Diploma Thesis

Maintenance Strategies for Large Offshore Wind Farms

Composed by:	cand. tema. Matti Scheu
Matriculation Number:	2371766
Date of Birth:	13th December 1985
Contact:	matti.scheu@gmail.com
Composed at:	[1]Department of Civil and Transport Engineering, Norwegian University of Science and Technology, Trondheim
	[2]Endowed Chair of Wind Energy (Stiftungslehrstuhl Windenergie), University of Stuttgart
Supervised by:	[1]Prof. Michael Muskulus
	[2]Dipl.- Ing. Denis Matha

Stuttgart, April 2012

NTNU
Norwegian University of
Science and Technology

University of Stuttgart
Germany

SWE Stiftungslehrstuhl Windenergie
am Institut für Flugzeugbau

To the most important people in my life.

MY MUM & DAD

MY BROTHER

MY FAMILY

MY CLOSEST FRIENDS

ACKNOWLEDGMENTS

I would like to express my appreciation to my advisory committee of the Norwegian University of Science and Technology (NTNU) and the University of Stuttgart: Prof. Geir Moe†, Prof. Michael Muskulus, Prof. Po Wen Cheng and Dipl.-Ing. Denis Matha.

Prof. Moe sadly passed away during my stay in his former "Department of Civil and Transport Engineering" at the NTNU. Not only have I been inspired by his positive attitude, his great knowledge and idealism.

Special thanks to Prof. Michael Muskulus for all the time and patience he spent during my six month stay in Norway. Due to his extraordinary knowledge, inspiration and own understanding of research practice, we found a great way to work in a very aim-oriented, but also free manner. I am proud to say that it was him who motivated me to establish two publications on international conferences within this thesis.

My gratitude also goes to the "Endowed Chair of Wind Energy (SWE)" of the University of Stuttgart under the direction of Prof. Po Wen Cheng, who enabled me to perform this thesis abroad, and who gave a great support through my second supervisor Denis Matha. I would also like to thank the SWE for all the chances they offered me during my studies, especially in person of Tim Fischer, who took a great part in establishing this Diploma project.

For interesting discussions and knowledge exchange, I thank the SINTEF Energy Research group in person of Matthias Hofmann and Jørn Heggset.

Special thanks also to the European Centre for Medium-Range Weather Forecasts for data support. This data was the basis for a major part of this thesis, the weather model.

I thank all colleagues at both Universities, who assisted me in difficult situations, for their great support. Without you, Matthias Brommundt and Wenjun Lu, I would have spent much more time solving my Matlab problems.

CONTENTS

LIST OF ABBREVIATIONS / NOMENCLATURE

Abbreviations	
CDF	Cumulative distribution function
O	Observed
O&M	Operation and maintenance
OWF	Offshore wind farm
OWT	Offshore wind turbine
S	Simulated
WT	Wind turbine

Nomenclature

Symbol	**Full term**	**Unit**
A	Availability	%
AC	Accessibility	%
CF	Capacity factor	%
COE	Cost of energy	€/kWh
DT	Downtime	h
E_{annual}	Annual energy yield	kWh
FCR	Annual fixed charge rate	%
FIT	Feed in tariff	€/kWh
Hs	Significant wave height	m
ICC	Initial capital cost	€
KS	Kolmogorov-Smirnov statistics (maximum difference between two CDFs)	-
MTBF	Mean time between failures	h
MTTR	Mean time to repair	h
n	Operating years	Year
n_s	Sample size	-
p	Significance probability	-
p_{ij}	Transition probability	-
P_{rated}	Rated power	kW

PL	Production losses	€
$P(x)$	Probability depending on X	-
r	Discount rate	-
r_P	Pearson's correlation coefficient	-
SE	Standard error	-
T	Time	h
T_m	Transition matrix	-
TTF	Time to failure	h
$TTFF$	Time to first failure	h
v	Wind speed	m/s
v_{actual}	Actual wind speed	m/s
v_{rated}	Rated wind speed	m/s
λ	Annual failure rate	a^{-1}
σ	Standard deviation	-

LIST OF FIGURES

LIST OF TABLES

LIST OF PUBLICATIONS

9th Deep Sea Offshore Wind R&D Seminar, 19 - 20 January 2012, Trondheim, Norway

Maintenance strategies for large offshore wind farms

Matti Niclas Scheu[1,2], Denis Matha1, Matthias Hofmann[3], Michael Muskulus[2]

Abstract: "Up to one third of the total cost of energy from offshore wind generation is contributed by operation and maintenance (O&M). Compared to its onshore counterpart, this fraction is significantly higher. Costs are not only caused by spare-parts and repair actions, but also by production losses due to downtime. The accessibility of a turbine in case of a failure is one main aspect affecting downtime. Therefore, a tool has been developed and implemented in MATLAB to simulate the operating phase of a wind farm with special emphasis toward the modeling of failures and repair. As an example application, a site at the UK east coast was chosen, and a few distinct scenarios were considered. Results include how sensitive availability changes with respect to changes in maintenance fleet and maintenance scheduling strategy. A quantification of potential cost savings due to an increase in availability is also stated" [Sch12a].

The International Society of Offshore and Polar Engineers Conference, 17 - 23 June 2012, Rhodes, Greece

Validation of a Markov-based weather model for simulation of O&M for offshore wind farms

Matti Niclas Scheu[1,2], Denis Matha[1], Michael Muskulus[2]

Abstract: "A novel method to model wind and wave conditions for offshore sites based on historical data has been developed, in order to simulate the operating phase of offshore wind farms. Significant wave heights are modeled by a finite state Markov chain in discrete time. Wind speeds are simulated according to their conditional probability distribution, dependent on the corresponding wave height. Results show the accuracy of modeled wind speed and wave height time series, by comparing their second-order statistics, linear correlations and cumulative distribution functions for persistence with observed time series" [Sch12b].

[1] *Endowed Chair of Wind Energy, University Stuttgart, Allmandring 5B, 70569 Stuttgart, Germany*
[2] *Department of Civil and Transport Engineering, Norwegian University of Science and Technology, 7491 Trondheim, Norway*
[3] *SINTEF Energy Research, Sem Sælands vei 11, 7034 Trondheim, Norway*

1 ABSTRACT

Which equipment is needed and how shall tasks be scheduled in order to implement the economically most efficient operation and maintenance strategy for large offshore wind farms? This is the question motivating this research project. Considering production losses due to turbine downtime as well as local geographical and weather conditions, an efficient operation and maintenance (O&M) solution shall be achieved for two reference sites at the UK east coast.

For this purpose, a Matlab-based tool has been developed, consisting of the following five main modules: Weather, Failures, Resources, Strategy and Cost.

The "Weather" module is able to generate future sea states and wind speeds based on historical data. It uses a finite state Markov chain in discrete time to model significant wave heights. Wind speeds are then generated according to their conditional probability distribution at the corresponding wave height. In order to validate the weather module, several time series were generated and compared with existing data. For comparison, the mean values, standard errors, linear correlations and cumulative distribution functions for persistence of operational weather windows were chosen, both for synthetic and observed wind speed and wave height time series. Both reference sites in the UK North Sea were considered for validation.

Failure rates are the basis for the "Failure" module. As an input, data gathered from onshore reliability investigations are used, which can be updated once more detailed data is available for offshore turbines. The outcomes of this module are turbine-failures occurring at a certain time.

Within the “Resources” module, it is defined which equipment and personnel is available for O&M activities. The equipment is specified by its technical characteristics, e.g., the maximum transportable personnel and the operational wave height boundary.

Another key parameter is the "Strategy". The main goal of this module is to take the decision whether to perform an operation or not. Within this thesis, one specific strategy has been used, but references are made to possible modifications in the according paragraphs.

The measurement of economic performance is done in the "Cost" module. Here, production losses are quantified by combining the wind speed during a failure with the linearized power curve of the turbine and the local buyback price system. Therefore, the worth of additional or better maintenance equipment can be seen directly as an increase in availability and a decrease of production losses.

Results show how sensitive availability and therefore production losses change with respect to changes in the maintenance fleet, reliability characteristics of components and distance to shore. Major improvements of availability were achieved by applying maintenance vessels with a higher operational wave height boundary. An increase of this constraint from 1.0 m to 1.8 m significant wave height raised the availability by up to 30 percent, leading to a much better economic performance. The influence of the weather forecast accuracy on the number of maintenance vessel and crane deployments is also stated, showing a more efficient equipment deployment if the weather forecast is accurate for longer times. An improvement of component-reliability, modeled as a 50% decreased annual failure rate, could save up to 440 k€ of yearly production losses for each modeled wind turbine. Higher transit times, due to a greater distance to shore, strongly decrease the wind park availability.

2 INTRODUCTION

The offshore wind power market has been growing strongly over the last years. The installed offshore capacity in Europe is planned to increase up to 40-55 GW in 2020 [Ewe09]. Many advantages, like stronger and more constant wind, argue for installing wind farms off the shore. Anyhow, the investment in an offshore wind project involves high financial risks, mainly due to strongly varying operational costs. A lack of experience in this branch and low reliability and availability levels are the main reasons [Fen10]. Costs for operation and maintenance (O&M) are contributing up to 30% to the total cost of energy (COE) [Bus97]. These costs are not only caused by spare-parts and repair actions, but also by production losses due to downtime. The accessibility of a turbine in case of a failure is one main aspect affecting downtime. Even losses of a few percent can be critical, due to the marginal economic nature of offshore wind energy projects.

These challenges are addressed within this thesis by evaluating the experience gained from existing offshore wind parks, and by proposing a method which allows for finding more optimal solutions for future projects. This method is implemented in a Matlab-based tool, enabling to simulate the operating phase of offshore wind farms. The tool attempts to evaluate global parameters, such as the park availability and park production losses. In order to cover uncertainties, e.g., changing weather conditions in different years, a accurate probabilistic model has been set up. Applied strategies have to pass severe scenarios, as none simulated year equals another. A relatively quick simulation time (about 5 minutes for a simulation time of one year), allows for a great number of variations of different parameter.

The next chapter attempts to clarify basic terms and definitions used in the main body of this document. Several commonly applied maintenance strategies are explained in order to ease the understanding of the strategy applied in the O&M optimization tool. In chapter 3.3, experiences gained from UK round I wind parks[4] are shown. The importance of optimizing the operational phase of offshore wind farms is highlighted by comparing economical key parameters, e.g., the cost of energy, for four different parks. After discussing accessibility issues in the Horns Rev wind park, different means of access and their characteristics are presented. Practical O&M experience is concluded by assessing reliability data.

Starting with the basic principles of weather modeling, the methods implemented in the simulation tool are then explained in chapter 4. A major part is the validation of modeled wind speed and wave height time series at two reference sites. Validation with measured data shows a high level of accuracy. The description of the tool is completed by explaining the applied maintenance strategy.

The studies that were performed, respectively their results, are presented in chapter 5. All different parameter used for simulation are listed in order to clarify the simulation basis. Results show the effect of different levels of wave height constraints of maintenance equipment and different maintenance fleet compositions on availability. Significant differences between both investigated sites are shown; mainly in the magnitude of achieved availabilities. The effect of a change in reliability characteristics of the components is shown, as well as a study dealing with the distance from the wind park to shore. The effect of the accuracy of weather forecasts is shown in terms of the numbers of deployments of the maintenance equipment. Furthermore, results are quantified monetarily in order to get a sense for the effect of changes in the maintenance fleet on the global economic performance of an offshore wind farm.

Methods, results and possible future tasks are discussed and concluded in chapters 6 and 7.

The appendix of this document contains a user manual for executing all proposed methods and studies in Matlab. An instruction of how to model wind speeds, wave heights and the complete operation phase of an offshore wind farm is given. The documentation is supported by the complete source code.

[4] *http://www.bwea.com/offshore/round1.html(7.02.2012)*

3 PRINCIPLES OF OPERATIONS AND MAINTENANCE

In the following paragraphs, relevant terms are explained in order to enable a quick understanding of the main body of this thesis. First, different O&M strategies are discussed. Next, the terms "Availability", "Accessibility", "Capacity Factor", "Cost of Energy", "Fixed Charge Rate", "Feed in Tariff", "Failure Rate" and "Production Losses" are defined. Operation and maintenance in practice is considered by evaluating experiences gained from UK round I wind farms and by describing the basics of offshore access methods and turbine reliability characteristics.

3.1 STRATEGIES

In general, maintenance strategies can be divided into preventive and corrective maintenance. Preventive maintenance strategies aim to prevent component or system failures. Corrective maintenance strategies are required if a component or system has already failed. As shown in Figure 1, preventive maintenance can be split up in calendar-, respectively, condition-based maintenance. Corrective maintenance is either planned or unplanned [Obd07].

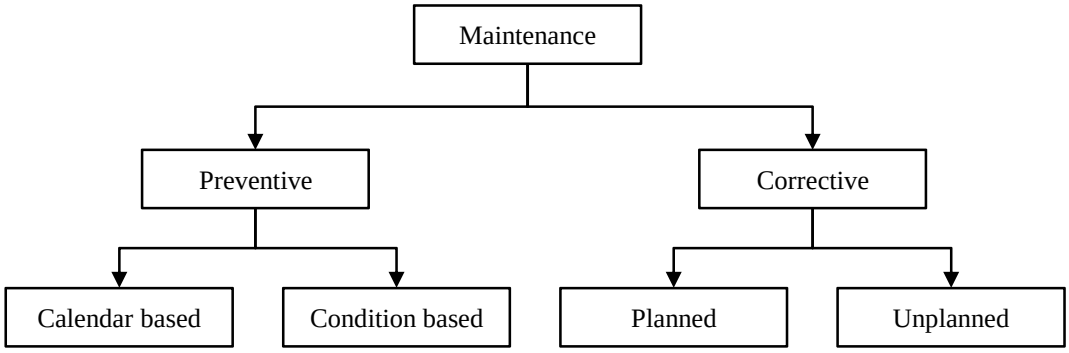


Figure 1: Basic maintenance strategies - overview [Obd07]

Calendar-based maintenance is executed after certain time intervals or a defined number of operating hours. For offshore applications, this service is usually done twice a year [Bus01a]. Condition-based maintenance and planned-corrective maintenance are initiated depending on the status of the system. This status or degradation of a system can, for instance, be supervised by remote monitoring systems [Mcg08]. Planned corrective maintenance is applied for components that are expected to fail during their operational lifetime. It is closely related to condition-based preventive maintenance in practice, with the only difference that the latter is considered in the design from the beginning. Unplanned corrective maintenance is applied if a system, which is not expected to fail (as the tower or substructure of a turbine), fails and repair has to be carried out [Obd07]. Planned and unplanned corrective maintenance is the focus of this research project.

3.2 TERMS & DEFINITIONS

The following paragraphs contain definitions and mathematical expressions of selected terms used within this thesis.

The technical availability (A) is the fraction of time during which a system meets its demands in relation to a defined time interval (usually one year). Mathematically, it can be expressed as follows [Bus03]:

$$A = \frac{T_{running}}{T_{running} + T_{down}} * 100\ [\%] \quad \textbf{(3-1)}$$

Accessibility (AC) can be expressed as the fraction of time a turbine can be accessed, divided by the considered time interval (usually one year) [Fau11]:

$$AC = \frac{T_{accessible}}{T_{accessible} + T_{unaccessible}} * 100\ [\%] \quad \textbf{(3-2)}$$

A commonly used term to describe the productivity of a wind turbine or farm is the "Capacity Factor" (CF). It is defined as the percentage of the achieved annual energy production over the maximum (theoretically possible) annual energy production determined by the rated power [Fen10].

$$CF = \frac{E_{annual}}{P_{rated} * 8760\ h} * 100\ [\%] \quad \textbf{(3-3)}$$

The fixed charge rate (FCR) is used for discounting investments. It is calculated with the discount rate *r* (sum of real interest rate and inflation) and the expected number of operating years *n* [Fen10].

$$FCR = \frac{r}{[1 - (1 + r)^{-n}]} * 100\ [\%] \quad \textbf{(3-4)}$$

Under the term "Cost of Energy" (COE), the expected yearly average costs a wind turbine causes over its lifetime are composed and divided by the expected mean annual energy production [Fau11]. Yearly costs are the sum of the initial capital cost (ICC) discounted by the fixed charge rate (FCR) and annual cost for operation and maintenance [Fen10].

$$COE = \frac{ICC * FCR + O\&M_{cost}}{E_{expected}} \left[\frac{€}{kWh}\right] \quad \textbf{(3-5)}$$

The "Feed in Tariff" (FIT) determines the amount of money a supplier or utility has to pay per kWh of renewable electricity to the operator of a wind farm. It is aimed to represent the COE preferably accurate with the FIT in order to avoid losses for the operator, but to also limit the cost of renewable energy.

The failure rate λ expresses the mean number of failures that a system experiences in a certain time interval. Commonly used, and also applied for this thesis, is the annual failure rate. Mathematically, it is defined as the reciprocal of the mean time between failures (MTBF) [Bus01b]. The mean time to repair (MTTR) defines the average downtime a failed component or system causes after it has failed [Bla04].

$$\lambda = \frac{1}{MTBF} \left[\frac{failures}{year}\right] \quad \textbf{(3-6)}$$

A number to quantify the potential income, which is not obtained due to downtime, is the "Production Loss" (PL). For the purpose of this thesis, monetary PL is calculated by combining the current wind speed with a linearized power curve and the local FIT during periods a turbine is not converting energy.

$$PL = FIT * \int_{t_1}^{t_2} v(t) * P\big(v(t)\big)\, dt\ [€] \quad \textbf{(3-7)}$$

3.3 OPERATIONS AND MAINTENANCE IN PRACTICE

In this chapter, the performance of currently operating offshore wind farms is briefly evaluated by discussing selected key parameters. Main challenges are distinguished by highlighting major draw-backs in the economic performance of those wind parks. As reference, mainly UK round I offshore wind farms and the Horns Rev wind park are considered, as sufficient data is publicly available for these [Fen10,Ves05]. Turbine reliability data is extracted from an onshore survey [Fau11].

3.3.1 EXPERIENCE FROM UK ROUND I WIND PARKS

At present, the most attractive FIT for offshore wind generation in Europe has been proposed by the UK government, with a value of 0.181 €/kWh. Considering entire Europe, this buyback pricing system for renewable energy causes the biggest growth of the offshore wind energy market in the United Kingdom [Koe10]. Due to the significance of this market, it is concentrated on wind parks in the UK in this chapter.

UK round I offshore wind parks North Hoyle and Barrow achieved availabilities of 87.7%, respectively 67.4%, on average in their first years of operation from 2004 to 2007. For onshore application, availabilities of 98 % and more can be assumed. O&M costs for North Hoyle have been 26 €/kWh, whereas they were only 14 €/kWh at Barrow. The COE was 80 €/MWh for North Hoyle and 102 €/MWh for Barrow[5]. Both numbers lead to the assumption that the high COE seen in the Barrow wind farm is mainly caused by significant downtime losses. Low O&M costs indicate a low number of failures, respectively, a low number of O&M offshore deployments. The high COE in the Barrow wind farm alludes to a low energy yield, respectively, capacity factor, caused either by low wind speeds, or by a high downtime level. In fact, a gearbox replacement taking place from July to October 2007 caused a stop of the turbines. Pitch system and gearbox issues also contributed to a low CF of only 24.1% in the Barrow wind farm [Fen10]. It can be concluded that downtime due to non-reliability of components leads to a low availability and thereby increases the cost of energy significantly.

3.3.2 ACCESS SYSTEMS

Access systems can be grouped into three main categories. Ordinary maintenance vessels for transport of crew, tools and small spare parts, crane vessels with heavy lifting equipment, and helicopters. Recently proposed innovative systems such as the "Ampelmann" concept are taken into account [Tem05]. Major differences between these means of transport are their capacity and their operational limits. Table 1 compares operational limits of ordinary vessels, advanced access systems, and helicopters.

Table 1: Offshore access systems and their operational limits [Bus03]

Access system	**Wave height boundary [m]**	**Wind speed boundary [m/s]**
Ordinary Vessel	1.5	10
Advanced access system	2.5	11.5
Helicopter	-	20

An ordinary vessel can be understood as a boat from which the personnel has to jump on a ladder at the turbine in order to access it. For safety restrictions, significant wave heights can only be handled up to 1.5 m. Advanced systems (such as the “Ampelmann”) could bear with wave heights up to 2.5 m and also higher wind

[5] *Exchange rate from GBP to EUR from the 29th January 2012 (1 GBP ~ 1.19 EUR)*

speeds. So-called "SWATH" catamarans are even useable up to 3.5 m significant wave height [Lóp10]. Helicopters are primarily not affected by wind and waves (although snow and fog can be an issue), but involve high operational expenses [Bus03]. For the purpose of this study, ordinary maintenance vessels, crane boats and advanced access systems [Tem05] are taken into account. Figure 2 shows examples of an ordinary maintenance vessel[6] (left), a crane vessel[7] (middle) and an Ampelmann[8] access system (right).

Figure 2: Offshore access systems

For all access systems, either a port at land or an offshore base has to be available. Smaller projects and ones being close to shore usually rely on a port at the coastline. For big parks, located farther offshore, an offshore base can be an interesting alternative, as transit times and costs might be too high with support from a port. Proposals include, for instance, the adaption of a heavy-lifting jack-up barge to an offshore maintenance base [Ewe09].

The influence of accessibility on availability can clearly be seen in the case of the Horns Rev wind park completed in 2002 and located in the Danish North Sea. The wind farm consists of 80 Vestas V80 turbines of 2 MW size. Due to gearbox issues, it was decided to replace all nacelles in the summer of 2004 [Ves05]. Harsh weather conditions led to scarce accessibility and therefore a downtime of several months, causing heavy economical losses, mainly for the manufacturer Vestas, who had granted a 5-year guarantee.

3.3.3 RELIABILITY

All turbine downtimes (DT) are initially caused by non-reliability of turbines, respectively, their subsystems and components. The consequence is a decreased availability and therefore a lower monetary income for the operator. The following paragraph shows currently achieved reliability levels for onshore wind turbines. An outlook for turbines deployed offshore is also given. Onshore reliability data has been gathered from an ongoing onshore reliability study performed at the Fraunhofer Institute for Wind Energy and Energy System Technology [Fau11]. In accordance with that study, turbine failures for 12 subsystems were considered. All considered systems and their according annual failure rate and average downtime are summarized in Table 2. According to this survey, the electrical system and electronic control are causing by far the most failures, with an average of one failure per year. Mechanical subassemblies fail less frequently, but cause longer downtimes if they do. Combining the failure rate with the downtime, it can be seen that gearbox failures cause higher mean annual downtimes than sensors, even if sensors fail 2.5 times more often. On the average, a turbine fails about twice a year.

[6] *http://www.renewbl.com (28.01.2012)*

[7] *http://www.offshorewind.biz (28.01.2012)*

[8] *http://www.frs-offshore.de (28.01.2012)*

Table 2: Annual failure rate and downtime of different subsystems [Fau11]

	λ [failures/year]	DT [days]
Electrical System	0.57	1.53
Electronic Control	0.43	1.59
Sensors	0.25	1.41
Hydraulic System	0.23	1.36
Yaw System	0.18	2.70
Rotor Hub	0.17	3.71
Mechanical Brake	0.13	2.89
Rotor Blades	0.11	2.60
Gearbox	0.10	6.21
Generator	0.11	5.39
Support & Housing	0.10	4.90
Drive Train	0.05	5.71

Note however that due to teething problems in the beginning, and endurance limits towards the end of a turbine's lifetime, the failure frequency is higher in these periods [Bli00]. Also, an increase of failure rates in high wind speed regions can be observed. In these regions, especially for offshore applications, the accessibility of a turbine can become critical for a long period of time, e.g., in stormy winter months. The level of reliability achieved nowadays for onshore turbines is therefore too low for offshore applications. High offshore transit costs also demand for a highly reliable design of offshore wind turbines (OWT) [Fau11].

4 A Simulation Tool for Operation and Maintenance Studies

This chapter outlines the general methodology of wind speed and wave height modeling as well as the applied method for failure modeling and the implemented maintenance strategy for a reference offshore wind park. After showing examples of existing tools for the simulation of operation and maintenance from offshore wind farms, a detailed introduction of the simulation-functionality of the tool developed within this thesis is given.

4.1 Review of Existing Tools for Simulation of Operation and Maintenance

Many tools attempting to simulate the operating phase of offshore wind farms have been developed in the past. While many of them are available to the public, others are undisclosed due to commercial interests. Two famous research institutions handling with O&M tools are the Energy Research Centre of the Netherlands (ECN) and the Delft University of Technology. The tools CONTOFAX, RECOFF and the ECN O&M Tool have been developed at ECN. Further research institutions and businesses handling with O&M are mentioned in [Hof11]. Within this reference, some thirteen tools focusing mainly on operation and maintenance are presented, which include well-known names such as Garrad Hassan, the BMT Group Ltd., Det Norske Veritas (DNV) and the National Renewable Energy Laboratory (NREL).

All tools are used to simulate and/or to optimize the operating phase of offshore wind farms in the long-term. Most of the available tools use costs or production losses during downtime to quantify results. The ECN

O&M Tool (ECNT) [Rad08] has already been commercialized and is assumed to be one of the leading tools on the market. Therefore, it is explained in more detail and several aspects are compared to the tool developed within this thesis project.

The ECNT has been developed in cooperation between three partners, namely the ECN, Germanischer Lloyd Industrial Services GmbH Business Segment Wind Energy (GL Wind) and Hochschule für Angewandte Wissenschaften (HAW) Hamburg. Three different types of maintenance are considered in the ECNT: calendar-based, condition-based and unplanned corrective maintenance. The latter is considered to be the most difficult to predict since it contains the most uncertainties. Unplanned corrective maintenance is the only type of maintenance studied with the tool developed here. In contrast to the ECNT, the first two types above are disregarded, but could be added to the model. Both tools use annual failure rates for determining the occurrence of failures. In contrast to the tool developed here, the ECNT considers costs for repair containing labor, material and ship costs. Production losses during downtime are taken into account by in both models. Both tools mainly focus on parameter variations regarding maintenance access systems, i.e., their technical characteristics.

Especially in terms of weather modeling, the tool presented within this thesis differs from all other software known to the author. In the ECNT, wave heights are simulated with a statistical approach using spectral analysis. Wind speeds, as also done in the tool developed here, are assumed to correlate with the actual wave height. The applicability of the Markov theory on wave height modeling has been shown before [Ana96], but the application in relation to offshore wind energy is new. The accuracy of the model has been validated in detail (see section 4.3) and is assumed to be a solid base for further wind park simulation. The weather model can be used independently from the remaining tool and can therefore be easily combined with other tools.

In contrast to others, the tool developed here is not available with a graphical user interface yet. This allows for many changes potential users or developers want to apply, but obviously compromises user friendliness. As the tool is publically available (see Appendix), these changes and modifications can be applied easily using the widespread software Matlab.

4.2 Simulation Details

The tool simulates the operating phase of offshore wind farms in discrete time with a resolution of six hours. Basis for generating synthetic wind speeds and wave heights at the corresponding discrete time steps, is a probabilistic model, described in detail in chapter 4.1. For simulation, different objects, states and information are defined. Figure 3 overviews all objects handled within the simulation tool. Objects are *"Crew"* (1), *"Maintenance Vessel"* (2), *"Crane Vessel"* (3), *"Turbine"* (4) and *"Components"* (5). All objects are characterized by a state and by general information. The basis for the simulation is the “state” of an object, which defines its condition. If, e.g., a component fails, its state changes from *"running"* to *"down"*, which means that the turbine cannot produce electricity until all components are *“running”* again. General information can be static (e.g. rated power of a turbine, maximum capacity of a boat), or dynamic (e.g. number of crews assigned to a vessel, next turbine to be repaired). Static information is defined before the actual simulation starts, dynamic values are initiated in the beginning, and change during simulation.

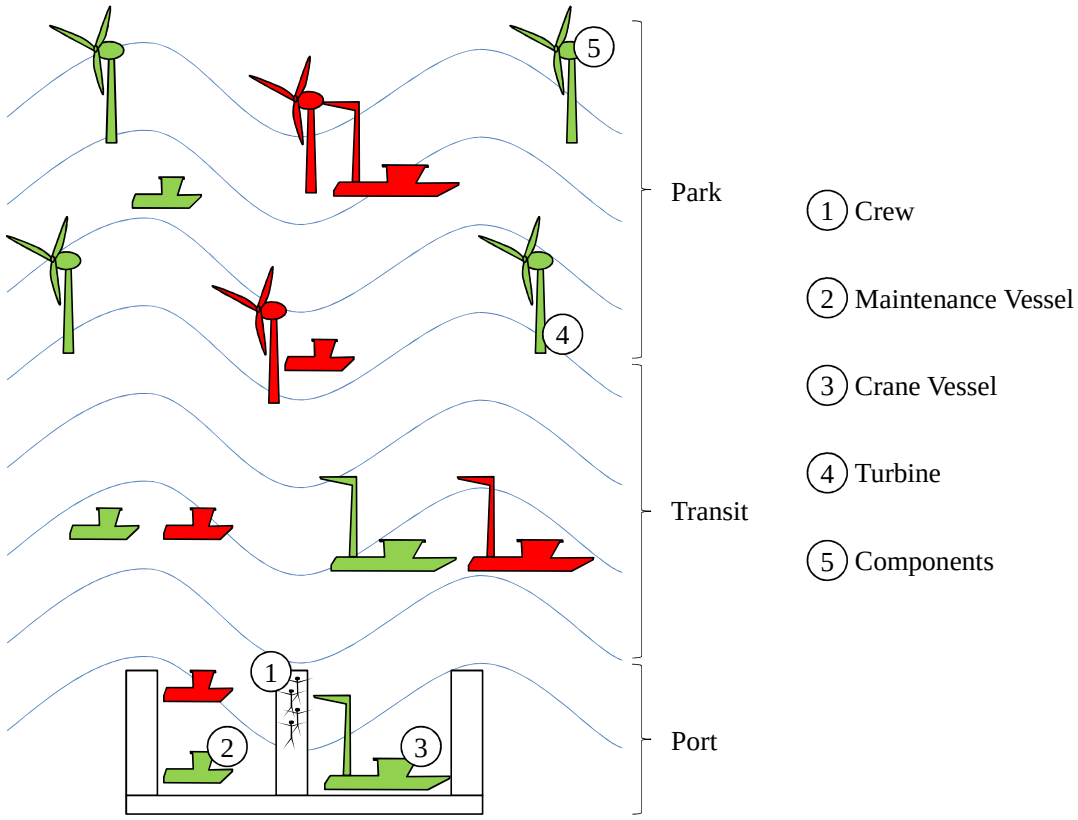


Figure 3: Overview: Objects handled in simulation tool

Table 3 to Table 7 give an overview of the information and all possible states of each object:

Table 3: States and information of crew

Crew	
State:	**Information:**
(a) In Repair	(1) Crew ID
(b) Waiting	(2) No. People in Crew
(c) In Ship	(3) Time Repair Done
(d) Reserved	(4) At which Component
	(5) At which Turbine
	(6) Next Component
	(7) Next Turbine

Crews are dynamically created if required, respectively suspended as soon as they are not needed anymore. A crew can either be created onshore (if repair of a component is required and shall be solved by a new crew from the harbor), or offshore (if a crew finished a task, returned to the vessel and changed its state to *"Personnel without Task"*– explanation at the end of this paragraph). A crew can either be in the state *"In Repair"*, *"Waiting"*, *"In Ship"* or *"Reserved"*. A crew can be in state (a), if it is dropped off at a turbine for repair of a component it is reserved for. As soon as a crew is established, it is *"Reserved"* for repair of a

certain component at a certain turbine. If it finishes repair and returns on the boat (always the same boat it came from – boats can only handle their own crews), the state of the crew changes to *"Personnel without Task"*, from which a new crew could be established offshore. If a crew is reserved for repair, but the vessel it shall be carried with is waiting in harbor, e.g., because of bad weather conditions, it is in state (b). A crew is *"In Ship"* as long it is in transit to or from the park. State (d) is activated as soon as a crew is reserved for repair.

All crews are distinguished by a so-called *"Crew ID"* (1), a simple identification number (ID), used to identify where the required crew information can be found. Vessels use the Crew ID to figure out which crews are assigned for transport. Information item (2) can be used to figure out how many people are loaded on a boat, in order not to exceed transportation limits, which are defined in advance. The information of the time a crew is busy until it finishes the repair of a component, is given in (3) – see also chapter 4.4. Information items (4) and (5) define at which component and which turbine a crew is busy with repair. In information item (6) and (7), the next turbine and component a crew is reserved for is defined. If the crew is not reserved for further repair, these information remain unspecified.

Table 4: States and information of an ordinary maintenance vessel

Ordinary Maintenance Vessel	
State:	**Information:**
(a) Available Harbor	(1) Max. People to Carry
(b) Reserved Harbor	(2) No. People on Board
(c) Transit To	(3) No. Free People on Board
(d) Transit Back	(4) No. People Loaded
(e) Busy Park	(5) No. of Crews
	(6) Crew IDs
	(7) Transit Time
	(8) Potential Transit Time
	(9) Time Offshore
	(10) Max. Significant Wave Height
	(11) Estimated Time Free
	(12) Estimated Weather Window Needed

Under the term "Ordinary Maintenance Vessel", a ship being able to transport crews and minor spare parts is understood here. To a ship located in the harbor, which is *"Available Harbor"* (a), no crew is assigned for repair. As soon as a crew assigns for repair to a ship in harbor, the ship turns to the state *"Reserved Harbor"* (b). As soon as weather conditions allow for, a reserved ship located in harbor goes into state (c), the transit to park. A ship, which leaves the park in order to return to the port is in the state *"Transit Back"* (d). That happens either because all crews assigned to the ship finished their tasks, because of bad weather conditions, or due to the exceedance of the so-called *"Maximum Time Offshore"*, a term, which limits the time a ship, respectively a crew, is allowed to work offshore at a stretch, which is necessary to consider effectual employment laws. A ship is in state *"Busy Park"* (e), as long as it drops and collects crews at turbines, or is waiting for crews, in the park. It is possible, that a ship contains personnel without tasks, e.g., because the repair-task of one assigned crew takes more time than the repair-task of another assigned crew – crews which finished their tasks and returned to the vessel dissipate into *"Personnel without Task"* or *"Free People"*. During that period, it is possible that a crew is established offshore if a new failure occurs, in order to prevent another vessel (if existent) to be deployed from harbor and to therefore hold transit times on a preferably low level.

A ship has information about the maximum number of people it can carry (1), in order to respect transportation limits. Combining the number of people on board (2), with the number of free people on board (3), informs the ship if a new crew could be established from the personnel loaded. Information item (4) is used to record the number of people that has been loaded on the boat in harbor. Before a ship goes into the transit back to harbor, it compares the number of people initially loaded with the number of people on board, in order to see if all crews have been collected. The number of crews (5) and the *"Crew IDs"* (6), are information on which crews are handled by a vessel. The crew information can then be evaluated in order to know which turbine or component has to be repaired next, respectively, which component at which turbine has been repaired. In information item *"Transit Time"* (7), the transit time counts down every time step a vessel is in state (c) or (d) until zero. Then, depending on the actual direction of the vessel, it either reaches the park, or the port. The *"Potential Transit Time"* (8) defines the value the actual transit time starts to count from. Information item (8) enables the variation of distances to the evaluated wind park. This item can also be used to test the effect of faster vessels on the wind park performance. As mentioned before, there is a maximum time a crew, respectively a ship, is allowed to operate offshore. This time is counted in information field (9), the "*Time Offshore"*. It is set to zero before a vessel goes into transit to park. From the transit to park, until it returns to the port, a value of one is added each time step. If the maximum time offshore is exceeded, the ship initiates an interruption of all ongoing repairs, collects all assigned crews, and returns to the harbor. The wave height boundary is defined by information (10). It is the only restriction considered for offshore operations within this study. If a ship is in harbor, it checks the weather forecast against its wave height boundary. If, in the time the ship is expected to be offshore, which is defined by the *"Estimated Weather Window"* (12) (assumed transit time + longest assumed repair time), the wave height boundary is exceeded, the ship stays reserved in harbor. If the weather forecast is not available for the duration defined in information item (12), the weather-check is done for the maximum length of the forecast. If a ship is busy in park, it remains until all tasks are done, the significant wave height exceeds the ship's limits, or the maximum time offshore is exceeded. The *"Estimated Time Free"* (11) represents the longest repair time a crew assigned to a ship located in park expects. If the ship is located in the port, the transit time has to be added. The estimated time free is mainly used in order to find out if a vessel located in park can carry out a demanded repair, before a ship in harbor can do the same. This function is mainly implemented in order to decrease the number of vessel deployments.

Table 5: States and information of a crane vessel

Crane Vessel	
State:	**Information:**
(a) Available Harbor	(1) At which Turbine
(b) Reserved Harbor	(2) At which Component
(c) Transit To	(3) Next Turbine
(d) Transit Back	(4) Next Component
(e) In Repair	(5) Time to Repair
	(6) Transit Time
	(7) Potential Transit Time
	(8) Time Offshore
	(9) Max. Significant Wave Height
	(10) Estimated Weather Window Needed
	(11) Estimated Time Free

Crane vessels can be in the same states as ordinary maintenance vessels, except state (e), which is called *"In Repair"* for crane vessels. Crane vessels are assumed to handle big spare parts, such as generators or blades.

Therefore, for this study, crane vessels have to return to the port after every repair action, in order to load new spare parts for their next repair. During repair, in contrast to ordinary maintenance vessels, crane vessels always have to be stationary at the turbine they are repairing. Hence, the information needed for scheduling crane boats is a combination from crew information and the information used for ordinary maintenance vessels.

Crane vessels use the information at which turbine they are (1), in order to “inform” the component (2) being repaired about its remaining repair time (see component information, Table 7). In harbor, the information of which component (3) at which turbine (4) is going to be repaired, is necessary to schedule the ships’ offshore operations. The *"Time to Repair"* (5) defines the duration a crane vessel is expected to take until a repair it is reserved for is completed. In harbor, it is composed of the assumed transit time and the assumed repair time. Information items (6) to (11) behave analogously to (7) to (12) from an ordinary maintenance vessel.

Table 6: States and information of a turbine

Turbine	
State:	**Information:**
-	(1) Rated Power
	(2) Cut-In Wind Speed
	(3) Rated Wind Speed
	(4) Cut-Out Wind Speed
	(5) Turbine Name
	(6) Repair Time
	(7) No. Unreserved Failures

As explained later in detail (chapter 5), the rated power, cut-in-, rated-, and cut-out wind speed of a turbine are used to linaerize its power curve and to be able to calculate production losses during down time. The *"Turbine Name"* (5), is solely used for identification of a turbine. The *"Repair Time"* (6) is zero, as long as all components are running. As soon as a component breaks, it adds its assumed repair time to the repair time of the turbine. The number of unreserved failures (7) is convenient information in order to efficiently find out which component needs repair; not all components of all turbines have to be checked in every time step, but only the ones which are located in turbines with unreserved failures.

Table 7: States and information of a component

Component	
State:	**Information:**
(a) Running	(1) Name
(b) Down	(2) Turbine
(c) Unreserved	(3) Time to Failure
(d) Reserved	(4) Repair Time
	(5) People Required
	(6) Crew ID
	(7) Category

A component can either be in the state *"Running"*, *"Down"*, *"Unreserved"* or *"Reserved"*. Initially, all components are running. They break down, as soon as the so-called *"Time to Failure"* (3) is zero. The time to failure is determined by a probabilistic model, explained in more detail in chapter 4.4. As soon as (3) is zero, the assumed repair time for the component is set (4). In case of a minor repair (see information (7)), a number

indicating how many people are required for repair is set (5). For big components, which are assumed to require a crane boat for repair, this information item remains empty, because crane vessels can only handle one repair in each offshore deployment, and it is assumed that the number of people required is equal to the number of people loaded on the crane vessel before the transit to park. Which crew is reserved for repair of a component is stored in information item (6). The *"Category"* of a component is used to identify whether it can be repaired by a crew only, or if it requires a crane vessel. For the purpose of this study, the categorization of the components follows Table 8, and all components shown in chapter 3.3.3 are considered. The respective assumed repair time of each component is indicated in brackets behind the component name. All assumed repair times are parametric; the following estimated values indicate those which have been used for the studies carried out within this thesis.

Table 8: Categorization of components

Maintenance Vessel + Crew:	**Crane Vessel:**
Electrical System (12 h)	Rotor Hub (24 h)
Electronic Control (12 h)	Rotor Blades (36 h)
Sensors (12 h)	Gearbox (36 h)
Hydraulic System (18 h)	Generator (24 h)
Yaw System (18 h)	Support & Housing (24 h)
Mechanical Brake (18 h)	Drive Train (24 h)

The number of people assumed to be required for repair of a component which can be repaired without crane is two. According to repair and transit times, this parametric value can be varied for any purpose of a study, but is held fixed for all simulations carried out within this thesis.

4.3 WEATHER MODELING

In order to simulate the operating phase of an offshore wind farm, an accurate representation of weather conditions is crucial, as important performance parameter, such as accessibility and availability, mainly depend on these conditions. By far the most important limiting condition is the wave height, as maintenance or crane vessels can only operate up to a certain limit. According to Monbet and Marteau [Mon01], there are three main classes of methods that are able to generate realistic synthetic sea state time series: One can consider simulations based on Gaussian statistics, autoregressive-moving average (ARMA) processes, and stochastic processes that assume the Markov property.

Gaussian methods solely rely on the probability density function of a parameter. This function defines which function value can be expected (mean value) and how much variation around this (quantified by the variance, or its square-root, the standard variation) is present [Gra09]. In order to implement a process based on Gaussian statistics into a simulation, a random number has to be drawn in every simulation step. Coupling this number with the probability density function of a parameter then leads to the function outcome. Consequently, values for distinct time steps are independent of each other, leading to an unrealistic representation of weather conditions, as long term weather phenomena, such as the seasonality of weather conditions, cannot be displayed. In long term simulations, the mean values and standard deviations of a sample can be represented very well. In terms of representing a realistic persistence of a parameter, e.g. of a certain wave height level, Gaussian-based methods are, however, not applicable and are therefore not taken into account for weather modeling.

Due to the simplicity of Markov processes, and the ability to model weather phenomena with realistic persistence, a Markov process for the simulation of significant wave height time series was chosen. The

applicability of Markov processes for accurate weather modeling with respect to persistence statistics has been validated in detail by Anastasiou and Tsekos in 1996 [Ana96]. Wind speed time series were here derived according to their conditional probability distribution, conditioned on the corresponding wave height. The following paragraph outlines the basics of applying a discrete-time Markov chain.

Markov chains are used to describe stochastic processes with finite memory. Processes, generated based on a finite number of state-observations, are the main scope of application. As shown in this thesis, these states can be historical wave height time series which are used to generate future sea states.

For practical reasons, the memory of a Markov process is typically limited to the last state of the system. This is obviously a great simplification, but allows for a simple and efficient method. Since measurements are naturally sampled at discrete time points, it is most natural to also consider a Markov chain in discrete time [Kem76]. Different states, representing different values of significant wave height, were defined and characterized by transition probabilities. No constraints on the allowable transitions were imposed, in contrast to the approach of Rothkopf et al., in which a tridiagonal transition matrix is proposed [Rot74]. The transition probabilities were derived directly from the data, which allows for working with data sampled at arbitrary time intervals. The transition probabilities (p_{ij}) were estimated by the average frequencies of transitions observed in the data, and can be conveniently displayed in matrix form:

$$T_M = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1s} \\ p_{21} & p_{22} & \cdots & p_{2s} \\ \cdots & \cdots & \cdots & \cdots \\ p_{s1} & p_{s2} & \cdots & p_{ss} \end{bmatrix} \quad \textbf{(4-1)}$$

The probability p_{ij} represents the average number of transitions from wave height state *i* to wave height state *j* [Ana96]. Synthetic time series are then generated, from an initial state, by randomly choosing the next state according to the relevant transition probabilities. It is clear that the next state only depends on the previous state (Markov property).

In order to cover seasonality, transition matrices were estimated on a monthly basis for this study. The intersection between the different months followed a simple approach: The last value of the previous month is considered as the first value of the present month. If this exact wave height does not exist in the present month, the next smaller possible value is chosen.

4.3.1 Validation Principles for Modeled Weather Conditions

Both modeled wind speed and wave height time series were validated by comparing simulated time series with observations. Mean values were compared in order to detect possible biases in the wave heights and wind speeds. Standard errors quantify whether the range of fluctuations can be considered realistic. In wind energy applications typically the wind speed is modeled and wave heights are derived from the wind speed. Since the wave height is crucial for accessibility, however, it was decided to model the wave heights directly, and derive wind speeds based on them. As wind speed and wave height correlate strongly, it was tested if this property also holds true for the modeled time series. Finally, as suggested by Anastasiou and Tsekos [Ana96], the accuracy of wave height and wind speed persistence, i.e., the length of possible weather windows for operation and maintenance fulfilling certain threshold criteria, was validated by comparing their cumulative distribution functions. A brief definition of the terms used for validation is given in the following paragraph, concluded by selected figures and discussions.

The standard error of the mean (*SE*) is used to describe the standard deviation in relation to the sample size (n_s). Mathematically, the standard error of the standard deviation is defined as follows **(4-2)**:

$$SE = \frac{\sigma}{\sqrt{n_s}} \quad \textbf{(4-2)}$$

A common way to describe the linear correlation between two variables is Pearson's correlation coefficient. It uses the covariance (*cov*) and standard deviation (σ) of both variables, here wind speed and wave height, to define the correlation coefficient r_P in a range of -1 (complete anti-correlation) to +1 (complete correlation). Typically, values of r_P above 0.2 (respectively, below -0.2) are considered a slight degree of correlation, above 0.6 (below -0.6) are considered as a medium degree of correlation, and above 0.8 (below -0.8) are considered as high correlation. A mathematical description is given in **(4-3)**. [Gra09].

$$r_P = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \quad \textbf{(4-3)}$$

The cumulative distribution function (CDF) is defined as follows [Kar07]:

$$P_X(x) = P(X \leq x) \quad \textbf{(4-4)}$$

$P_X(x)$ expresses the probability that a random variable (X) takes a value less or equal to x. X has to be a real number in the range from $-\infty$ to $+\infty$. For the purpose of this study, X represents the duration of a weather window fulfilling certain threshold criteria, and $P_X(x)$ is the probability that this weather window is less or equal the length x.

Differences between CDFs for the synthetic data and the observed data were assessed with the Kolmogorov-Smirnov test. It measures the difference between two CDFs, quantified by the significance probability p (for the null hypothesis that the two CDFs are equal); the test statistic *KS* is the maximum absolute difference between the two CDFs [Cra05].

4.3.2 Details of the Validation Procedure

The resolution of the significant wave height series is one of the most important parameters for obtaining an accurate model and depends on the available data. As with most discretization procedures, this involves a compromise between bias and variance: If the resolution, i.e., the length of a typical wave height bin, is chosen too large, the transition probabilities between wave heights are estimated accurately, but fluctuations on smaller scales are effectively filtered out, leading to relatively large variance compared with the original data. On the other hand, if the resolution is chosen too small, the synthetic wave height series include more fluctuations, but estimates of transition probabilities become unreliable due to fewer data points on which they are based.

To validate the method, we chose two sites in the North Sea close to the UK east coast, which is of interest due to the large future wind farm projects planned in the area. Results are shown for both sites, whereas one is close to shore (site **1**), the other far from the cost (site **2**), as shown on a map in section 5.1. The ERA Interim dataset, available from the European Centre for Medium-Range Weather Forecasts[9] in 6 hr resolution and for a period of 22 years (1989 to 2010), was used. Although longer datasets were available, this dataset includes more recent satellite observations of the sea surface in the reanalysis, and can therefore be assumed to feature more realistic wave height data than previous reanalyzes. A resolution of 0.4 m for the significant wave

[9] *http://www.ecmwf.int/*

heights resulted in relatively stable estimates of transition probabilities for the ensuing 18 wave height states, ranging from zero to 7.2 m.

The first state of the simulation was chosen randomly according to its marginal distribution. In order to capture seasonality effects, transition matrices were estimated separately for each month. Transitions between adjacent months were based on a simplified approach: from the last value in the current month the next value was drawn as if it were still in the same month, then it was switched to the transition matrix for the next month. If that particular wave height state did not occur in the new transition matrix, the state was adjusted to the first smaller wave height which occurs (with nonzero probability) in the Markov description for that month. Such adjustments occurred very seldom.

For all observed wave heights, the corresponding conditional wind speeds were recorded and discretized in 1 m/s resolution, covering wind velocities of 1 m/s to 30 m/s [Sch12b].

4.3.3 Validation Results

The means of significant wave height and wind speeds, estimated for each month separately, agree well, as does their variability (Figure 5 – Figure 12). Figure 4 shows the distribution of yearly wind speed (V) and wave height (Hs) mean values for both sites (site **1**: close to shore, site **2**: far from shore), simulated (S) and observed (O) in the form of boxplots. The center line in the boxplots represents the sample median. Differences in median values reflect the different character of the two sites: the second site farther from shore features significantly larger wave heights and slightly larger wind speeds. The weather model captures both the average values and their yearly fluctuations reasonably well.

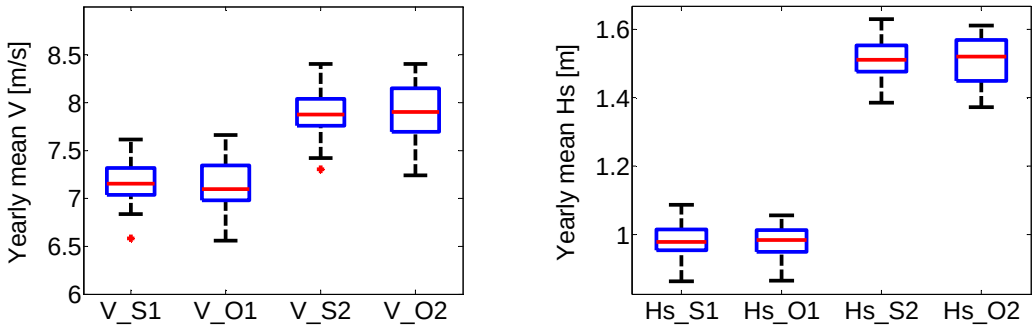


Figure 4: Yearly wind speed and significant wave height mean values

The correlation between wind speed and wave height time series was assessed by Pearson's correlation coefficient (Table 9), assuming linear correlation, as common in wind energy applications. High values of the correlation coefficients of more than 0.83 confirm the validity of this assumption. The synthetic data exhibits slightly lower correlations, but the difference in linear correlations lies under 2.5 %. Better correlations would be obtainable if one would also use a Markov model for the wind speed time series, but since the parameters of this model depend on the respective wave height values, such a hierarchical Markov model would suffer from large bias due to insufficient statistics on which the estimation would be based (even for 22 years of observed data) or would seriously limit the resolution of the wave height representation otherwise. Therefore, it was preferred to use the simple correlation method, especially since its performance can be deemed adequate: the interaction between wind and waves seems to be captured relatively well.

Table 9: Linear correlation between wind and wave height time series

	Observed site **1**	Modeled site **1**	Observed site **2**	Modeled site **2**
r	0.878	0.859	0.845	0.834

Monthly statistical summaries for observed (Figure 5, Figure 7) and simulated wave heights (Figure 6, Figure 8) further confirm the adequateness of the method. Similar results were obtained for the wind speeds (Figure 9, Figure 11 and, respectively, Figure 10, Figure 12).

The necessity of estimating transition matrices for each month separately can be clearly seen. No major differences between observed and simulated data can be discerned, although both simulated wind and wave height time series exhibit a slightly larger number of outliers compared to the observed data.

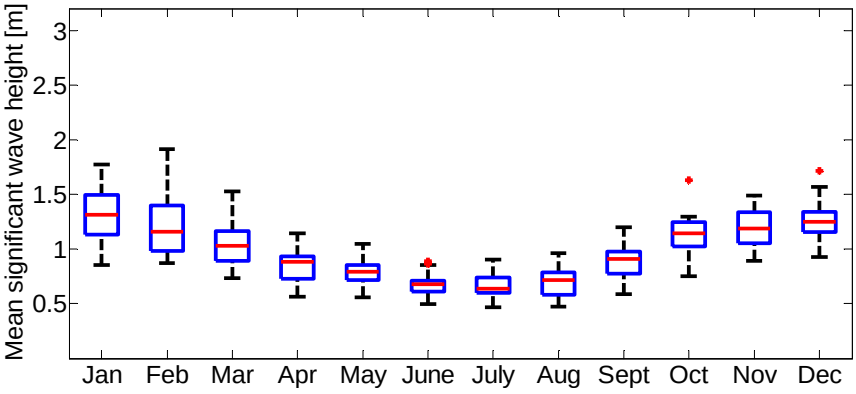


Figure 5: Wave height mean value distribution 1989 to 2010 – site **1**

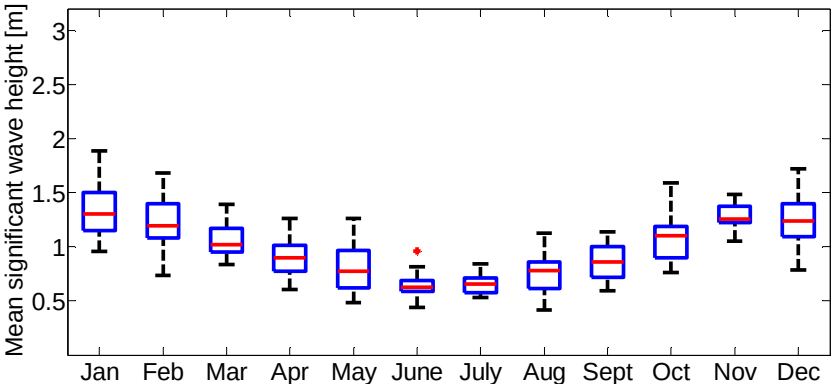


Figure 6: Wave height mean value distribution 22-year simulation - site **1**

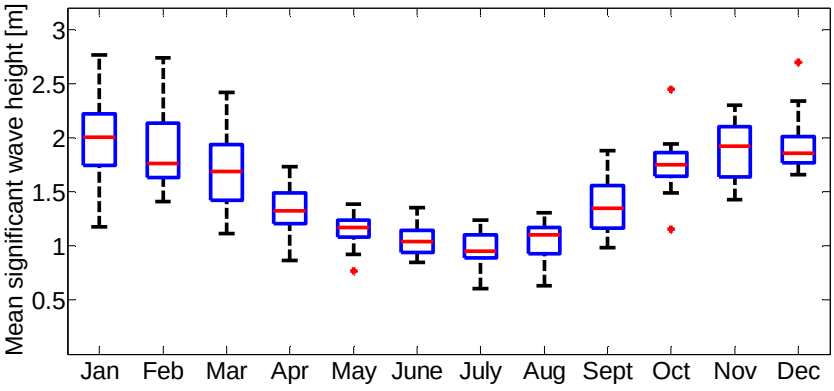


Figure 7: Wave height mean value distribution 1989 to 2010 - site **2**

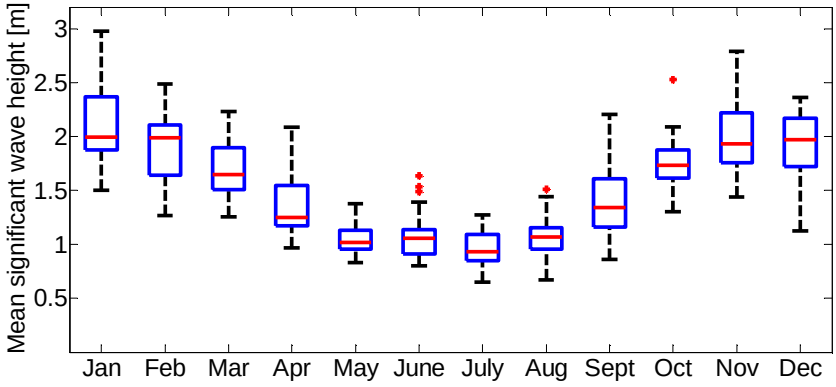


Figure 8: Wave height mean value distribution 22-year simulation - site **2**

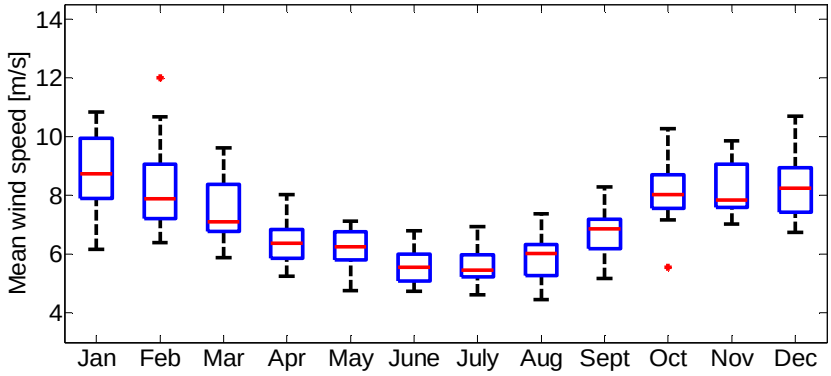


Figure 9: Wind speed mean value distribution 1989 to 2010 – site **1**

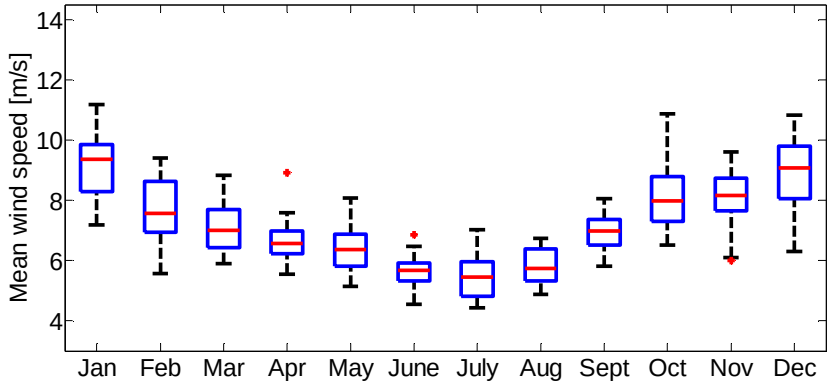


Figure 10: Wind speed mean value distribution 22-year simulation - site **1**

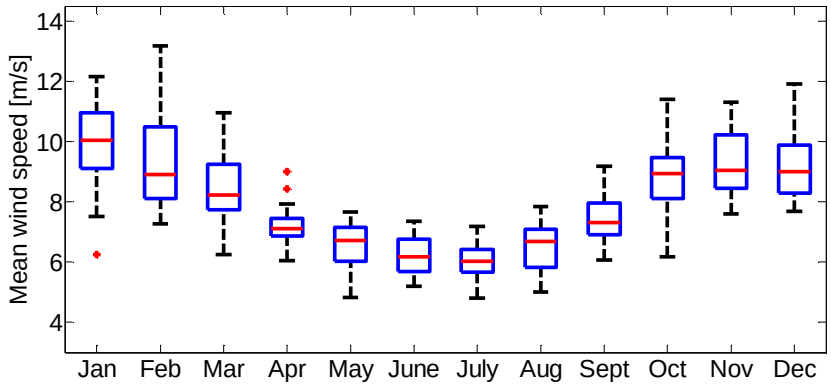


Figure 11: Wind speed mean value distribution 1989 to 2010 - site **2**

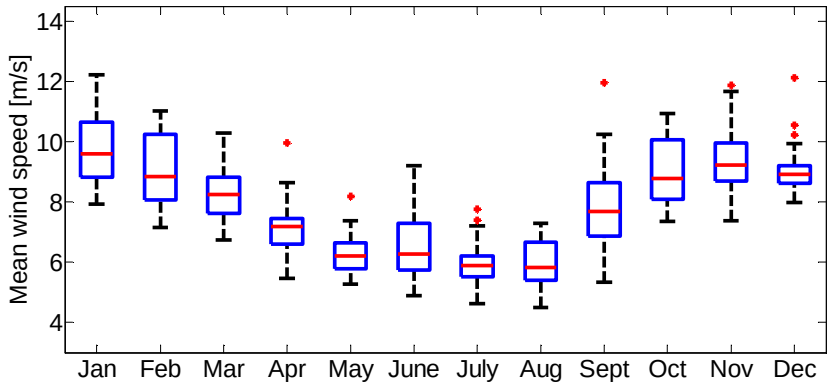


Figure 12: Wind speed mean value distribution 22-year simulation - site **2**

The wind speeds are also showing a slight increase in variability for a few months, which can be attributed to using the simple correlation method that cannot directly take temporal correlations between adjacent values of the wind speed into account. In other words, the persistence of wind speeds is likely to be somewhat affected.

As mentioned previously, the persistence of certain weather phenomena is of particular interest for operation and maintenance activities. Therefore, the cumulative distribution functions (CDFs) of run-length distributions for both, modeled and observed time series, were studied (Figure 13 - Figure 20). Two different wave height boundaries of 1.5 m/s and 2.5 m/s, respectively, were chosen. These correspond to typical operational limits for maintenance vessels and their access systems. The latter value can only be achieved by an advanced access system that will likely be more costly. The two different wind velocity boundaries have been chosen (8 m/s and 15 m/s) to cover both a value close to the mean wind speed, and a higher value which still can occur frequently.

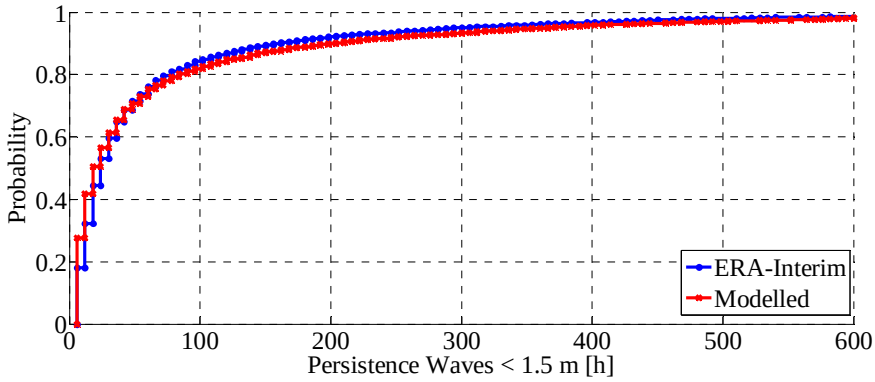


Figure 13: CDF wave height persistence - Hs < 1.5 m - site **1**

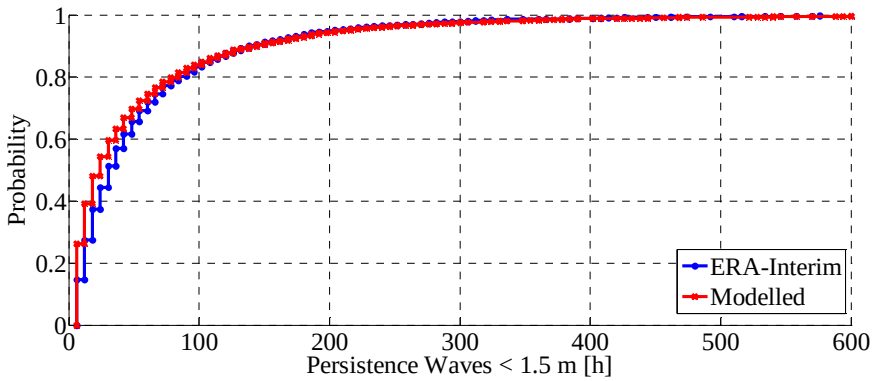


Figure 14: CDF wave height persistence - Hs < 1.5 m - site **2**

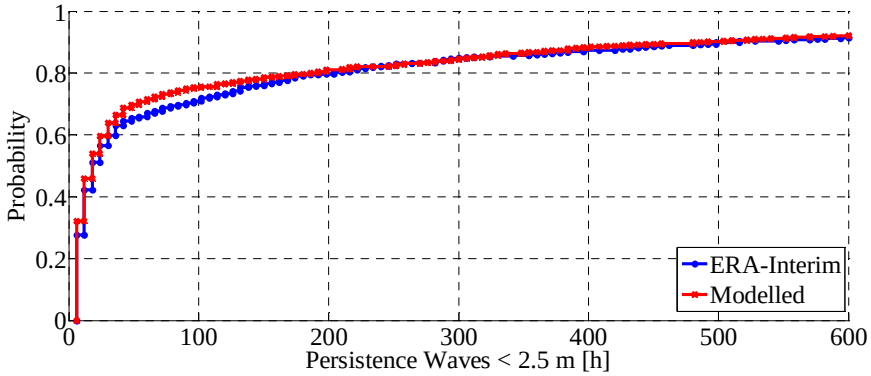


Figure 15: CDF wave height persistence - Hs < 2.5 m - site **1**

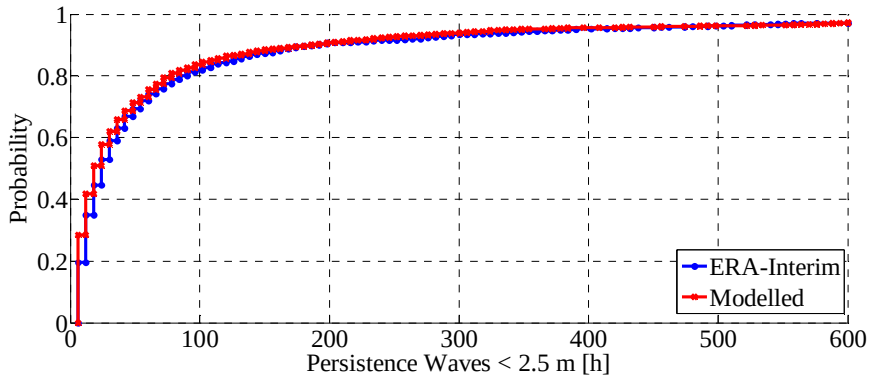


Figure 16: CDF wave height persistence - Hs < 2.5 m - site **2**

CDFs for site **1** and significant wave heights lower than 1.5 m show a slight overestimation of events shorter than 50 h for the modeled data, compared to the observations (Figure 13). In contrast, longer weather windows with wave heights below this boundary occur slightly less often in the simulations than in the measurements.

For a wave height boundary of 2.5 m at the same site, both curves are converging for weather windows of 170 h duration and longer (Figure 15). For site **2**, farther off the shore, the overestimation of low-duration events is slightly more pronounced. At the same time, the curves for both wave height boundaries are converging from around 100 h and more (Figure 14, Figure 16).

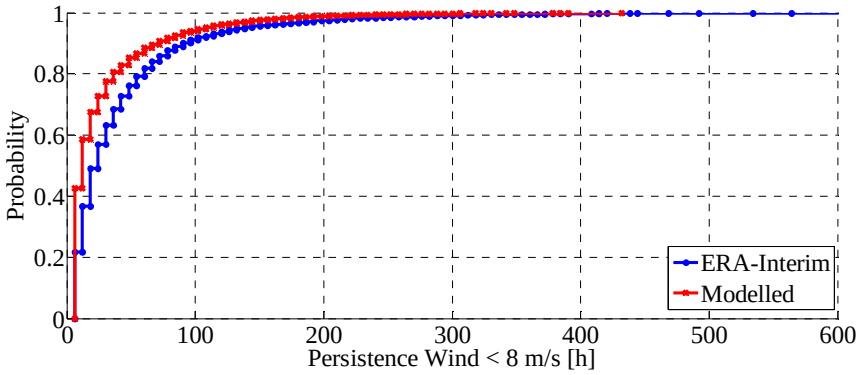


Figure 17: CDF wind speed persistence - wind speed < 8 m/s - **1**

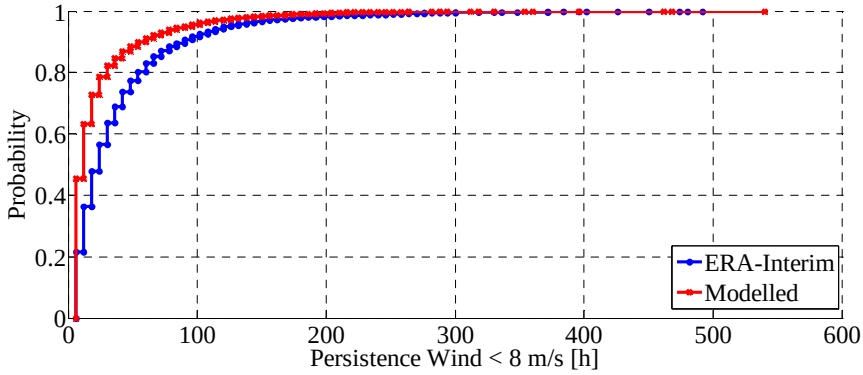


Figure 18: CDF wind speed persistence - wind speed < 8 m/s - site **2**

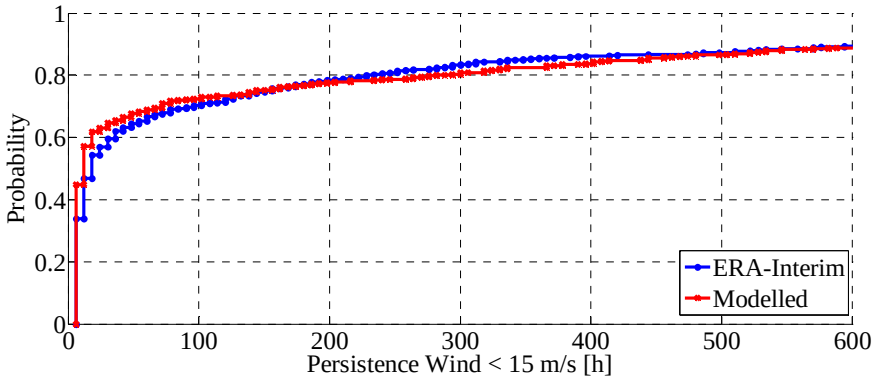


Figure 19: CDF wind speed persistence - wind speed < 15 m/s - site **1**

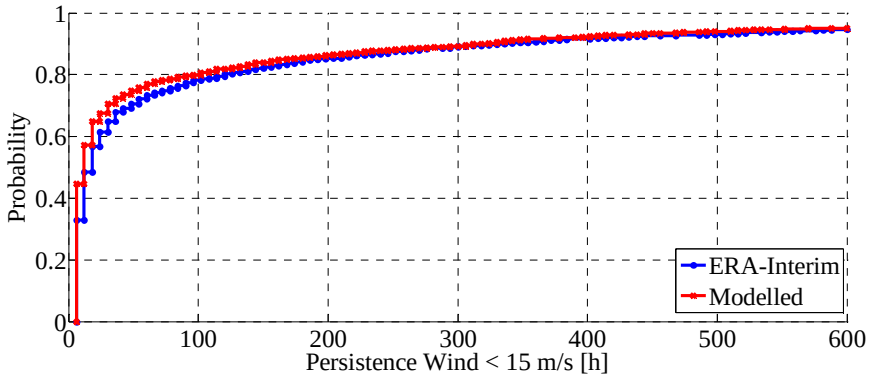


Figure 20: CDF wind speed persistence - wind speed < 15 m/s - site **2**

As expected from the above, the discrepancy between both CDFs (modeled and observed data) is somewhat higher for the wind speeds. A significant overestimation of weather windows shorter than 100 h (with wind speeds lower than 8 m/s) can be observed for both sites (Figure 17, Figure 18). Low-duration events are represented more accurately for the larger wind speed boundary of 15 m/s, but still exhibit a systematic overestimation. Longer weather windows are captured with some slight underestimation at site **1**, and agree relatively well at site **2** (Figure 19, Figure 20).

The Kolmogorov-Smirnov two-sample test was used to quantify these differences in persistence. Table 10 shows the differences between CDFs of run-length distributions for modeled and observed data by the significance probability (p) and the maximum difference between the two CDFs (KS).

Table 10: Differences between modeled and observed run-length distributions

Site 1: Hs < 1.5 m		**Site 2**: Hs < 1.5 m	
p	*KS*	*p*	*KS*
$<10^{-10}$	0.096	≈0	0.119
Site 1: Hs < 2.5 m		**Site 2**: Hs < 2.5 m	
p	*KS*	*p*	*KS*
0.261	0.048	$<10^{-7}$	0.089

Site 1: Wind speed < 8 m/s		**Site 2:** Wind speed < 8 m/s	
p	*KS*	*p*	*KS*
≈0	0.219	≈0	0.269
Site 1: Wind speed < 15 m/s		**Site 2:** Wind speed < 15 m/s	
p	*KS*	*p*	*KS*
$<10^{-3}$	0.110	$<10^{-7}$	0.116

Assuming a significance level of 0.1, *p*-values lower than this level indicate a significant difference between the persistence of modeled and original data. However, this test is very conservative, especially given the large number of samples on which it is based, and all run-length distributions test positive for significant differences. More informative are the Kolmogorov-Smirnov test statistics themselves. Their largest values occur for wind speeds lower than 8 m/s. That is also reflected by the shape of the respective CDF, as discussed above. The values for the significant wave height time series remain below 0.12, i.e., the probability of encountering weather windows of a certain duration differs at most by 12 percent for the simulated time series. Given the difficulty of realizing agreement in persistence across different scales in time [Ana96], the model can be considered to perform relatively well.

To summarize: also the persistence of weather windows for significant wave height values could be captured reasonably well by this approach. Moreover, as short-duration events are overestimated, this weather model is always conservative for the simulation of wind turbine maintenance, since longer persistence allows for prolonged offshore maintenance and repair operations. Weather windows for wind speeds are represented with much less accuracy (a maximum deviation of 27 percent in their probability of occurrence), but depending on the application, this can also be considered adequate. It is more important to capture the overall statistics of the wind speeds properly, in order to obtain good estimates of production losses during maintenance operations; and this has been confirmed before [Sch12b].

4.4 FAILURE MODELING

Turbine, respectively component or subassembly failures are modeled according to their assumed reliability characteristic, in terms of an annual failure rate λ (chapter 3.3.3). The simulation time step in which a failure occurs, expressed by the time to failure (TTF) random variable with respect to the last repair (or the start of the simulation), is determined by simulating from an exponential distribution, as in **(4-5)**:

$$P(TTF = x) = \lambda e^{-\lambda x} \qquad \textbf{(4-5)}$$

The probability of *TTF* exhibiting the value *x* is denoted by *P(TTF = x)*. The number of time steps until the next failure is then determined by an exponentially distributed random number with mean value taken to be the MTBF, i.e., λ^{-1}.

In the first simulation time step, the time to first failure (TTFF) is calculated for all twelve considered subsystems in all turbines. Due to the stochastic nature of this process, it is possible that failures for a certain component occur at the first day of operation from one turbine, and never occur over a lifetime of 20 years of another turbine. This means that a sufficient number of simulations need to be run in order to gain statistically reliable results – and therefore the ability to generate arbitrarily many synthetic time series of weather data is desirable and has motivated the weather model.

After a subsystem failure, repair has to take place before the time to next failure (TTF) is determined again, which happens after the remaining repair time is set to zero and the subsystem is therefore running again. Failures can be repaired non-continuously. An interruption of repair due to bad weather conditions or violation of other limits, e.g., the maximum time offshore, is permitted. Failures of different subassemblies do not affect each other. In the model it is therefore possible that, for instance, the pitch system of a turbine breaks, even if it is not running due to another failure. This case is very scarce and the influence on simulation results is therefore ignorable. Bathtub curves or increased failure rates according to higher wind speeds are also neglected. These effects might have an influence on the totally achieved availability, but in order to find out about major trends, the chosen approach still is applicable. Component failures are assumed to always cause a breakdown of the whole turbine.

4.5 Applied Maintenance Strategy

The maintenance scheduling strategy, applied within this thesis, is described in the following paragraph. A general overview of the procedure after a new failure occurs is shown in Figure 21 for an ordinary vessel (the scheduling of a crane ship being similar).

In case of the occurrence of a new failure, all ships operating in the wind park check if they transport enough personnel (from previous, completed repairs) without present task. If this is the case, a new crew is established for which the component is scheduled for repair, provided that the maintenance personnel has not been offshore too long and the weather conditions are sufficient.

If a failure cannot be scheduled for one of the ships located in the park, it stays unscheduled (unreserved) and can then be scheduled for ships residing in harbor. First, all ships already containing crews (from previous time steps with bad weather) are trying to add a new crew, which is scheduled for this failure. Depending on each ship's capacity, either a crew for this failure is created, or, if the maximum load is already reached, leaves the component unreserved and available for scheduling by other ships.

If there are no suitable ships available, the failure remains unreserved and the procedure starts afresh six hours later, in the next time step.

If a ship in harbor carries a crew, it will, as soon as weather conditions allow for, enter into transit to the park. Before a ship leaves the harbor, all scheduling phases are passed again in order to consider changes, such as a new failure for instance. For the weather check, a so-called *look-ahead-time* represents the accuracy of the weather forecast. This parameter determines the maximum time a ship can assume 100% reliable information about future weather conditions. Weather conditions from dates after *the look-ahead-time* are assumed to be unreliable, which is not accurately realistic, but eases the simulation significantly. Weather conditions are sufficient, as long as the wave height is under the operational threshold level for the total intended time offshore (maximum of all crews' expected time to repair, separately assessed for each ship or crane vessel carrying crews).

In the next time step after transit, the ship drops off all crews at the turbines with components that are scheduled for repair. This task is done after checking the actual weather conditions and the time offshore (both check-ups are done in every time step while a vessel is offshore). It is assumed, that all transits in park can be handled within one time step of 6 h. In very large wind farms, actual transit times in park could play a role, but are neglected for the purpose of this study. After a crew arrives at a component, the respective discrete repair time counts down every time step until zero. The crew is then collected by the ship in the next time step and can immediately schedule further failures. Every ship can only handle the crews that were assigned to it in the harbor. It is thereby presumed that all components which can be repaired without a crane do not require additional equipment from land.

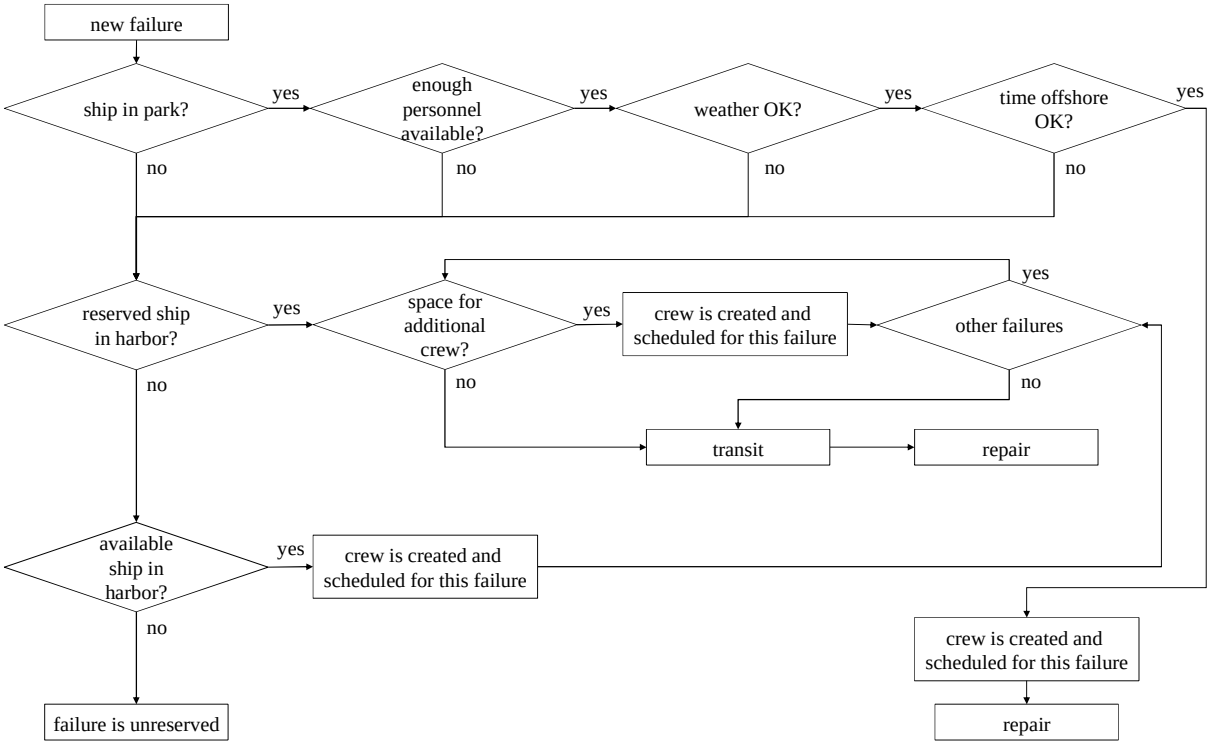


Figure 21: Scheduling strategy flow chart

In contrast, a crane can only handle one repair task before it needs to return back to the harbor. If a crew has to interrupt a repair, failures are reset to the state *unreserved* and repair times stay as they were before the interruption, assuming the next crew can continue immediately working on the failure [Sch12a].

5 Results of Wind Park Simulation

Simulation results obtained with the described tool are shown for two different reference offshore wind parks in the British North Sea (Section 5.1). Variations of maintenance fleet, reliability characteristics of components, distance to shore and weather forecast accuracy have been performed and studied with respect to park-availability, cost savings due to availability changes and deployment of equipment. Each simulation represents one year of operation. This relatively short simulation time enables the possibility of a great parameter variation. For a representative case study, it is recommended to simulate a longer period of operation, e.g., 20 years. A simulation of this length has exemplarily been carried out, with the result that the level of availability decreases significantly, compared to the availability achieved in one year simulations. The reason for this lies in a shift of unsolved turbine failures towards longer simulation times. The effect is getting stronger, the less the maintenance crews and maintenance equipment have the ability to solve all failures

during a given period, for instance, one year. All failures, which could not be solved during that period, are then added to the next year. This trend pursues continuously and leads to a high number of unsolved failures after a long simulation period. However, it is expected, that achieved results for one year of operation are representative for the behavior of wind park performance depending on the discussed parameter variations. In order to present correct absolute values, however, longer simulation times are necessary. All presented results aim to provide a trend of how the operating cost of an offshore wind farm could react on changes of the named parameters.

5.1 Design Basis

As described in chapter 3.3.1, the offshore wind power market is growing strongly in British waters due to an attractive FIT. Large offshore wind projects are planned within UK round III. One example is the Dogger Bank area, in which the FOREWIND[10] consortium is planning to achieve 9 GW installed capacity by 2020. The first tranches of the project are two 1.4 GW parks.

Figure 22: Map simulation site **1** and **2**[11]

Taking into account the importance of the British offshore wind energy market, two fictive reference projects in the UK North Sea were chosen. One close to the shore (Site **1**), the other far from shore (Site **2**). Each reference park consists of 500 turbines, each with a rated power (P_{rated}) of 5 MW, resulting in 2.5 GW in total. The dimension is therefore comparable to large UK round III offshore wind projects.

The turbines used are characterized by their rated power in order to calculate production losses by means of a linearized power curve in combination with the current wind speed (v) during downtime. The power curve is linearized for all wind speeds (v_{actual}) from 3 m/s (cut-in wind speed) to 11.4 m/s (rated wind speed - v_{rated}) by the following equation:

$$P = \frac{v_{actual}}{v_{rated}} * P_{rated} \qquad \textbf{(5-1)}$$

Above 25 m/s wind speed (cut-out), respectively under 3 m/s, electricity production is assumed to be zero. In the area between rated wind speed and cut-out wind speed, power production is assumed to be according to the full rated power (here: 5 MW). The respective turbine data is taken from the 5 MW reference turbine developed by the National Renewable Energy Laboratory [Jon09].

[10] *RWE; SSE; Statkraft; Statoil - http://www.forewind.co.uk/*

[11] *© 2012 Europa Technologies © 2012 Google US Dept of State Geographer © MapLink/Tele Atlas*

The simulation period is one year, respectively 1460 time steps à 6 h, which is the resolution of the applied weather model. Results show mean values of five simulation runs in order to cover fluctuations.

For selected studies (section 5.2 and 5.6), both sites have been taken into account. The exemplary study shown in chapter 5.5 has only been performed for the first site. It is expected that the trend of the number of vessel deployment depending on the accuracy of the weather forecast does not change due to an increased distance to shore. Studies shown in section 5.3 and 5.4, refer to site **2**, far from shore. Results achieved for the reliability study are assumed to be more significant with a greater distance to shore (section 5.3). As long distances to shore are studied in section 5.4, weather conditions for a site located far offshore are applicable.

The applied maintenance strategy is fully corrective. All kinds of other services and inspections are neglected.

5.2 Influence of Wave Height Constraint and Equipment Variation on Park Availability

Four different compositions of the maintenance fleet were taken into account for this study, as major effects can be clearly seen in those configurations (Figure 23, Figure 24):

Wave height boundaries were varied from 1.0 to 2.6 m in steps of 0.4 m (corresponding to the resolution of the sea state simulation). All other simulation parameters were held constant. The look-ahead-time was set to 48 h. The maximum number of people that could be carried on a ship was four. Each calculated availability value represents a mean over five runs, with standard errors less than a few percent.

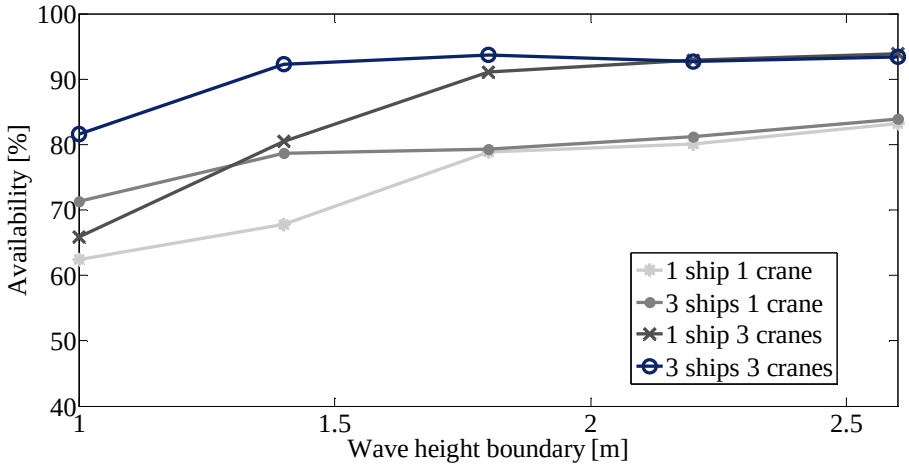


Figure 23: Park availability against equipment characteristics - site **1**

Regarding fleet variation, it can be seen that the highest availabilities have been achieved with three ships and three cranes, the largest fleet considered at both sites (Figure 23, Figure 24). In reverse, lowest values occurred by considering only one ship and one crane.

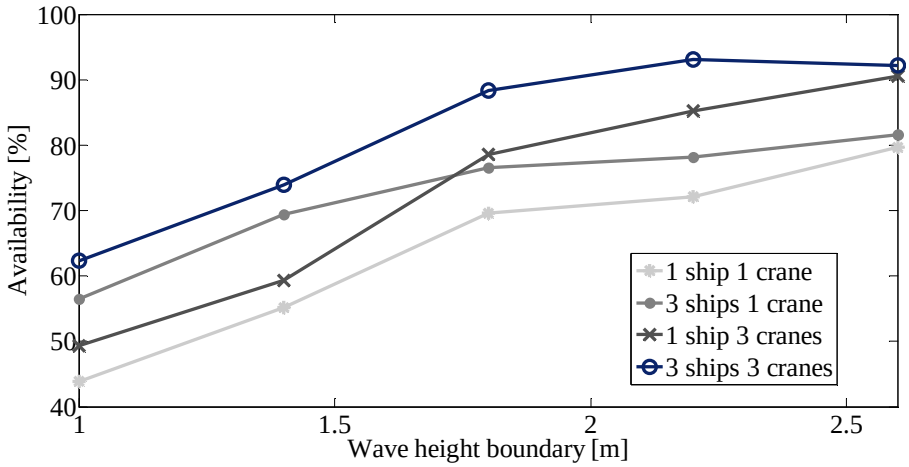


Figure 24: Park availability against equipment characteristics - site **2**

Examining site one (Figure 23), for assumed wave height boundaries less than 1.4 m, additional ships and cranes both lead to an increase of availability of the same magnitude, i.e. the availability changes (almost) linearly with the number of resources. For the regime from 1.4 to 2.2 m, additional ships have a lower influence on the availability than additional cranes have. The availability of more than one ship that can perform for boundaries greater than 2.2 m has almost no effect on availability anymore, i.e., there exist sufficient weather windows that one such ship can handle all repairs of the 500 turbine wind park. The number of crane vessels still has an important influence, as each crane vessel can only handle one repair at a time. In case of several failures, a greater number of cranes would directly lead to more repairs during a weather window.

A significantly lower level of availability can be observed at site two (Figure 24). Curve shapes from the first study seem to be shifted towards higher wave height boundaries. Additional ships are more effective than additional cranes for wave height boundaries lower than 1.4 m. At a wave height boundary of 1.8 m, this trend changes slightly to almost the same level for both configurations with either three cranes or three ships. As also observed for site one, the result for the "one-ship-three-cranes" configuration converges to one consisting of three cranes and three ships, however not from 1.8 m wave height boundary, but from 2.6 m. As additional ships have almost no influence on the availability from this wave height boundary, values for the "one-ship-one-crane" configuration converge to the one from the "three-ships-one-crane" configuration.

Going farther offshore, entailing higher wind speeds and wave heights, therefore decreases the availability if maintenance scheduling strategy and maintenance fleet are held constant. In order to achieve the same availability as for site **1**, significant wave height boundaries for maintenance equipment have to be raised at least by 0.8 m from 1.8 m to 2.6 m. Even if this modification is done, achieved availabilities are slightly lower for site **2** [Sch12a]. Anyhow, potential cost savings are tremendous if this technical modification is taken into account. For site two, taking into account a "one-ship-three-cranes" configuration, more than 290 M€ could be cut down on the production losses every year, considering the assumed simulation conditions. Applying the same scenario at site one, yearly cost savings of not even 50 M€ could be achieved. At the same time, considering a wave height boundary from 1.8 m, total production losses for site **2** are at around 535 M€, whereas they account for 215 M€ at site **1**. Consequently, if a park is located farther offshore, higher-quality maintenance equipment is essential in order to keep availability, and therefore production losses, on an economic acceptable level.

5.3 Influence of Component Reliability

Three different reliability characteristics of all components have been taken into account for this study. The regular reliability, as described in chapter 3.3.3, one for which failure rates are 50% lower, and one for which the failure rate is 50% higher, than for the baseline. All other simulation parameter were held constant. Exemplarily for this study, a "three-ships-three-cranes" fleet composition, with a wave height boundary of 1.8 m for the equipment, was chosen. The simulation has been carried out for site **2**, far from shore. In Figure 25, results from this study are composed.

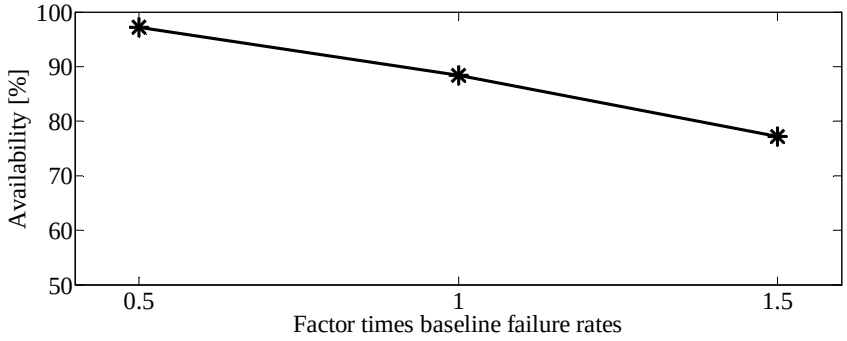


Figure 25: Park availability against component reliability

It can clearly be seen that the availability increases towards a lower multiplication factor for applied failure rates. Even the fleet configuration with the most optimal results regarding achieved availabilities depends strongly on the reliability of components. If the level of reliability could be increased, and failure rates would be 50% lower than assumed here, achieved availabilities could reach the same level as for onshore wind turbines, about 97%. For this simulation, according to **(5-2)** in chapter 5.6, yearly cost savings of 220 M€ for the park could be achieved by a better reliability. That is 440 k€ yearly cost savings per turbine, a huge potential in a considered lifetime of 20 years.

5.4 Influence of the Distance to Shore

As briefly discussed in chapter 3.3.2, an offshore base can be applicable in order to access wind parks located far from shore. For this study, such a base is virtually implemented in the simulation for ordinary maintenance vessels. It is assumed, that crews, vessels and spare-parts can be accessed offshore. Due to the size of spare parts crane vessels have to handle with, it is assumed that they still need port support. The transit time of ordinary maintenance vessels is set to zero, different distances from the port to the park for crane vessels are modeled with transit times of 12 h, 24 h and 36 h. These modifications are evaluated against achieved availabilities in a "three-ships-three-cranes" configuration and an assumed restriction for significant wave height of 1.8 m at site **2**, far from shore. Achieved simulation results are displayed in Figure 26.

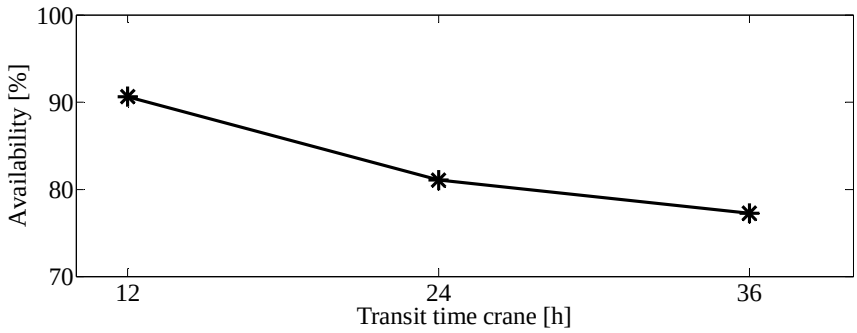


Figure 26: Distance to shore against park availability

A significant decrease of availabilities according to a greater transit time for crane vessels can be seen in the graphic above, even if a transit time of zero is applied for ordinary maintenance vessels. The origin of this observation lies in greater and more frequent weather windows required for each crane deployment. Under the simulation conditions applied for this study, an offshore base including big spare-parts and a crane vessel seems to be desirable. As also stated in chapter 3.3.2, a jack-up platform, including a crane and space for a large number of all kinds of spare-parts, is a possibility to improve the park performance with respect to the achieved park availability.

5.5 Effects of Accuracy of the Weather-Forecast

The decision whether an operation is going to be performed mainly depends on weather conditions. In case of a theoretical, fully accurate forecast (with a look-ahead-time of infinite length), ships or crane boats are only deployed if it is assured that they can finish their tasks. The possibility of splitting up repair actions on purpose, for instance in winter months when weather windows tend to be shorter, is here neglected. If the look-ahead-time is finite, as it is in reality, vessels have to cancel offshore operations from time to time, and need to access the turbine more than once for each repair (if the intended time offshore is longer than the available weather forecast – see also section). This is represented in Figure 27, where the yearly number of deployments for an ordinary maintenance vessel (ship) and a crane vessel (crane) are displayed for five different cases. The analysis was performed for site one and the look-ahead-time has been varied in a range from 6 h to 72 h.

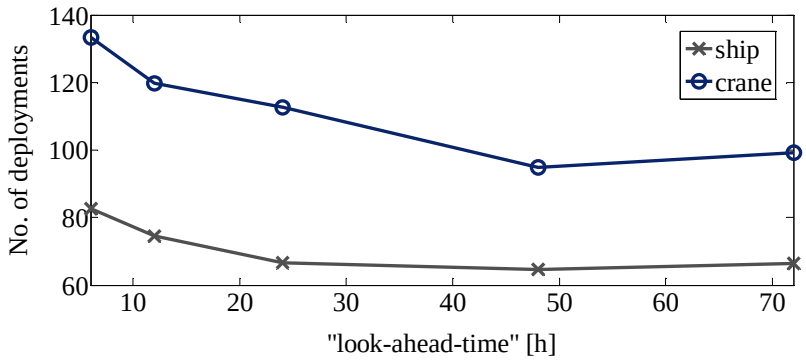


Figure 27: No. of ship and crane deployments against look-ahead-time

The total number of average operations carried out by both vessels is shown for a one year simulation in a "three-ships-three-cranes" configuration, where five runs have been taken into account. Achieved availabilities have been at 90% for all runs. Both curves show a strong dependence of the amount of operations on the reliability of the weather forecast in regions of short look-ahead-times. Crane boats arrive at their optimal operating point if an accurate forecast of two days or more can be provided. Due to shorter transit and repair times, the saturation for ships is reached after 24 h, i.e., long-term information about future weather conditions is not necessary for an optimum performance in this configuration. Especially for crane vessels, a slight increase of the number of deployments can be observed for long look-ahead-times. This phenomenon is assumed to be of statistical nature due to the limited amount of simulation runs [Sch12a].

5.6 Cost

For an estimation of potential cost savings due to higher availability, downtime losses have been quantified monetarily. For the FIT, data from a KPMG market survey has been used [Koe10], leading to an income of 0.1801 €/kWh for British waters. The dependence of yearly production losses on availability is visualized in

Figure 5, where data is provided for each entire simulated park. All different fleet combinations and wave height boundaries have been evaluated for this diagram.

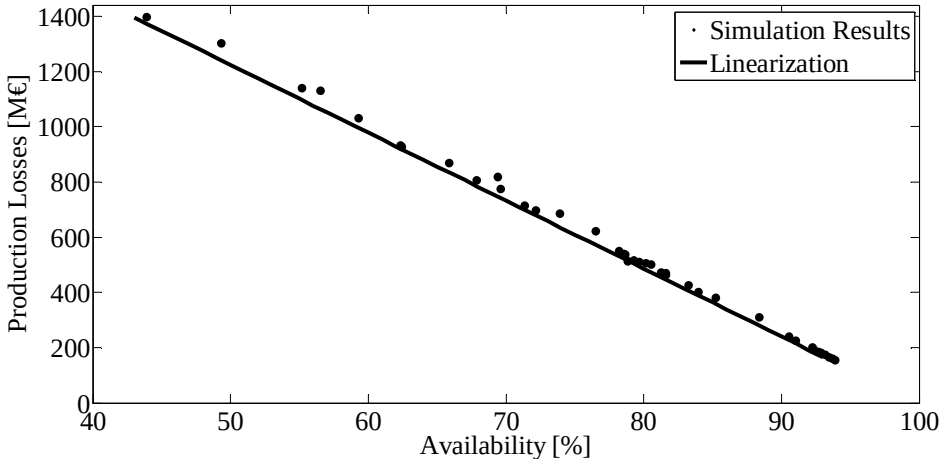


Figure 28: Production losses against availability

A strong linear correlation between availability and production losses can be seen from Figure 28. In numbers, a Pearson's correlation coefficient of 0.99 is achieved. Slight variations from the linear trend occur in regions of lower availabilities. Wind speed variations are having a greater influence here, due to the total amount of considered realizations.

Investigating the worth of higher-quality maintenance equipment, an exemplary change of wave height boundary from 1.0 m to 1.8 m for both access methods in a "one-ship-one-crane" configuration would decrease downtime losses by about 30 percent. In numbers, a yearly decrease of downtime losses of 393 M€ for one 2.5 GW park, respectively, more than 780 k€ on average per turbine, could be achieved by using a more advanced access system, according to the assumptions made for this investigation. An investment in higher-quality maintenance equipment therefore seems to be highly recommendable, almost no matter how much that costs.

For both simulated wind parks, the correlation between production losses (PL) and availability (A) can be formulated as in equation **(5-2)**, when considering availability values from 43-93 % in order to approximate cost savings by variation of parameters:

$$PL(A) = (0.62 - \mathrm{A}) \cdot 2460\ \mathrm{M€} + 929\ \mathrm{M€}. \qquad \textbf{(5-2)}$$

A small increase in availability can therefore lead to high cost savings. Which parameters could effectively affect availability is stated in the above sections. As shown in Figure 23 and Figure 24, the application of a "three-ships-one-crane" configuration with an assumed wave height boundary of 1.4 m leads to the same availability level as a "one-ship-one-crane" configuration with a wave height boundary of 1.8 m at both reference sites. The decision on which solution is providing the overall economic optimum has therefore to be decided considering aspects on various cost drivers [Sch12a]. The major conclusion however, is the recommendation to invest in maintenance equipment which is applicable in regions of high wave heights. In order to perform economically well far offshore, this recommendation becomes even more inevitable.

6 CONCLUSION

Within this Diploma Thesis, an approach of simulating the operating phase of an offshore wind farm has been implemented. The application of presented methodologies shows promising results.

An accurate weather model has been developed, based on Markov theory, which, to the authors knowledge has not been applied for wind park simulation before. The weather model was validated successfully by several tests. Synthetic wave height time series thereby are more accurately represented than modeled wind speed time series. Two sites, one relatively close, the other far from shore, were evaluated.

Significant changes in availability, monetarily quantified by production losses, have been presented with respect to changes in maintenance fleet and vessel characteristics for both reference wind farms. The restrictions for significant wave height boundary of the fleet emerged to be the most crucial parameter influencing availability. Regarding fleet composition (number of maintenance ships, respectively crane vessels), achievable availability levels saturate from a certain wave height boundary. Consequently, a higher number of vessels does not necessarily cause a higher park-availability. In order to achieve availability levels higher than 90%, a maintenance fleet consisting of three ships and three cranes with a wave height boundary of 2.2 m for a 500 turbine wind park far from shore is recommended. The same reference park, located close to shore, requires the same maintenance fleet, whereas the equipment is sufficiently applicable with a wave height boundary of only 1.4 m. If the wave height boundary for maintenance equipment would also be 2.2 m for the park located close to shore, a fleet composition of one ship and three crane vessels would lead to the same results. In order to achieve availability levels which are comparable to those from onshore wind generation, the reliability of components has to be improved when considering all maintenance fleet compositions used for this study. Applying an offshore base for ordinary maintenance vessels, minor spare-parts and crew, the effect of a longer distance to the park has been shown by increased transit times for crane vessels against the achieved park availability. A greater distance to shore leads to a significant decrease of availability, leading to the recommendation of a jack-up based offshore platform, including big spare-parts and a crane.

The strong linear correlation between production losses and availability has graphically been visualized and concluded by a simplified equation. The equation is applicable for both sites, even if wind speeds are higher for the park located farther from shore.

A relation between weather forecast accuracy and number of vessel deployments has also been presented. An accurate long-term weather forecast leads to a more efficient vessel-deployment, whereas a saturation can be observed from a certain length of the forecast. Ordinary maintenance vessels experience this saturation from a forecast-length of around 48 h, crane deployments saturate from 24 h already.

The economical potential was shown with the perspective of implementing these methods in tools for wind park life cycle cost models. More detailed studies regarding the economical effect of different scheduling strategies and equipment specifications could be performed on this basis. Simplifications, e.g., for characterization of restrictions for maintenance equipment, could be replaced by more accurate representations. In order to represent an actual planned wind farm, more detailed information about failure rates and repair times is necessary. Including activation times for vessels, labor, equipment and spare-part cost could also allow for more detailed studies.

7 DISCUSSION

The intention of this project was to propose a method to figure out how different parameter influence the cost for the operating phase of offshore wind parks from a global point of view. As stated in chapter 5, the sensitivity of the availability of an offshore wind park, under the variation of certain parameters became the exploratory focus. Calculated availabilities and potential cost savings shall not be understood as ultimate values, but allow for developing a sense for how the availability and costs could react to parameter variations. Several simplifications have been made in order to concentrate on the parameters of interest:

Regarding the weather module, slightly better statistics could have been achieved by decreasing the wave height resolution, i.e., by decreasing the number of states of the Markov chain. At the same time, that would eliminate the possibility of varying wave height boundaries as it was done for this study. Selected tests of the weather module also show a sufficient accuracy of generated time series. Furthermore, the generation of wind speed time series based on wave heights would not have been possible to the same level of accuracy. Especially for the cost module, a precise representation of wind speeds is essential.

Considered failure rates are based on an onshore survey. Values for offshore turbines might differ significantly, but, as the market still is quite young, only incomplete data is available. Failure rates for all considered systems can be readily adjusted, if required. The simplification of a total turbine breakdown, independent of which system fails, might be refined subject to each component.

The possibility of continuing a repair from the point it has been interrupted might also, especially for components repaired or replaced by a crane, not be realistic.

The definition of maintenance resources was held very simple. Only two different access systems were considered, an ordinary maintenance vessel and a crane vessel. Ships were only specified by the maximum number of personnel they can carry, a maximum weight capacity of transported material is neglected. In terms of restrictions, both access systems are only underlying wave height boundaries. In reality, depending on the planned operation, wind speed, fog, temperature or rainfall might also play a role. Alternative access methods, e.g., by helicopter, have not been taken into account. For ships in harbor, personnel is assumed to be always available in the demanded quantity.

The scheduling strategy applied for this study could be refined by implementing more factors a ship, respectively a crane, can base its decisions on, including probabilistic aspects for weather forecast accuracy or repair time variations. If transit times and repairs could be supplied by costs, more optimal overall strategies could be developed. For the variation of maintenance fleet, wave height boundary and weather forecast accuracy, the applied scheduling is showing sufficient results.

All results are shown for a simulation time of one year. An exemplary study has been carried out, showing that the total park availability decreases with an increased simulation time, here 20 years. This comes due to the shift of failures, that could not be solved in one year, to the next year. Therefore, if the existent maintenance equipment is not able to solve all failures, they are cumulating towards longer simulation durations. All shown results are expected to become more significant in a realistic long-term simulation.

Economic performance is solely investigated by evaluating production losses during downtime. Significant changes have been achieved by parameter variation, showing that especially the wave height boundary for access systems has a great influence on the economic performance. An implementation of the described methods to other wind farm cost models could be a possibility of accessing life cycle cost for an overall economic optimization.

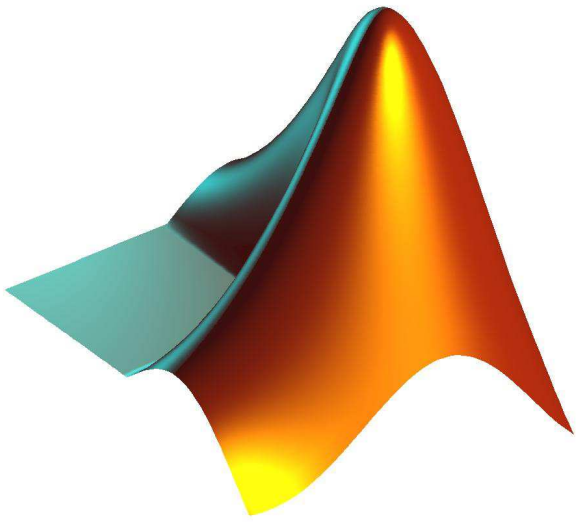
BIBLIOGRAPHY

[Ana96]	Anastasiou K, Tsekos C, "Persistence statistics of marine environmental parameters from Markov theory, Part 1: analysis in discrete time", *Applied Ocean Research 18*, P.187-199, 1996
[Bla04]	Blank R, "The Basics of Reliability", Productivity Press, New York, USA, 2004
[Bli00]	Blischke WR, Murthy DNP, "Reliability - Modeling, Prediction, and Optimization", Wiley, Denver MA, USA, 2000
[Bus97]	Bussel GJW van, Schöntag C, "Operation and Maintenance Aspects of Large Offshore Windfarms", *Delft University of Technology*, The Netherlands, 1997
[Bus01a]	Bussel GJW van, Zaaijer MB,"Reliability, Availability and Maintenance aspects of large-scale offshore wind farms, a concept study", *ImarE Conference Volume 113*, P. 119 - 126, The Netherlands, 2001
[Bus01b]	Bussel GJW van, Zaaijer MB, "DOWEC concepts study, reliability, availability and maintenance aspects", *Proceedings European Wind Energy Conference*, P. 557 - 560, Copenhagen, Denmark, 2001
[Bus03]	Bussel GJW van, Bierbooms WAAM, "Analysis of different means of transport in the operation and maintenance strategy for the reference DOWEC offshore wind farm", *Proceedings OWEMES offshore wind energy seminar*, Naples, Italy, 2003
[Cra05]	Crawley MJ, "Statistics: an introduction using R", West Sussex, England, 2005
[Ewe09]	European Wind Energy Association, "Wind Energy - The Facts", London, UK, 2009
[Fau11]	Faulstich S, Hahn B, Tavner PJ, "Wind turbine downtime and its importance for offshore deployment", *Wind Energy 14*, P. 327 - 337, 2011
[Fen10]	Feng Y, Tavner PJ, Long H, "Early experiences with UK round I offshore wind farms", *Proceedings of the Institution of Civil Engineers: Energy 163*, P. 167 - 181, 2010
[Gra09]	Gravetter FJ, Wallnau LB, "Essentials of Statistics for the Behavioral Sciences", Belmont, USA, 2009
[Hof11]	Hofmann M, "A Review of Decision Support Models for Offshore Wind Farms with an Emphasis on Operation and Maintenance Strategies", *Wind Engineering 35*, P. 1-16, No. 1, 2011
[Jam11]	Jamieson P, "Innovation in Wind Turbine Design", West Sussex, United Kingdom, 2011
[Jon09]	Jonkman J, Butterfield S, Musial W, Scott G, "Definition of a 5-MW reference wind turbine for offshore system deployment", *National Renewable Energy Laboratory, NREL/TP-500-38060*, Golden, Colorado, USA, 2009
[Kar07]	Karris ST, "Mathematics for Business, Science and Technology", Fremont, USA, 2007
[Kem76]	Kemeny JG, Snell JL, "Finite Markov Chains", Harrisonburg VA., USA
[Koe10]	Koeppe O, Schulze K, "Offshore-Windparks in Europa", *KPMG*, Berlin, Germany, 2010
[Lóp10]	López JA et al., "Offshore Access: A key driver to increase Offshore Wind Farms Efficiency", *Iberdrola Ingeniería y Construcción*, Madrid, Spain, 2010

[Mcg08]	McGugan M, Larsen GC, Sørensen BF, "Fundamentals for remote condition monitoring of offshore wind turbines", *Risø National Laboratory for Sustainable Energy, Technical University of Denmark*, Roskilde, Danmark, 2008
[Mon01]	Monbet V, Marteau PF, "Continuous Space Discrete Time Markov Models for Multivariate Sea State Parameter Processes", *Proceedings International Offshore and Polar Engineering Conference*, Stavanger, Norway, 2001
[Obd07]	Obdam T, Rademakers L, Braam H, Eecen P, "Estimating Costs of Operation & Maintenance for Offshore Wind Farms", *Proceedings European Wind Energy Conference*, Milan, Italy, 2007
[Rad08]	Rademakers LWMM, Braam H, Obdam TS, Frohboese P, Kruse N, "Tools for estimating operation and maintenance costs of offshore wind farms: State of the Art" *ECN Wind Energy*, Petten, The Netherlands, 2008
[Rot74]	Rothkopf MH, McCarron JK, Fromovitz S, "A Weather Model for Simulating Offshore Construction Alternatives", *Management Scienc, Vol. 20 No. 10,* P. 1345-1349, 1974
[Sch12a]	Scheu M, Matha D, Hofmann M, Muskulus M, "Maintenance strategies for large offshore wind farms," *Energy Proceedia 2012*
[Sch12b]	Scheu M, Matha D, Muskulus M, "Validation of a Markov-based weather model for simulation of O&M for offshore wind farms" *Proceedings International Offshore and Polar Engineering Conference*, Rhodes, Greece, 2012
[Tem05]	Tempel J van der et al., "Scale model testing of the Ampelmann - Safe and easy access to Offshore Wind Turbines", *Proceedings of the Copenhagen Offshore Wind Conference*, Copenhagen, Danmark, 2005.
[Ves05]	Vestergaard S, "Offshore Technology / Operation & Maintenance - accessibility", *Copenhagen Offshore Wind*, Fredericia, Danmark, 2005

A APPENDIX - MATLAB USER INSTRUCTION

The following paragraphs describe how a wind park simulation can be performed in Matlab (The Mathworks, Inc. Version R2011a) according to the proposed methods. Starting with the modeling of wind and wave conditions based on historical data (chapters A-1 and A-2), possible modifications for the maintenance strategy and their effects are described in chapter A-3 . It is also referred to the Matlab code in chapter A-4 . In the following, an additional list of contents for the appendix is given.



[12]

[12] http://savannah.gatech.edu

APPENDIX - LIST OF CONTENTS

A-1 MODELING WAVE CONDITIONS

The basis for using the module to model wave conditions are wave height time series for a certain offshore site. In this example, a site at the East Coast of the UK was chosen (site **1**) and time series from 1989 to 2010 are used in a 6 h resolution. The time series has to be in plaintext (ASCII) format and shall have the following exemplary form:

Table 11: Excerpt from an exemplary wave height time series

Year	Month	Day	Hour	Hs
1989	1	1	0	2.750874e-001
1989	1	1	6	2.631577e-001
1989	1	1	12	2.591035e-001
1989	1	1	18	2.390890e-001
1989	1	2	0	2.223317e-001
1989	1	2	6	1.984636e-001
1989	1	2	12	2.183822e-001

In order to enable monthly generated sea states, input files shall be available for every month separately and should be named as follows:

"YEAR_Month (abbr.)_DAY_to_Month(abbr.)_DAY" e.g.: *"1989_Jan_1_to_Jan_31"*

YEAR and DAY values are numbers. Abbreviated months follow the order:

January ~ Jan
February ~ Feb
March ~ Mar
April ~ Apr
May ~ May
June ~ June
July ~ July
August ~ Aug
September ~ Sept
October ~ Oct
November ~ Nov
December ~ Dec

With the function *"read_output_generate_markov_FINAL"* (Appendix A-4), Markov matrices can then be generated. The output of the function are monthly generated Markov matrices in plaintext format. Following steps have to be followed in order to run this function:

1. Set input path
2. Set output path
3. Set number of years (default: 22)
4. Choose month (default: 1 ~ January / 2 ~ February, etc.)

(1) Set the path of the folder in which wave height time series files are stored manually in the Matlab source code. (2) Here, the generated Markov matrices will be stored. Watch out carefully to not generate the matrix for one month twice, because values are going to be added to the existing output file, which will result in errors in further steps. (3) Should be the maximum number of years (files) available to consider the maximum

number of observed transitions for the Markov matrix. (4) One month at a time can be chosen. The definition of which month to handle is set in the code, whereas only one month can be handled at a time.

After this step is performed for all months, the function *"simulate_wave_time_series01"* can be used to generate wave height time series. The first step of this function is to read in previously generated Markov matrices with the function *"read_markov_FINAL"* (Appendix (b)). A matrix containing all single Markov matrices is the output. Before using this function, the path at the end of the code has to be changed to the one containing the matrices. Markov matrices have to exist for all 12 months in a size of 18 x 18. Then, the following steps have to be done to run *"simulate_wave_time_series01"* (Appendix (c)):

1. Define start and end year
2. Define start and end month
3. Set path for the function *"read_markov_FINAL"* (if function is not in the current folder)

(1) Start and end year defining the period to model time series. (2) Start and end month shall be set to one respectively twelve. (3) Change the path for the *"read_markov_FINAL"* function if necessary and make sure the definition of the bins is the same for Markov matrices.

A-2 MODELING WIND CONDITIONS

The function *"simulate_wind_time_series"* is used to model wind speed time series. Before being able to model wind conditions, wave height time series have to be generated as synthetic wind speeds are modeled based on them. This function is included in the code (Appendix (d)). As a first step, existent data has to be evaluated in order to define a matrix containing the conditional probability distribution of wind speeds at a given wave height state. Therefore, the following steps have to be performed:

1. Run *"simulate_wave_time_series01"*
2. Run "*wind_wave_comp*" (Appendix (e))
3. Run *"sort_data"* (Appendix (f))
4. Run *"wind_distribution"* (Appendix (g))

For (1) use instructions as given in Appendix chapter A-1 . One always needs to ensure that the handled offshore site is the same for wind and wave modeling. In (2), define paths in the functions *"read_output02_wind"* (Appendix (h))and *"read_output02_waves"* (Appendix (i)). (3) clusters observed wind speeds according to their corresponding wave height. In (4), an output ASCII file is generated to display the probability distribution of wind speeds at the according wave height. Therefore, define the desired output path in the code. One should run the function only once, as newly generated files are added to existing ones and therefore cause errors in further steps.

After the above described steps have been performed once, wind time series can be generated with the function *"simulate_wind_time_series"*. Consider all instructions stated for *"simulate_wave_time_series01"*, "*wind_wave_comp*" and *"sort_data"*.

A-3 SIMULATING THE OPERATION OF A WIND FARM

To finally model the operating phase of an offshore wind farm, the script *"park_simulation"* (Appendix (j)) has to be executed. This script accesses several functions depending on the state of simulation. The functions are not explained in detail, but shown in commented form in Appendix (k) to (ee). The park simulation requires the adjustment of multiple parameters. The following steps have to be performed, provided all plaintext weather files have been generated before. Define:

1. Turbine rated power *"P_turb"* (Default: 5 MW)
2. Cut in wind speed *"cut_in"* (Default: 3 m/s)
3. Rated wind speed *"v_rated"* (Default: 11.4 m/s)
4. Cut out wind speed *"cut_out"* (Default: 25 m/s)
5. Simulation duration in 6 h steps *"t_sim"* (Default: 1460 ~ 1 year)
6. Time of accurate weather forecast *"lookahead_time"* in 6 h steps (Default: 4 ~ 24 h)
7. Maximum time a ship is busy in park but reserves failures *"factor_time_for_failure_ship*" (Default: 3 ~ 18 h)
8. Maximum duration a ship can be busy offshore *"max_time_offshore"* (Default: 30 ~ 7.5 days)
9. Assumed transit time for ships *"assumed_transit_time_s"* (Default: 1 ~ 6 h)
10. Assumed transit time for cranes *"assumed_transit_time_c"* (Default: 2 ~12 h)
11. Assumed (fix) repair times for the twelve subsystems *"assumed_repair_times"*
12. Assumed No. of people required for repair (only if no crane is necessary) *"assumed_people_required"*
13. Assumed yearly failure rates for all twelve subsystems *"assumed_failure_rate"*
14. Number of turbines in park *"n_turb"* (Default: 500)
15. Number of subsystems *"n_components"* (Default: 12)
16. Which component can be handled by a crew only *"crew_component"*
17. Which component needs a crane in case of a failure *"crane_component"*
18. No. of ships available for repair *"n_ships"* (Default: 1)
19. No. of people a ship can carry *"max_people"* (Default: 4)
20. Maximum significant wave height for offshore deployment ship *"max_hs"* (Default: 1.5 m)
21. No. of cranes available for repair *"n_cranes"* (Default: 3)
22. Maximum significant wave height for offshore deployment crane *"max_hs"* (Default: 1.5 m)

After all steps have been taken and therefore all parameters have been set, check that the weather modeling is done for the desired offshore site. Define any required outputs at the end of the script. Some examples for outputs are given in commented form. After that, the simulation can be started. Depending on the computing power and demanded outputs, the simulation of one year takes about 5 minutes.

A-4 MATLAB SOURCE CODE

(a) Source code: *"read_output_generate_markov_FINAL"*

```
% Input: Time series .txt data

% Output: Single Markov matrices for each month as .txt

% Instruction: Define path for input data at the bottom of the code (line 73).
    % Define "y1" and "y2" as start and end year. Here: 1 & 22.
    % Define which month the Markov matrix shall be generated for. Define
    % the output path in line 126

% Attention: This function is only possible for one month at a time. File
    % names have to be in the form "1January",...
    % New generated Markov matrices will be added to existing ones', make
    % sure there is no matrix with the same name already stored.

clear all

global y1 y2

% Start and end year (1 = 1989 / 22 = 2010)
y1 = 1;
y2 = 22;

for path_i = 1 % Define the month here!

    if path_i == 1
        path_ii = '1January';
    end

    if path_i == 2
        path_ii = '2February';
    end

    if path_i == 3
        path_ii = '3March';
    end

    if path_i == 4
        path_ii = '4April';
    end

    if path_i == 5
        path_ii = '5May';
    end

    if path_i == 6
        path_ii = '6June';
    end

    if path_i == 7
        path_ii = '7July';
    end
    if path_i == 8
        path_ii = '8August';
    end


    if path_i == 9
        path_ii = '9September';
```

```
    end


    if path_i == 10
        path_ii = '10October';
    end

    if path_i == 11
        path_ii = '11November';
    end

    if path_i == 12
        path_ii = '12December';
    end


%Define input data path here!
enterpath = ['D:\UNI\DA\Data_processing\Output\Waves\' path_ii '\'];


path = [enterpath '*.txt'];
liste = dir(path);
files = {liste.name};

    % Generate "Timeseries_ALL" for each month and every recorded year
    for i = y1:y2
        % import files
        D(1,i) = importdata([enterpath liste(i,1).name]);
        % consider different month-lengths and leap years
        maxLen(1,i) = length(D(1,i).data(:,13));
        % Read out chronological time series (here in column 13!)
        Timeseries_ALL(1:maxLen(1,i),i) = D(1,i).data(:,13);
    end

% Generate .txt Markov matrix for the required month

% Define wave height bins here!
bindef = [0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4...
 0.4 0.4 0.4];

% Calculate the cumulative sum of the bins
bins = cumsum(bindef);

%Define the size of the Mrkov matrix
k = length(bins);

%Set the Markov matrix blanc
M = zeros(k,k);

[d f] = size(Timeseries_ALL);

for a = y1:y2;
    % Consider differing lengths of monthly data (28,29,30,31 days)
    Len = length(Timeseries_ALL(:,a));
    for t = 1 :Len-1;
        % define which values are greater than the "bins"
        p = bins > Timeseries_ALL(t,a);
        % defines next value greater than "bins"
        q = bins > Timeseries_ALL(t+1,a);
        % find first value beeing greater than bins
        i = find(p,1);
        % find next value beeing greater than bins
        j = find(q,1);
        % generate Markov transition matrix by adding +1 to the above
        % defined transitions
```

```
        M(i,j)=M(i,j)+1;
    end
M;
end
C = sum(M,2);
M_norm = zeros(k,k);

    % normalize Markov matrix
    for f = 1:k;
        M_norm(f,:) = M(f,:)/C(f);
    end

    % replaces all NaN values by a "0"
    ii=isnan(M_norm);
    M_norm(ii)=0;

    % Define output path here!

    % Generate .txt output file for Markov matrices
    filename = ['D:\UNI\DA\Data_processing\Output\Waves\Markov\Markov_' path_ii '.txt'];
    fid=fopen(filename,'a');
    row=size(M_norm,1);
    column=size(M_norm,2);
    for i=1:row;
        for j=1:column      ;
            fprintf(fid,'%+2.8E\t',M_norm(i,j)) ;
        end;
            fprintf(fid,'\r\n');
    end;
end
```

(b) Source Code: *"read_markov_FINAL"*

```
% Input: Monthly generated 18 x 18 Markov matrices in .txt format

% Output: "Markov_year" matrix. Markov matrix for each month

% Instruction: Define path at the bottom of this document

% Attention: This code is written for 18 x 18 Markov matrices. The
    % resolution of wave height bins cannot be changed here. Markov matrices
    % have to be existent for all 12 months.
    % .txt files have to be named according to the code below
    % (1January,...)

for path_i = 1:12

    if path_i == 1
        path_ii = '1January';
    end

    if path_i == 2
        path_ii = '2February';
    end

    if path_i == 3
        path_ii = '3March';
    end

    if path_i == 4
        path_ii = '4April';
    end

    if path_i == 5
        path_ii = '5May';
    end

    if path_i == 6
        path_ii = '6June';
    end

    if path_i == 7
        path_ii = '7July';
    end

    if path_i == 8
        path_ii = '8August';
    end

    if path_i == 9
        path_ii = '9September';
    end

    if path_i == 10
        path_ii = '10October';
    end
```

Appendix - Matlab User Instruction

```
if path_i == 11
        path_ii = '11November';
    end

    if path_i == 12
        path_ii = '12December';
    end

% Define path here!
path = ['D:\UNI\DA\Data_processing\Output\Waves2\Markov\Markov_' path_ii '.txt'];

% Import data from paths and define matrix "Markov_year"
Markov_year(:,:,path_i) = importdata(path);

end
```

(c) Source Code: *"simulate_wave_time_series01"*

```
% Input: .txt files monthly Markov matrices (18 x 18)

% Output: Significant wave height time series

% Instruction: Define start and end year and month. It is only possible to
    % start with January as the first month. Define the path where
    % "read_markov_FINAL" can be found.

% Attention: Make sure "bindef" is identical to the one defined for the
    % Markov matrices.

% Import .txt Markov matrices MAKE SURE "bindef" IS SIMILAR TO THE ONE FROM
"read_markov_FINAL"!!!!
% addpath('D:\UNI\DA\Data_processing\Txt_data');
read_markov_FINAL;

% start in year "y1"
y1 = 1;
% finish in year "y2"
y2 = 22;
% start in month "start_month" --> January!
start_month = 1;
% end in month "end_month" --> December!
end_month = 12;

bindef = [0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4];
bins = cumsum(bindef);

for x = 1;
    wh(x) = bins(x)/2;
end

for x = 2:length(bins);
    wh(x) = (bins(x) + bins(x-1))/2;
end

wh;

for month = 1:12

      C_norm = Markov_year(:,:,month)/sum(Markov_year(:,:,month));
      F(:,:,month) = cumsum(C_norm);

end

steps0 = 1;
steps1 = 124; %how many simulation steps have to be performed January
steps2 = 112;  %February
steps3 = 124;  %March
steps4 = 120;  %April
steps5 = 124;  %May
steps6 = 120;  %June
steps7 = 124;  %July
steps8 = 124;  %August
steps9 = 120;  %September
steps10 = 124; %October
steps11 = 120; %November
steps12 = 124; %December

for y = y1:y2;
% generate random variable r1
r1 = rand;
```

Appendix - Matlab User Instruction

```
% search for the first value greater than r1 for start point
u = F(:,:,start_month) > r1; %Has to be January up to now!
Z0 = find(u,1);

    for jan = steps0:steps1;
        % choose Markov matrix for January
        while sum(Markov_year(Z0,:,1),2) == 0
            Z0 = Z0-1;
        end
        container_jan1(jan)=Z0;
        r2 = rand;
        G = cumsum(Markov_year(Z0,:,1));

        v = G > r2;
        Z1= find(v,1);

        Z0 = Z1;
        container_jan2(jan)=Z1;
        wh2(jan,y,1) = wh(container_jan1(jan));
    end

    for feb = steps0:steps2;
        while sum(Markov_year(Z1,:,2),2) == 0
            Z1 = Z1-1;
        end
        container_feb2(feb)=Z1;
        r2 = rand;
        G = cumsum(Markov_year(Z1,:,2));

        v = G > r2;
        Z2= find(v,1);
       Z1 = Z2;
        container_feb2(feb)=Z2;
        wh2(feb,y,2) = wh(container_feb2(feb));
    end

    for mar = steps0:steps3;
        while sum(Markov_year(Z2,:,3),2) == 0
            Z2 = Z2-1;
        end
        container_mar3(mar)=Z2;
        r2 = rand;
        G = cumsum(Markov_year(Z2,:,3));

        v = G > r2;
        Z3= find(v,1);

        Z2 = Z3;
        container_mar3(mar)=Z3;
        wh2(mar,y,3) = wh(container_mar3(mar));
    end
```

Appendix - Matlab User Instruction

```
for apr = steps0:steps4;
        while sum(Markov_year(Z3,:,4),2) == 0
            Z3 = Z3-1;
        end
        container_apr4(apr)=Z3;
        r2 = rand;
        G = cumsum(Markov_year(Z3,:,4));

        v = G > r2;
        Z4= find(v,1);

        Z3 = Z4;
        container_apr4(apr)=Z4;
        wh2(apr,y,4) = wh(container_apr4(apr));
    end

    for may = steps0:steps5;
        while sum(Markov_year(Z4,:,5),2) == 0
            Z4 = Z4-1;
        end
        container_may5(may)=Z4;
        r2 = rand;
        G = cumsum(Markov_year(Z4,:,5));

        v = G > r2;
        Z5= find(v,1);

        Z4 = Z5;
        container_may5(may)=Z5;
        wh2(may,y,5) = wh(container_may5(may));
    end
    for june = steps0:steps6;
        while sum(Markov_year(Z5,:,6),2) == 0
            Z5 = Z5-1;
        end
        container_june6(june)=Z5;
        r2 = rand;
        G = cumsum(Markov_year(Z5,:,6));

        v = G > r2;
        Z6= find(v,1);

        Z5 = Z6;
        container_june6(june)=Z6;
        wh2(june,y,6) = wh(container_june6(june));
    end

    for july = steps0:steps7;
        while sum(Markov_year(Z6,:,7),2) == 0
            Z6 = Z6-1;
        end
        container_july7(july)=Z6;
        r2 = rand;
        G = cumsum(Markov_year(Z6,:,7));

        v = G > r2;
        Z7= find(v,1);

        Z6 = Z7;
        container_july7(july)=Z7;
        wh2(july,y,7) = wh(container_july7(july));
    end
```

```
for aug = steps0:steps8;
    while sum(Markov_year(Z7,:,8),2) == 0
        Z7 = Z7-1;
    end
    container_aug8(aug)=Z7;
    r2 = rand;
    G = cumsum(Markov_year(Z7,:,8));

    v = G > r2;
    Z8= find(v,1);

    Z7 = Z8;
    container_aug8(aug)=Z8;
    wh2(aug,y,8) = wh(container_aug8(aug));
end

for sept = steps0:steps9;
    while sum(Markov_year(Z8,:,9),2) == 0
        Z8 = Z8-1;
    end
    container_sept9(sept)=Z8;
    r2 = rand;
    G = cumsum(Markov_year(Z8,:,9));

    v = G > r2;
    Z9= find(v,1);

    Z8 = Z9;
    container_sept9(sept)=Z9;
    wh2(sept,y,9) = wh(container_sept9(sept));
end

for oct = steps0:steps10;
    while sum(Markov_year(Z9,:,10),2) == 0
        Z9 = Z9-1;
    end
    container_oct10(oct)=Z9;
    r2 = rand;
    G = cumsum(Markov_year(Z9,:,10));

    v = G > r2;
    Z10= find(v,1);

    Z9 = Z10;
    container_oct10(oct)=Z10;
    wh2(oct,y,10) = wh(container_oct10(oct));
end

for nov = steps0:steps11;
    while sum(Markov_year(Z10,:,11),2) == 0
        Z10 = Z10-1;
    end
    container_nov11(nov)=Z10;
    r2 = rand;
    G = cumsum(Markov_year(Z10,:,11));

    v = G > r2;
    Z11= find(v,1);

    Z10 = Z11;
    container_nov11(nov)=Z11;
    wh2(nov,y,11) = wh(container_nov11(nov));
end
```

Appendix - Matlab User Instruction

```
for dec = steps0:steps12;
        while sum(Markov_year(Z11,:,12),2) == 0
            Z11 = Z11-1;
        end
        container_dec12(dec)=Z11;
        r2 = rand;
        G = cumsum(Markov_year(Z11,:,12));

        v = G > r2;
        Z12= find(v,1);

        Z11 = Z12;
        container_dec12(dec)=Z12;
        wh2(dec,y,12) = wh(container_dec12(dec));
    end

    wh2_full = wh2;
for o = 1:12
        for p = 1:121
            wh_year(p+121*(o-1),1) = wh2_full(p,1,o);
        end
    end
end

for a = 1:124
    for b=1:22
        for c = 1:12
            wh2_full_1(a,c,b) = wh2_full(a,b,c);
        end
    end
end

%displays ALL values over every year
wh2_full_1 = wh2_full_1(wh2_full_1~=0);

% yearly data for 22 years
WAVE_year = zeros(1460,22);
start_1 = 1;
end_1 = 1460;
WAVE_year(1:1460,1) = wh2_full_1(1:1460,1);

for p = 2:22
    start_1 = start_1 + 1460;
    end_1 = end_1 + 1460;
    WAVE_year(1:1460,p) = wh2_full_1(start_1:end_1,1);
end

% calculate standard errors yearly

ERROR_WAVE = std(WAVE_year)/(sqrt(length(WAVE_year)));
STDerror_waves = zeros(length(ERROR_WAVE),1);

for i = 1:length(std(WAVE_year))
    STDerror_waves(i,1) = ERROR_WAVE(1,i);
end
```

(d) Source Code: *"simulate_wind_time_series"*

```
% Input: Function "simulate_wave_time_series01" / "wind_wave_comp" /
% "sort_data"

% Output: Wind speed time series

% Instruction: Change the path for wind_distr to the one containing the
% matrix f_wwb_ALL_norm.txt

% Attention: Make sure all path definition in underlying functions are
% correct!


simulate_wave_time_series01;

wind_wave_comp;

sort_data;
% addpath('D:\UNI\DA\Data_processing\Txt_data');

wind_distr =
importdata('D:\UNI\DA\Data_processing\Output\Wind2\Distribution\f_wwb_ALL_norm.txt');

for wwx = 1:length(wh2_full_1)
    if wh2_full_1(wwx)<= bins(1)
        ww = rand;
        Q = cumsum(wind_distr(1,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) = bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end
    if  wh2_full_1(wwx) > bins(1) && wh2_full_1(wwx) <=bins(2)
        ww = rand;
        Q = cumsum(wind_distr(2,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) = bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end
    if  wh2_full_1(wwx) > bins(2) && wh2_full_1(wwx) <=bins(3)
        ww = rand;
        Q = cumsum(wind_distr(3,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) = bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end
```

```
if  wh2_full_1(wwx) > bins(3) && wh2_full_1(wwx) <=bins(4)
        ww = rand;
        Q = cumsum(wind_distr(4,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) = bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end
    if  wh2_full_1(wwx) > bins(4) && wh2_full_1(wwx) <=bins(5)
        ww = rand;
        Q = cumsum(wind_distr(5,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) = bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end
    if  wh2_full_1(wwx) > bins(5) && wh2_full_1(wwx) <=bins(6)
        ww = rand;
        Q = cumsum(wind_distr(6,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) = bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end

    if  wh2_full_1(wwx) > bins(6) && wh2_full_1(wwx) <=bins(7)
        ww = rand;
        Q = cumsum(wind_distr(7,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) = bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end
    if  wh2_full_1(wwx) > bins(7) && wh2_full_1(wwx) <=bins(8)
        ww = rand;
        Q = cumsum(wind_distr(8,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) = bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end
    if  wh2_full_1(wwx) > bins(8) && wh2_full_1(wwx) <=bins(9)
        ww = rand;
        Q = cumsum(wind_distr(9,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) = bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end
    if  wh2_full_1(wwx) > bins(9) && wh2_full_1(wwx) <=bins(10)
        ww = rand;
        Q = cumsum(wind_distr(10,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) = bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end
    if  wh2_full_1(wwx) > bins(10) && wh2_full_1(wwx) <=bins(11)
        ww = rand;
        Q = cumsum(wind_distr(11,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) = bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end
```

```
   if  wh2_full_1(wwx) > bins(11) && wh2_full_1(wwx) <=bins(12)

        ww = rand;
        Q = cumsum(wind_distr(12,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) = bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end
    if  wh2_full_1(wwx) > bins(12) && wh2_full_1(wwx) <=bins(13)
        ww = rand;
        Q = cumsum(wind_distr(13,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) = bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end

    if  wh2_full_1(wwx) > bins(13) && wh2_full_1(wwx) <=bins(14)
        ww = rand;
        Q = cumsum(wind_distr(14,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) = bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end
    if  wh2_full_1(wwx) > bins(14) && wh2_full_1(wwx) <=bins(15)
        ww = rand;
        Q = cumsum(wind_distr(15,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) = bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end
    if  wh2_full_1(wwx) > bins(15) && wh2_full_1(wwx) <=bins(16)
        ww = rand;
        Q = cumsum(wind_distr(16,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) =bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end
    if  wh2_full_1(wwx) > bins(16) && wh2_full_1(wwx) <=bins(17)
        ww = rand;
        Q = cumsum(wind_distr(17,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) = bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end

if  wh2_full_1(wwx) > bins(17) && wh2_full_1(wwx) <=bins(18)
        ww = rand;
        Q = cumsum(wind_distr(18,:));

        wp = Q > ww;
        wpr = find(wp,1);
        wind_ALL(wwx,1) = bins_wind(wpr)-(bins_wind(wpr+1)-bins_wind(wpr))/2;
    end
end

WIND_WAVE_ALL = [wh2_full_1 wind_ALL];

corr_WIND_WAVE_ALL_sim = corr(WIND_WAVE_ALL(:,1),WIND_WAVE_ALL(:,2));
```

(e) Source Code: *"wind_wave_comp"*

```
% Input: .txt files of existent wind and wave time series & function
% "simulate_wave_time_series01"

% Output: Matrix "wind_wave_ALL" containing all observed wind speeds and
% wave heights in two columns

% Instruction: Change the path in "read_output02_wind" and
% "read_output02_waves" to the one containing the desired time series

% Attention: Make sure both paths are the same, so that the correct time
% series are combined for the matrix!

read_output02_wind;

read_output02_waves;

[a b] = size(wh_year);

wind_wave = zeros(a,b);
RHO = zeros(2*b,1);


wspeed_ALL = wspeed_year_real(:);

wave_ALL = wh_year_real(:);

wind_wave_ALL = [wspeed_ALL wave_ALL];
```

(f) Source Code: *"sort_data"*

```
% Input: Function "wind_wave_comp"

% Output: Matrices "wind_wave_binX" containing all observed wind speeds in
% a certain wave height bin

% Attention: Make sure bin sizes are consistent! Default: wave height bin
% size = 0.4 m, wind speed 1 m/s

A = sortrows(wind_wave_ALL,2);

bindef = [0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4];

bins = cumsum(bindef);

bindef_wind = [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1];

bins_wind = cumsum(bindef_wind);

for b = 1:length(bins)
    f(1,b) = length(find(A(:,2)<bins(b)));
end

for k = 2:length(bins)
    f(1,k) = f(1,k) - sum(f(1:(k-1)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin1_0 = A(1:f(1));
wind_wave_bin1 = wind_wave_bin1_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin2_0 = A(f(1)+1:sum(f(1:2)));
wind_wave_bin2 = wind_wave_bin2_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin3_0 = A(sum(f(1:2))+1:sum(f(1:3)));
wind_wave_bin3 = wind_wave_bin3_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin4_0 = A(sum(f(1:3))+1:sum(f(1:4)));
wind_wave_bin4 = wind_wave_bin4_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin5_0 = A(sum(f(1:4))+1:sum(f(1:5)));
wind_wave_bin5 = wind_wave_bin5_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin6_0 = A(sum(f(1:5))+1:sum(f(1:6)));
wind_wave_bin6 = wind_wave_bin6_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin7_0 = A(sum(f(1:6))+1:sum(f(1:7)));
```

```
wind_wave_bin7 = wind_wave_bin7_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin8_0 = A(sum(f(1:7))+1:sum(f(1:8)));
wind_wave_bin8 = wind_wave_bin8_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin9_0 = A(sum(f(1:8))+1:sum(f(1:9)));
wind_wave_bin9 = wind_wave_bin9_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin10_0 = A(sum(f(1:9))+1:sum(f(1:10)));
wind_wave_bin10 = wind_wave_bin10_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin11_0 = A(sum(f(1:10))+1:sum(f(1:11)));
wind_wave_bin11 = wind_wave_bin11_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin12_0 = A(sum(f(1:11))+1:sum(f(1:12)));
wind_wave_bin12 = wind_wave_bin12_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin13_0 = A(sum(f(1:12))+1:sum(f(1:13)));
wind_wave_bin13 = wind_wave_bin13_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin14_0 = A(sum(f(1:13))+1:sum(f(1:14)));
wind_wave_bin14 = wind_wave_bin14_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin15_0 = A(sum(f(1:14))+1:sum(f(1:15)));
wind_wave_bin15 = wind_wave_bin15_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin16_0 = A(sum(f(1:15))+1:sum(f(1:16)));
wind_wave_bin16 = wind_wave_bin16_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin17_0 = A(sum(f(1:16))+1:sum(f(1:17)));
wind_wave_bin17 = wind_wave_bin17_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wind_wave_bin18_0 = A(sum(f(1:17))+1:sum(f(1:18)));
wind_wave_bin18 = wind_wave_bin18_0(:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

(g) Source Code: *"wind_distribution"*

```
% Input: Function sort_data

 % Output: .txt file containing the conditional probability distribution of
 % wind speeds at a certain wave height.

 % Instruction: Define output path "filename" at the bottom of the code

 % Attention: Carry out this function only once for a time series.
 % Additional runs will change the existent .txt output file!


for wwb1 = 1:length(bins_wind)
    f_wwb1(1,wwb1) = length(find(wind_wave_bin1(:,1)<bins_wind(wwb1)));
end
for wwb1 = 2:length(bins_wind)
    f_wwb1(1,wwb1) = f_wwb1(1,wwb1) - sum(f_wwb1(1:(wwb1-1)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for wwb2 = 1:length(bins_wind)
    f_wwb2(1,wwb2) = length(find(wind_wave_bin2(:,1)<bins_wind(wwb2)));
end
for wwb2 = 2:length(bins_wind)
    f_wwb2(1,wwb2) = f_wwb2(1,wwb2) - sum(f_wwb2(1:(wwb2-1)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for wwb3 = 1:length(bins_wind)
    f_wwb3(1,wwb3) = length(find(wind_wave_bin3(:,1)<bins_wind(wwb3)));
end
for wwb3 = 2:length(bins_wind)
    f_wwb3(1,wwb3) = f_wwb3(1,wwb3) - sum(f_wwb3(1:(wwb3-1)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for wwb4 = 1:length(bins_wind)
    f_wwb4(1,wwb4) = length(find(wind_wave_bin4(:,1)<bins_wind(wwb4)));
end
for wwb4 = 2:length(bins_wind)
    f_wwb4(1,wwb4) = f_wwb4(1,wwb4) - sum(f_wwb4(1:(wwb4-1)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for wwb5 = 1:length(bins_wind)
    f_wwb5(1,wwb5) = length(find(wind_wave_bin5(:,1)<bins_wind(wwb5)));
end
for wwb5 = 2:length(bins_wind)
    f_wwb5(1,wwb5) = f_wwb5(1,wwb5) - sum(f_wwb5(1:(wwb5-1)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for wwb6 = 1:length(bins_wind)
    f_wwb6(1,wwb6) = length(find(wind_wave_bin6(:,1)<bins_wind(wwb6)));
end
for wwb6 = 2:length(bins_wind)
    f_wwb6(1,wwb6) = f_wwb6(1,wwb6) - sum(f_wwb6(1:(wwb6-1)));
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for wwb7 = 1:length(bins_wind)
    f_wwb7(1,wwb7) = length(find(wind_wave_bin7(:,1)<bins_wind(wwb7)));
end
for wwb7 = 2:length(bins_wind)
    f_wwb7(1,wwb7) = f_wwb7(1,wwb7) - sum(f_wwb7(1:(wwb7-1)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for wwb8 = 1:length(bins_wind)
    f_wwb8(1,wwb8) = length(find(wind_wave_bin8(:,1)<bins_wind(wwb8)));
end
for wwb8 = 2:length(bins_wind)
    f_wwb8(1,wwb8) = f_wwb8(1,wwb8) - sum(f_wwb8(1:(wwb8-1)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for wwb9 = 1:length(bins_wind)
    f_wwb9(1,wwb9) = length(find(wind_wave_bin9(:,1)<bins_wind(wwb9)));
end
for wwb9 = 2:length(bins_wind)
    f_wwb9(1,wwb9) = f_wwb9(1,wwb9) - sum(f_wwb9(1:(wwb9-1)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for wwb10 = 1:length(bins_wind)
    f_wwb10(1,wwb10) = length(find(wind_wave_bin10(:,1)<bins_wind(wwb10)));
end
for wwb10 = 2:length(bins_wind)
    f_wwb10(1,wwb10) = f_wwb10(1,wwb10) - sum(f_wwb10(1:(wwb10-1)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for wwb11 = 1:length(bins_wind)
    f_wwb11(1,wwb11) = length(find(wind_wave_bin11(:,1)<bins_wind(wwb11)));
end
for wwb11 = 2:length(bins_wind)
    f_wwb11(1,wwb11) = f_wwb11(1,wwb11) - sum(f_wwb11(1:(wwb11-1)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for wwb12 = 1:length(bins_wind)
    f_wwb12(1,wwb12) = length(find(wind_wave_bin12(:,1)<bins_wind(wwb12)));
end
for wwb12 = 2:length(bins_wind)
    f_wwb12(1,wwb12) = f_wwb12(1,wwb12) - sum(f_wwb12(1:(wwb12-1)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for wwb13 = 1:length(bins_wind)
    f_wwb13(1,wwb13) = length(find(wind_wave_bin13(:,1)<bins_wind(wwb13)));
end
for wwb13 = 2:length(bins_wind)
    f_wwb13(1,wwb13) = f_wwb13(1,wwb13) - sum(f_wwb13(1:(wwb13-1)));
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for wwb14 = 1:length(bins_wind)
    f_wwb14(1,wwb14) = length(find(wind_wave_bin14(:,1)<bins_wind(wwb14)));
end
for wwb14 = 2:length(bins_wind)
    f_wwb14(1,wwb14) = f_wwb14(1,wwb14) - sum(f_wwb14(1:(wwb14-1)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for wwb15 = 1:length(bins_wind)
    f_wwb15(1,wwb15) = length(find(wind_wave_bin15(:,1)<bins_wind(wwb15)));
end
for wwb15 = 2:length(bins_wind)
    f_wwb15(1,wwb15) = f_wwb15(1,wwb15) - sum(f_wwb15(1:(wwb15-1)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for wwb16 = 1:length(bins_wind)
    f_wwb16(1,wwb16) = length(find(wind_wave_bin16(:,1)<bins_wind(wwb16)));
end
for wwb16 = 2:length(bins_wind)
    f_wwb16(1,wwb16) = f_wwb16(1,wwb16) - sum(f_wwb16(1:(wwb16-1)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for wwb17 = 1:length(bins_wind)
    f_wwb17(1,wwb17) = length(find(wind_wave_bin17(:,1)<bins_wind(wwb17)));
end
for wwb17 = 2:length(bins_wind)
    f_wwb17(1,wwb17) = f_wwb17(1,wwb17) - sum(f_wwb17(1:(wwb17-1)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for wwb18 = 1:length(bins_wind)
    f_wwb18(1,wwb18) = length(find(wind_wave_bin18(:,1)<bins_wind(wwb18)));
end
for wwb18 = 2:length(bins_wind)
    f_wwb18(1,wwb18) = f_wwb18(1,wwb18) - sum(f_wwb18(1:(wwb18-1)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

f_wwb_ALL = [f_wwb1;f_wwb2;f_wwb3;f_wwb4;f_wwb5;f_wwb6;f_wwb7;f_wwb8;...
    f_wwb9;f_wwb10;f_wwb11;f_wwb12;f_wwb13;f_wwb14;f_wwb15;f_wwb16;f_wwb17;f_wwb18];

k = length(f_wwb_ALL);

%sums up the rows of markov matrix
U = sum(f_wwb_ALL,2);
%blanc matrix for normalized markov matrix
f_wwb_ALL_norm = zeros(length(bins),length(bins_wind));

for f = 1:length(bins);
    f_wwb_ALL_norm(f,:) = f_wwb_ALL(f,:)/U(f); %normalizes markov matrix
end

ii=isnan(f_wwb_ALL_norm);   %replaces all NaN values by a "0"
f_wwb_ALL_norm(ii)=0;
```

```
[row column] = size(f_wwb1);

filename = ['D:\UNI\DA\Data_processing\Output\Wind2\Distribution\f_wwb_ALL_norm1.txt']
fid=fopen(filename,'a');
row=size(f_wwb_ALL_norm,1);
column=size(f_wwb_ALL,2);
for i=1:row;
for j=1:column      ;
fprintf(fid,'%+2.8E\t',f_wwb_ALL_norm(i,j)) ;
end;
fprintf(fid,'\r\n');
end;
```

(h) Source Code: *"read_output02_wind"*

```
% Instruction: Change input path for .txt formatted wind speed time series

for path_i = 1:12

    if path_i == 1
        path_ii = '1January';
    end

    if path_i == 2
        path_ii = '2February';
    end

    if path_i == 3
        path_ii = '3March';
    end

    if path_i == 4
        path_ii = '4April';
    end

    if path_i == 5
        path_ii = '5May';
    end

    if path_i == 6
        path_ii = '6June';
    end

    if path_i == 7
        path_ii = '7July';
    end

    if path_i == 8
        path_ii = '8August';
    end

    if path_i == 9
        path_ii = '9September';
    end

    if path_i == 10
        path_ii = '10October';
    end

    if path_i == 11
        path_ii = '11November';
    end

    if path_i == 12
        path_ii = '12December';
    end
```

Appendix - Matlab User Instruction

```
enterpath = ['D:\UNI\DA\Data_processing\Output\Wind2\' path_ii '\'];


path = [enterpath '*.txt'];
liste = dir(path);
files = {liste.name};

    for year = 1:22
        D(1,year,path_i) = importdata([enterpath liste(year,1).name]);
        Len = length(D(1,year,path_i).data(:,13));
        All_Timeseries(1:Len,path_i,year) = D(1,year,path_i).data(:,13);
    end


end

for t = 1:22
    for o = 1:12
        for p = 1:121
            wspeed_year_real(p+121*(o-1),t) = All_Timeseries(p,o,t) ;
        end
    end
end

wspeed_year_real(wspeed_year_real(:,1)==0,:)=[];
```

(i) Source Code: *"read_output02_waves"*

```
% Instruction: Change input path for .txt formatted wave height time series


for path_i = 1:12

    if path_i == 1
        path_ii = '1January';
    end

    if path_i == 2
        path_ii = '2February';
    end

    if path_i == 3
        path_ii = '3March';
    end

    if path_i == 4
        path_ii = '4April';
    end

    if path_i == 5
        path_ii = '5May';
    end

    if path_i == 6
        path_ii = '6June';
    end

    if path_i == 7
        path_ii = '7July';
    end

    if path_i == 8
        path_ii = '8August';
    end

    if path_i == 9
        path_ii = '9September';
    end

    if path_i == 10
        path_ii = '10October';
    end

    if path_i == 11
        path_ii = '11November';
    end

    if path_i == 12
        path_ii = '12December';
    end



enterpath = ['D:\UNI\DA\Data_processing\Output\Waves2\' path_ii '\'];


path = [enterpath '*.txt'];
liste = dir(path);
```

Appendix - Matlab User Instruction

```
files = {liste.name};

    for year = 1:22
        D(1,year,path_i) = importdata([enterpath liste(year,1).name]);
        Len = length(D(1,year,path_i).data(:,13));
        All_Timeseries(1:Len,path_i,year) = D(1,year,path_i).data(:,13);
    end


end

for t = 1:22
for o = 1:12
    for p = 1:121
    wh_year_real(p+121*(o-1),t) = All_Timeseries(p,o,t) ;
    end
end
end

wh_year_real(wh_year_real(:,1)==0,:)=[];

wh_year_real = wh_year_real(wh_year_real~=0); %displays ALL values over every year
```

(j) Source Code: *"park_simulation"*

```
% Input: All weather information!

% Output: Desired outputs can be defined at the bottom of this code

% Instruction: Given seperately in the user manual

% Attention: Make sure that all data for weather modeling is correct!
% functions "simulate_wave_time_series01" and "simulate_wind_time_series"
% are used within this function!

% General Information
% Turbine Information for linearized power curve:
global P_turb % rated power in KW
global cut_in % cut in wind speed [m/s]
global v_rated % rated wind speed [m/s]
global cut_out % cut out wind speed [m/s]
global FIT % feed in tariff [€/kWh] --> here: 0.1801
P_turb = 5000;
cut_in = 3;
v_rated = 11.4;
cut_out = 25;
FIT = 0.1801;

% simulation time in 6h intervals
global t_sim
t_sim = 1460;

global LOG_s_transit_to
global LOG_s_transit_back
global LOG_c_transit_to
global LOG_c_transit_back
LOG_s_transit_to = zeros(1,1); % number of ships! only one run possible!!
LOG_s_transit_back = zeros(1,1); % number of ships! only one run possible!!
LOG_c_transit_to = zeros(1,1); % number of cranes! only one run possible!!
LOG_c_transit_back = zeros(1,1); % number of cranes! only one run possible!!



xx = 0;
for max_hs = 1.5 % or several runs! e.g.[1.0 1.4 1.8 2.2 2.6]

    xx = xx + 1;
    for runs = 1:5

        clearvars -except xx max_hs runs t_sim Availability LOG_s_transit_to...
             LOG_s_transit_back LOG_c_transit_to LOG_c_transit_back ...
             P_turb cut_in cut_out v_rated FIT Availability...
       LOG_s_transit_to Production_losses...
             Deployment_ship Deployment_crane

        COST = zeros(5,500);

        % initialize foo1 and baz1 for production losses (bottom of script)
        foo1 = 0;
        baz1 = 0;
        % xx = xx + 1;
        % weather simulation
        % (1)

        simulate_wave_time_series01
```

```
simulate_wind_time_series

% Importing wave data --> max. 32120 steps! If more required, go to
% "simulate_wave_time_series_01"
hs = WIND_WAVE_ALL(1:10000,1); % significant wave height modeled starting
at 1. january!!!
ws = WIND_WAVE_ALL(1:10000,2); % wind speed modeled starting at 1.
january!!!
% global simulation parameters
global lookahead_time ; % time weather can be predicted 100 % in 6 h ...
% intervals
global factor_time_for_failure_ship;
global factor_time_for_failure_crane;
global max_t_unreserved_park;
global max_time_offshore;
lookahead_time = 4;
factor_time_for_failure_ship = 3.0; %including a general assumption for
... % transit time!!
factor_time_for_failure_crane = 4.0;
max_t_unreserved_park = 10;
max_time_offshore = 30;


% states for components
global s_component_running;
global s_no_crew;
global s_component_unreserved;
global s_component_unreserved_park;
global s_component_reserved;
global s_component_down;
s_component_running = 1;
s_no_crew = 2;
s_component_reserved = 3;
s_component_unreserved = 4;
s_component_unreserved_park = 5;
s_component_down = 6;

% states for ship
global state_s_available_harbor;
global state_s_reserved_harbor;
global state_s_transit_to;
global state_s_transit_back;
global state_s_busy_park;
state_s_busy_park = 101;
state_s_available_harbor = 102;
state_s_reserved_harbor = 103;
state_s_transit_to = 104;
state_s_transit_back = 105;

 % states for crew
global s_in_repair;
global s_in_ship;
global s_waiting;
global s_reserved;
s_in_repair = 1;
s_in_ship = 2;
s_waiting = 3;
s_reserved = 4;

% states for crane
global state_c_available_harbor;
global state_c_reserved_harbor;
global state_c_transit_to;
global state_c_transit_back;
global state_c_busy_park;
global state_c_reserved;
global state_c_in_repair;
```

```
        state_c_busy_park = 201;
        state_c_available_harbor = 202;
        state_c_reserved_harbor = 203;
        state_c_transit_to = 204;
        state_c_transit_back = 205;
        state_c_reserved = 206;
        state_c_in_repair = 207;

        % ship transit time
        % (2)
        global assumed_transit_time_s
        assumed_transit_time_s = 1;

        % crane transit time
        % (2)
        global assumed_transit_time_c
        assumed_transit_time_c = 2;

        % assumed_repair_times
        % (3)
        global assumed_repair_times
        assumed_repair_times = [2 2 2 3 3 4 3 6 6 4 4 4];

        % assumed_people_required --> "10" means component is considered to be
        % repaired by crane with a fixed personnel.
        % (4)
        global assumed_people_required
        assumed_people_required = [2 2 2 2 2 10 2 10 10 10 10 10];

        % assumed failure_rates according to S. Faulstich et.al "Wind turbine
        % downtime and its importance for offshore deployment
        % (5)
        global assumed_failure_rate
        assumed_failure_rate = [0.57 0.43 0.25 0.23 0.18 0.17 0.13 0.11 0.10...
      0.11 0.10 0.05];

% initialize park info
        global n_turb
        global i_turb
        n_turb = 500;
        for i = 1:n_turb
            foo =
      struct('name_turbine',i,'repair_time',0,'n_unreserved_failures'...
                ,0,'n_people_at_turbine',0);
            bar(i) = foo;
        end
        i_turb = bar;

        % initialize component info - failure rates & info in Faulstich et. al.
        global n_components
        global crew_component
        global crane_component
        n_components = 12;
        % determines which components can be handeled by crew only
        crew_component = [1 2 3 4 5 7];
        % determindes which components can be handeled only by crane
        crane_component = [6 8 9 10 11 12];
```

Appendix - Matlab User Instruction

```
        % initialize component info for components assumed to be repaired by crew
        % only
        global i_component
        for i = 1:n_turb
            for j = crew_component
                foo = struct('name_component',j,'turbine',i,'ttf',999999,...
                'repair_time',0,'state',s_component_running,'people_required'...
                ,assumed_people_required(j),'crew_id',0,...
                 't_unreserved_in_park',0,...
                'crew_component',1,'crane_component',0);
                % foo.ttf = new_ttf(i,j);
                baz(i,j) = foo;
            end
            i_component = baz;
        end

        % initialize component info for components assumed to be repaired by         %
crane only
        for i = 1:n_turb
            for j = crane_component
                foo = struct('name_component',j,'turbine',i,'ttf',999999,...
                    'repair_time',0,'state',s_component_running, ...
                    'people_required',0,'crew_id',s_no_crew,...
                     't_unreserved_in_park',0,...
                    'crew_component',0,'crane_component',1);
                % foo.ttf = new_ttf(i,j);
                baz(i,j) = foo;
            end
            i_component = baz;
        end


      % initialize crew info
        global n_crews
        global i_crew
        n_crews = 1;
        for i = 1:n_crews
            foo = struct('crew_id',i,'state',0,'n_people_in_crew'...
             ,0,'t_repair_done',0,'at_which_turbine',0,...
             'at_which_component',0,'next_turbine',0,...
            'next_component',0);
            baz_crew(i) = foo;
        end
        i_crew = baz_crew;

        % initialize ship info
        global n_ships
        global i_ship
        n_ships = 1;
        for s = 1:n_ships
            foo = struct('state',state_s_available_harbor,'max_people',4,...
                'n_people',0,'n_people_free',0,'n_people_loaded',0,...
                'n_crews',0,'crew_ids',0,'n_crews_assigned',0,'transit_time',...
                assumed_transit_time_s,...
                'pot_transit_time',assumed_transit_time_s,'time_offshore',0,...
                'max_hs',max_hs,'min_weather_window_needed'...
                 ,0,'max_windspeed',50,'estimated_time_free'...
                 ,0,'estimated_weather_window_needed',0);
            i_ship1(s) = foo;
        end
        i_ship = i_ship1;
```

Appendix - Matlab User Instruction

```
% initialize crane info
       global n_cranes
       global i_crane
       n_cranes = 1;
       for s = 1:n_cranes
           foo = struct('state',state_c_available_harbor,...
                'at_which_turbine',0,'at_which_component',0,...
                'next_turbine',0,'next_component',0,...
                'time_to_repair',0,'transit_time',...
               assumed_transit_time_c,...
               'pot_transit_time',assumed_transit_time_c,'time_offshore',0,...
               'max_hs',max_hs,'min_weather_window_needed'...
                ,0,'estimated_time_free',0,'max_windspeed',50);
           i_crane1(s) = foo;
       end
       i_crane = i_crane1;

       turb_down = zeros(n_turb,1);


     for t = 1:t_sim
           fprintf('\n==> TIME STEP %8i <==\n',t);
           for i = 1:n_turb
               if i_turb(i).repair_time ~= 0
                   % production losses due to downtime
                   if ws(t) >= cut_in && ws(t) <= v_rated
                       foo1 = ws(t)*(P_turb/v_rated)*FIT*6;
                   end
                   if ws(t) > v_rated && ws(t) <= cut_out
                       foo1 = P_turb*FIT*6;
                   end
                   if ws(t) <= cut_in || ws(t) >= cut_out
                       foo1 = 0;
                   end
                   if ws(t+1) >= cut_in && ws(t+1) <= v_rated
                       baz1 = ws(t+1)*(P_turb/v_rated)*FIT*6;
                   end
                   if ws(t+1) > v_rated && ws(t+1) <= cut_out
                       baz1 = P_turb*FIT*6;
                   end
                   if ws(t+1) <= cut_in || ws(t+1) >= cut_out
                       baz1 = 0;
                   end
                   % linear interpolation of production losses!
                   COST(runs,i) = COST(runs,i) + (foo1+baz1)/2;

                   turb_down(i) = turb_down(i) + 1;
               end
               for j = 1:n_components
                   d_new_failure_01(i,j);
               end
           end
           for k = 1:n_ships
               d_s_busy_park(k,t,hs);
           end
           for c = 1:n_cranes
               d_c_busy_park_01(c,t,hs);
           end
           for k = 1:n_ships
               d_s_reserved_harbor(k,t,hs);
           end
           for k = 1:n_ships
               d_s_available_harbor(k,t,hs);
           end
           for c = 1:n_cranes
               d_c_reserved_harbor(c,t,hs);
           end
```

```
            for c = 1:n_cranes
                d_c_available_harbor(c);
            end
            for p = 1:length(i_crew)
                d_repair_crew(p); % TODO: repair time - 1 in next time step
            end
            for q = 1:n_cranes
                d_repair_crane(q,t,hs);
            end

% output
            for i = 1:n_turb
                for j = 1:n_components
                    if i_component(i,j).state == s_component_unreserved
                        fprintf('*** Component %3i at turbine %3i ...
                                 has unscheduled failure ***\n',j,i);
                    end
                end
            end

        end

        % Output

        % Availability
        Availability(runs,xx) = ((n_turb*t_sim) - sum(turb_down))/(n_turb*t_sim);

        % Production losses
        Production_losses(runs,xx) = sum(COST(runs,:),2);

        % No. of deployments
        Deployment_ship(runs,xx) = mean(LOG_s_transit_to);
        Deployment_crane(runs,xx) = mean(LOG_c_transit_to)/2;

    end
end

COST = sum(COST,2);
LOG_c_transit_to = (LOG_c_transit_to/2);
```

(k) Source Code: *"d_c_available_harbor"*

```
% function if crane is available in harbor

function d_c_available_harbor(c)

% initialize global variables
global i_crane
global state_c_available_harbor
global i_component
global state_c_reserved
global s_component_unreserved
global state_c_reserved_harbor
global i_turb
global s_component_reserved
global n_turb
global n_components

% make sure that crane is available harbor, because all cranes are
% passing this funtion!
if i_crane(c).state == state_c_available_harbor
    for i = 1:n_turb
        % check if crane has been reserved in the meantime
        if i_crane(c).state == state_c_reserved || i_crane(c).state == ...
                state_c_reserved_harbor ;
            break
        else
            % crane is not reserved --> check if turbine has an unreserved
            % failure
            % TODO what happens if unreserved park?!?
            if i_turb(i).n_unreserved_failures == 0
                continue
            else
                % turbine with unreserved failure --> check components
                for j = 1:n_components
                    % check if crane has been reserved in the meantime
                    if i_crane(c).state == state_c_reserved || ...
                            i_crane(c).state == state_c_reserved_harbor;
                        break
                    else
                        if i_component(i,j).state ~= s_component_unreserved ...
                                || i_component(i,j).crane_component ~= 1
                            % TODO s_component_unreserved_park
                            continue
                        else
                            % component that could be scheduled. check if
                            % component is unreserved and crane is
                            % available in harbor
                            assert(i_crane(c).state == state_c_available_harbor);
                            assert(i_component(i,j).state ...
                                 == s_component_unreserved);
                            assert(i_component(i,j).crane_component == 1);
                            fprintf('\n+++> Crane %3i tries to ...
                               schedule unreserved component %3i in ...
                               turbine %3i\n',c,j,i);
```

```
            if i_component(i,j).state == s_component_unreserved
                                % crane reserves component
                                      fprintf('\n+++> Crane %3i reserves failure...
                                      of component %3i in turbine %3i ...
                                      (in harbor)\n',c,j,i);
                                assert(i_component(i,j).state...
                                      == s_component_unreserved);
                                % update crane info
                                i_crane(c).state = state_c_reserved_harbor;
                                i_crane(c).next_turbine ...
                                      = i_component(i,j).turbine;
                                i_crane(c).next_component ...
                                      = i_component(i,j).name_component;
                                % i_crane(c).time_to_repair ...
                                      = i_component(i,j).repair_time;
                                i_crane(c).transit_time ...
                                      = i_crane(c).pot_transit_time;
                                i_crane(c).estimated_time_free ...
                                      = i_component(i,j).repair_time ...
                                      + i_crane(c).pot_transit_time;
                                % i_crane(c).min_weather_window_needed ...
                                      = i_component(i,j).repair_time + ...
                                   % i_crane(c).pot_transit_time;
                                % update turbine info
                                i_turb(i).n_unreserved_failures ...
                                      = i_turb(i).n_unreserved_failures - 1;
                                % update component info
                                i_component(i,j).state = ...
                                      s_component_reserved;
                           else
                                errordlg('\n+++> Crane %3i does NOT ...
                                   reserve failure of component %3i in ...
                                   turbine %3i (in harbor -- ERROR)\n',c,j,i);
                           end
                      end
                 end
            end
       end
  end
end
```

(l) Source Code: *"d_c_busy_park_01"*

```
function d_c_busy_park_01(c,t,hs)

global i_crane
global max_time_offshore
global lookahead_time
global state_c_busy_park
global state_c_reserved
global state_c_in_repair
global state_c_transit_back

% crane either finished a repair in park, or just arrived in park
if i_crane(c).state == state_c_busy_park || i_crane(c).state == state_c_reserved
    i_crane(c).time_offshore = i_crane(c).time_offshore + 1;
    if i_crane(c).time_offshore >= max_time_offshore
        % crane offshore too long
         fprintf('\nWWW> Crane %3i goes back because it was offshore too ...
          long (WARNING)<WWW\n',c);
         i_crane(c).state = state_c_transit_back;
         d_transit_back_crane(c);
    else
        % disp(i_crane(c)) % --> crane data
        d_scheduling_crane_offshore_01(c);
        if i_crane(c).state == state_c_reserved
            % weather check
            if i_crane(c).estimated_time_free <= lookahead_time
                baz = i_crane(c).estimated_time_free;
            else
                baz = lookahead_time;
            end
            a = hs(t:(t+baz));
            foo = find(i_crane(c).max_hs <= a,1);
            % TODO: hs matrix + other weather constraints? wind speed matrix?
            if ~isempty(foo)
                 fprintf('\nWWW> Crane %3i goes back because of bad ...
                  weather (WARNING)<WWW\n',c);
                 disp(i_crane)
                 i_crane(c).state = state_c_transit_back;
                 d_transit_back_crane(c);
                 % disp(i_crane(c)) % --> crane data
            else
                 fprintf('\n+++> Crane %3i arrived at turbine for repair\n',c);
                i_crane(c).at_which_turbine = i_crane(c).next_turbine;
                i_crane(c).next_turbine = 0;
                i_crane(c).at_which_component = i_crane(c).next_component;
                i_crane(c).next_component = 0;
                i_crane(c).state = state_c_in_repair;
            end
        else
            fprintf('\n+++> Crane %3i goes back because no tasks!\n',c);
            i_crane(c).state = state_c_transit_back;
            d_transit_back_crane(c);
        end
    end
end
if i_crane(c).state == state_c_transit_back
    d_transit_back_crane(c);
end
```

(m) Source Code: *"d_c_reserved_harbor"*

```
function d_c_reserved_harbor(c,t,hs)

global state_c_reserved_harbor
global i_crane
global lookahead_time
global state_c_transit_to

% if crane is already in transit
if i_crane(c).state == state_c_transit_to
    d_transit_to_crane(c);
end

% make sure that crane is reserved harbor, because all cranes are
% passing this funtion!
if i_crane(c).state == state_c_reserved_harbor
     fprintf('\n+++> Reserved crane %3i checks the weather\n',c);
    % check if estimated time free or lookahead time are the driver. the
    % smaller value is allways the driver. if the forecast (lookahead time)
    % is short, but OK, the crane shall go!
    if i_crane(c).estimated_time_free <= lookahead_time
        baz = i_crane(c).estimated_time_free;
    else
        baz = lookahead_time;
    end
    % find values of wave height in "hs" matrix
    a = hs(t:(t+baz));
    % check if there is a wave height value in the considered time period
    % which is higher than the wave height boundary "max_hs"
    foo = find(i_crane(c).max_hs <= a,1);
    % TODO: hs matrix + other weather constraints? wind speed matrix
    if isempty(foo)
        % weather OK --> crane goes to transit
        assert(i_crane(c).state == state_c_reserved_harbor);
        fprintf('\n+++> Weather OK for reserved crane %3i (goes ...
         into transit)\n',c);
        i_crane(c).state = state_c_transit_to;
        d_transit_to_crane(c);
    else
         fprintf('\n+++> Weather too bad (max_hs = %3.2d > s_max_hs = %3.2d) for
reserved crane %3i (stays reserved in harbor)\n',...
          max(foo),i_crane(c).max_hs,c);
    end
end
```

(n) Source Code: *"d_collect"*

```
function d_collect(k)

global i_ship
global i_turb
global i_crew
global i_component
global s_component_unreserved


something_done = true;
while something_done
    something_done = false;
    for i = [i_ship(k).crew_ids]
        if i ~= 0
            % crew reserved for component
            if i_crew(1,i).next_turbine ~= 0
                fprintf('\n---> ship %3i had reserved crew and unreserves ...
failure before transit back\n',k);
                i_component(i_crew(1,i).next_turbine,i_crew(1,i).next_component).state
=...
                    s_component_unreserved;
                i_turb(i_crew(1,i).next_turbine).n_unreserved_failures = ...
                    i_turb(i_crew(1,i).next_turbine).n_unreserved_failures + 1;
            end

            if i_crew(1,i).at_which_turbine ~= 0 % crew assigned to this ...
                    ship is at a turbine!
                % crew is abrogated! if repair wasn't finished, crew to
                % continue will be defined in scheduling!
                if i_component(i_crew(1,i).at_which_turbine,...
                           i_crew(1,i).at_which_component).repair_time ~= 0
                    fprintf('\n---> ship %3i was at repair and unreserves ...
                      failure before transit back\n',k);
                    i_turb(1,i_crew(1,i).at_which_turbine).repair_time = ...
        i_turb(1,i_crew(1,i).at_which_turbine).repair_time...
                        + i_crew(1,i).t_repair_done;
                    i_turb(1,i_crew(1,i).at_which_turbine) ...
               .n_unreserved_failures = ...
                    i_turb(1,i_crew(1,i).at_which_turbine)...
                     .n_unreserved_failures + 1;
                    i_component(i_crew(1,i).at_which_turbine,...
                      i_crew(1,i).at_which_component)...
                      .repair_time = i_crew(1,i).t_repair_done;
                    i_component(i_crew(1,i).at_which_turbine,...
                      i_crew(1,i).at_which_component).state = ...
                    s_component_unreserved;
                end
```

```
                    i_component(i_crew(1,i).at_which_turbine,...
                    i_crew(1,i).at_which_component).crew_id = 0;
              i_ship(k).n_people = i_ship(k).n_people + ...
        i_crew(1,i).n_people_in_crew;
                    i_ship(k).n_crews = i_ship(k).n_crews - 1;
                    i_turb(1,i_crew(1,i).at_which_turbine)....
                    .n_people_at_turbine = ...
                    i_turb(1,i_crew(1,i).at_which_turbine).n_people_at_turbine ...
                    - i_crew(1,i).n_people_in_crew;
                    i_crew(1,i).state = 0;
                    i_crew(1,i).t_repair_done = 0;
                    i_crew(1,i).n_people_in_crew = 0;
                    i_crew(1,i).at_which_turbine = 0;
                    i_crew(1,i).at_which_component = 0;
                    i_crew(1,i).next_turbine = 0;
                    i_crew(1,i).next_component = 0;
                    i_crew(1,i).crew_id = 0;
                    something_done = true;
              end
          end
      end
end
```

(o) Source Code: *"d_cr_crew_res_failure_harbor"*

```
function d_cr_crew_res_failure_harbor(i,j,k)

global i_crew
global s_reserved
global i_component
global i_ship
global i_turb
global state_s_reserved_harbor
global s_component_reserved
global assumed_people_required
global state_s_busy_park
global s_component_unreserved


if i_component(i,j).state == s_component_unreserved && i_component(i,j).crew_component ==
1
    % new_crew_id = 1;
    assert(i_component(i,j).crew_component == 1);
    for c = 1:length(i_crew)
        if i_crew(c).crew_id == 0
            new_crew_id = c;
            break
        end
    end
    if c == length(i_crew)
        new_crew_id = c + 1;
    end
    assert(i_component(i,j).state == s_component_unreserved)
    fprintf('\n-->Ship %3i reserves failure of component %3i in ...
    turbine %3i\n',k,j,i);
    i_crew(new_crew_id).crew_id = new_crew_id;
    i_crew(new_crew_id).state = s_reserved;
    i_crew(new_crew_id).n_people_in_crew = assumed_people_required(j);
    i_crew(new_crew_id).t_repair_done = i_component(i,j).repair_time;
    i_crew(new_crew_id).next_turbine = i_component(i,j).turbine;
    i_crew(new_crew_id).next_component = i_component(i,j).name_component;
    i_ship(k).state = state_s_busy_park;
    i_ship(k).n_crews = i_ship(k).n_crews + 1;
    i_ship(k).crew_ids(i_ship(k).n_crews) = i_crew(new_crew_id).crew_id;
    i_ship(k).n_crews_assigned = i_ship(k).n_crews_assigned + 1;
    if i_ship(k).estimated_time_free < (i_crew(new_crew_id).t_repair_done...
            + i_ship(k).pot_transit_time)
        i_ship(k).estimated_time_free = (i_crew(new_crew_id).t_repair_done...
            + i_ship(k).pot_transit_time);
    end
    i_ship(k).estimated_weather_window_needed = i_ship(k).estimated_time_free;
    i_ship(k).state = state_s_reserved_harbor;
    i_ship(k).n_people = i_ship(k).n_people + ...
    i_crew(new_crew_id).n_people_in_crew;
    i_ship(k).n_people_loaded = i_ship(k).n_people_loaded + ...
    i_crew(new_crew_id).n_people_in_crew;
    i_turb(i).n_unreserved_failures = i_turb(i).n_unreserved_failures - 1;
    i_component(i,j).state = s_component_reserved;
end
end
```

(p) Source Code: *"d_cr_crew_res_failure_offshore"*

```
function d_cr_crew_res_failure_offshore(i,j,k)

global i_crew
global s_reserved
global i_component
global i_ship
global i_turb
global s_component_reserved
global state_s_busy_park
global assumed_people_required
global s_component_unreserved

if i_component(i,j).state == s_component_unreserved % TODO: s_unreserved_park
    % new_crew_id = 1;
    assert(i_component(i,j).crew_component == 1);
    for c = 1:length(i_crew)
        if i_crew(c).crew_id == 0
            new_crew_id = c;
            break
        end
    end
    if c == length(i_crew)
        new_crew_id = c + 1;
    end
    fprintf('\n---> Ship %3i reserves failure of component %3i in ...
    turbine %3i (in park)\n',k,j,i);
    assert(i_component(i,j).state == s_component_unreserved)
    i_crew(new_crew_id).crew_id = new_crew_id;
    i_crew(new_crew_id).state = s_reserved;
    i_crew(new_crew_id).n_people_in_crew = assumed_people_required(j);
    i_crew(new_crew_id).t_repair_done = i_component(i,j).repair_time;
    i_crew(new_crew_id).next_turbine = i_component(i,j).turbine;
    i_crew(new_crew_id).next_component = i_component(i,j).name_component;
    % at_which_turbine / component = 0??
    i_ship(k).state = state_s_busy_park;
    i_ship(k).n_crews = i_ship(k).n_crews + 1;
    % i_ship(k).crew_ids(i_ship.n_crews + 1) = i_crew(new_crew_id).crew_id;
    i_ship(k).crew_ids(i_ship(k).n_crews) = i_crew(new_crew_id).crew_id;
    i_ship(k).n_crews_assigned = i_ship(k).n_crews_assigned + 1;
    i_ship(k).n_people_free = i_ship(k).n_people_free - ...
    i_crew(new_crew_id).n_people_in_crew;
    if i_ship(k).estimated_time_free < i_crew(new_crew_id).t_repair_done
        i_ship(k).estimated_time_free = i_crew(new_crew_id).t_repair_done;
    end
    i_ship(k).estimated_weather_window_needed = i_ship(k).estimated_time_free;
    i_turb(i).n_unreserved_failures = i_turb(i).n_unreserved_failures - 1;
    i_component(i,j).state = s_component_reserved;
else
     fprintf('\n---> Ship %3i does NOT reserve failure of component %3i ...
     in turbine %3i (in park -- ERROR)\n',k,j,i);
end
end
```

(q) Source Code: *"d_crane_res_failure_offshore"*

```
function d_crane_res_failure_offshore(i,j,c)

global i_component
global i_crane
global i_turb
global s_component_reserved
global state_c_reserved
global s_component_unreserved

if i_component(i,j).state == s_component_unreserved && i_component(i,j).crane_component ==
1 % TODO: s_unreserved_park
    fprintf('\n+++> Crane %3i reserves failure of component %3i in turbine ...
    %3i (in park)\n',c,j,i);
    assert(i_component(i,j).state == s_component_unreserved);
    i_crane(c).state = state_c_reserved;
    i_crane(c).next_turbine = i_component(i,j).turbine;
    i_crane(c).next_component = i_component(i,j).name_component;
    % i_crane(c).time_to_repair = i_component(i,j).repair_time;
    i_crane(c).estimated_time_free = i_component(i,j).repair_time;
    i_turb(i).n_unreserved_failures = i_turb(i).n_unreserved_failures - 1;
    i_component(i,j).state = s_component_reserved;
else
     errordlg('\n+++>Crane %3i does NOT reserve failure of component %3i ...
     in turbine %3i (in park -- ERROR)\n',c,j,i);
end
end
```

(r) Source Code: *"d_drop_collect"*

```
function d_drop_collect(k)

% TODO: flag for crew left boat

global i_ship
global s_waiting s_in_repair s_reserved
global i_crew
global i_component
global i_turb

something_done = true;
while something_done
  something_done = false;
  % collect
  for i = [i_ship(k).crew_ids]
      if i ~= 0
          if i_crew(1,i).state == s_waiting % crew assigned in this ship before
              i_ship(k).n_people = i_ship(k).n_people + ...
               i_crew(1,i).n_people_in_crew;
              i_ship(k).n_people_free = i_ship(k).n_people_free + ...
               i_crew(1,i).n_people_in_crew;
              i_crew(i).crew_id = 0;
              i_crew(i).state = 0;
              i_crew(i).at_which_turbine = 0;
              i_crew(i).at_which_component = 0;
              something_done = true;
          end
      end
  end
  % drop
  for i = [i_ship(k).crew_ids]
      if i ~= 0
          if i_crew(1,i).state == s_reserved % crew assigned to this ...
           ship is reserved
              i_crew(1,i).at_which_turbine = i_crew(1,i).next_turbine; ...
               % = turbine that booked this crew
              i_crew(1,i).at_which_component = i_crew(1,i).next_component; ...
               % = component that booked this crew
              i_crew(1,i).state = s_in_repair;
              i_crew(1,i).next_turbine = 0;
              i_crew(1,i).next_component = 0;
              i_crew(1,i).t_repair_done =
        i_component(i_crew(1,i).at_which_turbine,...
               i_crew(1,i).at_which_component).repair_time;
              i_component(i_crew(1,i).at_which_turbine, ...
               i_crew(1,i).at_which_component).crew_id = ...
               i_crew(1,i).crew_id;
              i_component(i_crew(1,i).at_which_turbine, ...
               i_crew(1,i).at_which_component).people_required = ...
              i_component(i_crew(1,i).at_which_turbine, ...
        i_crew(1,i).at_which_component).people_required ...
              - i_crew(1,i).n_people_in_crew;
              i_component(i_crew(1,i).at_which_turbine, ...
        i_crew(1,i).at_which_component).t_unreserved_in_park = 0;
              i_ship(k).n_people = i_ship(k).n_people - ...
               i_crew(1,i).n_people_in_crew;
              i_turb(1,i_crew(1,i).at_which_turbine).n_people_at_turbine = ...
                  i_turb(1,i_crew(1,i).at_which_turbine).n_people_at_turbine ...
                    + i_crew(1,i).n_people_in_crew;
                    something_done = true;
          end
       end
    end
end
```

(s) Source Code: *"d_new_failure_01"*

```
function d_new_failure_01(i,j)

global i_component
global assumed_failure_rate
global s_component_unreserved
global assumed_repair_times
global i_turb


if i_component(i,j).ttf == 999999 % initial value --> generating first ttf
    i_component(i,j).ttf = round(exprnd(1/(assumed_failure_rate(j)/(365*4))));
end

if i_component(i,j).ttf > 0
    i_component(i,j).ttf = i_component(i,j).ttf  - 1;
end

if i_component(i,j).ttf == 0 && i_component(i,j).repair_time == 0
    i_component(i,j).state = s_component_unreserved;
    fprintf('\n*** Component %3i at turbine %3i has new unreserved ...
    failure ***\n',j,i);
    i_component(i,j).repair_time = assumed_repair_times(j);
    i_turb(i).n_unreserved_failures = i_turb(i).n_unreserved_failures + 1;
    i_turb(i).repair_time = i_turb(i).repair_time + assumed_repair_times(j);
    % TODO: failure unreserved park
end
```

(t) Source Code: *"d_repair_crane"*

```
function d_repair_crane(q,t,hs)

global i_crane
global s_component_running
global i_component
global i_turb
global assumed_failure_rate
global state_c_busy_park % necessary if crane can handle other components ...
after repair - see below
global state_c_in_repair
global assumed_repair_times
global s_component_unreserved
global state_c_transit_back
global max_time_offshore


if i_crane(q).state == state_c_in_repair && i_crane(q).at_which_turbine ~= 0 % crane at
turbine!
    % weather check
    if i_crane(q).time_offshore >= max_time_offshore
        % crane offshore too long
        fprintf('\nWWW> Crane %3i goes back because it was offshore too ...
         long (WARNING)<WWW\n',c);
        i_crane(q).state = state_c_transit_back;
        d_transit_back_crane(q);
    else
        a = find(hs(t) > i_crane(q).max_hs,1);
        if  isempty(a)
            % weather OK
            i_component(i_crane(q).at_which_turbine, ...
             i_crane(q).at_which_component).repair_time = ...
            i_component(i_crane(q).at_which_turbine, ...
      i_crane(q).at_which_component).repair_time - 1;
            i_turb(i_crane(q).at_which_turbine).repair_time = ...
i_turb(i_crane(q).at_which_turbine).repair_time - 1;
            i_crane(q).time_to_repair = i_crane(q).time_to_repair - 1;
            % i_crane(q).min_weather_window_needed = ...
             i_crane(q).min_weather_window_needed - 1;
            i_crane(q).estimated_time_free = i_crane(q).estimated_time_free - 1;
            i_crane(q).state = state_c_in_repair;
            fprintf('\n+++>CRANE Component %3i at turbine %3i is in repair! ...
             time left %3i\n',i_crane(q).at_which_component, ...
             i_crane(q).at_which_turbine...
            ,i_component(i_crane(q).at_which_turbine, ...
             i_crane(q).at_which_component).repair_time)
```

```
if i_component(i_crane(q).at_which_turbine,...
       i_crane(q).at_which_component).repair_time == 0
                    fprintf('\n+++> Component %3i at turbine %3i has been ...
                    repaired!\n',i_crane(q).at_which_component,...
                    i_crane(q).at_which_turbine);
              i_component(i_crane(q).at_which_turbine, ...
                    i_crane(q).at_which_component).ttf = ...
                 round(exprnd(1/(assumed_failure_rate ...
              (i_crane(q).at_which_component)/(365*4))));
                    i_component(i_crane(q).at_which_turbine, ...
              i_crane(q).at_which_component).state = ...
                    s_component_running;
                    i_component(i_crane(q).at_which_turbine, ...
                    i_crane(q).at_which_component).t_unreserved_in_park = 0;
                    % crane can handle other tasks
                    % i_crane(q).state = state_c_busy_park;
                    % crane has to go back after repair
                    fprintf('\n+++> Crane %3i goes back because repair ended\n',q);
                    i_crane(q).state = state_c_transit_back;
                    d_transit_back_crane(q);
              end
           else
              % weather too bad
              fprintf('\nWWW> Crane %3i goes back because of bad weather ...
               during repair(WARNING)<WWW\n',q);
              i_crane(q).state = state_c_transit_back;
              d_transit_back_crane(q);
              % update component info
              if i_component(i_crane(q).at_which_turbine, ...
               i_crane(q).at_which_component).repair_time ~= 0
                 i_turb(1,i_crane(q).at_which_turbine).repair_time =
            i_turb(1,i_crane(q).at_which_turbine).repair_time...
                    + assumed_repair_times(i_crane(q).at_which_component);
                 i_turb(1,i_crane(q).at_which_turbine).n_unreserved_failures = ...
                 i_turb(1,i_crane(q).at_which_turbine).n_unreserved_failures + 1;
                 i_component(i_crane(q).at_which_turbine, ...
                  i_crane(q).at_which_component).repair_time = ...
                    assumed_repair_times(i_crane(q).at_which_component);
                 i_component(i_crane(q).at_which_turbine, ...
                  i_crane(q).at_which_component).state = ...
                    s_component_unreserved;
                 i_crane(q).state = state_c_transit_back;
              end
           end
       end
end
```

(u) Source Code: *"d_repair_crew"*

```
function d_repair_crew(p)

global i_crew
global s_in_repair
global s_component_running
global s_waiting
global i_component
global i_turb
global assumed_failure_rate


if i_crew(p).state == s_in_repair
       % decreasing turbine repair time
       i_turb(i_crew(p).at_which_turbine).repair_time =
       i_turb(i_crew(p).at_which_turbine).repair_time - 1;
       % decreasing component repair time
       i_component(i_crew(p).at_which_turbine, ...
       i_crew(p).at_which_component).repair_time = ...
      i_component(i_crew(p).at_which_turbine, ...
       i_crew(p).at_which_component).repair_time - 1;
       i_crew(p).t_repair_done = i_crew(p).t_repair_done - 1;
       if i_component(i_crew(p).at_which_turbine, ...
              i_crew(p).at_which_component).repair_time == 0
              fprintf('\n---> Component %3i at turbine %3i has been ...
       repaired!\n',i_crew(p).at_which_component, ...
              i_crew(p).at_which_turbine);
              i_component(i_crew(p).at_which_turbine, ...
              i_crew(p).at_which_component).ttf = ...
            round(exprnd(1/(assumed_failure_rate ...
       (i_crew(p).at_which_component)/(365*4))));
              i_component(i_crew(p).at_which_turbine, ...
              i_crew(p).at_which_component).state = s_component_running;
              i_component(i_crew(p).at_which_turbine, ...
              i_crew(p).at_which_component).people_required = 0;
              i_component(i_crew(p).at_which_turbine, ...
              i_crew(p).at_which_component).crew_id = 0;
              i_component(i_crew(p).at_which_turbine,...
              i_crew(p).at_which_component).t_unreserved_in_park = 0;
              i_crew(p).state = s_waiting;
       end
end
```

(v) Source Code: *"d_s_available_harbor"*

```
function d_s_available_harbor(k,t,hs)

global i_ship
global state_s_available_harbor
global state_s_reserved_harbor
global lookahead_time

if i_ship(k).state == state_s_available_harbor
    d_scheduling_ship_harbor_01(k);
    if i_ship(k).n_people_free == i_ship(k).n_people_loaded
        %fprintf('===> Ship %3i resets to available\n',k);
        i_ship(k).state = state_s_available_harbor;
    else
         fprintf('\n---> Ship %3i checks the weather\n',k);
        if i_ship(k).estimated_weather_window_needed <= lookahead_time
            baz = i_ship(k).estimated_weather_window_needed;
        end
        if i_ship(k).estimated_weather_window_needed > lookahead_time
            baz = lookahead_time;
        end
        a = hs(t:(t+baz));
        foo = find(i_ship(k).max_hs <= a,1);
        % TODO: hs matrix + other weather constraints? wind speed matrix?
        if isempty(foo)
            if i_ship(k).state == state_s_reserved_harbor
                    fprintf('\n---> Weather OK for ship %3i (goes ...
                    into transit)\n',k);
                    d_transit_to(k);
            else
                 fprintf('\n---> Ship %3i not reserved? (WARNING)\n',k);
            end
        else
            fprintf('\n---> Weather too bad for ship %3i (stays ...
             in harbor)\n',k);
        end
    end
end
```

(w) Source Code: *"d_s_busy_park"*

```
function d_s_busy_park(k,t,hs)

global i_ship
global max_time_offshore
global lookahead_time
global state_s_busy_park


if i_ship(k).state == state_s_busy_park
    i_ship(k).time_offshore = i_ship(k).time_offshore + 1;
    if i_ship(k).time_offshore >= max_time_offshore
        % ship offshore too long
        fprintf('\nWWW> Ship %3i goes back (offshore too long)<WWW\n',k);
        d_collect(k);
        d_transit_back(k);
    else
        % disp(i_ship(k)) % --> ship data
        d_scheduling_ship_offshore_01(k);
        if i_ship(k).n_people_free == i_ship(k).n_people_loaded
            fprintf('\n---> Ship %3i goes back (all jobs done)\n',k);
            d_collect(k);
            d_transit_back(k);
        else
            fprintf('\n---> Ship %3i is busy in park\n',k);
            a = find(hs(t) > i_ship(k).max_hs,1);
            if  ~isempty(a) %~isempty(foo)
                    fprintf('\nWWW> Ship %3i goes back because of bad ...
                    weather (WARNING)<WWW\n',k);
                    d_collect(k);
                    d_transit_back(k);
                    % disp(i_ship(k)) % --> ship data
            else
                    d_drop_collect(k);
                    d_scheduling_ship_offshore_01(k);
            end
        end
    end
end
```

(x) Source Code: *"d_s_reserved_harbor"*

```
function d_s_reserved_harbor(k,t,hs)

global i_ship
global state_s_reserved_harbor
global state_s_available_harbor
global lookahead_time
global state_s_transit_to

if i_ship(k).state == state_s_reserved_harbor
    fprintf('\n---> Ship %3i is reserved and looking for more jobs\n',k);
    d_scheduling_ship_harbor_01(k);
    if i_ship(k).n_people_free == i_ship(k).n_people_loaded
       fprintf('\nWWW> Reserved ship %3i resets to available ...
       (WARNING)<WWW\n',k);
       i_ship(k).state = state_s_available_harbor;
    else
         fprintf('\n---> Reserved ship %3i checks the weather\n',k);
         if i_ship(k).estimated_time_free <= lookahead_time
            baz = i_ship(k).estimated_time_free;
        end
        if i_ship(k).estimated_time_free > lookahead_time
            baz = lookahead_time;
        end
        a = hs(t:(t+baz));
        foo = find(i_ship(k).max_hs <= a,1);
        if isempty(foo)
            if i_ship(k).state == state_s_reserved_harbor
                fprintf('\n---> Weather OK for reserved ship %3i (goes ...
                 into transit)\n',k);
                d_transit_to(k);
                disp(i_ship(k))
            else
               fprintf('\nWWW> Reserved ship %3i not reserved? ...
                (WARNING)<WWW\n',k);
            end
        else
             fprintf('\n---> Weather too bad (max_hs = %3.2d > ...
              s_max_hs = %3.2d) for reserved ship %3i (stays reserved in
          harbor)\n',max(foo),i_ship(k).max_hs,k);
         end
    end
end


% if ship is already in transit
if i_ship(k).state == state_s_transit_to
    d_transit_to(k);
end
```

(y) Source Code: *"d_scheduling_crane_offshore_01"*

```
function d_scheduling_crane_offshore_01(c)

global n_turb
global n_components
global i_component
global i_turb
global i_crane
global state_c_busy_park
global factor_time_for_failure_crane
global s_component_unreserved
global state_c_reserved
global state_c_in_repair

if i_crane(c).state ~= state_c_in_repair
    if i_crane(c).next_turbine ~= 0
        i_crane(c).state = state_c_reserved;
    else
        for i = 1:n_turb
            if i_crane(c).state == state_c_reserved || ...
                    i_crane(c).state == state_c_in_repair;
                break
            else
                if i_turb(i).n_unreserved_failures == 0 ...
                    % TODO what happens if unreserved park?!?
                    continue
                else
                    for j = 1:n_components
                        if i_crane(c).state == state_c_reserved || ...
                                i_crane(c).state == state_c_in_repair;
                            break
                        else
                            if i_component(i,j).state ~= s_component_unreserved
                                continue
                            else
                                % component that could be scheduled
                                if i_component(i,j).crane_component == 0
                                    continue
                                end
                                if i_crane(c).state == state_c_busy_park
                                    if i_crane(c).estimated_time_free <= ...
                                              factor_time_for_failure_crane
                                              fprintf('\n+++> Crane %3i tries to ...
                                              schedule unreserved component ...
                                              %3i in turbine %3i\n',c,j,i);
                                              d_crane_res_failure_offshore(i,j,c);
                                    end
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
```

(z) Source Code: *"d_scheduling_ship_harbor_01"*

```
function d_scheduling_ship_harbor_01(k)
global i_ship
global i_turb
global i_component
global state_s_reserved_harbor
global state_s_available_harbor
global n_turb
global n_components
global s_component_unreserved
 something_done = true;
while something_done
    something_done = false;
    if i_ship(k).state == state_s_reserved_harbor
        if i_ship(k).n_people <= i_ship(k).max_people
            for i = 1:n_turb
                if i_turb(i).n_unreserved_failures == 0
                    continue
                else
                    for j = 1:n_components
                        if i_component(i,j).state ~= s_component_unreserved ||...
                                i_component(i,j).crew_component ~= 1
                            continue
                        else
                            if i_component(i,j).crew_component == 1 && ...
                                i_ship(k).n_people_loaded + ...
                                i_component(i,j).people_required <= ...
                                    i_ship(k).max_people
                                % engough space on ship
                                d_cr_crew_res_failure_harbor(i,j,k);
                                something_done = true;
                            end
                        end
                    end
                end
            end
        end
    end
 if i_ship(k).state == state_s_available_harbor
        if i_ship(k).n_people <= i_ship(k).max_people
            for i = 1:n_turb
                if i_turb(i).n_unreserved_failures == 0
                    continue
                else
                    for j = 1:n_components
                        if i_component(i,j).state ~= s_component_unreserved
                            continue
                        else
                            if i_ship(k).n_people <= i_ship(k).max_people
                                if i_component(i,j).crew_component == 1 && ...
                                        i_ship(k).n_people_loaded + ...
                                        i_component(i,j).people_required <= ...
                                             i_ship(k).max_people
                                    d_cr_crew_res_failure_harbor(i,j,k)
                                    i_ship(k).state = state_s_reserved_harbor;
                                    something_done = true;
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
```

(aa) Source Code: *"d_scheduling_ship_offshore_01"*

```
function d_scheduling_ship_offshore_01(k)


global n_turb
global n_components
global i_component
global i_ship
global i_turb
global i_crew
global state_s_busy_park
global factor_time_for_failure_ship
global s_component_unreserved

for a = [i_ship(k).crew_ids]
    if a ~= 0
        foo(a) = i_crew(a).t_repair_done;
    end
end
i_ship(k).estimated_time_free = max(foo);

something_done = true;
%fprintf('Enter loop\n');
while something_done
    something_done = false;
    if i_ship(k).n_people_free > 0;
        for i = 1:n_turb
            if i_ship(k).n_people_free > 0;
                if i_turb(i).n_unreserved_failures == 0
                continue
                else
                    for j = 1:n_components
                        if i_ship(k).n_people_free > 0;
                            if i_component(i,j).state ~= ...
                                 s_component_unreserved || ...
                                 i_component(i,j).crew_component ~= 1
                                   continue
                            else
                                % component that could be scheduled
                                 fprintf('\n---> Ship %3i tries to schedule ...
                                    unreserved component %3i in turbine ...
                                    %3i\n',k,j,i);
                                if i_component(i,j).crew_component == 0
                                    continue
                                end
                                if i_ship(k).state == state_s_busy_park
                                    if i_ship(k).n_people_free >= ...
                                              i_component(i,j).people_required
                                              fprintf('\n---> Ship %3i schedules ...
                                              failure of component %3i in ...
                                              turbine %i\n',k,j,i);
                                       d_cr_crew_res_failure_offshore(i,j,k);
                                              something_done = true;
                                              continue
                                    end
                                    if i_ship(k).estimated_time_free <= ...
                                              factor_time_for_failure_ship
```

```
                                                fprintf('\nWWW> Ship %3i schedules ...
                                                failure of component %3i in turbine ...
                                                %i (WARNING)<WWW\n',k,j,i);
                                                d_cr_crew_res_failure_offshore(i,j,k);
                                                something_done = true;
                                            end
                                        end
                                    end
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
%fprintf('Leave loop\n');
```

(bb) Source Code: *"d_transit_back"*

```
function d_transit_back(k)

global i_ship
global state_s_transit_back
global state_s_available_harbor
global LOG_s_transit_back

i_ship(k).state = state_s_transit_back;
i_ship(k).transit_time = i_ship(k).pot_transit_time;
i_ship(k).transit_time = i_ship(k).transit_time - 1;
fprintf('\n---> Ship %3i is in transit back\n',k);
LOG_s_transit_back(k) = LOG_s_transit_back(k) + 1;


if i_ship(k).transit_time == 0
    i_ship(k).state = state_s_available_harbor;
    i_ship(k).n_people = 0;
    i_ship(k).n_people_free = 0;
    i_ship(k).n_people_loaded = 0;
    i_ship(k).n_crews = 0;
    i_ship(k).crew_ids = 0;
    i_ship(k).n_crews_assigned = 0;
    i_ship(k).transit_time = i_ship(k).pot_transit_time;
    i_ship(k).time_offshore = 0;
    i_ship(k).estimated_time_free = 0;
    i_ship(k).estimated_weather_window_needed = 0;
    fprintf('\n---> Ship %3i is available in harbor after transit back\n',k);
end
```

(cc) Source Code: *"d_transit_back_crane"*

```
function d_transit_back_crane(c)

global i_crane
global state_c_transit_back
global state_c_available_harbor
global i_component
global s_component_unreserved
global i_turb
global LOG_c_transit_back

% crane in transit back
if i_crane(c).state == state_c_transit_back
    i_crane(c).transit_time = i_crane(c).transit_time - 1;
    LOG_c_transit_back(c) = LOG_c_transit_back(c) + 1;
    fprintf('\n+++> Crane %3i is in transit back\n',c);
    % if repair was not finished, UNreserve failure!
    if i_crane(c).at_which_turbine ~= 0
        if i_component(i_crane(c).at_which_turbine,...
             i_crane(c).at_which_component)....
                    repair_time ~= 0
            i_component(i_crane(c).at_which_turbine,...
             i_crane(c).at_which_component).state...
                = s_component_unreserved;
            i_turb(i_crane(c).at_which_turbine).n_unreserved_failures = ...
                i_turb(i_crane(c).at_which_turbine).n_unreserved_failures + 1;
            fprintf('\n+++> Crane %3i was not finished with repair --> ...
             unreserves failure\n',c);
        end
    end

    % if crane had reserved another component
    if i_crane(c).next_turbine ~= 0
        i_component(i_crane(c).next_turbine,i_crane(c).next_component).state...
            = s_component_unreserved;
        i_turb(i_crane(c).next_turbine).n_unreserved_failures = ...
            i_turb(i_crane(c).next_turbine).n_unreserved_failures + 1;
        fprintf('\n+++> Crane %3i was reserved before going back --> ....
         unreserves failure\n',c);
    end

    % crane info is initialized
    if i_crane(c).transit_time == 0
        i_crane(c).state = state_c_available_harbor;
        i_crane(c).at_which_turbine = 0;
        i_crane(c).at_which_component = 0;
        i_crane(c).next_turbine = 0;
        i_crane(c).next_component = 0;
        i_crane(c).time_to_repair = 0;
        i_crane(c).time_offshore = 0;
        % i_crane(c).min_weather_window_needed = 0;
        i_crane(c).estimated_time_free = 0;
        i_crane(c).transit_time = i_crane(c).pot_transit_time;
        fprintf('\n+++> Crane %3i is available in harbor after ...
         transit back\n',c);
    end
end
```

(dd) Source Code: *"d_transit_to"*

```
function d_transit_to(k)

global i_ship
global state_s_transit_to
global state_s_busy_park
global LOG_s_transit_to

fprintf('\n--->Ship %3i goes into transit \n',k);
i_ship(k).state = state_s_transit_to;
i_ship(k).transit_time = i_ship(k).transit_time - 1;
i_ship(k).estimated_time_free = i_ship(k).estimated_time_free - 1;
i_ship(k).time_offshore = i_ship(k).time_offshore + 1;
LOG_s_transit_to(k) = LOG_s_transit_to(k) + 1;

if i_ship(k).transit_time == 0
    fprintf('\n--->Ship %3i arrives at park\n',k);
    i_ship(k).state = state_s_busy_park;
    i_ship(k).transit_time = i_ship(k).pot_transit_time;
end
```

(ee) Source Code: *"d_transit_to_crane"*

```
function d_transit_to(k)

global i_ship
global state_s_transit_to
global state_s_busy_park
global LOG_s_transit_to

fprintf('\n--->Ship %3i goes into transit \n',k);
i_ship(k).state = state_s_transit_to;
i_ship(k).transit_time = i_ship(k).transit_time - 1;
i_ship(k).estimated_time_free = i_ship(k).estimated_time_free - 1;
i_ship(k).time_offshore = i_ship(k).time_offshore + 1;
LOG_s_transit_to(k) = LOG_s_transit_to(k) + 1;

if i_ship(k).transit_time == 0
    fprintf('\n--->Ship %3i arrives at park\n',k);
    i_ship(k).state = state_s_busy_park;
    i_ship(k).transit_time = i_ship(k).pot_transit_time;
end
```