Analysis of Power Systems: A Student's Introduction*. Cham: Springer, 2020.

[17] R. Billinton and R. Allan, "Power-system reliability in perspective," *Electronics and Power*, vol. 30, no. 3, p. 231, 1984. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/ep.1984.0118.

[18] IkamusumeFan, *PDF of the uniform probability distribution using the maximum convention at the transition points*, Wikimedia Commons, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Continuous_uniform_distribution#/media/File:Uniform_Distribution_PDF_SVG.svg.

[19] N. Thomopoulos, *Essentials of Monte Carlo Simulation: Statistical Methods for Building Simulation Models*. Nov. 2013, pp. 1–171.

[20] M. Bhuiyan, "Modelling multistate problems in sequential simulation of power system reliability studies," *IEE Proceedings - Generation, Transmission and Distribution*, vol. 142, no. 4, p. 343, 1995. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/ip-gtd_19951871.

[21] R. Billinton and W. Li, "A system state transition sampling method for composite system reliability evaluation," *IEEE Transactions on Power Systems*, vol. 8, no. 3, pp. 761–770, 1993.

[22] W. L. Dunn and J. K. Shultis, *Exploring Monte Carlo Methods*. Amsterdam, The Netherlands: Elsevier/Academic Press, 2012.

[23] Probability Methods Subcommittee, "IEEE Reliability Test System," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-98, no. 6, pp. 2047–2054, 1979.

[24] R. Billinton, S. Kumar, N. Chowdhury, *et al.*, "A reliability test system for educational purposes-basic data," *IEEE Transactions on Power Systems*, vol. 4, no. 3, pp. 1238–1244, 1989.

[25] S. Sulaeman, M. Benidris, J. Mitra, and C. Singh, "A Wind Farm Reliability Model Considering Both Wind Variability and Turbine Forced Outages," *IEEE Transactions on Sustainable Energy*, vol. 8, no. 2, pp. 629–637, Apr. 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7579211/.

[26] R. Billinton and D. Huang, "Basic Concepts in Generating Capacity Adequacy Evaluation," in *2006 International Conference on Probabilistic Methods Applied to Power Systems*, Stockholm, Sweden: IEEE, Jun. 2006, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/4202394/.

[27] P. Giorsetto and K. F. Utsurogi, "Development of a New Procedure for Reliability Modeling of Wind Turbine Generators," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-102, no. 1, pp. 134–143, 1983.

[28] C. Bak *et al.*, "The DTU 10-MW Reference Wind Turbine," Tech. Rep., 2013.

[29] WindAtlas.xyz, *Wind data for 2019 of the coast of trondheim*, Windsite data: lattitude: '64.0', longitude: '8.0', height: '100', date from: '2019-01-01', date to: '2019-12-31', 2019. [Online]. Available: https://windatlas.xyz.

[30] NumPy Developers. "NumPy documentation: Numpy.random." (2024), [Online]. Available: https://numpy.org/devdocs/reference/random/index.html.

[31] R. N. Allan, R. Billinton, and N. M. K. Abdel-Gawad, "The IEEE Reliability Test System - Extensions to and Evaluation of the Generating System," *IEEE Transactions on Power Systems*, vol. 1, no. 4, pp. 1–7, 1986.

[32] ClimateData.ca, *Historical wind data from the North Battleford A weather station*, 2024. [Online]. Available: https://climatedata.ca/download/#station-download.

[33] NextEra Energy, *Fact sheet: What happens at the end of a wind farm's useful life?* Available: https://www.nexteraenergyresources.com/pdf/NEER-Decommissioning-FactSheet.pdf, 2024.

[34] H. Shi, Z. Dong, N. Xiao, and Q. Huang, "Wind Speed Distributions Used in Wind Energy Assessment: A Review," *Frontiers in Energy Research*, vol. 9, Nov. 2021.

[35] Ember, Energy Institute, and Our World in Data, *Total electricity generation per person – Ember and Energy Institute*, Available: https://ourworldindata.org/grapher/per-capita-electricity-generation, Statistical Review of World Energy (2023); Population based on various sources (2023) – with major processing by Our World in Data. Dataset retrieved May 15, 2024, 2024.

[36] Trondheim Municipality, *Population statistics*, Available: https://www-trondheim-kommune-no.translate.goog/aktuelt/om-kommunen/statistikk/befolkningsstatistikk/?_x_tr_sl=auto&_x_tr_tl=en&_x_tr_hl=no, 2024.

[37] TenneT, *Lionlink*, Available: https://www.tennet.eu/lionlink, 2024.

[38] TenneT, *Target grid*, Available: https://www.tennet.eu/target-grid, 2024.

[39] The Danish Energy Agency, *Denmark's Energy Islands*, Available: https://ens.dk/en/our-responsibilities/offshore-wind-power/denmarks-energy-islands, 2024.

A
Methodological Approach for Constructing the Sequential MCS Scripts

The scripts that are used to obtain reliability indices using a sequential MCS method will be elaborated on below. The load curves and reliability metrics scripts will be the same as for the non-sequential MCS method and can be found in Appendices B.2 and C.3, while the generation profile can be obtained using two scripts. One has the option to add derated states, while the other does not to reduce the computation time. They can be found in Appendices D.1 and D.2.

A.1. Generation Profile

The state duration method has been chosen as the sequential MCS method, as the procedure to obtain an availability profile is much more straightforward than the state transition method, the other sequential MCS method. It uses a generator's failure and repair rates and the generator state profile again matches the duration of the load curve with 8736 hourly generation increments. The duration of the alternating **UP** and **DOWN** state is calculated using Equation 2.6. The MTTF and MTTR are either based on previous reliability data of generators or an educated guess is taken based on similar generators. The generated random numbers use the same **'numpy.random.rand()'** function as the state sampling method.
Every generator in the system is assumed to be operational at the start of the first year. Operational and repair states alternate until a generation profile for a full year is reached. To ensure that the generation profile doesn't exceed the 8736 hours in the IEEE load curve, it is checked whether the sum of the hours in the generation state profile is below the total hours before adding a new state.
When adding a new state, one final check is carried out to ensure that the addition of this new state won't exceed the total hours. This is done by taking the smaller value out of the new state's added time and the difference between the total hours and the sum of hours in the generation profile. The duration of the latest added state that is not making it into the generation profile will be stored and transferred over to the next iteration of the MCS.

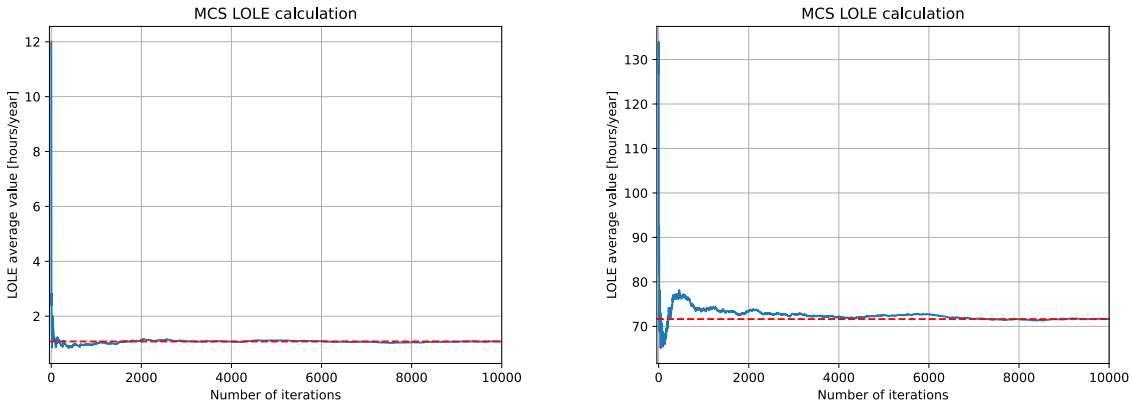
The generator state profile has the same format as the state sampling method. The generator contributions can be combined to obtain a summed generation profile for the entire system.

When a generator has derated states, one cannot simply assume that **UP** and **DOWN** states follow each other. At the start of the first simulation year, the generators are still assumed to be in the **UP** state. The failure rates from the **UP** state to the **DERATED** state(s) and **DOWN** state are used to calculate a TTF using Equation 2.6. The state with the smallest duration out of all the computed TTFs is chosen as the next state. The generator will then return to the **UP** state again, where the duration is calculated using the repair rate μ from the state that is being left. This is under the assumption that a generator is repaired immediately once it is not operating at full capacity anymore.

A.2. Reliability Metrics

The summed generation profiles from the state sampling and state duration method have the same format, so the **calc_LOLE_MCS** and **calc_EENS_MCS** functions can again be used. The number of simulation years is chosen to be 10000 years, as this value gives a CoV that is in the same range as the one for the state sampling method around 10^{-5}.

The convergence process for the LOLE value of the RBTS for both an HPL and CYPL model is shown in Figure A.1, while the LOLE values are shown in Table A.1 along with MCS benchmark values [26]. The average LOLE value convergence graphs for the IEEE-RTS can be found in Appendix H.1.



(a) average LOLE value for the HPL model

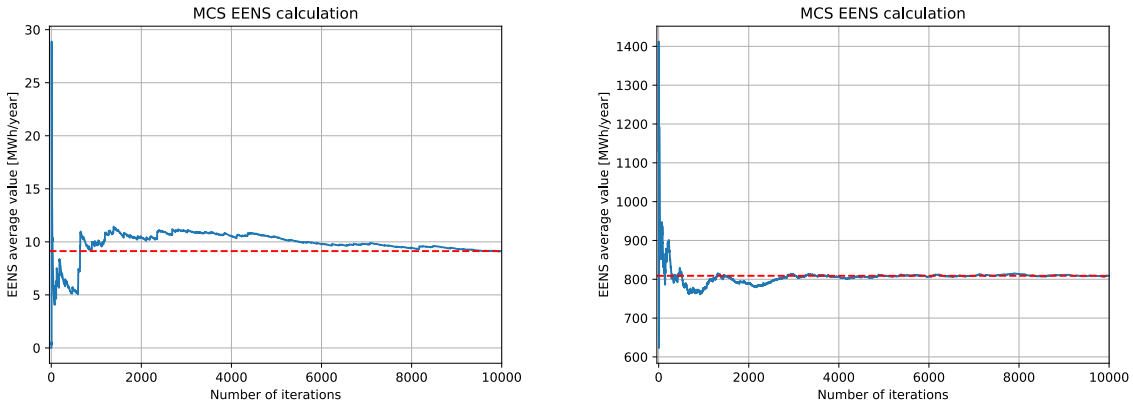
(b) average LOLE value for the CYPL model

Figure A.1: Converging process for the average LOLE value

Table A.1: Comparison between the analytical and MCS scripts for LOLE [26]

	LOLE HPL [hours/year]			LOLE CYPL [hours/year]		
Test System	MCS benchmark	MCS script	Δ [%]	MCS benchmark	MCS script	Δ [%]
IEEE RTS-79	9.3868	8.9038	5.42	-	717.3284	-
RBTS	1.0901	1.0695	1.93	72.6183	70.8222	2.54

A graphical representation of how the average EENS value for the RBTS converges can be found in Figure A.2, while performance metrics of the EENS calculation for the HPL and CYPL models are presented in Table A.2. The average EENS value convergence graphs for the IEEE-RTS are given in Appendix H.2.



(a) average EENS value for the HPL model

(b) average EENS value for the CYPL model

Figure A.2: Converging process for the average EENS value

Table A.2: Comparison between the analytical and MCS scripts for EENS [26]

	EENS HPL [MWh/year]			EENS CYPL [MWh/year]		
Test System	MCS benchmark	MCS script	Δ [%]	MCS benchmark	MCS script	Δ [%]
IEEE RTS-79	1192.5072	1125.7439	5.93	-	123977.6720	-
RBTS	9.9268	9.3133	6.59	816.8147	796.9362	2.49

The values for LOLE and EENS seem to be off by a few percent, and there are two main reasons. One of them is the fact that the TTF and TTR are rounded down to an integer value, i.e. the decimal part is dropped, to ensure that they fit within the summed generation profile and can be multiplied with a zero, one or a fraction between these two values for a derated state. This shortens the duration of both the TTF and the TTR, but it has a more significant effect on the TTR. In the test systems used in this thesis, the MTTF is in the order of 10^3, while that of the MTTR is in the order of 10^1. Getting rid of up to 1 hour on that scale for the TTR will reduce the repair time, and the system will thus be down for a shorter period. One would expect the system to be more reliable and have lower LOLE and EENS values, which can be observed in Tables A.1 and A.2.

Another factor that could contribute to the differences compared to the benchmark values is that the generation profile of each simulation year starts in the **UP** state, and the time overshoot from the previous year is currently not being carried over. This could lead to a system that appears more reliable than it actually is, as observed in the simulation results. An attempt has been made to carry over the overshoot to the next year in the script, but this wasn't fully achieved within the given timeframe of this thesis.

Results for the RBTS with a wind farm [12] using the state duration method couldn't be obtained, as the wind farm is only described by its FOR. The correct values for the failure and repair rate cannot be derived from the FOR, as multiple combinations of λ and μ satisfy the relation between availability and unavailability of the wind farm.

B
Analytical Method Scripts

B.1. COPT for Conventional Generators Including Derated States

```python
"""
File: COPT_derated_states.py
Author: Julian Wuijts
Date: 05-07-2024
Description: A Python script to create a COPT from a list of generators.
"""

#Importing packages
import pandas as pd
import numpy as np
from tabulate import tabulate


def remove_duplicates(dupl):         #Function for removing duplicate values in the capacity outage array
    return list(dict.fromkeys(dupl))

def calc_COPT(generator_data):

    number_of_generators = int(sum(pd.to_numeric(generator_data.iloc[1], errors='coerce').notna())/2) # Count the number of capacity outages and probabilities that are defined in a row, and divide by 2 to obtain the number of generators

    # Define the indices for the capacity and probability in the generator data variable
    select_capacity = [(2 * n)-1 for n in range(1,number_of_generators+1)]
    select_probability = [(2 * n) for n in range(1,number_of_generators+1)]

    # Create a matrix with the possbile capacity states the generators. There is one generator per row and one generator capacity state per column
    Gen_capacity_matrix = generator_data.iloc[1:,select_capacity].T
    Gen_capacity_matrix = Gen_capacity_matrix.reset_index(drop=True)
    Gen_capacity_matrix.columns = range(Gen_capacity_matrix.shape[1]) # Reset the indices of both the columns and rows to start from zero

    # Create a matrix with the probabilities of being in a certain capacity state. The probabilities for one generator are given in a row.
    Gen_unavailability_matrix = generator_data.iloc[1:,select_probability].T
    Gen_unavailability_matrix = Gen_unavailability_matrix.reset_index(drop=True)
    Gen_unavailability_matrix.columns = range(Gen_unavailability_matrix.shape[1]) # Reset the indices of both the columns and rows to start from zero

    #Initialise lists for the COPT recursive algorithm
    Gen_outage_capacity = []
    Gen_outage_capacity_update =[]

    Old_gen_outage_capacity = [0]
    Gen_outage_capacity = [0]
    P_old_list_prev_gen = []

    for row in Gen_capacity_matrix.index:          # Go through all generators one by one
        for column in Gen_capacity_matrix.columns: # Go through all capacity states of one generator
            Gen_outage_capacity_update_temp = [x+Gen_capacity_matrix.loc[row,column] for x in Gen_outage_capacity] # Add the capacity of the current generator outage state to all previous total capacity outages
            Gen_outage_capacity_update = Gen_outage_capacity_update + Gen_outage_capacity_update_temp              # Append the new total capacity outages for the current generator outage state to the end of a list with the previous total capacity outages
```

```
        Gen_outage_capacity = Gen_outage_capacity + Gen_outage_capacity_update                                  # Append all the new total capacity outages for one generator to a list with the previous total capacity outages
        Gen_outage_capacity = remove_duplicates(Gen_outage_capacity)                                            # Remove duplicate capacity outage values from the list
        Gen_outage_capacity.sort()                                                                              # Sort the capacity outage values in ascending order

        P_new_list = [] # Initalise the list with the cumulative probabilities for the COPT

        for i in range(len(Gen_outage_capacity)): # Go through all outage states for the generator that is added to the COPT

            # Initialise the lists for temporarily storing probability values
            P_new = []
            P_old_list = []
            P_new_sum = 0

            for j in range(len(Gen_unavailability_matrix.columns)): # Go through all capacity states of one generator
                Capacity_outage_minus_new_gen = [Gen_outage_capacity[i] - x for x in Gen_capacity_matrix.values[row]] # Create a list with the generation surplus / deficit

                if Capacity_outage_minus_new_gen[j] <= 0 : # If x_j - g_i is smaller than 0, the cumulative probability is one (improve this explanation)
                    P_old = 1
                elif Capacity_outage_minus_new_gen[j] in Old_gen_outage_capacity:                # If x_j - g_i matches a capacity outage level from the previous iteration,
                    P_old_index = Old_gen_outage_capacity.index(Capacity_outage_minus_new_gen[j]) # assign the probability from the previous iteration to the current
                    P_old = P_old_list_prev_gen[P_old_index]
                elif Capacity_outage_minus_new_gen[j] > max(Old_gen_outage_capacity): # If x_j - g_i is larger than the largest capacity outage from the previous iteration,
                    P_old = 0                                                         # the cumulative probability is zero
                else: # If x_j - g_i is in between outage capacity values, it gets the probability from the upper outage capacity value
                    for k in range(len(Old_gen_outage_capacity)-1):
                        if Old_gen_outage_capacity[k] <= Capacity_outage_minus_new_gen[j] < Old_gen_outage_capacity[k+1]:
                            P_old = P_old_list_prev_gen[k+1]
                            break
                        else:
                            P_old = 0

                P_old_list.append(P_old)  # Add the probability for the current capacity outage state to a list
                P_new_temp = Gen_unavailability_matrix.values[row,j] * P_old_list[j] # Mulitply the probability of the capacity outage state with the (un)availability of the current generator capacity state
                P_new.append(P_new_temp)  # Add the new probability to a list

            P_new_sum = np.sum(P_new)     # Sum all probabilities for one outage state
            P_new_list.append(P_new_sum)  # Add the probability for one outage state to a list with all outage states

        #Update the outage capacity and probability lists by converting the new list to the old list
        Old_gen_outage_capacity = Gen_outage_capacity
        P_old_list_prev_gen = P_new_list
    return Gen_outage_capacity, P_new_list

if __name__ == "__main__": # Only print these graphs if the COPT_derated_states script is ran


    ## UPDATE THE PATH BELOW TO WHERE YOUR FILE IS LOCATED. USE / INSTEAD OF THE \ GENERATED BY WINDOWS ##
    generator_data = pd.read_excel("/Users/jswui/Desktop/COPT_example.xlsx", sheet_name="Generators") #Import generator data from Excel


    Gen_outage_capacity, P_new_list = calc_COPT(generator_data)

    COPT_table_headers = ['Generator Outage Capacity [MW]','Cumulative probability']
    P_new_list_rounded = ["{:.5f}".format(x) for x in P_new_list]
    COPT = list(zip(Gen_outage_capacity,P_new_list_rounded))
```

```
    COPT_format = tabulate(COPT, headers=COPT_table_headers, tablefmt="grid", numalign="center",stralign="center")

    print('\n\n')
    print(COPT_format)

    # This command saves the COPT as a .txt file to the current folder. It can be uncommented if the table is too large to print in the terminal

    # with open("COPT_table.txt", "w") as f:
    #     print(COPT_format, file=f)
```

B.2. IEEE Load Profile

```
"""
File: Load_curves.py
Author: Julian Wuijts
Date: 05-07-2024
Description: A Python script to create the IEEE load curves.
"""

from matplotlib import pyplot as plt
import numpy as np
import pandas as pd

def CreateLoadCurve(YPL):
    #Import the load data from excel
    WPL_excel = pd.read_excel("/Users/jswui/Desktop/Load_curves.xlsx", sheet_name="WPL")
    DPL_excel = pd.read_excel("/Users/jswui/Desktop/Load_curves.xlsx", sheet_name="DPL")
    HPL_excel = pd.read_excel("/Users/jswui/Desktop/Load_curves.xlsx", sheet_name="HPL")

    time = list(range(1,8737)) # Create a list with hourly increments for the duration of a year (52 weeks * 7 days * 24 hours = 8736 hours)

    ############### YPL ###############

    YPL_plot = YPL * np.ones(8736) # Create hourly increments for the YPL model


    ############### WPL ###############

    #Adapt the WPL data from excel and store them in a list
    WPL = WPL_excel.to_numpy()
    WPL = WPL[np.ix_([1],np.arange(1, WPL.shape[1]))]
    WPL = WPL.flatten()
    WPL_list = []

    for i in range(len(WPL)):                    # Add hourly increments to each WPL value (7 days * 24 hours = 168 hours)
        WPL_list.extend(WPL[i]*np.ones(168))

    WPL_plot = [x * YPL / 100 for x in WPL_list] # Divide WPL_list by 100 to get a percentage and multiply it with the YPL to get the absolute value
    WPL_plot_sorted = np.sort(WPL_plot)[::-1]    # Order the WPL_plot list in descending order for the Load Duration Curve


    ############### DPL ###############

    # Adapt the DPL data from excel and store them in a list
    DPL = DPL_excel.to_numpy()
    DPL = DPL[np.ix_([1],np.arange(1, DPL.shape[1]))]
    DPL = DPL.flatten()
    DPL_list_temp= []
    DPL_list = []

    for j in range(len(DPL)): # Duplicate DPL 24 times for each day to obtain a week of load data with hourly increments
```

```
50          DPL_list_temp.extend(list(DPL[j]*np.ones(24)))
51
52      for k in range(52):       # Duplicate DPL_list_temp 52 times to go from a week to a year
53          DPL_list.extend(DPL_list_temp)
54
55      DPL_plot = np.multiply(WPL_plot, DPL_list) / 100 # Divide DPL_list by 100 to get a percentage and multiply it with
        ↪  the WPL_plot values to get the absolute value
56      DPL_plot_sorted = np.sort(DPL_plot)[::-1]        # Order the DPL_plot list in descending order for the Load Duration
        ↪  Curve
57
58
59      ############### HPL ###############
60
61      # Adapt the HPL data from excel and store them in a list
62      # There are separate lists for a weekday and a weekend day, as well as for the three specified season categories
63      HPL = HPL_excel.to_numpy()
64      HPL_winter_weekday = HPL[np.ix_([1],np.arange(3, HPL.shape[1]))].flatten()
65      HPL_summer_weekday = HPL[np.ix_([7],np.arange(3, HPL.shape[1]))].flatten()
66      HPL_spring_fall_weekday = HPL[np.ix_([13],np.arange(3, HPL.shape[1]))].flatten()
67
68      HPL_winter_weekend = HPL[np.ix_([4],np.arange(3, HPL.shape[1]))].flatten()
69      HPL_summer_weekend = HPL[np.ix_([10],np.arange(3, HPL.shape[1]))].flatten()
70      HPL_spring_fall_weekend = HPL[np.ix_([16],np.arange(3, HPL.shape[1]))].flatten()
71
72      # Initialising the lists with the hourly data for a week
73      HPL_winter = []
74      HPL_summer = []
75      HPL_spring_fall = []
76
77
78      for l in range(7): # Create variables that contain the data of a whole week by combining 5 weekdays and 2 weekend
        ↪  days in a new variable
79          if l < 5:
80              HPL_winter.extend(list(HPL_winter_weekday))
81              HPL_summer.extend(list(HPL_summer_weekday))
82              HPL_spring_fall.extend(list(HPL_spring_fall_weekday))
83          else:
84              HPL_winter.extend(list(HPL_winter_weekend))
85              HPL_summer.extend(list(HPL_summer_weekend))
86              HPL_spring_fall.extend(list(HPL_spring_fall_weekend))
87
88      # Initialising the list with the hourly data for a year
89      HPL_list = []
90
91      for m in range(52): # Check which seasons corresponds to the current week number and add the hourly data for a week
        ↪  to the list for a
92          if m < 8 or 43 <= m < 52:
93              HPL_list.extend(HPL_winter)
94          elif 17 <= m < 30:
95              HPL_list.extend(HPL_summer)
96          elif  8 <= m < 17 or 30 <= m < 43:
97              HPL_list.extend(HPL_spring_fall)
98          else:
99              print('something went wrong!') # A check to ensure that all 52 weeks are assigned
100
101
102     HPL_plot = np.multiply(DPL_plot, HPL_list) / 100 # Divide HPL_list by 100 to get a percentage and multiply it with
        ↪  the DPL_plot values to get the absolute value
103     HPL_plot_sorted = np.sort(HPL_plot)[::-1]
104     return time, YPL_plot, WPL_plot, DPL_plot, HPL_plot, WPL_plot_sorted, DPL_plot_sorted, HPL_plot_sorted
105
106 if __name__ == "__main__": # Only print these graphs if the Load_curves script is ran
107
108     #Define the Yearly Peak Load for the test systems
109     YPL_RBTS = 185
110     YPL_RTS = 2850
111
112     #Choose which one of the two test systems should be used by uncommenting that entry
113     YPL = YPL_RBTS
```

```
    #YPL = YPL_RTS

    time, YPL_plot, WPL_plot, DPL_plot, HPL_plot, WPL_plot_sorted, DPL_plot_sorted, HPL_plot_sorted =
    ↪  CreateLoadCurve(YPL)

    plt.figure(1)
    plt.plot(time, YPL_plot, label='YPL')
    plt.plot(time, WPL_plot, label='WPL')
    plt.plot(time, DPL_plot, label='DPL')
    plt.xlabel('Time [hours]')
    plt.ylabel('Load [MW]')
    plt.title('Load Curves for the RBTS')
    plt.title('IEEE load curves in Per-unit')
    plt.legend()

    plt.figure(2)
    plt.plot(time, YPL_plot, label='YPL')
    plt.plot(time, WPL_plot_sorted, label='WPL')
    plt.plot(time, DPL_plot_sorted, label='DPL')
    plt.plot(time, HPL_plot_sorted, label='HPL')
    plt.xlabel('Time [hours]')
    plt.ylabel('Load [MW]')
    plt.title('Load Duration Curves for the RBTS in descending order')
    plt.title('IEEE Load Duration Curves in Per-unit')
    plt.legend()
    plt.show()
```

B.3. Reliability Indices

```
"""
File: Reliability_indices.py
Author: Julian Wuijts
Date: 05-07-2024
Description: A Python script to obtain the reliability indices LOLP, LOLE and EENS.
"""

import pandas as pd

from Load_curves import CreateLoadCurve
from COPT_derated_states import calc_COPT


def calc_LOLP(Gen_outage_capacity,P_outage,load):
    Installed_generation = Gen_outage_capacity[-1]
    Max_outage = Installed_generation - load
    j = 0
    if Max_outage <= 0 :
        P= 1
    elif Max_outage in Gen_outage_capacity:
        P_index = Gen_outage_capacity.index(Max_outage)
        P = P_outage[P_index]
    elif Max_outage > max(Gen_outage_capacity):
        P = 0
    else:
        for j in range(len(Gen_outage_capacity)-1):
            if Gen_outage_capacity[j] <= Max_outage < Gen_outage_capacity[j+1]:
                P = P_outage[j+1]
                break
            else:
                print('Something went wrong!')
    return P

def calc_LOLE(Gen_outage_capacity,P_outage, load):
    i = 0
    j = 0
    P_list = []
```

```
    Installed_generation = Gen_outage_capacity[-1]
    for i in range(len(time)):
        Max_outage = Installed_generation - (load[i])
        if Max_outage <= 0 :
            P= 1
        elif Max_outage in Gen_outage_capacity:
            P_index = Gen_outage_capacity.index(Max_outage)
            P = P_outage[P_index+1]
        elif Max_outage > max(Gen_outage_capacity):
            P = 0
        else:
            for j in range(len(Gen_outage_capacity)-1):
                if Gen_outage_capacity[j] <= Max_outage < Gen_outage_capacity[j+1]:
                    P = P_outage[j+1]
                    break
                else:
                    pass
        P_list.append(P)
    LOLE = sum(P_list)
    return LOLE

def calc_EENS(Gen_outage_capacity,P_individual, load):
    i = 0
    j = 0
    EENS = 0
    #EENS_list = []
    Installed_generation = Gen_outage_capacity[-1]
    for i in range(len(time)):
        E_sum = 0
        for j in range(len(Gen_outage_capacity)):
            Capacity_outage = (Gen_outage_capacity[j] -(Installed_generation - load[i]))
            if Capacity_outage > 0:
                E_individual = Capacity_outage*P_individual[j]
                E_sum += E_individual
            else:
                pass
        EENS += E_sum
    return EENS

generator_data = pd.read_excel("/Users/jswui/Desktop/COPT_RBTS.xlsx", sheet_name="Generators") #Import generator data
↪  from Excel
YPL_RBTS = 185
YPL_RTS = 2850

YPL = YPL_RBTS

Gen_outage_capacity, P_outage = calc_COPT(generator_data)
time, YPL_plot, WPL_plot, DPL_plot, HPL_plot, WPL_plot_sorted, DPL_plot_sorted, HPL_plot_sorted = CreateLoadCurve(YPL)

P_outage2 = P_outage
P_outage2.append(0)
P_individual = [P_outage2[i - 1] - P_outage2[i] for i in range(1, len(P_outage2))]


# Calculate LOLP
# P = calc_LOLP(Gen_outage_capacity,P_outage,YPL)
# print('\nLOLP', P,'\n')

# Calculate LOLE
LOLE_HPL = calc_LOLE(Gen_outage_capacity,P_outage,HPL_plot)
print('\nLOLE HPL',LOLE_HPL,'hours/year \n')
print('\nLOLP dash HPL',LOLE_HPL/8736,'\n')

# LOLE_DPL = calc_LOLE(Gen_outage_capacity,P_outage,DPL_plot)/24
# print('\nLOLE DPL',LOLE_DPL,'days/year \n')

LOLE_YPL = calc_LOLE(Gen_outage_capacity,P_outage,YPL_plot)
print('\nLOLE YPL',LOLE_YPL,'hours/year \n')
print('\nLOLE YPL',LOLE_YPL/24,'days/year \n')
```

```
print('\nLOLP dash YPL',LOLE_YPL/8736,'\n\n')

# Calculate EENS
EENS_HPL = calc_EENS(Gen_outage_capacity,P_individual, HPL_plot)
print('\nEENS HPL', EENS_HPL, 'MWh / year\n')
print('\nENDS HPL', EENS_HPL/8736,'\n')

EENS_YPL = calc_EENS(Gen_outage_capacity,P_individual, YPL_plot)
print('\nEENS YPL', EENS_YPL, 'MWh / year\n')
print('\nENDS YPL', EENS_YPL/8736,'\n')
```

B.4. COPT for Wind Turbines

```
"""
File: COPT_wind.py
Author: Julian Wuijts
Date: 05-07-2024
Description: A Python script to create a COPT for wind turbines.
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom
from tabulate import tabulate

def remove_duplicates(dupl):         #Function for removing duplicate values in the capacity outage array
    return list(dict.fromkeys(dupl))

def CreatePowerProfile(wind_speed,Turbine_power_profile_excel):

    #Extract the turbine power profile data from the excel file
    Turbine_power_profile = Turbine_power_profile_excel.to_numpy()
    Turbine_power_profile = Turbine_power_profile[:,1:]
    P_rated = Turbine_power_profile[0,1]
    V_cut_in = float(Turbine_power_profile[1,1])
    V_rated = float(Turbine_power_profile[2,1])
    V_cut_out = float(Turbine_power_profile[3,1])

    #Calculate the constants for creating the power profile
    A = 1/(V_cut_in-V_rated)**2 * (V_cut_in*(V_cut_in+V_rated)-4*V_cut_in*V_rated*((V_cut_in+V_rated)/(2*V_rated))**3)
    B = 1/(V_cut_in-V_rated)**2 * (4*(V_cut_in+V_rated)*((V_cut_in+V_rated)/(2*V_rated))**3-(3*V_cut_in+V_rated))
    C = 1/(V_cut_in-V_rated)**2 * (2-4*((V_cut_in+V_rated)/(2*V_rated))**3)

    P_output_list = []

    for i in wind_speed:
        if i < V_cut_in:
            P_output = 0
        elif V_cut_in <= i < V_rated:
            P_output = (A+B*i+C*(i)**2)*P_rated
        elif V_rated <= i < V_cut_out:
            P_output = P_rated
        else:
            P_output = 0
        P_output_list.append(P_output)

    return P_output_list, P_rated, wind_speed

def combine_wind_mechanical_outage (all_states, number_of_turbines, output_states,
↪  turbine_power_profile_probability_test):
    #Enter the power outage and mechanical outage from 0 MW / turbines to the maximum MW / number of turbines
    multi_state_COPT =[]
    for k, value1 in enumerate(all_states):
        probability_all_turbines =[]
        for i in range(number_of_turbines+1):
```

```
            result_sum = 0
            probability_one_turbine_list = []
            for j in range(number_of_wind_output_states):
                result = 0
                result_temp =0
                if i == 0:
                    result = turbine_power_profile_probability_test[j]
                else:
                    valid_state_check = (all_states[k]/i - output_states[j])
                    if k == 0:
                        if j == 0:
                            result = turbine_power_profile_probability_test[j]
                    else:
                        if valid_state_check >= 0:
                            result_temp = turbine_power_profile_probability_test[j]
                result_sum+=result_temp
                probability_one_turbine = result*individual_prob[i]
                probability_one_turbine_list.append(probability_one_turbine)

            probability_one_turbine_sum = result_sum*individual_prob[i]
            probability_one_turbine_list_sum = sum(probability_one_turbine_list)
            probabilities_summed = probability_one_turbine_sum+probability_one_turbine_list_sum
            probability_all_turbines.append(probabilities_summed)

        multi_state_COPT.append(sum(probability_all_turbines))

    multi_state_COPT_individual_temp = [0] + multi_state_COPT
    multi_state_COPT_individual = [multi_state_COPT_individual_temp[i+1] - multi_state_COPT_individual_temp[i] for i in
    ↪  range(len(multi_state_COPT_individual_temp)-1)]
    return multi_state_COPT_individual

def Apportioning (Apportioned_states, all_states_outage, Outage_multi_state_COPT_individual):

    #Initialising the apportioning variables
    Apportioned_probability_lower = 0
    Apportioned_probability_upper = 0
    Apportioned_list_lower_outage = []
    Apportioned_list_upper_outage = []
    Apportioned_list = [0] *len(Apportioned_states)


    #print('COPT individual probabilities', Outage_multi_state_COPT_individual)

    for index_i, i in enumerate(Apportioned_states):
        count = 0
        for index_j, j in enumerate(all_states_outage):
            if index_i < len(Apportioned_states)-1:
                if i == j and count == 0:
                    Apportioned_probability_lower_temp = Outage_multi_state_COPT_individual[index_j]
                    Apportioned_probability_upper_temp = 0
                    count = 1
                elif Apportioned_states[index_i] < j < Apportioned_states[index_i+1]:
                    # Using the apportioning formula
                    Apportioned_probability_lower_temp = Outage_multi_state_COPT_individual[index_j] *
                    ↪  ((Apportioned_states[index_i+1]-j) / (Apportioned_states[index_i+1]-Apportioned_states[index_i]))
                    Apportioned_probability_upper_temp = Outage_multi_state_COPT_individual[index_j] *
                    ↪  ((j-Apportioned_states[index_i]) / (Apportioned_states[index_i+1]-Apportioned_states[index_i]))
                else:
                    Apportioned_probability_lower_temp = 0
                    Apportioned_probability_upper_temp = 0
                Apportioned_probability_lower+=Apportioned_probability_lower_temp
                Apportioned_probability_upper+=Apportioned_probability_upper_temp
            else:
                if i == j:
                    Apportioned_probability_lower_temp = Outage_multi_state_COPT_individual[index_j]
                    Apportioned_probability_upper_temp = 0
                Apportioned_probability_lower+=Apportioned_probability_lower_temp
                Apportioned_probability_upper+=Apportioned_probability_upper_temp
        Apportioned_list[index_i]+=Apportioned_probability_lower
```

```
        if index_i < len(Apportioned_states)-1:
            Apportioned_list[index_i+1]+=Apportioned_probability_upper

    Apportioned_list_cumulative_outage = Apportioned_list
    Apportioned_list_individual_temp_outage = [0] + Apportioned_list_cumulative_outage
    Apportioned_list_individual_outage = [Apportioned_list_individual_temp_outage[i+1] -
    ↪  Apportioned_list_individual_temp_outage[i] for i in range(len(Apportioned_list_individual_temp_outage)-1)]

    return Apportioned_list_individual_outage, Apportioned_list_cumulative_outage

def Windplots(time, wind_speed, P_output_list, P_output_test):
    # Plot the wind speed distribution
    plt.figure(1)
    plt.hist(wind_speed, bins=25, edgecolor='black',density=True)
    plt.title('Wind speed probability distribution for winds of the coast of Trondheim in 2019')
    plt.xlabel('Wind Speed [m/s]')
    plt.ylabel('Probability')

    # Plot the wind speed chronologically
    plt.figure(2)
    plt.plot(time, wind_speed, label='Wind speeds')
    plt.xlabel('Time [hours]')
    plt.ylabel('Wind Speed [m/s]')
    plt.title('Wind speeds of the coast of Trondheim in 2019')

    # Plot the power output probability distribution individually or cumulatively
    plt.figure(3)
    plt.hist(P_output_list, bins=11, edgecolor='black',density=True) # Plot the individual probabilites
    #plt.hist(P_output_list, bins=20, density=True, cumulative=True) # Plot the cumulative probabilities
    plt.xlabel('Output Power [MW]')
    plt.ylabel('Probability')
    plt.title('Power output probability distribution for a turbine of the coast of Trondheim in 2019')

    # Plot the power output of a turbine chronologically
    plt.figure(4)
    plt.plot(time, P_output_list, label='Power output')
    plt.xlabel('Time [hours]')
    plt.ylabel('Power [MW]')
    plt.title('Power output for the DTU 10 MW reference turbine of the coast of Trondheim in 2019')

    # Plot the power curve for a wind turbine
    plt.figure(5)
    plt.plot(wind_speed_test, P_output_test, label='Power output')
    plt.xlabel('Wind Speed [m/s]')
    plt.ylabel('Power Output [MW]')
    plt.title('Power output profile for the DTU 10 MW reference wind turbine')

    plt.show()
    return

time = list(range(1,8761))
############### Specify wind turbine generator data ###############

number_of_wind_output_states = 5
FOR = 0.04
number_of_turbines = 10
P_rated = 2 # in MW
turbine_power_profile_probability_test = [0.50897,0.24450,0.11688,0.05944,0.07021] #5

####### Construct the mechanical outage probability table ########
if __name__ == "__main__":
    availability = 1 - FOR
    x = range(number_of_turbines+1)
    cumulative_prob = binom.cdf(x, number_of_turbines,availability)
    individual_prob = binom.pmf(x, number_of_turbines,availability)

    individual_prob_outage = individual_prob[::-1]

    Installed_capacity = P_rated*number_of_turbines
```

```
    Capacity_outage = np.linspace(0,Installed_capacity,number_of_turbines+1)

    Mechanical_outage_headers = ['Capacity Outage [MW]','Individual probability']
    individual_prob_outage_rounded = ["{:.5f}".format(x) for x in individual_prob_outage]
    Mechanical_outage_table = list(zip(Capacity_outage,individual_prob_outage_rounded))
    Mechanical_outage_table_format = tabulate(Mechanical_outage_table, headers=Mechanical_outage_headers,
    ↪  tablefmt="grid", numalign="center",stralign="center")

    print('Mechanical outage probability table \n')
    print(Mechanical_outage_table_format)

    print('\n mechanical availability',individual_prob*100,'\n')


output_states = np.linspace(0,P_rated,number_of_wind_output_states)

all_states = []
for i in range(1,number_of_turbines+1):
    for j in range(len(output_states)):
        possible_state=i*output_states[j]
        all_states.append(possible_state)

all_states = remove_duplicates(all_states)


all_states.sort()
all_states_outage = all_states
all_states_outage = [number_of_turbines*P_rated - x for x in all_states_outage]
all_states_outage.sort()

Wind_speed_excel = pd.read_excel("/Users/jswui/Desktop/Thesis codes/Analytical/COPT_wind.xlsx", sheet_name="Wind_speed")
Wind_speed_test = pd.read_excel("/Users/jswui/Desktop/Thesis codes/Analytical/COPT_wind.xlsx",
↪  sheet_name="Wind_speed_power_curve")
Turbine_power_profile_excel = pd.read_excel("/Users/jswui/Desktop/Thesis codes/Analytical/COPT_wind.xlsx",
↪  sheet_name="Turbine_power_profile")

#Extract the wind speeds from the excel file
wind_speed = Wind_speed_excel.to_numpy()
wind_speed = wind_speed[np.ix_(np.arange(2, wind_speed.shape[0]),[1])]
wind_speed = wind_speed.flatten()

wind_speed_test = Wind_speed_test.to_numpy()
wind_speed_test = wind_speed_test[2:,1]
wind_speed_test = wind_speed_test.flatten()



if __name__ == "__main__":

    P_output_list, P_rated, wind_speed = CreatePowerProfile(wind_speed,Turbine_power_profile_excel)
    P_output_test, P_rated, wind_speed_test = CreatePowerProfile(wind_speed_test,Turbine_power_profile_excel)

    # Go from a list of power outputs to a state probability table

    State_probability_list = []

    for j, value in enumerate(output_states):
        counter = 0
        for k in P_output_list:
            if output_states[j] < output_states[-1]:
                if output_states[j] <= k < output_states[j+1]:
                    counter+=1
            elif output_states[j] == output_states[-1] and k == output_states[-1]:
                    counter+=1
        State_probability = counter/len(P_output_list)
        State_probability_list.append(State_probability)


multi_state_COPT_individual = combine_wind_mechanical_outage(all_states, number_of_turbines, output_states,
↪  turbine_power_profile_probability_test)
```

```
253
254 Output_table_headers = ['Capacity \n Output \n[MW]',' Individual\n probability']
255 multi_state_COPT_individual_rounded = ["{:.5f}".format(x) for x in multi_state_COPT_individual]
256 Output_table = list(zip(all_states,multi_state_COPT_individual_rounded))
257 Output_table_format = tabulate(Output_table, headers=Output_table_headers, tablefmt="grid",
    ↪ numalign="center",stralign="center")
258
259 print('\n Capacity Output Probability Table \n')
260 print(Output_table_format)
261
262 Outage_table_headers = ['Capacity Outage [MW]','Individual probability']
263 Outage_multi_state_COPT_individual = multi_state_COPT_individual[::-1]
264 Outage_multi_state_COPT_individual_rounded = ["{:.5f}".format(x) for x in Outage_multi_state_COPT_individual]
265 Outage_table = list(zip(all_states_outage,Outage_multi_state_COPT_individual_rounded))
266 Outage_table_format = tabulate(Outage_table, headers=Outage_table_headers, tablefmt="grid",
    ↪ numalign="center",stralign="center")
267
268 print('\n Capacity Outage Probability Table\n')
269 print(Outage_table_format)
270
271 # Specify the states to reduce the COPT to
272 Apportioned_states = [0,5,10,15,20]
273
274 Apportioned_list_individual_outage, Apportioned_list_cumulative_outage = Apportioning(Apportioned_states,
    ↪ all_states_outage, Outage_multi_state_COPT_individual)
275
276 # Create the apportioned COPT
277 Outage_table_reduced_headers = ['Capacity Outage [MW]','Individual probability']
278 Apportioned_list_individual_outage_rounded = ["{:.5f}".format(x) for x in Apportioned_list_individual_outage]
279 Outage_table_reduced = list(zip(Apportioned_states,Apportioned_list_individual_outage_rounded))
280 Outage_table_reduced_format = tabulate(Outage_table_reduced, headers=Outage_table_reduced_headers, tablefmt="grid",
    ↪ numalign="center",stralign="center")
281
282 print('\n Apportioned COPT\n')
283 print(Outage_table_reduced_format)
284
285 Windplots(time, wind_speed, P_output_list, P_output_test)
```

C
Non-Sequential MCS Scripts

C.1. Generator Profile for Both Wind and Conventional Generators without Derated States

```
"""
File: MCS_state_sampling_derated.py
Author: Julian Wuijts
Date: 05-07-2024
Description: A Python script to create a generation profile for a non-sequential MCS with the option for wind generation,
but without derated states to speed up the computational time.
"""

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from COPT_wind import CreatePowerProfile

# Precompute random values
def generate_random_values(total_hours):
    U = np.random.rand(total_hours)
    return U

# Generate wind speed list using precomputed random values
def generate_wind_speed_list(total_hours, alpha, beta):
    U = generate_random_values(total_hours)
    wind_speed_list = alpha * ((-np.log(U))**(1/beta))
    return wind_speed_list

# Generate generator profile state using precomputed random values
def generate_generator_profile_state(total_hours, FOR):
    U = generate_random_values(total_hours)
    generator_profile_state = (U >= FOR).astype(int)
    return generator_profile_state

# Optimize generation_profile function
def generation_profile(generator_data, alpha, beta, turbine_specs):
    generator_data = generator_data.to_numpy()

    FOR = generator_data[2, 2::2]
    generator_capacity_list = generator_data[2, 1::2]
    generator_type_list = generator_data[3, 1::2]
    number_of_generators = generator_data[4, 1::2].astype(int)
    generator_number = sum(number_of_generators)

    total_hours = 8736
    wind_speed_list = generate_wind_speed_list(total_hours, alpha, beta)
    P_output_list, _, _ = CreatePowerProfile(wind_speed_list, turbine_specs)
    wind_power_profile = P_output_list[:total_hours]

    generator_data_filtered = np.repeat([generator_type_list, generator_capacity_list, FOR], number_of_generators,
    ↪  axis=1)
    generator_type_list = generator_data_filtered[0]
    generator_capacity_list = generator_data_filtered[1]
    FOR = generator_data_filtered[2]
```

```

    summed_generator_profile = np.zeros(total_hours)

    for i in range(generator_number):
        generator_profile_state = generate_generator_profile_state(total_hours, FOR[i])

        if generator_type_list[i] == 0:
            generator_profile_capacity = generator_capacity_list[i] * generator_profile_state
        elif generator_type_list[i] == 1:
            generator_profile_capacity = wind_power_profile * generator_profile_state
        else:
            raise ValueError('Invalid generator type')

        summed_generator_profile += generator_profile_capacity

    return summed_generator_profile

# Use the optimized generation_profile function
if __name__ == "__main__":
    beta = 1.8459062218183633
    alpha = 4.703721846769401

    generator_data = pd.read_excel("/Users/jswui/Desktop/RBTS_MCS_wind_state_sampling.xlsx", sheet_name="Generators")
    ↪  #Import generator data from Excel
    turbine_specs = pd.read_excel("/Users/jswui/Desktop/RBTS_MCS_wind_state_sampling.xlsx", sheet_name="Turbines")

    summed_generator_profile = generation_profile(generator_data, alpha, beta, turbine_specs)
    time = np.arange(1, 8737)

    plt.plot(time, summed_generator_profile)
    plt.xlabel('Time [hours]')
    plt.ylabel('Output [MW]')
    plt.title('Generator Availability')
    plt.grid(True)
    plt.show()
```

C.2. Generator profile for Both Wind and Conventional Generators with Derated States

```
"""
File: MCS_state_sampling_derated.py
Author: Julian Wuijts
Date: 05-07-2024
Description: A Python script to create a generation profile for a non-sequential MCS with the option for wind generation
↪  and derated states.
"""

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from COPT_wind import CreatePowerProfile

def object_to_numpy_array(object):
    processed_list = []
    for value in object:
        try:
            processed_list.append(float(value))
        except (ValueError, TypeError):
            processed_list.append(np.nan)
    return np.array(processed_list)

# Precompute random values
def generate_random_values(total_hours):
    U = np.random.rand(total_hours)
```

```
    return U

# Generate wind speed list using precomputed random values
def generate_wind_speed_list(total_hours, alpha, beta):
    U = generate_random_values(total_hours)
    wind_speed_list = alpha * ((-np.log(U))**(1/beta))
    return wind_speed_list

# Generate generator profile state using precomputed random values
def generate_generator_profile_state(total_hours, state_probability,generator_capacity):
    U = generate_random_values(total_hours)

    state_probability_cleaned = state_probability[~np.isnan(state_probability)]
    state_probability_boundaries = np.cumsum(state_probability_cleaned)

    state = np.searchsorted(state_probability_boundaries, U, side='right')
    state_outage = generator_capacity[state]
    state_outage = np.array(state_outage, dtype=float)
    generator_profile_state = 1-(state_outage/max(generator_capacity))

    return generator_profile_state

# Optimize generation_profile function
def generation_profile(generator_data, alpha, beta, turbine_specs):

    generator_data = generator_data.to_numpy()

    state_probability = generator_data[3:, 2::2]
    generator_capacity_list = generator_data[3:, 1::2]
    generator_type_list = generator_data[0, 1::2]
    number_of_generators = generator_data[1, 1::2].astype(int)
    number_of_states = generator_data[3,0]
    generator_number = sum(number_of_generators)

    repeated_generator_type_list = np.repeat(generator_type_list, number_of_generators)
    repeated_generator_capacity_list = np.repeat(generator_capacity_list, number_of_generators, axis=1)
    repeated_state_probability = np.repeat(state_probability, number_of_generators, axis=1)

    # Concatenate the repeated arrays into the final structure
    generator_data_filtered = np.vstack([repeated_generator_type_list, repeated_generator_capacity_list,
    ↪  repeated_state_probability])

    # Splitting the concatenated data into individual components
    generator_type_list = generator_data_filtered[0]
    generator_capacity_list = generator_data_filtered[1:number_of_states+1]
    state_probability_list = generator_data_filtered[number_of_states+1:]
    state_probability_list = np.array([object_to_numpy_array(row) for row in state_probability_list])

    total_hours = 8736
    wind_speed_list = generate_wind_speed_list(total_hours, alpha, beta)
    P_output_list, _, _ = CreatePowerProfile(wind_speed_list, turbine_specs)
    wind_power_profile = P_output_list[:total_hours]

    summed_generator_profile = np.zeros(total_hours)

    for i in range(generator_number):
        generator_profile_state = generate_generator_profile_state(total_hours,
        ↪  state_probability_list[:,i],generator_capacity_list[:,i])


        if generator_type_list[i] == 0:
            generator_profile_capacity = max(generator_capacity_list[:,i]) * generator_profile_state
        elif generator_type_list[i] == 1:
            generator_profile_capacity = wind_power_profile * generator_profile_state
        else:
            print('Invalid generator type')

        summed_generator_profile += generator_profile_capacity

```

```
    return summed_generator_profile

# Use the optimized generation_profile function
if __name__ == "__main__":
    beta = 1.8459062218183633
    alpha = 4.703721846769401

    ## CHANGE THE DATAPATH HERE ##
    generator_data = pd.read_excel("/Users/jswui/Desktop/RBTS_MCS_wind_state_sampling_derated.xlsx", sheet_name="Generators") #Import generator data from Excel
    turbine_specs = pd.read_excel("/Users/jswui/Desktop/RBTS_MCS_wind_state_sampling_derated.xlsx", sheet_name="Turbines")

    summed_generator_profile = generation_profile(generator_data, alpha, beta, turbine_specs)
    time = np.arange(1, 8737)

    plt.plot(time, summed_generator_profile)
    plt.xlabel('Time [hours]')
    plt.ylabel('Output [MW]')
    plt.title('Generator Availability')
    plt.grid(True)
    plt.show()
```

C.3. Reliability Indices

```
"""
File: Reliability_indices_MCS.py
Author: Julian Wuijts
Date: 05-07-2024
Description: A Python script to obtain the reliability indices LOLE and EENS using a non-sequential MCS method.
"""

import pandas as pd
import numpy as np
import time

from Load_curves import CreateLoadCurve
from MCS_state_sampling_no_derated import generation_profile
#from MCS_state_sampling_derated import generation_profile
from matplotlib import pyplot as plt

def calc_LOLE_MCS(load,stopping_criteria):
    print('LOLE MCS calculation \n')
    num_of_iterations = 0
    LOLE_list = []
    LOLE_average_value_list = []
    LOLE_average_value = 0

    while num_of_iterations < stopping_criteria:
        time_unit = 0

        summed_generator_profile = generation_profile(generator_data,alpha,beta,turbine_specs)

        net_energy = summed_generator_profile-load
        time_unit = np.sum(net_energy < 0)
        LOLE_list.append(time_unit)

        num_of_iterations +=1
        LOLE_average_value = np.mean(LOLE_list)
        LOLE_average_value_list.append(LOLE_average_value)

    print('number of iterations', num_of_iterations)
    print('LOLE value',LOLE_average_value)
    num_of_iterations_list = np.arange(1, num_of_iterations + 1)

    std_dev_temp_list = (LOLE_list - LOLE_average_value) ** 2
```

```
    std_dev_temp = np.sum(std_dev_temp_list)

    std_dev = (1/(num_of_iterations*(num_of_iterations-1)))*std_dev_temp
    CoV = (std_dev/np.sqrt(num_of_iterations))/LOLE_average_value
    print('CoV LOLE',CoV)

    end_time = time.time()
    execution_time = end_time - start_time
    print('execution time: ',execution_time,' seconds')

    plt.figure(1)
    plt.plot(num_of_iterations_list, LOLE_average_value_list)
    plt.axhline(y=LOLE_average_value, color='r', linestyle='--', label='LOLE Average')
    plt.xlabel('Number of iterations')
    plt.ylabel('LOLE average value [hours/year]')
    plt.title('MCS LOLE calculation')
    plt.grid(True)
    plt.show()

    return


def calc_EENS_MCS(load,stopping_criteria):
    print('EENS MCS calculation \n')
    num_of_iterations = 0
    EENS_list = []
    EENS_average_value_list = []
    EENS_average_value = 0

    while num_of_iterations < stopping_criteria:
        energy_not_served = 0
        summed_generator_profile = generation_profile(generator_data,alpha,beta,turbine_specs)

        net_energy = summed_generator_profile-load
        net_energy_negative = np.minimum(net_energy, 0)
        energy_not_served = np.sum(-net_energy_negative)
        EENS_list.append(energy_not_served)

        num_of_iterations +=1
        EENS_average_value = np.mean(EENS_list)
        EENS_average_value_list.append(EENS_average_value)

    print('number of iterations', num_of_iterations)
    print('EENS value',EENS_average_value)
    num_of_iterations_list = list(range(1,num_of_iterations+1))

    std_dev_temp_list = []

    std_dev_temp_list = (EENS_list - EENS_average_value) ** 2
    std_dev_temp = np.sum(std_dev_temp_list)
    std_dev = (1/(num_of_iterations*(num_of_iterations-1)))*std_dev_temp
    CoV = (std_dev/np.sqrt(num_of_iterations))/EENS_average_value
    print('CoV EENS',CoV)

    end_time = time.time()
    execution_time = end_time - start_time
    print('execution time: ',execution_time,' seconds')

    plt.figure(2)
    plt.plot(num_of_iterations_list, EENS_average_value_list)
    plt.axhline(y=EENS_average_value, color='r', linestyle='--', label='EENS Average')
    plt.xlabel('Number of iterations')
    plt.ylabel('EENS average value [MWh/year]')
    plt.title('MCS EENS calculation')
    plt.grid(True)
    plt.show()

    return

```

```

start_time = time.time()

# Specify the alpha and beta parameters based on the wind characteristics #
alpha = 4.703721846769401
beta  = 1.8459062218183633

## CHANGE THE PATH HERE ##
generator_data = pd.read_excel("/Users/jswui/Desktop/RBTS_MCS_wind_state_sampling.xlsx", sheet_name="Generators")
↪  #Import generator data from Excel
turbine_specs = pd.read_excel("/Users/jswui/Desktop/RBTS_MCS_wind_state_sampling.xlsx", sheet_name="Turbines")

YPL_RBTS = 185
YPL_RTS = 2850

YPL = YPL_RBTS

_ , YPL_plot, WPL_plot, DPL_plot, HPL_plot, WPL_plot_sorted, DPL_plot_sorted, HPL_plot_sorted = CreateLoadCurve(YPL)


calc_LOLE_MCS(HPL_plot,10000)
#calc_EENS_MCS(HPL_plot,10000)
```

D
Sequential MCS Scripts

D.1. Generator Profile for Conventional Generators without Derated States

```
"""
File: MCS_state_duration_no_derated.py
Author: Julian Wuijts
Date: 05-07-2024
Description: A Python script to create a generation profile for a sequential MCS.
"""

import numpy as np
import math
import matplotlib.pyplot as plt
import pandas as pd

generator_data = pd.read_excel("/Users/jswui/Desktop/RBTS_MCS_no_wind_state_duration.xlsx", sheet_name="Generators")
↪  #Import generator data from Excel

## Create a generator profile ##
def generation_profile(generator_data):

    generator_data = generator_data.to_numpy()

    generator_capacity =generator_data[0,1:]
    MTTF = generator_data[1,1:]
    MTTR = generator_data[2,1:]

    generator_number = len(generator_capacity)
    total_hours = 8736

    summed_generator_profile = np.zeros(total_hours)

    for i in range(generator_number):

        sum_hours = 0
        generator_profile_state = []

        while sum_hours < total_hours:
            U1, U2 = np.random.rand(2)
            TTF = int(-MTTF[i] * math.log(U1))
            TTR = int(-MTTR[i] * math.log(U2))

            generator_profile_state.extend([1] * min(TTF, total_hours - sum_hours))
            sum_hours += TTF

            if sum_hours < total_hours:
                generator_profile_state.extend([0] * min(TTR, total_hours - sum_hours))
                sum_hours += TTR

        generator_profile_state = generator_profile_state[:total_hours]

        generator_profile_capacity = generator_capacity[i] * np.array(generator_profile_state)

        summed_generator_profile += generator_profile_capacity
```

```
    return summed_generator_profile


if __name__ == "__main__":

    #print('profile length',len(generator_profile_state))

    summed_generator_profile = generation_profile(generator_data)
    time = list(range(1,8737))

    plt.plot(time, summed_generator_profile)
    plt.xlabel('Time [hours]')
    plt.ylabel('Output [MW]')
    plt.title('Generator Availability')
    plt.grid(True)
    plt.show()
```

D.2. Generator Profile for Conventional Generators with Derated States

```
"""
File: MCS_state_duration_derated.py
Author: Julian Wuijts
Date: 05-07-2024
Description: A Python script to create a generation profile for a sequential MCS with the option to include derated states.
"""

import numpy as np
import math
import matplotlib.pyplot as plt
import pandas as pd


generator_data = pd.read_excel("/Users/jswui/Desktop/RBTS_MCS_no_wind_state_duration.xlsx", sheet_name="derated") #Import generator data from Excel
print('generator data',generator_data)


## Create a generator profile ##
def generation_profile(generator_data):

    generator_data = generator_data.to_numpy()

    generator_capacity = generator_data[0,1:]
    num_of_transition_states = int(generator_data[1,1])
    derated_states_nan = (generator_data[2,1:])
    derated_states = []

    for value in derated_states_nan:
        try:
            numeric_value = float(value)
            if not np.isnan(numeric_value):
                derated_states.append(numeric_value)
        except ValueError:
            pass

    derated_states = np.array(derated_states)

    repair_rate = generator_data[3:3+num_of_transition_states,1:]
    failure_rate = generator_data[3+num_of_transition_states:3+2*num_of_transition_states,1:]

    generator_number = len(generator_capacity)
    total_hours = 8736

    summed_generator_profile = np.zeros(total_hours)
```

```
45
46      for i in range(generator_number):
47
48          sum_hours = 0
49          generator_profile_state = []
50
51          while sum_hours < total_hours:
52
53              TTF = np.zeros(num_of_transition_states)
54
55              for j in range(num_of_transition_states):
56                  U = np.random.rand()
57                  if math.isnan(failure_rate[j,i]):
58                      TTF[j] = 1e6
59                  else:
60                      TTF[j] = int(-(8760/failure_rate[j,i]) * math.log(U))
61
62              state_duration_TTF = int(min(TTF))
63              state_TTF = np.argmin(TTF)
64              generator_profile_state.extend([1] * min(state_duration_TTF, total_hours - sum_hours))
65
66              sum_hours+=state_duration_TTF
67
68              if sum_hours < total_hours:
69                  U = np.random.rand()
70                  state_duration_TTR = 0
71                  state_duration_TTR = int((-8760/repair_rate[state_TTF,i])* math.log(U))
72                  generator_profile_state.extend([(derated_states[state_TTF])/100] * min(state_duration_TTR, total_hours -
                    ↪  sum_hours))
73
74                  sum_hours+=state_duration_TTR
75
76          generator_profile_state = generator_profile_state[:total_hours]
77
78          generator_profile_capacity = generator_capacity[i] * np.array(generator_profile_state)
79
80          summed_generator_profile += generator_profile_capacity
81
82      return summed_generator_profile
83
84
85  if __name__ == "__main__":
86
87      #print('profile length',len(generator_profile_state))
88
89      summed_generator_profile = generation_profile(generator_data)
90      time = list(range(1,8737))
91
92      plt.plot(time, summed_generator_profile)
93      plt.xlabel('Time [hours]')
94      plt.ylabel('Output [MW]')
95      plt.title('Generator Availability')
96      plt.grid(True)
97      plt.show()
```

E

Wind Turbine FOR Power Output Comparison and Weibull Distribution Fitting Scripts

E.1. Wind Turbine FOR Power Output Comparison

```
"""
File: Wind_turbine_reliability_comparison.py
Author: Julian Wuijts
Date: 05-07-2024
Description: A Python script to compare the power output distribution of a wind farm for various FORs .
"""

from COPT_wind import CreatePowerProfile
from MCS_state_sampling_no_derated import *

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#import math

def wind_power_distribution_FOR(num_of_years,alpha,beta,generator_data,turbine_specs,FOR_value):
    power_distribution_plot = []
    generator_data = generator_data.to_numpy()
    FOR = generator_data[4, 2::2]
    generator_capacity_list = generator_data[4, 1::2]
    generator_type_list = generator_data[0, 1::2]
    number_of_generators = generator_data[1, 1::2].astype(int)
    generator_number = sum(number_of_generators)

    generator_data_filtered = np.repeat([generator_type_list, generator_capacity_list, FOR], number_of_generators,
    ↪  axis=1)
    generator_type_list = generator_data_filtered[0]
    generator_capacity_list = generator_data_filtered[1]
    FOR = generator_data_filtered[2]

    FOR = np.full(generator_number,FOR_value)

    total_hours = 8760

    for i in range(num_of_years):
        summed_generator_profile = np.zeros(total_hours)

        wind_speed_list = generate_wind_speed_list(total_hours, alpha, beta)
        P_output_list, _, _ = CreatePowerProfile(wind_speed_list, turbine_specs)
        wind_power_profile = P_output_list[:total_hours]

        for i in range(generator_number):
            generator_profile_state = generate_generator_profile_state(total_hours, FOR[i])

            if generator_type_list[i] == 0:
                generator_profile_capacity = generator_capacity_list[i] * generator_profile_state
            elif generator_type_list[i] == 1:
```

```
                generator_profile_capacity = wind_power_profile * generator_profile_state
            else:
                raise ValueError('Invalid generator type')

            summed_generator_profile += generator_profile_capacity
        power_distribution_plot.extend(summed_generator_profile)

    return power_distribution_plot

num_of_years = 25

## Trondheim ##
alpha = 10.2239
beta = 1.8865

turbine_specs =  pd.read_excel("/Users/jswui/Desktop/Thesis codes/MCS/Wind_reliability_comparison.xlsx", sheet_name="Turbine_power_profile")
generator_data = pd.read_excel("/Users/jswui/Desktop/Thesis codes/MCS/Wind_reliability_comparison.xlsx", sheet_name="Generators")

FOR1 = 0.04
FOR2 = 0.08

FOR_test = np.arange(0.00,0.22,0.02)
yearly_power_outputs = []
perfect_reliability_output = 0

for FOR in FOR_test:
    power_distribution_plot = wind_power_distribution_FOR(num_of_years, alpha, beta, generator_data, turbine_specs, FOR)
    yearly_power_output = sum(power_distribution_plot) / num_of_years
    yearly_power_outputs.append(yearly_power_output)

    if FOR == 0.00:
        perfect_reliability_output = yearly_power_output

    # Sort the power distribution plot
    power_distribution_plot = np.sort(power_distribution_plot)

    # Calculate the cumulative probabilities
    cumulative_power_distribution_plot = np.arange(len(power_distribution_plot)) / float(len(power_distribution_plot))

    median_index = len(power_distribution_plot) // 2
    median_power_output = power_distribution_plot[median_index]
    #print(f'FOR = {FOR:.2f}, Median power output: {median_power_output:.2f} MW')

    if FOR == 0.00:
        perfect_reliability_median = median_power_output

    percentage_median_output = (median_power_output / perfect_reliability_median) * 100 if perfect_reliability_median != 0 else 100
    print(f'FOR = {FOR:.2f}, Median power output: {median_power_output:.2f} MW, {percentage_median_output:.2f}% of perfect reliability')

    # Plot the cumulative distribution function
    plt.plot(power_distribution_plot, cumulative_power_distribution_plot, linestyle='-', label=f'FOR={FOR:.2f}')

# Convert the list of yearly power outputs to a NumPy array
yearly_power_outputs = np.array(yearly_power_outputs)
print('\n')
# Print the yearly power outputs as an absolute value and percentage
for FOR, output in zip(FOR_test, yearly_power_outputs):
    percentage_output = (output / perfect_reliability_output) * 100
    print(f'FOR = {FOR:.2f}, Yearly power output is: {output:.2f} MWh, {percentage_output:.2f}% of perfect reliability')


# Plot the cumulative distribution functions for different FOR
plt.xlabel('Wind Farm Output Power [MW]', fontsize=24)
plt.ylabel('Probability', fontsize=20)
#plt.title('Cumulative distribution function for turbines with varying FOR')
```

```
plt.legend(loc='upper left', prop={'size': 14})
plt.show()
```

E.2. Weibull Distribution Fitting

```
"""
File: Weibull_distribution_fitting.py
Author: Julian Wuijts
Date: 05-07-2024
Description: A Python script to fit a Weibull distribution with parameters alpha and beta to a wind speed distribution.
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import weibull_min

## wind site Trondheim ##

Wind_speed_excel = pd.read_excel("/Users/jswui/Desktop/Thesis codes/MCS/windatlas-xyz-data_39_years_Trondheim.xlsx",
↪  sheet_name="windatlas-xyz-data_39_years")

# #Extract the wind speeds from the excel file
wind_speed = Wind_speed_excel.to_numpy()
wind_speed = wind_speed[2:, 1]
wind_speed = wind_speed.flatten()

cleaned_wind_speed = []

# Iterate over each element in wind_speed
for val in wind_speed:
    try:
        # Try converting the element to float
        num_val = float(val)
        # Check if it's not NaN
        if not np.isnan(num_val):
            # Append to cleaned_wind_speed
            cleaned_wind_speed.append(num_val)
    except ValueError:
        # If conversion to float fails, continue to the next element
        continue

# Convert cleaned_wind_speed to a numpy array
cleaned_wind_speed = np.array(cleaned_wind_speed)

# Fit the Weibull distribution
alpha, loc, beta = weibull_min.fit(cleaned_wind_speed, floc=0)


print('alpha, beta',alpha,beta)

plt.figure(1)
plt.hist(wind_speed, bins=25, edgecolor='black',density=True)
plt.title('Wind speed probability distribution for winds of the coast of Trondheim 1980-2019')
plt.xlabel('Wind Speed [m/s]',fontsize=20)
plt.ylabel('Probability',fontsize=20)
plt.xlim(right=30)


x = np.linspace(wind_speed.min(), wind_speed.max(), 100)
plt.plot(x, weibull_min(alpha,0,beta).pdf(x))

plt.show()
```

F
IEEE Load Curve

F.1. Weekly Peak Load

Table F.1: WPL as a percentage of the YPL

Week number	WPL as a % of YPL	Week number	WPL as a % of YPL
1	86.2	27	75.5
2	90.0	28	81.6
3	87.8	29	80.1
4	83.4	30	88.0
5	88.0	31	72.2
6	84.1	32	77.6
7	83.2	33	80.0
8	80.6	34	72.9
9	74.0	35	72.6
10	73.7	36	70.5
11	71.5	37	78.0
12	72.7	38	69.5
13	70.4	39	72.4
14	75.0	40	72.4
15	72.1	41	74.3
16	80.0	42	74.4
17	75.4	43	80.0
18	83.7	44	88.1
19	87.0	45	88.5
20	88.0	46	90.9
21	85.6	47	94.0
22	81.1	48	89.0
23	90.0	49	94.2
24	88.7	50	97.0
25	89.6	51	100.0
26	86.1	52	95.2

F.2. Daily Peak Load

Table F.2: DPL as a percentage of the WPL

Day	DPL as a % of WPL
Monday	93
Tuesday	100
Wednesday	98
Thursday	96
Friday	94
Saturday	77
Sunday	75

F.3. Hourly Peak Load

Table F.3: HPL as a percentage of the DPL

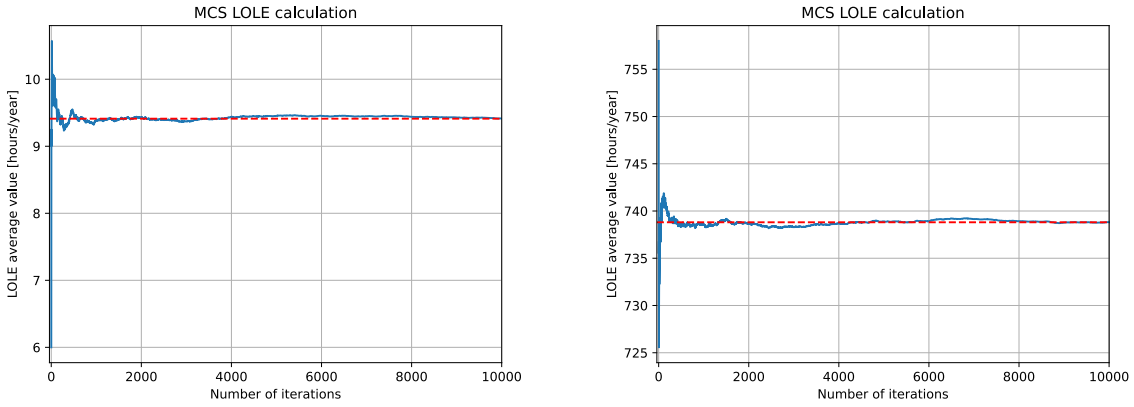
Season	Winter		Summer		Spring / Fall	
Weeks	1-8 & 44-52		18-30		9-17 & 31-43	
Day type / Hour	Weekday	Weekend	Weekday	Weekend	Weekday	Weekend
00.00-01.00	67	78	64	74	63	75
01.00-02.00	63	72	60	70	62	73
02.00-03.00	60	68	58	66	60	69
03.00-04.00	59	66	56	65	58	66
04.00-05.00	59	64	56	64	59	65
05.00-06.00	60	65	58	62	65	65
06.00-07.00	74	66	64	62	72	68
07.00-08.00	86	70	76	66	85	74
08.00-09.00	95	80	87	81	95	83
09.00-10.00	96	88	95	86	99	89
10.00-11.00	96	90	99	91	100	92
11.00-12.00	95	91	100	93	99	94
12.00-13.00	95	90	99	93	93	91
13.00-14.00	95	88	100	92	92	90
14.00-15.00	93	87	100	91	90	90
15.00-16.00	94	87	97	91	88	86
16.00-17.00	99	91	96	92	90	85
17.00-18.00	100	100	96	94	92	88
18.00-19.00	100	99	93	95	96	92
19.00-20.00	96	97	92	95	98	100
20.00-21.00	91	94	92	100	96	97
21.00-22.00	83	92	93	93	90	95
22.00-23.00	73	87	87	88	80	90
23.00-00.00	63	81	72	80	70	85

G

MCS State Sampling Convergence Process for the IEEE RTS

The following figures show the convergence process for LOLE and EENS values for the IEEE RTS using the state sampling method.

G.1. LOLE

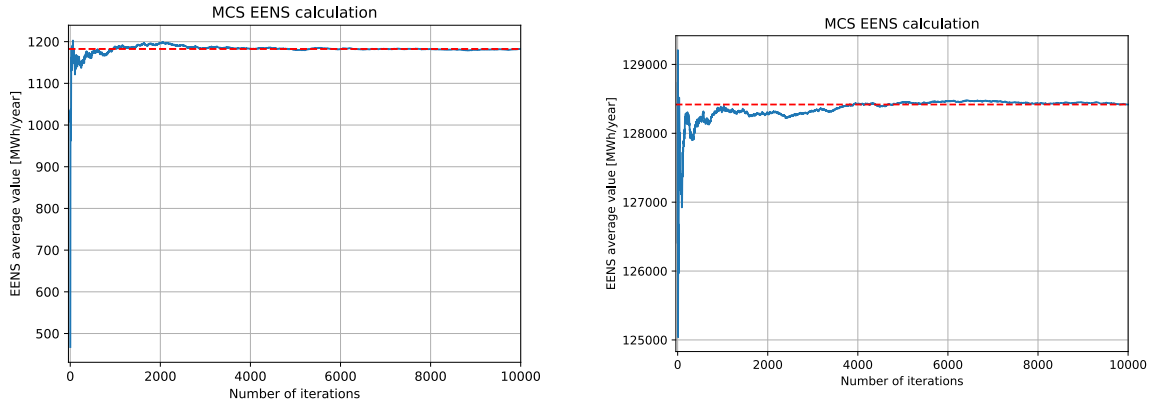


(a) average LOLE value for the HPL model

(b) average LOLE value for the CYPL model

Figure G.1: Converging process for the average LOLE value

G.2. EENS



(a) average EENS value for the HPL model

(b) average EENS value for the CYPL model

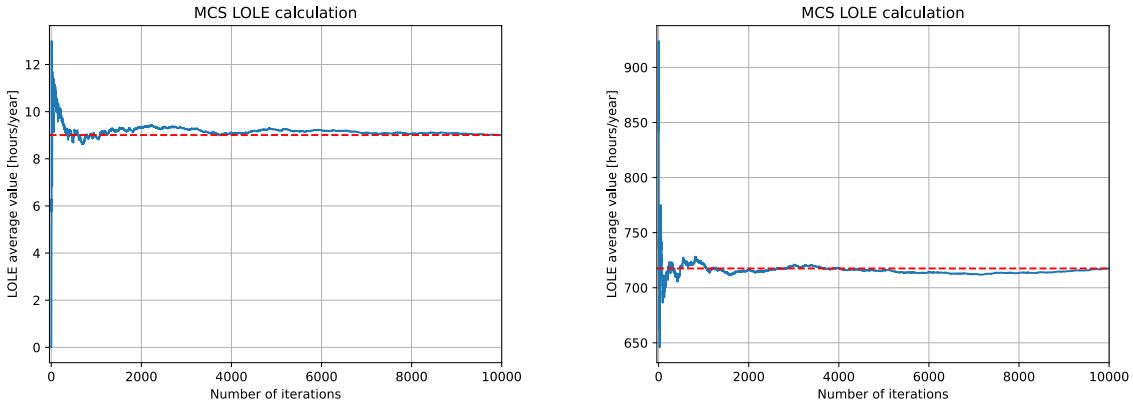
Figure G.2: Converging process for the average EENS value

H

MCS State Duration Convergence Process for the IEEE RTS

The following figures show the convergence process for LOLE and EENS values for the IEEE RTS using the state duration method.

H.1. LOLE

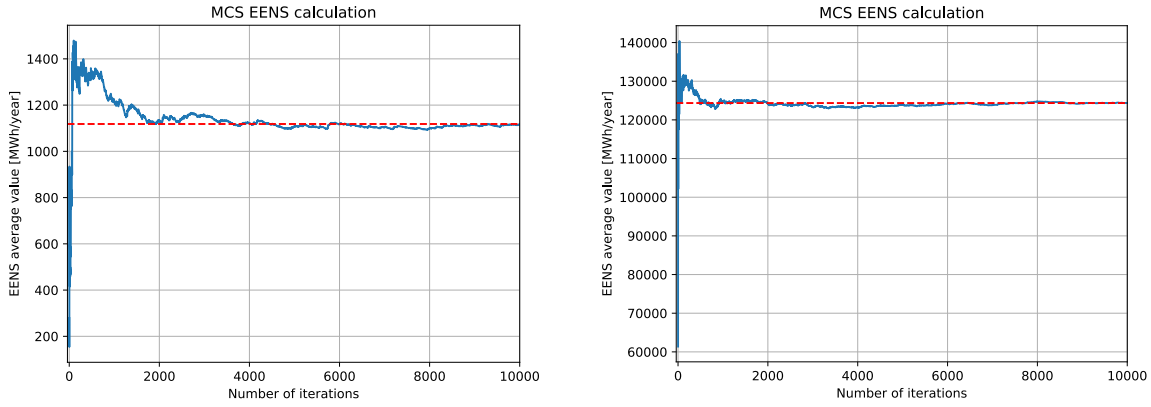


(a) average LOLE value for the HPL model

(b) average LOLE value for the CYPL model

Figure H.1: Converging process for the average LOLE value

H.2. EENS



(a) average EENS value for the HPL model

(b) average EENS value for the CYPL model

Figure H.2: Converging process for the average EENS value