A Machine Learning Framework to Model Extreme Events for Nonlinear Marine Dynamics

by

Wenzhe Xu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Naval Architecture and Marine Engineering)
in The University of Michigan
2020

Doctoral Committee:

Associate Professor Kevin J. Maki, Chair
Associate Professor Matthew D. Collette
Professor Charles R. Doering
Professor Armin W. Troesch

Wenzhe Xu
wenzhe@umich.edu
ORCID id: 0000-0002-4268-1196

Dedicated to my family

ACKNOWLEDGEMENTS

From the bottom of my heart, I would like to express my gratitude to my advisor Prof. Kevin Maki for his dedicated mentorship, encouragement, and support throughout my Ph.D. research. He is talented, professional, and open-minded in teaching me domain knowledge, presenting me different cultures and histories, and supporting me career explorations. The meetings and conversations with him always inspire me from multiple perspectives to form a comprehensive and objective attitude towards both my academic and life.

I would like to thank my other committee members, Prof. Matthew Collette, Prof. Charles Doering, and Prof. Armin Troesch for their stimulating questions and advice on my research work. I would also like to thank my peers at the Computational Ship Hydrodynamics Lab (CSHL) for the friendships and collaborations.

Last but not least, I would like to say big thank you to my family for their unconditional love and support in my life.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

LIST OF APPENDICES

Appendix

LIST OF ABBREVIATIONS

A-R Acceptance-Rejection

CDF Cumulative Distribution Function

CFD Computational Fluid Dynamics

CLT Central Limit Theorem

CNN Convolutional Neural Network

DLG Design Loads Generator

DRE Design Response Estimator

EVD Extreme Value Distribution

FORM First Order Reliability Method

FSI Fluid Structure Interaction

INID Independent and Not-Identically Distributed

LSTM Long Short-Term Memory

MAE Mean Absolute Error

MCS Monte Carlo Simulation

MLP Multi-Layer Perceptron

MSE Mean Squared Error

ODE Ordinary Differential Equation

PDE Partial Differential Equation

PDF Probability Density Function

RNN Recurrent Neural Network

SI System Identification

TEG Threshold Exceedance Generator

ABSTRACT

Extreme events such as large motions and excess loadings of marine systems can result in damage to the device or loss of life. Since the system is exposed to a random ocean environment, these extreme events need to be understood from a statistical perspective to design a safe system. However, analysis of extreme events is challenging because most marine systems operate in the nonlinear region, especially when extreme events occur, and observation of the extreme events is relatively rare for a proper design. Conducting high-fidelity simulations or experimental tests to observe such events is cost-prohibitive.

In the current research, a novel framework is proposed to randomly generate test environments that lead to a large response of the system. With the generated environment, large responses that would take a very long time to achieve can be observed within a much shorter time window. The time-domain context around the extreme event provides the user with rich insights towards the improvement of the design. The proposed framework consists of two modules, which are named as Threshold Exceedance Generator (TEG) and Design Response Estimator (DRE). The framework is data-driven, and its application requires minimal knowledge about the system from the user. The DRE module can identify a nonlinear marine system based on collected data. The TEG module can generate ocean environments that lead to large system response based on the system identification by the DRE module.

Machine learning methods, especially neural networks, are heavily used in the proposed framework. In the thesis, the extreme generation problem in the marine field is described and addressed from a machine-learning perspective. To validate the framework, marine examples including linear wave propagation, nonlinear wave propagation, nonlinear ship roll, tank sloshing, and a floating object in waves are explored. Examples from such a wide range show that the framework can be used for linear or nonlinear systems and Gaussian or non-Gaussian environments. The cost and the amount of data to apply the method are estimated and measured. The comparison between the results from the framework and Monte Carlo Simulation fully demonstrates the accuracy and feasibility of using the data-driven approach.

CHAPTER I

Introduction

1.1 Motivation

As ocean technology develops and more advanced marine systems are built, human beings nowadays are capable of exploring more watery places of the earth that have never been touched before. Ship traffic could not be busier due to booming global trades. Oil platforms are being built farther away from the coast and deeper at sea. Modern marine units are necessarily designed to survive in harsher ocean environments. Environmental loadings like waves, winds, currents, tides, etc. significantly affect the behavior of marine systems at sea. Extreme behaviors under these environment loadings can result in damage or loss of life. Successful analysis of extreme behavior of marine units in rough seas is crucial to provide information for more reliable and robust designs, and safer operations.

However, such analysis is difficult due to many reasons. First of all, the ocean environment is a random phenomenon in a probability sense. Statistical reports, like ocean climate forecasts or sea states, are usually provided instead of deterministic descriptions. Hence, to fully present the system behavior in the probability space, an ensemble of nearly deterministic experiments or simulations with different random seeds is analyzed often with high cost. Secondly, extreme behaviors at sea, like ship roll ending in capsize, large bending moment leading to material fatigue or structural failure, severe wave loads impacting platform decks, etc, are considerably rare for a good design. Observing extreme events associated with high rareness naturally takes a very long time for conducting experiments or simulations. Nowadays, ships, platforms, and other marine units are typically designed to operate for decades. Moreover, to conclude a statistical summary, enough number of realizations associated with different environment seeds are required. Figure 1.1 shows that the cost of extreme event simulation grows as the length of the exposure window and the number

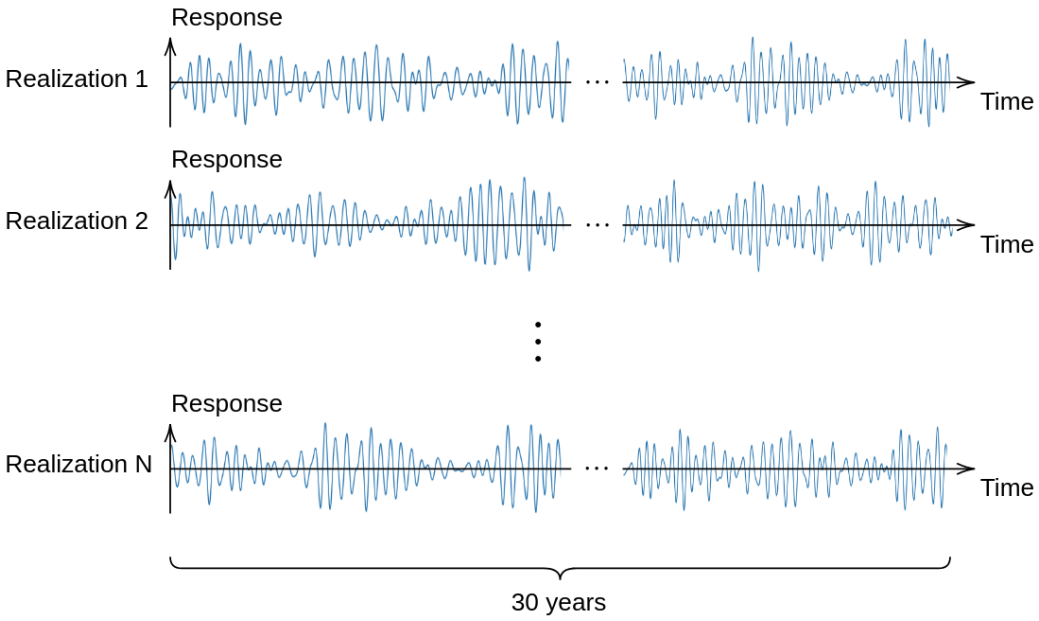

Figure 1.1: Cost scales with window length and probability space resolution

of the realizations increase. The extreme analysis is usually needed for long-term and short-term cases. Depending on the location and season of operations, marine structures may experience a wide range of ocean climate conditions during their lifetime. It is important from the long-term perspective to estimate the survivability of the design in mixed ocean environments during a very large exposure window. It is also important from the short-term perspective to evaluate the performance of the design during a relatively constant sea condition that lasts for a shorter period like hours of Sea State 9 (SS9) storm.

Classical linear theory may not be applied well for extreme scenarios since the superposition assumption used in linear systems is no longer valid. Moreover, ocean environments such as wave, wind, and current can produce many loadings on structures, leading to a complex nonlinear problem. More complex phenomena that do not occur on simple linear systems can show up in nonlinear marine systems, which play an important role in stability analysis, extreme prediction, failure estimation, etc. These phenomena include but are not limited to the dependency of system initial conditions, transient phase, chaos, and they form a barrier to apply many linear methods. Instead of forming closed-form solutions, experiment methods and numerical simulation focus more on a better characterization of the dynamics and achieve reasonable results for real-world engineering usage. Especially for numerical approaches, more computer-based methods have been developed as computational

power has grown exponentially over the past 50 years. Though powerful hardware and efficient algorithms ease the cost of lengthy simulations, characterizing system behavior precisely in the water domain requires complex mathematical models, which inhibit the ability of simulation to study large exposure times. For example, CFD can take hours to simulate seconds of ship motion in waves. Direct application of high-fidelity tools to simulate lifetime realizations is impossible.

A more promising way to combine the advantages from both theoretical and numerical methods is to conduct high-fidelity simulations for short time windows when the studied object exhibits a dangerous response. The remaining questions are how to generate deterministic loading environments that lead the nonlinear system to dangerous response, and how do the generated environments relate to the probability and statistics theory. Therefore, an intelligent method that accounts for the different nature of various nonlinear marine systems is needed.

1.2 Literature Review

In this section, previous works on modeling extreme events are briefly reviewed. Specifically, linear methods, and three methods from the state-of-the-art, which are the Design Loads Generator (DLG) method, the First Order Reliability Method (FORM), and various kinds of extrapolation methods, are discussed. In the end, their advantages and disadvantages are compared and summarized.

1.2.1 Gaussian Process and Linear Methods

The Gaussian process plays an important role in stochastic marine dynamic problems since many natural phenomena can be approximated by a Gaussian process. For example, the elevation of a wind-generated wave can be assumed to be a Gaussian process. The Gaussian assumption for ocean waves is considerably accurate for the deepwater condition according to many experiment observations (*Ochi*, 1990). The Gaussian assumption also eases the discussion of many problems due to its simple mathematical form. For example, a Gaussian process can be uniquely defined by its energy spectrum.

By assuming the elevation of ocean waves is Gaussian, many works have achieved concise and closed-form results in statistics. *Cartwright and Longuet-Higgins* (1956) derived the distribution of positive maxima (or crest height) for a narrowband or non-narrowband Gaussian process. *Lindgren* (1970) and *Tromans et al.* (1991) showed that the most likely wave profile around an a priori maximum crest height is the

autocorrelation curve scaled by the crest height. *Ochi et al.* (1973) gave the averaged number of positive maxima per unit time by considering the distribution of the mean-crossing period.

Another advantage of a Gaussian assumption is that results about extreme behavior are accessible. Closed-form solutions are usually available without running large numbers of simulations, which is often the case for other methods on extreme analysis. The tail behavior of a Gaussian process is fully described. The largest (or smallest) elevation (also known as the extreme values) of a Gaussian process during an exposure time window is a random variable. It is very interesting to ship and marine designers to know the Extreme Value Distribution (EVD) associated with a time window, which can be days during storms in the short-term sense, or decades of a lifetime in the long-term sense. By using Order Statistics Theory, *Ochi et al.* (1973) gave the distribution of the extreme value in terms of the spectrum parameters and the exposure window length. Due to its straightforward meaning and simple representation, the linear method becomes the first step for many reliability analysis and ocean climate forecasting.

Since an ocean wave is often modeled properly by a Gaussian process and the ocean is a primary source of forcing, many studies apply the Gaussian results to the response of marine systems. By assuming the studied system is linear, the system response under a Gaussian loading environment is another Gaussian process. Once the spectrum of the response is calculated, previous findings for the Gaussian process can be directly applied. This linear method is widely used when closed-form solutions are desired.

Though linear methods provide closed-form solutions and bypass expensive simulation, the linear assumption of the system dynamics may not be valid or accurate, especially when the system undergoes extreme response. When the system behaves in its nonlinear regime, the system output may not be Gaussian, and therefore results of the Gaussian process are not applicable.

1.2.2 Design Loads Generator

The Design Loads Generator (DLG) method by *Troesch* (1997), *Alford* (2008), and *Kim* (2012) steps further from the linear methods. The contribution of the DLG method is that it not only estimates the EVD but also provides an ensemble of deterministic environments to simulate extreme scenarios in the time domain. With the DLG, the time window around the moment (or design time) when the system undergoes extreme response is efficiently simulated. The time-domain simulations

add to the statistical summary to allow the user to understand a more complete story of the extreme response.

The method heavily utilizes Fourier analysis while maintaining the properties of the Gaussian process. It considers a Gaussian process $x(t)$ which can represent a linear water wave or linear system response under Gaussian loadings. The Gaussian process can be approximated by a Fourier series

$$x(t) = \sum_{i=1}^{N} a_i \cos(2\pi f_i t + \phi_i) \tag{1.1}$$

where f_i are discretized frequencies, Fourier amplitudes $a_i = \sqrt{2S(f)\Delta f}$ are evaluated based on the energy spectrum of the Gaussian process, and ϕ_i are random variables uniformly distributed from $-\pi$ to π. With the given representation, the only randomness of the Gaussian process comes from the random phase vector $(\phi_1, \cdots, \phi_N)$.

The objective of the DLG method is to randomly generate the phase vector such that the process at the design time (specified by the user) $x(t_d)$ follows the EVD associated to the exposure window length T for the Gaussian process. By simulating the scenario with the generated phase vector, a time-domain realization is produced, with which the user can observe an extreme response at the design time. Multiple simulations can be conducted to collect an ensemble of realizations, which produces a more complete time and probability domain analysis.

The procedure of DLG generating phase vectors is briefly illustrated in Figure 1.2. Since the desired exposure length T may be very long, a much shorter window T' is used to conduct a pre-experiment, which produces a collection of phase vector whose resultant response $x(t_d)$ follows the EVD associated to T'. The collected phases are then used to fit a model with parameter λ_i for each Fourier component. The parameter of each model is then optimized such that the resultant response follows the EVD associated with T as much as possible. Then each model can generate candidate phase angles for their own Fourier component. The generated phase vectors are then filtered by an Acceptance-Rejection (A-R) algorithm such that the resultant response follows the EVD associated with T.

The advantage of the DLG method is that it provides simulation environments in terms of phase vectors and the generated random seed is constrained to follow the EVD theory. *Filip et al.* (2017, 2018) and *Xu et al.* (2019) have applied the DLG method to generate wave environments for CFD cases like wave impacting on fixed deck, an extreme roll of a moving Tumblehome hull in head and following seas, etc. These nonlinear systems are first linearized to be applicable for the DLG method.

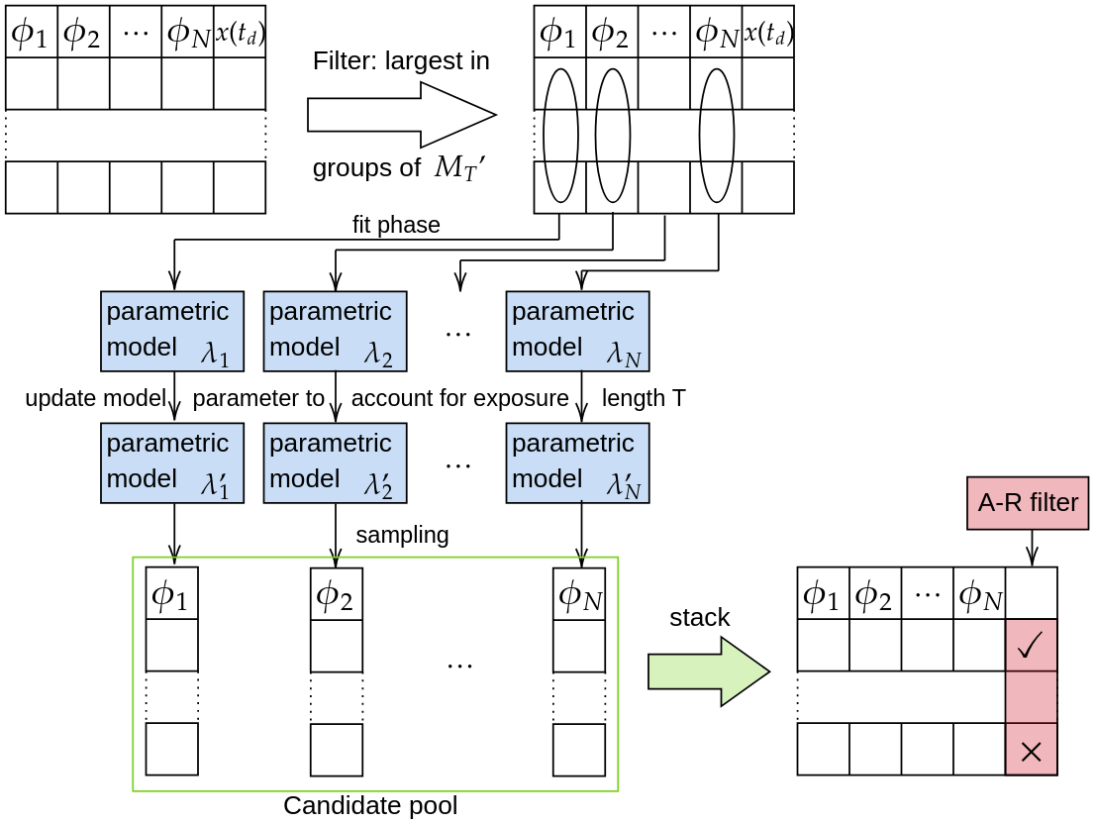

Figure 1.2: DLG generates phase vectors that satisfy the EVD theory

The DLG-generated wave environments are then used in the nonlinear simulations to collect statistical information of the nonlinear system.

The shortcomings of the DLG method come from two aspects. First, the approach is based on linear methods, which means it inherits the Gaussian assumption of the system response. The linear assumption also eases the dependency of the initial condition and transient effect, which occurs for nonlinear systems. It also provides the rule by giving a Gaussian EVD for the A-R algorithm to post-filter the candidate phase vectors. As a comparison, for a nonlinear system whose response is non-Gaussian, the EVD rule is still an open research question. The second aspect is the independence assumption for phases from different Fourier components. In the method, all parametric models are standalone with their own parameters. The process to generate the candidate pool of phase vectors is also performed in parallel for all models. However, the phases at the design time are in fact correlated even for a Gaussian process. Though the correlation among different Fourier components is improved by updating model parameters together and filtering the phases as a vector by the A-R algorithm, the correlation may still be unresolved or inaccurate.

1.2.3 First Order Reliability Method

The First Order Reliability Method (FORM) used in the marine domain by *Jensen* (1996, 2008, 2009) is developed beyond linear systems. For a nonlinear system in a random ocean environment parameterized by a random seed, the FORM method searches for the most probable seed that leads to the failure of the nonlinear system. Specifically, it considers a random wave environment parameterized by

$$\eta(x,t) = \sum_{i=1}^{N} a_i \sigma_i \cos(\omega_i t - k_i x) + b_i \sigma_i \sin(\omega_i t - k_i x) \tag{1.2}$$

where a_i and b_i are uncorrelated standard normal random variables, ω_i and k_i are the discretized angular frequencies and corresponding wave number, and σ is calculated based on the input wave spectrum.

$$\sigma_i = \sqrt{S(\omega_i)\Delta\omega_i} \tag{1.3}$$

The system response under a random seed $(a_1, b_1, \cdots, a_N, b_N)$ is then represented by

$$\phi(t_d; a_1, b_1, \cdots, a_N, b_N) \tag{1.4}$$

where t_d is the design time and $\phi(t)$ is the response time series.

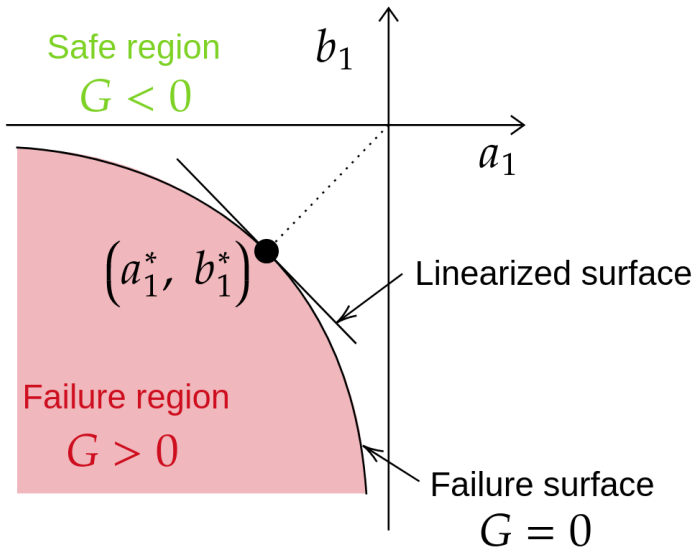

Figure 1.3: FORM method searches the most probable seed on the failure surface

A failure indicator is then defined by

$$G(a_1, b_1, \cdots, a_N, b_N) = \phi(t_d; a_1, b_1, \cdots, a_N, b_N) - \phi_0 \tag{1.5}$$

where ϕ_0 is the user-defined failure threshold. The indicator is positive when the system has response that exceeds the threshold (or system failure), and it is negative when the system has safe response. The equation

$$G(a_1, b_1, \cdots, a_N, b_N) = 0 \tag{1.6}$$

defines a failure surface in a high-dimension parameter space.

The objective of the FORM method is to search for a random seed

$$(a_1^*, b_1^*, \cdots, a_N^*, b_N^*) \tag{1.7}$$

that is closest to the origin on the failure surface. It represents the most probable seed that leads to a system failure at the design time. The name of the method "first order" means that the failure surface in the probability space is linearized, rather than linearizing system dynamics. Figure 1.3 visualizes the failure surface with a low-dimension example.

There are pros and cons to applying the FORM method. The method extends to the nonlinear systems which are not addressed well by linear methods. The method is also efficient in finding the optimal point $(a_1^*, b_1^*, \cdots, a_N^*, b_N^*)$ by using an iterative

searching algorithm on the failure surface. However, the iteration may be expensive if the time-domain simulation of $\phi(t)$ takes a long time, like a CFD simulation. The method is also not a generative model since it produces only the optimal parameter seed on the failure surface instead of a series of seeds in the failure region. The Fourier representation of a Gaussian process in Equation 1.2 generates a non-ergodic random process (*Shinozuka and Deodatis*, 1991).

1.2.4 Extrapolation Methods

Extrapolation methods estimate the extreme value distribution of a random process (Gaussian or non-Gaussian) during an exposure time window based on observed data. The time window length to be estimated is usually much larger than the time length when the observed data is collected. For example, an oceanographer is interested in the expected extreme wave elevation during 100 years given the observed data collected in one year. Many methods can be categorized into this extrapolation family. The extrapolation process usually consists of three steps in establishing mathematical models. The first step is to select mathematical assumptions. Example assumptions include observations are independent and identically distributed, or peaks of the random process appear Poisson-like. Mathematical models are often derived according to the selected assumptions. These models are usually a parametric probability distribution that describes how the observations or quantities of interest are distributed. The second step is to estimate the model parameters based on the observed data, which can also be regarded as model fitting. The last step is extending the fitted model into the region of interest with the help of order statistics or a basic assumption like Bernoulli, Binomial, etc.

Weihull (1951) probability paper, as traditional extrapolation methods, are simple and widely used in the field of oceanography. *Gumbel* have proved that the largest value among a group of independent and identically distributed random variables follows the Generalized Extreme Value (GEV) distribution as the group size becomes infinite (also referred to as the asymptotic condition). The CDF of the GEV distribution for the extreme random variable is

$$F(x;\mu,\sigma,\xi) = \exp\left[-\left(1+\frac{x-\mu}{\sigma}\xi\right)^{-1/\xi}\right] \tag{1.8}$$

where μ, σ, ξ are the location, scaling, and shape parameters respectively. Depending on the sign of ξ, the GEV distribution can be partitioned into three types, which are

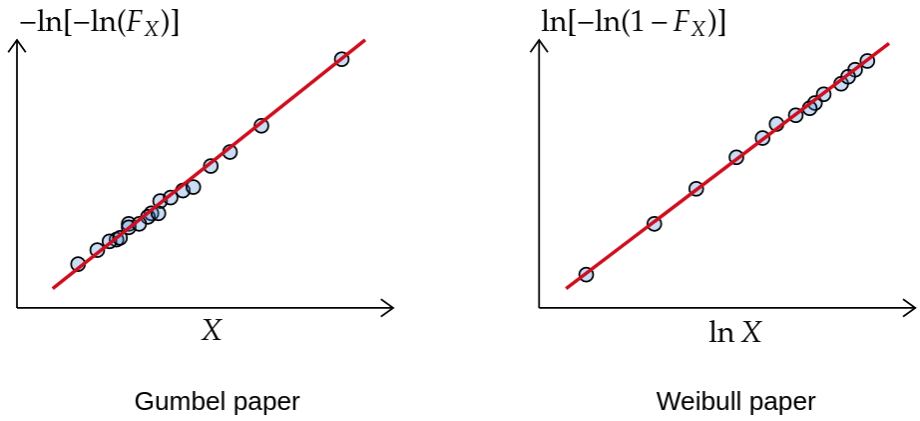

Figure 1.4: Example of using Gumbel and Weibull probability paper to fit observed data

Gumbel (Type I), Fréchet (Type II), and Weibull (Type III). By plotting the observation data and the corresponding empirical CDF in log log scale, the observations are expected to be fitted by a straight line if they follow a Gumbel distribution. Alternatively, the observations can also be plotted in $\log - \log\log$ scale to check whether they follow a Weibull distribution. The fitted line then automatically produces the parameters of the GEV distribution. Figure 1.4 illustrates the probability paper of the Gumbel and Weibull types.

The Peak-Over-Threshold (POT) method by *Davison and Smith* (1990), and *Leadbetter* (1991) extracts more data from observed time series compared to daily or yearly maxima. The method considers all peak values that exceed a certain threshold of u. It assumes that the exceedance events are independent, and occur with Poisson times. It also assumes that the excess values from the threshold follow the generalized Pareto distribution with a sufficiently large threshold.

$$F(y) = \Pr(Y < y) = 1 - \left(1 + c\frac{y}{a}\right)_{+}^{-1/c} \tag{1.9}$$

where a and c are scaling and shape parameters respectively, and Y is the excess value defined by $X - u$. By fitting the Pareto distribution and Poisson process to the observed data, the model is able to produce a statistical estimation of problems like the tail behavior and the return period (*Belenky et al.*, 2010, 2012; *Campbell et al.*, 2014).

While methods based on the GEV and POT concepts develop models for the asymptotic range, *Naess and Gaidai* (2008); *Næss and Gaidai* (2009); *Naess et al.* (2010); *Gaidai et al.* (2016) developed a new method of estimating the extreme dis-

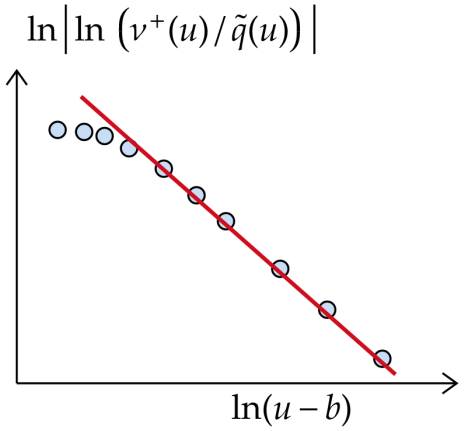

Figure 1.5: Example of using the parametric model to fit the processed mean upcrossing rates

tribution for a system response process in the sub-asymptotic range. The method considers the threshold crossing under the Poisson assumption. A parametric model is used to approximate the mean upcrossing rate $\nu^+(u)$ given by

$$\nu^+(u) = \tilde{q}(u)\exp\left[-a(u-b)^c\right] \tag{1.10}$$

where $\tilde{q}(u)$ behaves like constant, and a, b and c are shape parameters.

The mean upcrossing rates of various thresholds are estimated based on the observed data and used to determine the model parameters, as illustrated in Figure 1.5. Once the model of mean upcrossing rate is achieved, the rate for even a larger threshold that is located in the range of sparse observations can then be extrapolated. Statistical conclusions like the Extreme Value Distribution can then be calculated from the estimated mean upcrossing rate.

$$F_{M_T}(x) = \Pr(M_T \leq x) = \exp[-\nu^+(x)T] \tag{1.11}$$

where $M_T = \max\{X(t); 0 \leq t \leq T\}$ is the extreme value during the exposure time window of length T.

One of the advantages of extrapolation methods is their simplicity. Users can directly fit a parametric model to the observed data and they can also easily determine the confidence interval of the parameters, which depends on the sparsity of the data in a certain region. The model parameters are computed instantaneously since the involved optimization problem of finding the best-fit parameters is straightforward. By introducing more assumptions, relatively fewer parameters are needed to be de-

Perspective	Linear	DLG	FORM	Extrapolation
Gaussianity	Gaussian	Gaussian	Both	Both
Domain	time/probability	time/probability	time/probability	(time)/probability
Generative	No	Yes	No	No
Data-driven	No	No	Yes	Yes
Cost	low	low	It depends	It depends

Table 1.1: Compare state-of-the-art methods of extreme event modeling

termined. Therefore, extrapolation methods are often used to efficiently estimate the EVD for the studied process.

There are also shortcomings embedded in the extrapolation methods. These methods rarely provide a time-domain simulation environment to verify whether the estimated extreme event can be really realized. The methods are also applied with ideal assumptions like a high threshold value or asymptotic condition. Moreover, to reduce the variance of the model parameters, an adequate amount of data needs to be collected. The data collection task might be expensive for application scenarios like CFD simulations or experiments.

1.3 Identification of Research Need

The state-of-the-art methods of modeling and estimating extreme events have been briefly reviewed in Section 1.2 with their advantages and disadvantages discussed. Table 1.1 compares these methods from different perspectives.

The first perspective shown in the first row of the table is whether the method can be applied to a Gaussian or non-Gaussian process. The linear theory and the DLG method usually assume the studied process is a Gaussian process. The process can be a Gaussian environment like the elevation process of Gaussian ocean waves or the system response of a linear system under Gaussian loadings. The FORM method and the extrapolation methods can be applied to both Gaussian and non-Gaussian processes.

The study domain of these four methods is compared in the second row. Since the linear method uses the Gaussian assumption, it can give the theoretical closed-form distribution of positive maxima and their occurring times. The statistical results answer the question in both the time and probability domains. The DLG method is able to produce time-domain realizations while being constrained by the probability conclusions from the linear method. The FORM method iteratively evaluates the response in the time domain to search for the optimal random seed in the probability space. It also can produce a statistical summary like the failure probability of the

studied system. On the other hand, the extrapolation method often fits a model to the observed data, which usually comes from processed time-domain data. Moreover, conclusions of interest for this method are usually in probability description like return period, mean upcrossing rate, and etc.

The third dimension of comparison is whether the method is generative, which means whether the method is able to generate time-domain realizations to verify the estimated extreme event indeed happens. This measure is important since visualizing the dangerous behavior of the system presented in a more complete story can help ship and marine designers improve safety during the design iteration and understand failure more completely. Compared to other methods, the DLG method, as the name indicates, is the only method that is able to generate loading environments for extreme event simulation.

The fourth perspective for comparison is whether the methods rely on observed data. The observations can come from experiment measurement or numerical simulation. Since the linear method and the DLG method are derived or developed with Gaussian assumption, theoretical, or even closed-form results are often available without relying on observed data. However, for more general cases like a non-Gaussian process, observed data is often required to understand the system.

Finally, the computational cost of these methods is compared. Due to the simplicity of the Gaussian assumption, the linear method and the DLG method can finish the analysis quickly. However, for the FORM method and the extrapolation method, the cost mainly comes from data collection. The FORM method requires the response evaluation for a given random seed during iterations. The extrapolation method requires enough observed data to fit the parametric model. If the data collection process involves expensive procedures like CFD simulation or experiment measurements, the cost becomes very high. If the observed data comes from simplified mathematical simulations that can be efficiently conducted, the cost is low.

Considering the advantages and disadvantages of the state-of-the-art methods, the current research is proposing a new framework to model the extreme event of the system response. In the format of Table 1.1, the proposed framework is designed with the properties in Table 1.2.

The proposed framework can be applied to both Gaussian or non-Gaussian processes. Since many marine systems behave nonlinearly, especially in the context of extreme events, the response of nonlinear systems is generally non-Gaussian. Hence, the proposed framework is designed to work in this more general situation with as few assumptions as possible. Due to this general application situation and fewer as-

Perspective	the proposed method
Gaussianity	Both
Domain	time/probability
Generative	Yes
Data-driven	Yes
Cost	medium

Table 1.2: Properties of the proposed framework in current research

sumptions to be relied upon, the proposed framework is designed to be data-driven. Moreover, nowadays numerical simulations like CFD can achieve acceptably accurate results compared to experiments and at the same time produce large amounts of data. For example, a typical CFD case can have multi-million cells describing a flow domain and necessary fluid properties inside each cell can be retrieved at all time steps during the simulation. Many users can obtain access to the High-Performance Computing (HPC) service remotely, which gives a more promising application area for data-driven approaches. Though more data becomes accessible, acquiring valuable data is challenging. Hence, the proposed data-driven framework is designed to require less data, which reduces the cost of data collection for the user. The proposed framework is also generative by providing the user with simulation environments. By conducting simulations with the generated environments, the user is able to observe the extreme response of the system around a certain timestamp. These observed extreme events and their time ensemble allow a more complete analysis in both the time and probability domain.

1.4 Research Overview and Thesis Outline

As mentioned in the previous section, the proposed framework is expected to model extreme events for a linear or nonlinear system. The data-driven and generative requirements for the proposed framework makes the task challenging. Therefore, machine learning methods are heavily used in the proposed framework to better format and solve the problem. These machine learning concepts and methods have been applied successfully in other fields of study like computer vision, natural language processing, and speech recognition. As will be shown in later chapters, these machine learning methods used in the proposed framework achieve encouraging results for marine-related tasks.

The proposed framework considers a deterministic system exposed to a random environment. An example can be a ship moving in random ocean waves. Since the

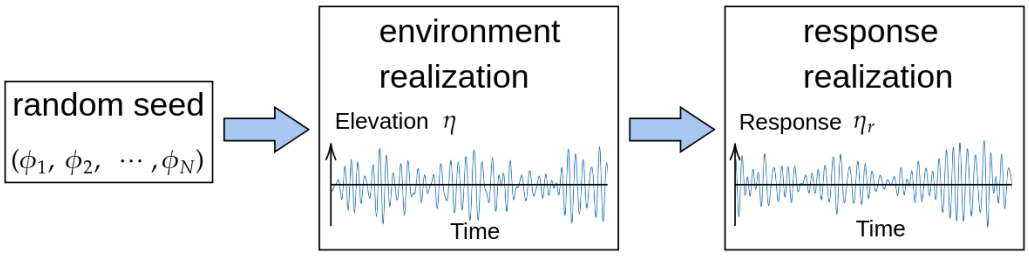

Figure 1.6: A random seed specifies an ocean environment, and determines a response realization.

only randomness comes from the loading environment, the ship motion can be fully described if the randomness of the environment is specified. The specific setting of the environment is here called the random seed. In other words, each random seed corresponds to one realization of the motion process. Mathematically, the ocean wave is a random environment and its elevation process can be approximated by a Fourier series

$$\eta(x,t) = \sum_{i=1}^{N} a_i \cos(2\pi f_i t - k_i x + \phi_i) \tag{1.12}$$

where N is the number of Fourier components, constants a_i, f_i, and k_i are amplitudes, frequencies, and wavenumbers respectively for different components. The only random variables ϕ_i are the corresponding phase angles, which can be regarded as the random seed in this example. Once the random seed in the format of phase vector $(\phi_1, \cdots, \phi_N)$ is specified, the wave environment is uniquely determined. The ship motion under such an environment can then be represented by

$$\eta_r(t; \phi_1, \cdots, \phi_N) \tag{1.13}$$

where t is time and $(\phi_1, \cdots, \phi_N)$ is the specified random seed. Figure 1.6 illustrates how a random seed is mapped to a environment realization and a response realization.

The objective of the proposed framework is to generate random seeds as many as the user wants and these seeds can lead to large responses that are over the user-specified threshold at the design time. Mathematically, the proposed framework desires to return the seeds sampled from the distribution

$$\Pr\left(\phi_1, \cdots, \phi_N | \eta_r(t_d; \phi_1, \cdots, \phi_N) > \zeta\right) \tag{1.14}$$

where t_d is the user-specified design time for the user to observe the extreme event, ζ is the user-specified threshold for the system response. With the generated seeds,

the user can simulate the extreme scenarios.

The proposed framework consists of two modules, which are named as Threshold Exceedance Generator (TEG), and Design Response Estimator (DRE). The TEG module is designed to learn the distribution (or pattern) of the random seeds in the collected dataset and then generate new seeds that share a similar distribution. When applying the TEG module, it needs to know how large (or dangerous) the response is at the design time for a given seed. For a simple dynamical system where the behavior can be simply modeled, evaluation of the response is usually simple and efficient. In this case, the DRE module is not needed. However, if the evaluation of the response is computationally expensive or no simple mathematical model can be built, it is suggested to use the DRE module to first characterize or identify the system. The DRE module is designed to estimate the response of the system at the design time for a given seed. Generally speaking, the TEG and DRE modules focus on different aspects of the task, as shown by

$$\Pr(\underbrace{\phi_1, \cdots, \phi_N}_{\text{TEG}} \,|\, \underbrace{\eta_r(t_d; \phi_1, \cdots, \phi_N)}_{\text{DRE}} > \zeta). \tag{1.15}$$

As data-driven modules, TEG and DRE ask for the dataset in different formats. The TEG module requires a seed table which consists of samples of seed vectors and their corresponding response. On the other hand, the DRE module, like many other system identification methods, requires pairs of input and output time series. Due to the wide range of applications from each module, they can be used with other existing methods. For example, the DRE module can benefit the FORM method in a quick evaluation of the system response. In the later chapters, the TEG module and the DRE module will be explained in great detail and supported by many examples. Figure 1.7 shows the structure map of the current thesis.

In Chapter II, the fundamental background of machine learning is briefly reviewed, which includes Multi-Layer Perceptron (MLP), Recurrent Neural Network (RNN), and Long Short-Term Memory (LSTM). Instead of listing rigorous mathematical equations, the chapter focus on the intuition and functionality of the machine learning methods, which are helpful to better understand the TEG and DRE modules.

Chapter III discusses the TEG module from the motivation and the methodology. The computation complexity of the module is theoretically analyzed. The size of the data collection required by the module is also discussed through experiments. To validate the module, examples of linear waves, nonlinear waves, and nonlinear ship roll are analyzed. The linear wave example starts from a bichromatic wave

to better visualize the joint distribution of Fourier phases without worrying about the high-dimension difficulty in the visualization of phase vectors. A five-component wave is studied to check the pairwise joint distribution among the phases. Finally, a 30 component wave example is introduced followed by a discussion on the relation between the threshold conditional probability and the Extreme Value Distribution. After the module succeeds in linear examples, examples with more nonlinearity are brought in. Specifically, examples are the second-order nonlinear wave and the ship roll excited by wave and wind loadings.

Chapter IV explains the DRE module. To validate the module for the identification of dynamical systems, examples from the previous chapter are tested. Two new examples from CFD simulations, tank sloshing and a floating object, are introduced to present the potential of the DRE module.

Chapter V combines the TEG and DRE modules with an integrated example in order to illustrate the usage of the proposed framework. The example is a floating object undergoing pitch motion in irregular waves. The example is simulated via CFD. As is shown in the chapter, the proposed framework achieves the goal.

Finally, Chapter VI summarizes the results and findings. Future work is also discussed to improve this data-driven framework.

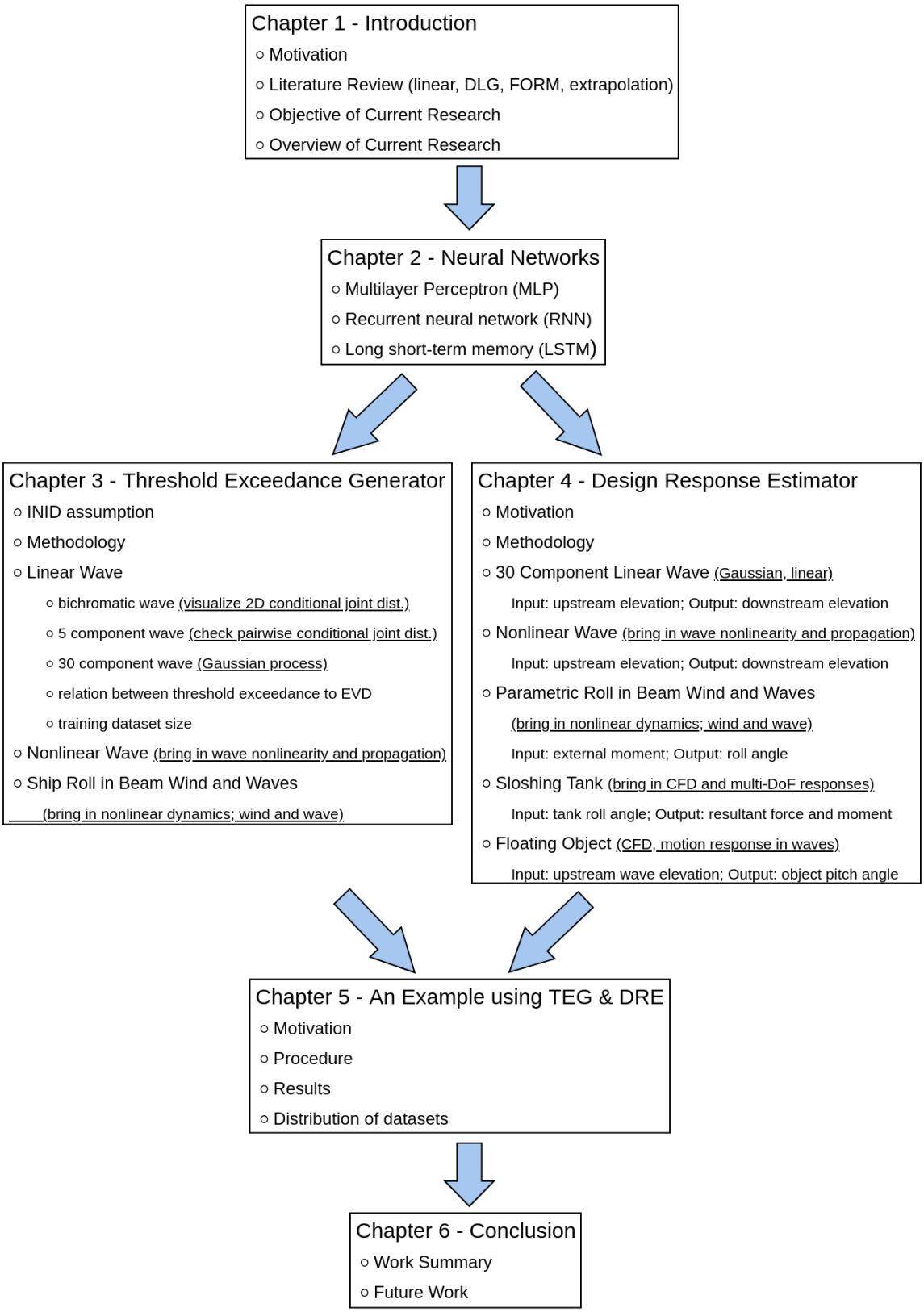

Figure 1.7: Chapter map

CHAPTER II

Neural Networks

Machine learning methods, especially neural networks, are heavily used in the proposed framework. This chapter focuses on the important concepts and functionality of relevant machine learning methods, instead of listing mathematical equations or providing rigorous proofs. The current chapter aims to help the readers who are not familiar with the machine learning domain to better understand the intuitions and ideas that appear in later chapters. Specifically, this chapter presents the basic concepts that are used to use Recurrent Neural Networks (RNN) including Long Short-Term Memory (LSTM).

2.1 What is Machine Learning?

Most machines complete their tasks by following concrete instructions that are explicitly programmed by humans. However, there are other advanced tasks where humans cannot give explicit instructions to the machine. Examples are the recognition of handwritten letters and retrieval and ranking of search results based on the user's query. The instructions of completing these tasks are no longer given by humans, and instead, they are learned from data. Humans are still involved in the process by preparing the data and designing the learning methods. Since most learning methods are data-driven, other names like pattern recognition or statistical inference are also used in different fields. As the amount of data has tremendously grown in the past several decades, machine learning methods have achieved breakthroughs in many fields. After several generations of development, machine learning methods perform well on many advanced tasks that humans are normally good at, such as perception and decision making.

Depending on various types of tasks, machine learning methods can be categorized into different families. A fundamental and widely-used family of methods is called

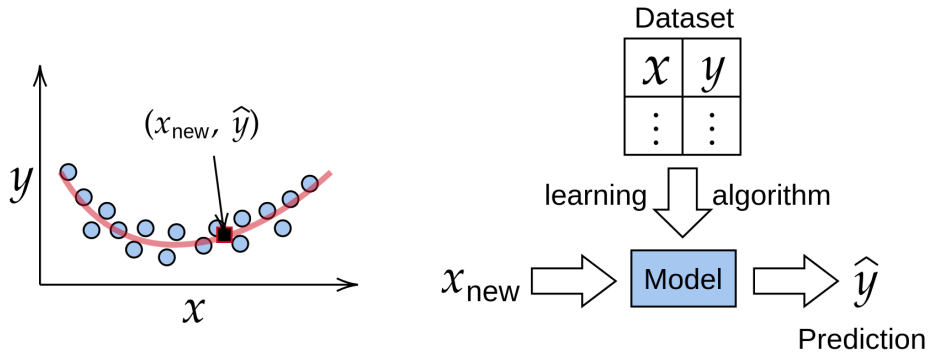

Figure 2.1: A supervised learning example: curve-fitting to data points

supervised learning. It develops predictive models based on both input and output data. The developed model acts as a mapping from the independent variables (or features) to the dependent variables (or labels). One example of a supervised learning task is to fit curves to data, shown in Figure 2.1. Given the dataset (blue circles), the fitted curve (red) is found by machine learning algorithms and it produces the prediction $\hat{y}$ for any new independent variable x_{new}. Other supervised learning examples are classifying an image into "cat" or "not cat", and controlling the steering angle and the accelerator based on the driving condition. Based on the convention, supervised learning is further broken into regression problems for cases when dependent variables are continuous, and classification problems in cases when dependent variables are discrete.

Machine learning models can also be described from the probabilistic perspective. These descriptions are important and widely-used since they are built on probability theory. For the previous classification example, producing the probability of the image being a "cat" image tells more story than just outputting a binary result (1: "cat"; 0: "not cat"). In this case, it may sound trivial to classify the image into the "cat" class if the predictive probability is larger than 0.5. However, such a decision threshold may be difficult to make for an example of classifying a tumor image into "benign" or "malignant". A too high threshold can result in an error that may bring an undue concern to a healthy patient. In the opposite case, too low of a threshold can result in an error of the cancer being not treated in time. Therefore, separate from the decision phase, the probabilistic description $\Pr(y|X)$ with X being the image and y being the class, in this case, tells a more complete story than just classification decision.

One probabilistic perspective, especially useful for classification models, is to determine whether the model is discriminative or generative. A discriminative model learns the conditional distribution $\Pr(y|X)$ from data, while a generative model learns

the joint distribution $\Pr(X, y)$. In the previous example, classifying an image into "cat" or "not cat" is a discriminative model. As a comparison, a corresponding generative model is to produce a new image and its class at the same time sampled from the learned distribution $\Pr(X, y)$. One can also ask the generative model to produce a "cat" image by sampling from $\Pr(X|y) = \Pr(X, y)/\Pr(y)$. Since the generative model learns a more complex (or higher dimensional) distribution than the discriminative model does, it can also serve to produce discriminative results by $\Pr(y|X) = \Pr(X, y)/\Pr(X)$. Though the generative model can produce more results than the discriminative model, it usually needs more time and data to be trained.

In later chapters, the concepts introduced here will reappear for the two modules of the proposed framework. As will be shown later, the Threshold Exceedance Generator (TEG) module is a classification and generative model, and the Design Response Estimator (DRE) module is a regression and discriminative model.

2.2 How does Machine Learning Work?

To build a machine learning model that approximates the mapping from features to labels, a typical workflow looks like Figure 2.2. The first step to build such a data-driven model is to collect the data which comes from either experimental measurements or numerical simulations. The collected data are usually tabulated as shown in Table 2.1, where each row presents a sample (or a record) and columns represent features or labels. The convention of laying out samples by rows is commonly-used in many machine learning frameworks and is adopted in the current thesis. The raw data needs to be preprocessed before being sent to the models. Common preprocessing includes but not limited to imputation, scaling, and encoding. Data imputation is to infer the missing value within the table. One of the imputation approaches is to fill the missing blanks with the mean value of the corresponding feature column. Data scaling is to normalize each feature and label column separately such that the values from these columns are comparable. Common scaling schemes are *Min-Max* and *Standardization*. The *Min-Max* scaling sets the lower bound of the column to zero and the upper bound of the column to one. Then the linearly scaled values from the column range from zero to one. The *Standardization* approach scales each value of the column by the mean and the standard deviation of that column. Data encoding is required when the categorical (or class) features appear in the dataset. *Integer-Encoding* assigns integers from 0 to $K - 1$ to each of K classes. However, sometimes the *Integer-Encoding* may bring in an unwanted relation, which means a

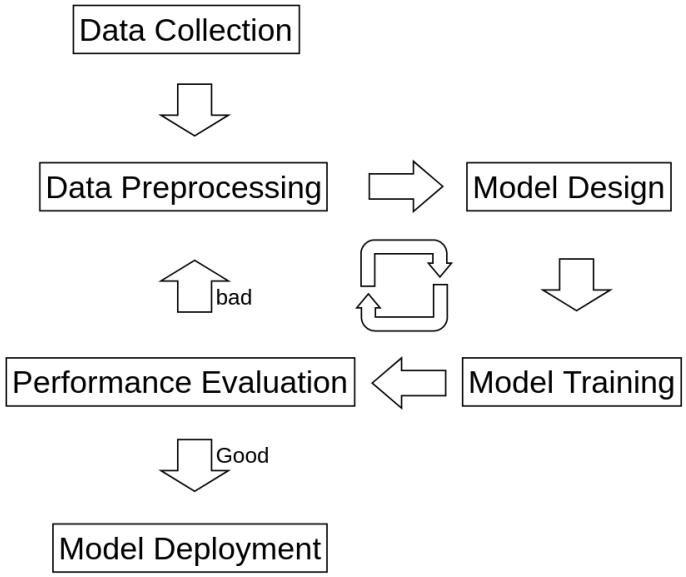

Figure 2.2: Workflow to build a machine learning model

X_1	X_2	$\cdots$	y_1	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 2.1: Layout (row-wise) of collected data.

class encoded with $K-1$ is not necessarily larger than a class encoded with 0. A more common encoding approach that addresses this problem is called *One-Hot-Encoding*, where the encoding integer is converted to a one-hot binary vector. The *One-Hot-Encoding* process to encode a color feature column is illustrated in Figure 2.3. Later chapters will show how the *Standardization* and the *One-Hot-Encoding* are used in both TEG and DRE modules.

After the data is preprocessed, it is given to the model to learn (or recognize) the pattern inside the data, which plays the core role of a machine-learning problem. Depending on the task complexity and the amount of data collected, different model

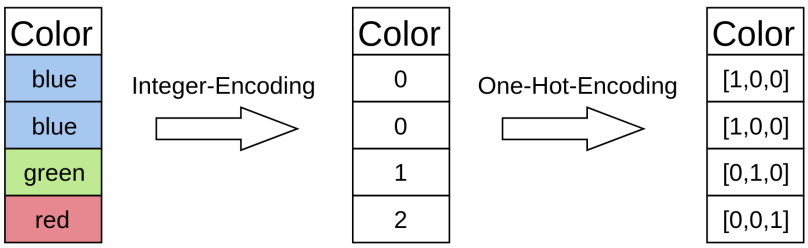

Figure 2.3: Categorical feature encoding

designs are considered. Many machine learning models are parametric and can be regarded as a function $f(X;\theta)$ where X are the given features and θ are the model parameters. Different model architectures mean different structures between X and θ. For the previous example of fitting a curve to the given data points, a polynomial can be regarded as the machine learning model with parameters being the coefficients in front of each power term. One of the well-known machine learning models, called *Neural Network*, is heavily used in the current thesis and is reviewed in the next section. Once a parametric model is selected, the model training process is to find the best (or nearly best) parameters such that the prediction $\hat{y} = f(X;\theta)$ from the model is as close as possible to the given ground-truth. A mathematical way to describe the training process is in the optimization format

$$\min_{\theta} \quad L(\hat{y}, y) \tag{2.1}$$

where $L(\cdot,\cdot)$ is the loss function to measure the error between the prediction and its ground-truth. Table 2.2 lists the common loss functions, where the MSE and MAE loss functions are used in regression problems and the Cross-Entropy loss functions are used in classification problems. Since there are multiple samples in the dataset, the loss is often evaluated based on the average across multiple samples. Since the closed-form solution to the optimization problem is often unavailable, an iterative approach like Gradient Descent is widely used. Gradient Descent updates the model parameters by considering the derivatives of the loss with respect to the model parameters

$$\theta \leftarrow \theta - \alpha \frac{\partial L}{\partial \theta} \tag{2.2}$$

where α is called the learning rate. Figure 2.4 demonstrates how the parameters are optimized during the training process for a 2D parameter space. Many optimizers based on Gradient Descent are developed, which include but are not limited to *Stochastic Gradient Descent*, *Adagrad* (*Duchi et al.*, 2011), *RMSprop* (*Hinton et al.*, 2012), and *Adam* (*Kingma and Ba*, 2014). The iterative optimization (or training process) stops when the loss converges, thereby producing a trained model with the optimal parameters.

Once the model is trained, it is important to evaluate its predictive performance. For a sufficiently complex model with a sufficient number of parameters relative to the amount of collected data, it can theoretically fit all the given data perfectly. Though the performance of the trained model is perfect on the given data, its performance may become worse for the new data. This phenomenon is commonly seen

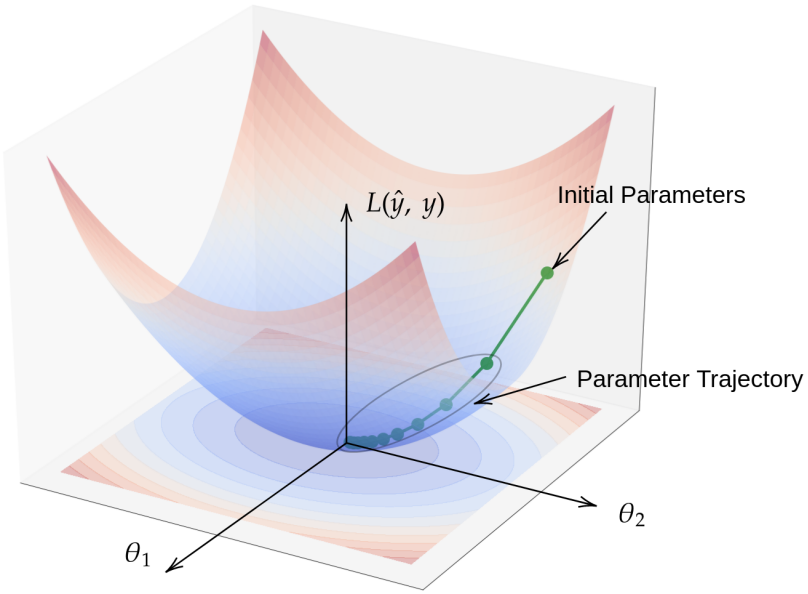

Figure 2.4: Gradient Descent illustration for a 2D parameter space

Name	Definition
Mean Squared Error (MSE)	$L(\hat{y}, y) = \frac{1}{2}(y - \hat{y})^2$
Mean Absolute Error (MAE)	$L(\hat{y}, y) = \lvert y - \hat{y} \rvert$
Cross Entropy (binary)	$L(\hat{y}, y) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$
Cross Entropy (multi-class)	$L(\hat{y}, y) = -\sum_{i=1}^{K} y_i \log(\hat{y}_i)$

Table 2.2: Common Loss Function for Neural Networks

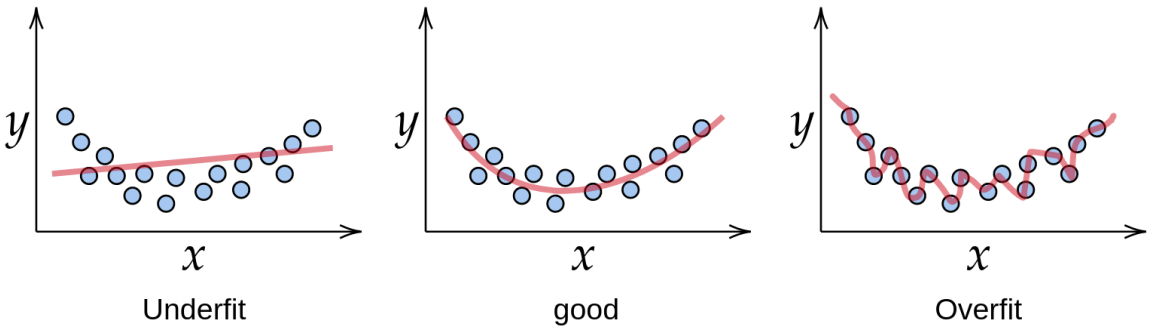

Figure 2.5: Models (red curves) underfit, well-fit, overfit the observations (blue dots)

SIZE OF THE RAW DATASET	RATIO FOR TRAINING-VALIDATION-TEST
10^3	0.6 - 0.2 - 0.2
10^4	0.7 - 0.2 - 0.1
$> 10^6$	0.98 - 0.01 - 0.01

Table 2.3: An example with common ratios for data splitting

in many cases, called *overfitting*. Figure 2.5 illustrates how a complex model overfits the given data. Generally speaking, the overfitted model tends to "memorize" the noise while recognizing the pattern in the data. To objectively measure the predictive performance of the trained model, one typical approach is to form a *validation* dataset. Hence, the collected data is usually split into the *training* and *validation* datasets. The training dataset is then used to fit the model parameters and the validation dataset is used to measure the model performance on new data. When multiple model candidates are considered, a third dataset called *test* dataset is often formed. In this case, the training dataset is again used to fit the parameters, and the performance of the model candidates on the validation dataset produces the best candidate. The performance of the best candidate on the test dataset is then reported as the performance for "unseen" data. As a rule of thumb, Table 2.3 lists the common ratios for data splitting.

To avoid overfitting, it is important to select a proper model complexity that is relative to the amount of collected data. As the amount of collected data increases, a more complex model can be adopted. Common approaches to avoid overfitting are collecting more data, or selecting a simpler model design. Another way to control the complexity of the model, called *early stopping*, is stopping the training process of a complex model if the performance on the validation dataset becomes worse. The model complexity can also be changed by regularizing the model parameters. For the curve fitting example, *regularization* is achieved by introducing the norm of the model parameters to the loss function.

$$\min_{\theta} \quad L(\hat{y}, y) + \lambda ||\theta||^2 \tag{2.3}$$

where $||\cdot||$ is some norm measure like L_1-norm or L_2-norm and λ is the regularization coefficient specified by the user.

As shown in later chapters, neural networks are selected as the model design for both TEG and DRE modules. Reducing the model complexity and early stopping are used to avoid overfitting. The training costs for both modules are relatively low and the trained models achieve good results on both training and validation datasets.

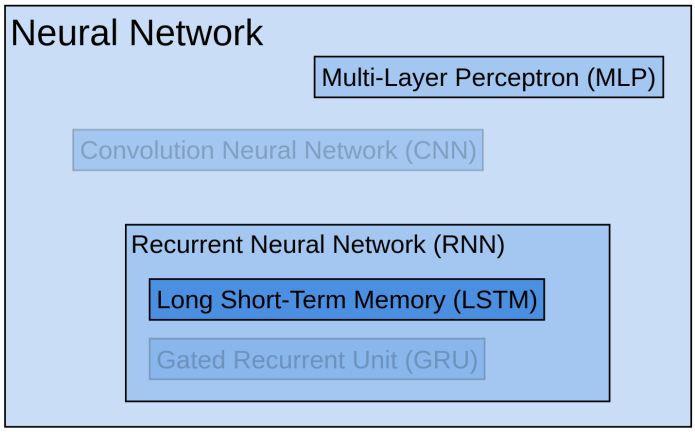

Figure 2.6: A Venn diagram showing Neural Networks family

2.3 Neural Network Structures

In this section, one family of Neural Networks is briefly reviewed. They are heavily used in the current design of the TEG and DRE modules. The introduction starts from a very basic vanilla type of neural network, called a Multi-Layer Perceptron (MLP), where its architecture and training process is reviewed. After familiarization with MLP, a deeper type of neural network called a Recurrent Neural Network (RNN) is discussed to better learn the pattern of sequential data, such as a time series. A concrete architecture of RNN, called Long Short-Term Memory (LSTM), is then introduced.

Figure 2.6 shows the relation among different neural network architectures via a Venn diagram. In this section, the MLP, RNN, and LSTM design are reviewed. Note that there exist other types of neural networks like Convolutional Neural Network (CNN) that are mainly used for image pixel data. Also, there exist other types of RNN like Gated Recurrent Unit (GRU) that may be suitable for the proposed framework and they are usually compared with the LSTM design. However, to demonstrate how machine learning methods can be applied to the extreme marine dynamical problems, the discussion and comparison are out of scope for the current thesis and only LSTM is considered when designing the proposed models.

2.3.1 Multi-Layer Perceptron (MLP)

Multi-Layer Perceptron (MLP), also known as Feedforward Neural Network, marks the beginning of neural networks. Due to its straightforward architecture, it is also mentioned as a “vanilla” neural network. The idea of neural networks first appeared

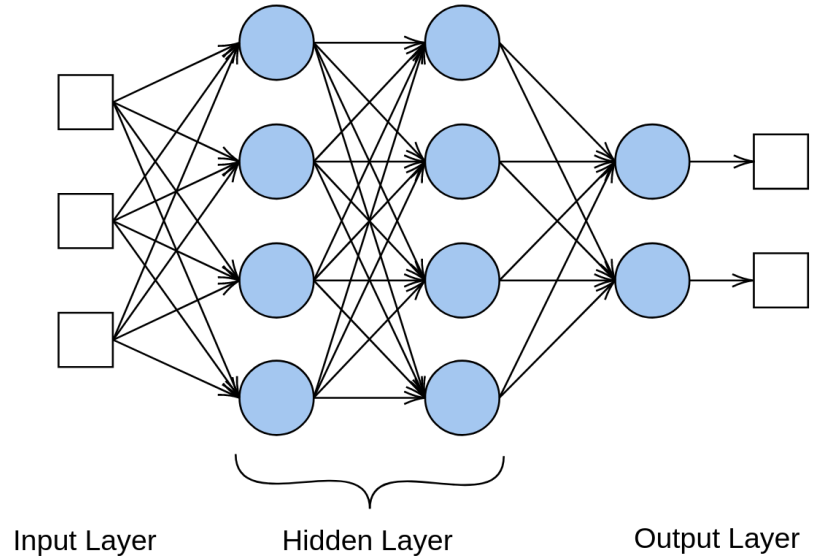

Figure 2.7: An example of Multi-Layer Perceptron architecture

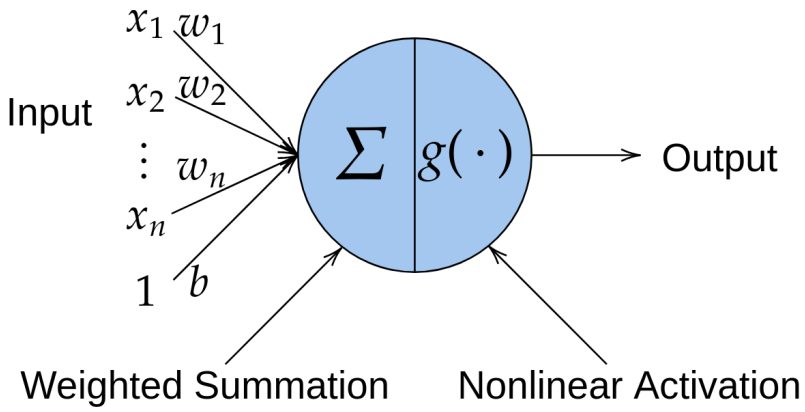

Figure 2.8: Node computation: weighted summation and nonlinear activation

as a mathematical model in the 1940's to imitate human intelligence. To describe how neurons in the human brain might work, the model tends to translate the biological connection among neurons into computer-based machines. The neural network models have been developed over several decades by many mathematicians, neuroscientist, and computer scientists (*McCulloch and Pitts*, 1943; *Hebb*, 2005; *Rosenblatt*, 1958; *Widrow and Hoff*, 1960; *Werbos*, 1990; *Rumelhart et al.*, 1986).

Figure 2.7 shows a typical MLP architecture as an example. The neural network maps the input on the left to the output on the right, which is called *Forward Propagation*. Its layers consist of one input layer, two hidden layers, and one output layer. The nodes and neurons inside each layer are represented as blue circles. The number

Name	Definition	Derivative
sigmoid	$g(z) = \frac{1}{1+e^{-z}}$	$g'(z) = g(1-g)$
tanh	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g'(z) = 1 - g^2$
ReLU	$g(z) = \max(0, z)$	$g'(z) = 1(z > 0), 0(\text{otherwise})$
leaky ReLU	$g(z) = \max(0.01z, z)$	$g'(z) = 1(z > 0), 0.01(\text{otherwise})$

Table 2.4: Common Activation Functions for Neural Networks

of nodes in each layer is also called *layer width*. For example, the width of both hidden layers is four. For the MLP architecture, the nodes inside each layer are connected to all the nodes in its previous and next layers. Therefore, these fully-connected layers are also called "dense" layers. The node computes its output based on its inputs in two steps, as shown Figure 2.8. The first step is a weighted summation of its inputs (and adding bias). Mathematically, it computes

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b \tag{2.4}$$

where $w_1, \cdots, w_n$ are the weight parameters and b is the bias parameter for the node. These parameters are optimized in the training process. The second step is a nonlinear activation.

$$a = g(z) \tag{2.5}$$

where $g(\cdot)$ is a nonlinear function. Common choices of the nonlinear activation functions are listed in Figure 2.4. Without the nonlinear activation, neural networks are not able to approximate possible nonlinear relations between the input and the output.

For the example architecture shown in Figure 2.7, the calculation of output for a training sample in a matrix representation is then

$$a^{[1]} = g^{[1]}([x_1, x_2, x_3]W^{[1]} + b^{[1]}) \tag{2.6}$$

$$a^{[2]} = g^{[2]}(a^{[1]}W^{[2]} + b^{[2]}) \tag{2.7}$$

$$[\hat{y}_1, \hat{y}_2] = a^{[3]} = g^{[3]}(a^{[2]}W^{[3]} + b^{[3]}) \tag{2.8}$$

where $W^{[1]} \in \mathbb{R}^{3\times4}$ and $b^{[1]} \in \mathbb{R}^{1\times4}$ are the parameters of the first hidden layer and $a^{[1]} \in \mathbb{R}^{1\times4}$ is the output of the first hidden layer. Similarly, $W^{[2]} \in \mathbb{R}^{4\times4}$ and $b^{[2]} \in \mathbb{R}^{1\times4}$ are the parameters of the second hidden layer and $a^{[2]} \in \mathbb{R}^{1\times4}$ is the output. Finally, $W^{[3]} \in \mathbb{R}^{4\times2}$ and $b^{[3]} \in \mathbb{R}^{1\times2}$ are the parameters for the output layer and $\hat{y} = a^{[3]} \in \mathbb{R}^{1\times2}$ is the final prediction. Therefore, the model has $16+20+10 = 46$

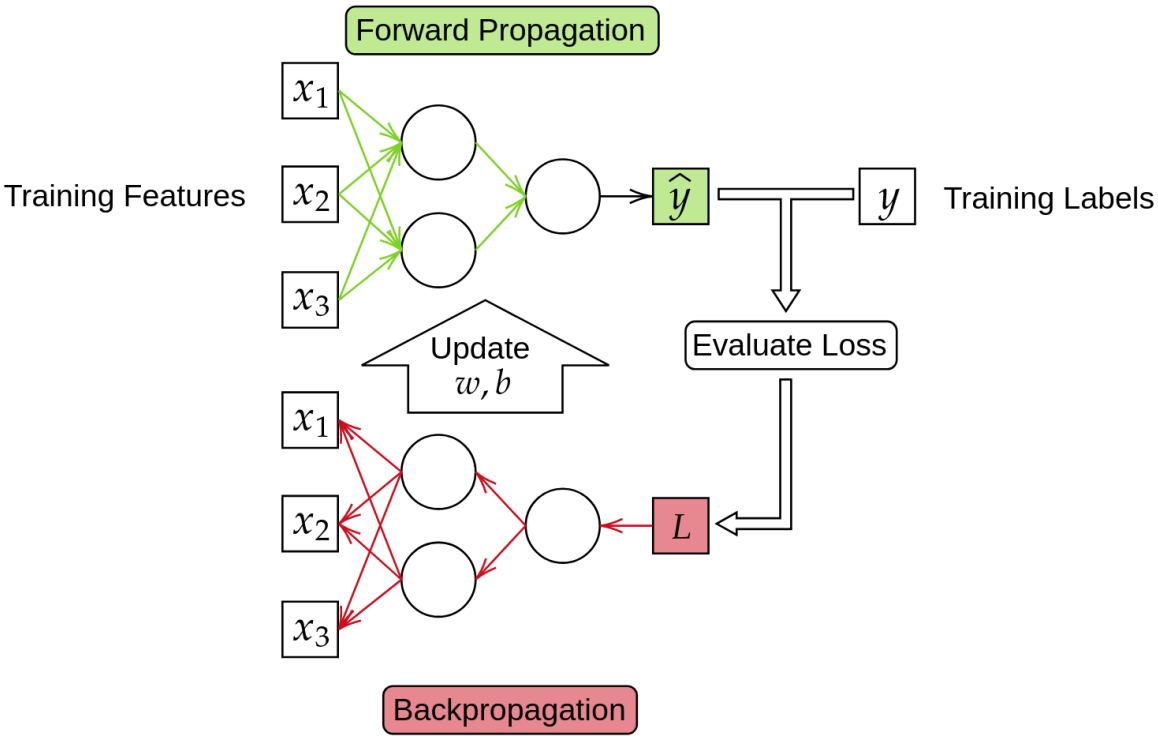

Figure 2.9: Training process: 1) forward propagation, 2) loss calculation, 3) backpropagation, 4) update parameters

parameters in total to fit the nonlinear relation between the input and the output.

The model training to determine the model parameters W and b is achieved through an iteration process shown in Figure 2.9. A batch of training samples is given to the model, which produces its output estimation of $\hat{y}$. Then estimates are compared against their according ground-truth y in the training dataset via a loss function. The averaged (or weighted) loss is then used to perform a *backpropagation*, where the derivatives of the loss with respect to all model parameters are calculated. The efficient computation of the derivatives in the backpropagation is performed from the output side to the input side. Once the derivatives are calculated, the model parameters are then updated via Gradient Descent.

As shown in later chapters, this vanilla type of neural network is used together with LSTMs in the TEG and DRE modules.

2.3.2 Recurrent Neural Network (RNN)

The Recurrent Neural Network (RNN) is developed by *Hopfield* (1982) and *Rumelhart et al.* (1986) to recognize the pattern of sequential data like time series, audio

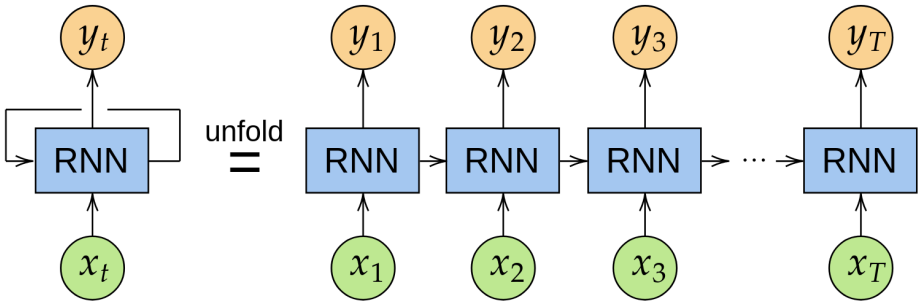

Figure 2.10: A simple RNN architecture

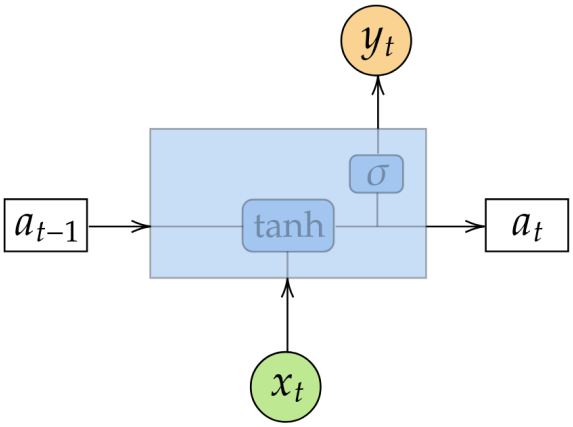

Figure 2.11: A diagram illustrating how the RNN unit becomes stateful

waves, and natural language. The supervised task is to predict the output sequence based on the given input sequence. Figure 2.10 shows a simple RNN structure. A RNN unit takes the current input x_t and produce the current output y_t. Different from a vanilla neural network, this RNN unit is stateful, which means the unit when calculating the next output y_{t+1} is different from the current unit. Therefore, it may be helpful by equivalently unfolding the concise representation (shown on the left) into a clearer long chain (shown on the right).

The RNN achieves its stateful behavior via an internal state variable, also known as hidden units. Figure 2.11 shows how the RNN unit updates its state over time. To calculate the output at the current timestamp, it first calculates its new hidden units based on the old hidden units and current input.

$$a_t = \tanh(W[a_{t-1}, x_t] + b) \tag{2.9}$$

where a_t and a_{t-1} are the hidden units at timestamp t (new) and $t-1$ (old), x_t is the input at timestamp t, W is the weight parameter and b is the bias parameter. The

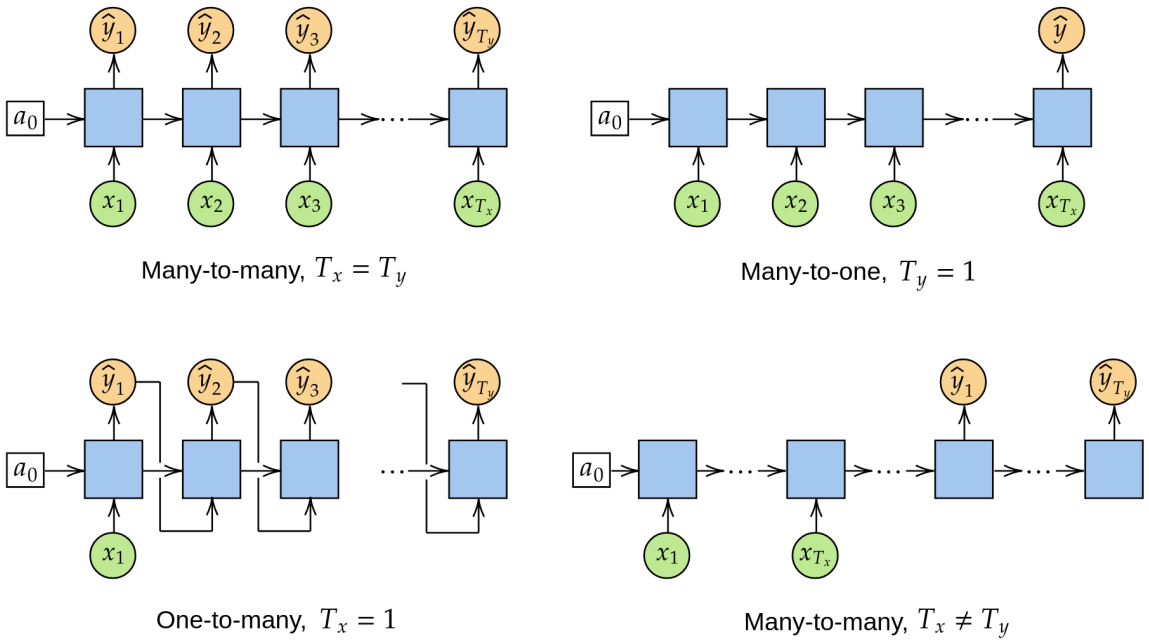

Figure 2.12: Different types of RNN architecture

prediction at current timestamp is then calculated as

$$y_t = \sigma(a_t) \tag{2.10}$$

where $\sigma(\cdot)$ is the sigmoid function given in Table 2.4. It is worth noting that the model parameter W and b are used for the prediction at all time steps while the hidden units vary over time. Therefore, compared to direct application of a MLP by treating the whole sequence as the input vector, the RNN reduces the number of parameters by sharing them over time. This efficient parameterization makes the RNN outperform the MLP in many sequential tasks. Another advantage of RNN over MLP is its flexible input dimension. The sequence length can be arbitrary without changing the RNN design.

Based on the lengths of the input and output sequences, different architectures of RNN are developed. Figure 2.12 shows how these RNNs are structured to account for different mapping conditions.

To increase the model complexity and learning ability for large datasets, a deep RNN is often used by "stacking up" multiple RNN layers, as shown in Figure 2.13. In the setting of Figure 2.13, the output sequence from each RNN layer is treated as the input sequence to the next (or upper) RNN layer. The output sequence from the very top RNN layer is hence the final output of this deep RNN structure.

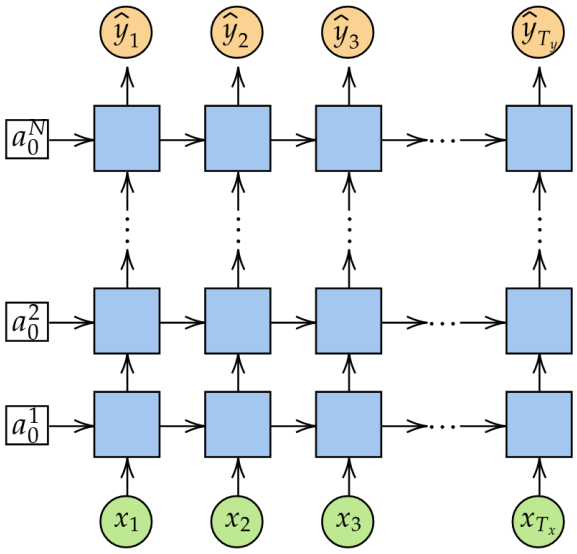

Figure 2.13: A diagram shows a deep RNN architecture

2.3.3 Long Short-Term Memory (LSTM)

Many types of RNN with similar structures are developed based on the idea of stateful units. A more complex design compared to Figure 2.10 is called LSTM, which is designed by *Hochreiter and Schmidhuber* (1997). It is proved to be successful in many machine learning examples and still widely used today. Figure 2.14 shows its computation unit, and the computation flow is well explained by *Olah* (2015). The important components of a LSTM unit are briefly reviewed as follows with a focus on intuition.

Similar to the hidden units in Figure 2.10, the cell state is used as a stateful memory in LSTM, shown in Figure 2.15. It acts like a "conveyor belt" carrying information flow according to Olah's description. The carried cargo on the belt can be thrown away and newly manufactured cargo can be added on the belt, which is controlled by two gates, called *forget gate* and *update gate* respectively. Figure 2.16 shows how the "forget" controller signal, f_t, is calculated based on the current input and previous output.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{2.11}$$

where h_{t-1} is the previous output, x_t is the current input, W_f is the weight parameter for the forget gate, and b_f is the bias parameter for the gate. The nonlinear sigmoid function produces a control signal that ranges from 0 to 1. Similar to the "forget"

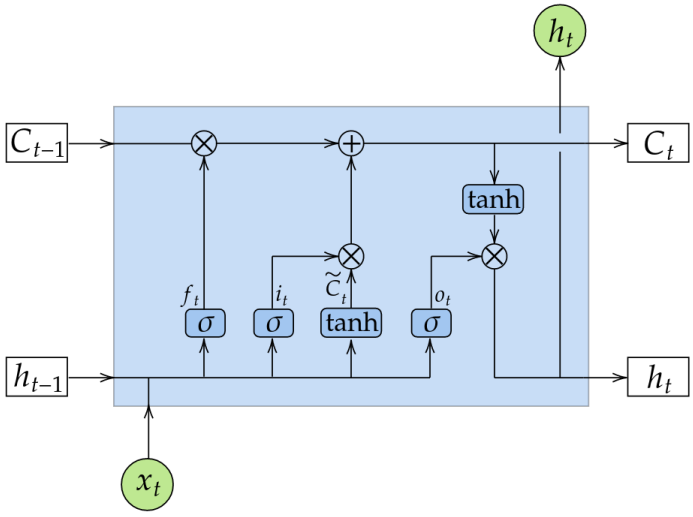

Figure 2.14: A diagram shows the LSTM unit

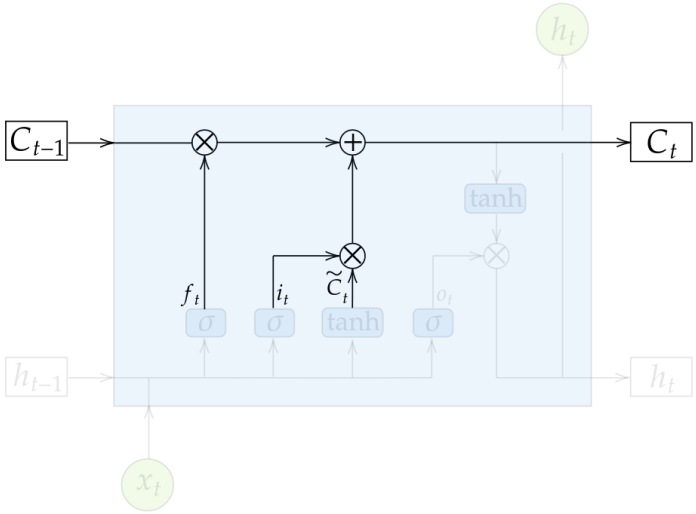

Figure 2.15: LSTM cell state

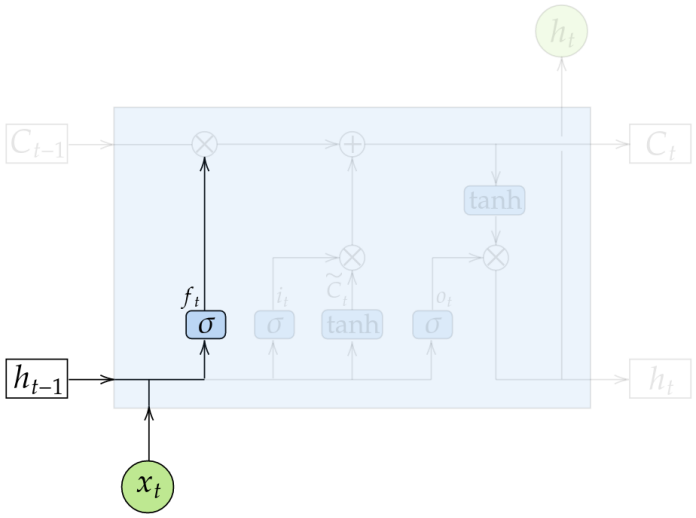

Figure 2.16: LSTM forget gate

controller, a update controller is also designed with its control signal calculated as

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2.12}$$

where W_i and b_i are the weight and bias parameters of the "update" gate, respectively. The "update" controller also produces a control signal that ranges from 0 to 1. The new cargo is "manufactured" by

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C) \tag{2.13}$$

where W_C and b_C are the weight and bias parameters for the new cargo. Note that the control signal are elementwise operations which allow each piece of cargo to be removed or added seperately. Therefore, the final cargo on the belt is then calculated by

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.14}$$

where $*$ is elementwise multiplication between the control signal and specific cargo.

Not all the cargo on the belt is needed in the output calculation. A *output gate* controls how much cargo on the belt are packed and produced as the output, as shown in Figure 2.18. Similar to the "update" controller and its newly manufactured cargo, a "output" controller and its packed cargo can be calculated as follows.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.15}$$

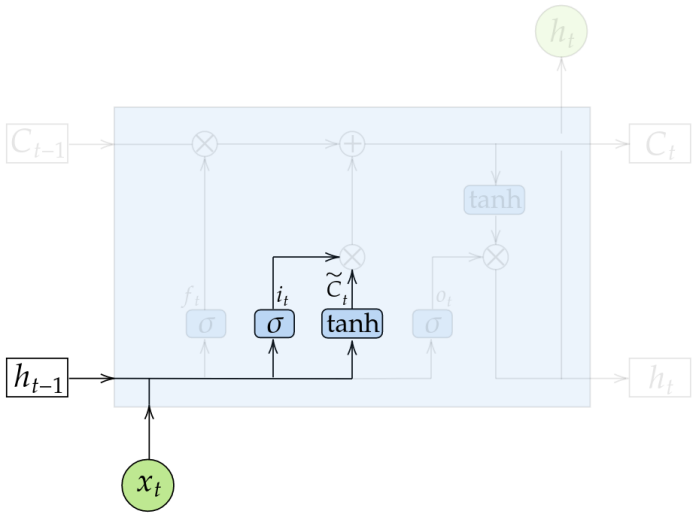

Figure 2.17: LSTM update gate

$$h_t = o_t * \tanh(C_t) \tag{2.16}$$

where W_o and b_o are weight and bias parameters for the "output" gate and h_t is the final output of the LSTM unit at timestamp t.

As shown in later chapters, both the TEG module and the DRE module use multiple layers of LSTM architecture together with Dense (or Fully-Connected) architecture to complete their different tasks. Specifically, the TEG module uses a many-to-one deep LSTM design and the DRE module uses a many-to-many (same length) deep LSTM design.

Table 2.5 compares the difference of machine learning tasks between the modules. The TEG module solves a generative classification problem, while the DRE module solves a regression problem. To generate a seed vector that specifies a loading environment, the TEG module treats the coordinates of the vector as sequential data. In terms of a Fourier representation, the TEG module can generate a Fourier phase sequence that follows the joint distribution of the given phase sets in a high-dimensional space. Different from the TEG module that deals with the Fourier phase in a frequency domain, the DRE module solves a system identification problem to predict the system response based on the system input in the time domain. In later chapters, different concrete architectures are designed to successfully complete the tasks for both modules. An open-source neural network library written in Python, called *Keras*, is used to implement the designs.

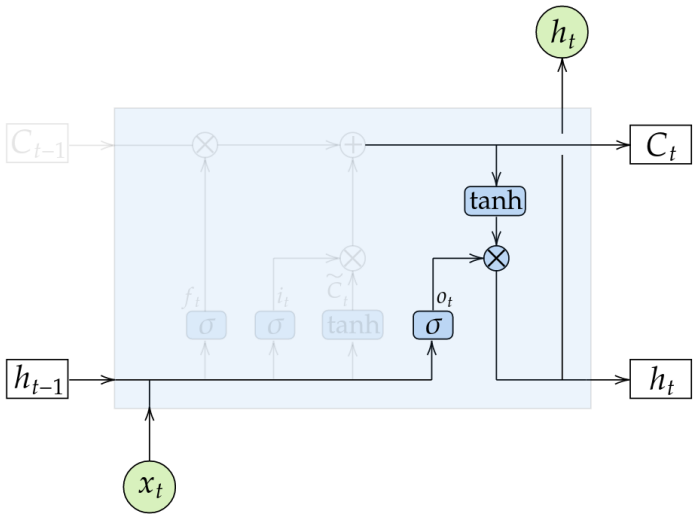

Figure 2.18: LSTM output gate

Compare	TEG	DRE
Generative	Yes	No
Regression	No	Yes
Independent variable	frequency domain	time domain
Dependent variable	discrete	real
Input	generation history	system input
Output	distribution of phases	system output

Table 2.5: Compare applying the LSTM network to TEG and DRE tasks

CHAPTER III

Module I - Threshold Exceedance Generator

In this chapter, a module named Threshold Exceedance Generator (TEG), is proposed to randomly generate deterministic wave environments that lead to large system response as many as the user desires. Simulation with the generated environments can explain the full time-domain story of the large system response. The chapter first demonstrates how the seed vector associated with a dangerous environment is distributed in high-dimensional space. Next, a machine learning approach, specifically a LSTM architecture, is proposed to recognize and generate new seed vectors based on the pattern in the probability space. Three examples, including linear waves, nonlinear waves, and nonlinear ship roll, are used to illustrate how the proposed model can be applied.

3.1 The Independent and Not-Identically Distributed Assumption

Consider a stochastic ocean wave elevation $\eta(x,t)$ represented as a Fourier series

$$\eta(x,t) = \sum_{i=1}^{N} a_i \cos(2\pi f_i t - k_i x + \phi_i) \tag{3.1}$$

where a_i, f_i and k_i are amplitudes, frequencies, and wave numbers repectively for each Fourier component. The frequencies and the wave numbers are related via the dispersion relation

$$(2\pi f_i)^2 = g k_i \tanh(k_i h) \tag{3.2}$$

where h is the finite water depth and g is the gravity constant. The phase angles ϕ_i are uniformly distributed from $-\pi$ to π. Due to the Central Limit Theorem (CLT), the resultant elevation process at an arbitrary location $\eta(x_0,t)$ is a Gaussian process

when N is sufficiently large. Since a Gaussian process is uniquely defined by its energy density spectrum, the discrete amplitudes can be determined from the discretized single-sided sea spectrum

$$a_i = \sqrt{2S(f_i)\Delta f} \tag{3.3}$$

where $S(f_i)$ is the spectral density evaluated at frequency f_i, and Δf is the frequency spacing for the discretized spectrum. The resultant wave elevation at time t_0 and location x_0, $\eta(x_0, t_0)$ then follows the zero-mean Gaussian distribution with Probability Density Function (PDF) as follows.

$$f_{\eta(x_0,t_0)}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \tag{3.4}$$

where the variance σ^2 relates to the spectrum by

$$\sigma^2 = \int_0^\infty S(f)\,\mathrm{d}f \approx \frac{1}{2}\sum_{i=1}^{N} a_i^2 \tag{3.5}$$

When the DLG method is used to generate the phase set $(\phi_1, \cdots, \phi_N)$ that leads to large elevation at $(x_0, t_0) = (0, 0)$, the Independent and Not-Identically Distributed (INID) among Fourier components is assumed. Specifically, the PDF for each phase ϕ_i random variable is parameterized by one model parameter λ_i

$$f_{\phi_i}(z) = \frac{1}{\lambda_i\sqrt{2\pi}} e^{-z^2/2\lambda_i^2} + \frac{1}{2\pi}\left[1 - \mathrm{erf}\left(\frac{\pi}{\lambda_i\sqrt{2}}\right)\right], -\pi \le z < \pi \tag{3.6}$$

where $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,\mathrm{d}t$. The phase model is a single-peak truncated Gaussian-like distribution. *Kim* (2012) showed the independently distributed assumption is incorrect. A similar experiment to verify his argument is demonstrated.

For the experiment, Figure 3.1 shows the JONSWAP discretized spectral density vs. frequency (left) and corresponding amplitudes vs. frequency (right) for the Gaussian wave. The spectrum has $N = 30$ Fourier components evaluated with uniform spacing in the range from 0.04 Hz to 0.09 Hz.

20,000 realizations are generated and the phase angle of each of the 30 components is uniformly distributed from $-\pi$ to π. The resultant elevation for each realization is evaluated using Eq. 3.1 at $(x_0, t_0) = (0, 0)$. Figure 3.2 shows the histogram of elevation normalized by the standard deviation σ calculated by Eq. 3.5. Among 20,000 realizations, 3283 of them lead to elevations that exceed one standard deviation, and they are colored in red in the histogram. The phase vectors that leads to $\eta > \sigma$

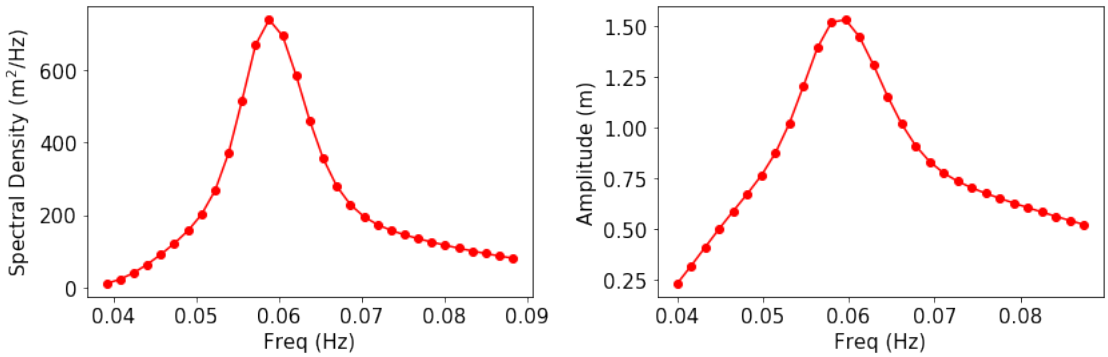

Figure 3.1: Single-sided spectrum and 30 discretized Fourier amplitudes

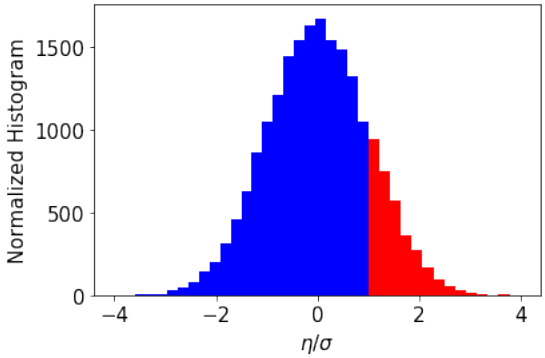

Figure 3.2: Histogram of elevation, η. 3283 out of 20,000 realizations have η that are greater than σ

are selected and visualized in Figure 3.3. The figure plots a histogram for each component in a 3D space where each histogram shows the phase angle distribution. Each 2D histogram is colored based on the corresponding amplitude a_i for that Fourier component shown in Figure 3.1. In other words, the figure presents the marginal distribution given the condition $\eta > \sigma$, which is $\Pr(\phi_i|\eta > \zeta)$.

Note that Figure 3.3 only portrays the marginal distribution and does not provide correlation information among Fourier components. Even though visualizing the joint distribution $\Pr(\phi_1, \cdots, \phi_{30}|\eta > \sigma)$ is hard, we can still discuss whether the phases are independent or not for the condition $\eta > \sigma$. In fact, *Kim* (2012) shows that the conditional phases are distributed dependently by contradiction. If the phases are distributed independently, then shuffling the phases for each Fourier component, as shown in Figure 3.4, would not affect the distribution of the elevation η. However, Figure 3.5 shows that the histogram of η from the shuffled dataset is significantly different from the original histogram of η. Therefore, the Independent and Not-Identically Distributed (INID) assumption is not suitable for the phases among

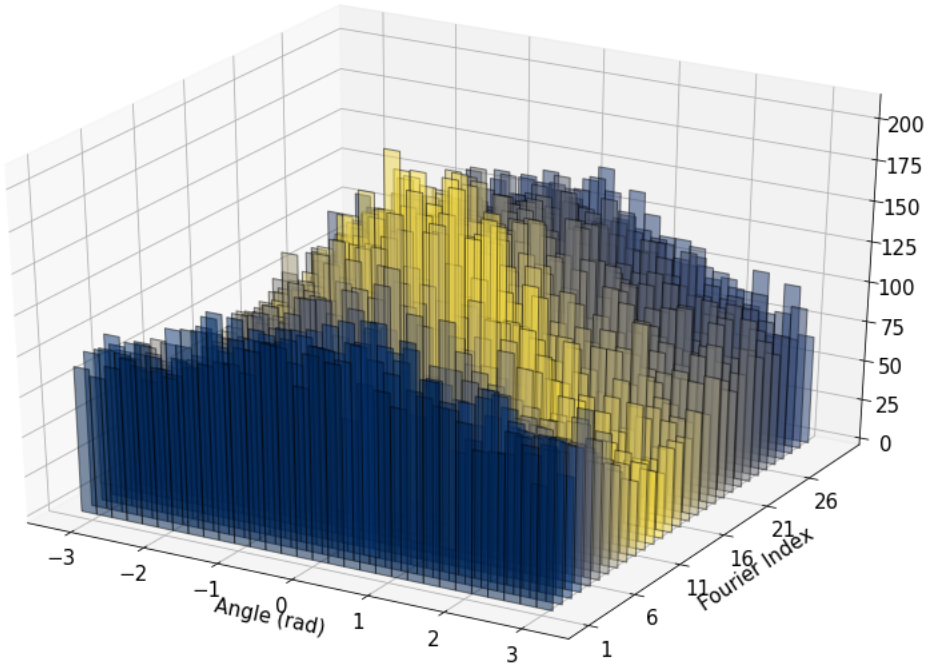

Figure 3.3: 30 2D histogram representing the marginal distribution $\Pr(\phi_i|\eta > \zeta)$, colored based on a_i (in Figure 3.1) for i-th Fourier components

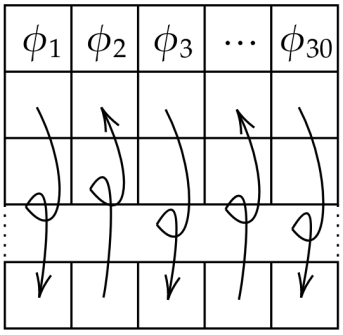

Figure 3.4: Shuffle the selected phases within the group for each Fourier component to check dependency given the condition $\eta > \sigma$

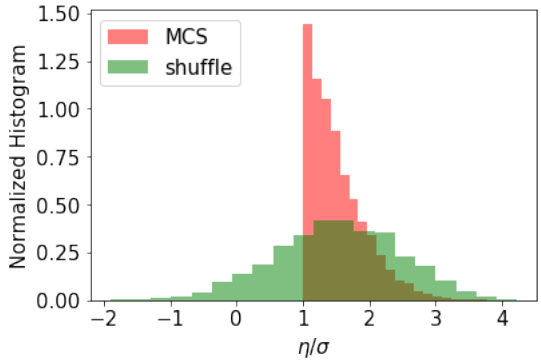

Figure 3.5: Compare the orignal histogram of the elevation ($\eta > \sigma$) and the histogram from the shuffled. Both histograms are "density" normalized so that the area is 1

different Fourier components. In other words, the phases from different Fourier indices are correlated. Thus it is more correct to sample the phase as a vector from the joint distribution, $\Pr(\phi_1, \cdots, \phi_N | \eta > \zeta)$.

Another limitation of the DLG phase model is that the distribution is symmetric at the origin. This limitation does not take into effect if the wave and the dynamical system are both linear since the distribution of the phases can always be symmetric if shifted in time and space. However, for nonlinear wave or dynamical systems, the distribution of the phases is no longer guaranteed to be symmetric. This part is demonstrated in nonlinear wave and nonlinear ship roll examples.

3.2 Methodology

3.2.1 Generative Model

The objective of the TEG as a Generative Model is to generate samples from the distribution

$$\Pr\left(\phi_1, \cdots, \phi_N | \eta_r(t_d; \phi_1, \cdots, \phi_N) > \zeta\right) \tag{3.7}$$

As mentioned in the Chapter II, supervised learning is to build a model that can predict the label y (dependent variable) based on the features X (independent variables). The process using the dataset to estimate the parameters of the model is called model training. For supervised learning, both features and labels are included in the dataset. Supervised learning tasks can be categorized from different perspectives. From the perspective of label type, regression tasks address the continuous labels, and classi-

Example	Classification
Discriminative	classify the image into "Dog" or "Cat"
Generative	generate "Dog" and "Cat" images (data augmentation)

Table 3.1: Example of Generative and Discriminative Models

Generation of images	Generation of environments
image pixels	seed vector (Fourier phases)
"Cat" category	response above threshold
"Dog" category	response below threshold

Table 3.2: Analogy between problems of generative models

fication tasks deal with the categorical labels. From the perspective of model usage, discriminative models learn the conditional probability $\Pr(y|X)$, and generative models learn the joint probability $\Pr(X, y)$. Discriminative models are used to estimate the distribution of the labels given the features, and generative models can be used to generate both features and labels following the similar distribution of the dataset. Table 3.1 compares the difference between discriminative and generative models via a simple example. In the example, a generative model can learn the pixel pattern from the given images and "create" new cat or dog images. With the similar idea used to generate ocean environments, a generative model can learn the pattern inside the environments that lead to large system response, and generate new environments that could also lead to large responses. Table 3.2 gives an analogy between generation of cat images and generation of environments that lead to large system response.

The TEG model is able to sample the environment seeds in a format of phase vector as many times as the user desires. Therefore, the TEG model is built as a generative model and the condition

$$\eta_r(t_d; \phi_1, \cdots, \phi_N) > \zeta \tag{3.8}$$

and

$$\eta_r(t_d; \phi_1, \cdots, \phi_N) \leq \zeta \tag{3.9}$$

where $\eta_r(t_d; \cdot)$ is the system response at the design time and $\phi_1, \cdots, \phi_N$ is the seed vector, can be treated as two labeling categories.

Two challenges have to be solved to train the TEG model. First, the model architecture needs to efficiently approximate and sample the joint distribution which spans a high-dimensional space with the size equal to the number of Fourier components.

The second challenge is related to the cost of generating a dataset when the ex-

ζ	$\Pr(\eta_r(t_d)) > \zeta$
0σ	0.5000
1σ	0.1587
2σ	0.02275
3σ	0.001350
4σ	3.167×10^{-5}
5σ	2.867×10^{-7}
6σ	9.866×10^{-10}
7σ	1.280×10^{-12}

Table 3.3: Probability of exceedance vs. rareness for a zero-mean Gaussian random variable

ceedance threshold value is high. A brute-force way to prepare the dataset for the model training is to generate phases independently and uniformly from $-\pi$ to π, and then select the phases that lead to threshold exceedance responses. The phases after being filtered are no longer uniformly and dependently distributed as shown in section 3.1, and they are used as the dataset to train the model. However, when the threshold value is set high, only a small set of the samples leads to an exceedance response, and most of them cannot be used for training. For a zero-mean Gaussian variable, Table 3.3 lists the exceedance probability (or 1 - Cumulative Distribution Function (CDF)) of a randomly generated phase sequence with the prescribed threshold ζ. Specifically, the probability of exceedance versus the threshold value in terms of the standard deviation of σ is listed in Table 3.3. Considering how small the probability of exceedance is for $\zeta \geq 4\sigma$, an efficient algorithm is required.

The new generative method addresses both challenges, as is shown in the remainder of this section.

3.2.2 Lauguage Model

The idea behind the TEG model architecture is taken from natural language processing. The language model in machine learning can estimate the likelihood of sequential data. For example, after training on a menu dataset, the language model can tell the probability difference between the following two phrases.

Pr("The", "apple", "and", "pair", "salad") $= 3.2 \times 10^{-13}$

Pr("The", "apple", "and", "pear", "salad") $= 5.7 \times 10^{-10}$

The language model can be built on different levels including character level and word level. The example above with a sequence of words is a word-level language model and an example of character-level models could be estimating the following probabilities.

$\Pr(\text{“a”, “p”, “p”, “l”, “e”}) = 1.6 \times 10^{-4}$

$\Pr(\text{“a”, “p”, “l”, “p”, “e”}) = 4.8 \times 10^{-7}$

Let $(x_1, \cdots, x_N)$ represent sequence data of length N. From the chain rule, the joint probability of observing the sequence can be decomposed into a product of conditional probabilities.

$$\Pr(x_1, \cdots, x_N) = \Pr(x_1) \Pr(x_2|x_1) \cdots \Pr(x_N|x_1, \cdots, x_{N-1}) \tag{3.10}$$

where $\Pr(x_n|x_1, \cdots, x_{n-1})$ is the conditional probability of x_n given the history of the generated sequence. Therefore, successfuly learning $\Pr(x_n|x_1, \cdots, x_{n-1})$ can not only estimate the joint probability but also provide the sampling method based on the generation history.

Since the features are sequential data, and the order of each feature in the sequence matters to the prediction, a neural net from the RNN architecture, called LSTM, is used to take the generated sequence as input and the distribution of the next element as the output. In Figure 2.12, a “many-to-one” RNN architecture is selected to approximate the mapping from the generated sequence to the probability vector for the next “to-be-sampled” element. Once the conditional probability mapping is learned by the model, it can generate a feature sequence of indefinite length. The sequence is generated by recursively sampling the next element from the predicted distribution $\Pr(x_n|x_1, \cdots, x_{n-1})$.

In the following subsections, the similar idea of building language models to generate samples from the joint distribution is introduced and customized to generate the correlated phase set step by step.

3.2.3 Data Preparation

Let Φ be the raw phase dataset sampled from the distribution of threshold exceedance phases in Eq. 3.7. Eq. 3.11 shows the dataset matrix Φ with shape (M, N) after being filtered by the threshold $\eta_r(t_d; \phi_1, \cdots, \phi_N) > \zeta$, where M is the number of realizations and N is the number of Fourier components. Each row of the matrix is one realization or sample and the column of the matrix corresponds to the corresponding Fourier indices.

$$\Phi = \begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1N} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{M1} & \phi_{M2} & \cdots & \phi_{MN} \end{bmatrix} \tag{3.11}$$

For the word-level text generation problem, the model outputs the probability vector for the next word in the dictionary. However, for the phase generation scenario, the phase angle is a continuous variable in the interval $[-\pi, \pi)$. Parameterization of the PDF on the continuous interval is required to format the model output. Moreover, successful sampling based on the distribution output is also required. The most (if not the most) straightforward parameterization on a bounded interval can be a discrete histogram. Hence, all continuous phase angles are digitized by a uniform bin grid. Let D represent the digitized phase angles.

$$D = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1N} \\ d_{21} & d_{22} & \cdots & d_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ d_{M1} & d_{M2} & \cdots & d_{MN} \end{bmatrix} \tag{3.12}$$

where $d_{ij} = \left\lfloor \frac{\phi_{ij} - (-\pi)}{2\pi/K} \right\rfloor + 1$ and K is the number of bins to digitize the angle range $[-\pi, \pi)$. As a result, the element in the dataset D is then integer-encoded, $d_{ij} \in \{1, 2, \cdots, K\}$. For a supervised learning task that approximates the mapping from indepedent variables X to depedent variables y, its training data naturally consists of data pairs (feature, label), where each pair stands for one data point, called a sample or record. To prepare the dataset for the TEG model, a "sliding-window" method is used to convert each row of D to multiple history records in X and y. Each row in D produces N records in the training dataset X and y and the feature matrix X is pre-padded with the dummy value 0. The feature sequence matrix $X = [x_{ij}]$ has shape of $(MN, N-1)$ and $x_{ij} \in \{0, 1, \cdots, K\}$. The label matrix $y = [y_i]$ has shape of $(MN, 1)$ and $y_i \in \{1, \cdots, K\}$.

INTEGER-ENCODING	ONE-HOT-ENCODING
0	$[1, 0, 0, \cdots, 0]$
1	$[0, 1, 0, \cdots, 0]$
2	$[0, 0, 1, \cdots, 0]$
$\vdots$	$\vdots$
K	$[0, 0, 0, \cdots, 1]$

Table 3.4: Convert Integer-Encoded to One-Hot-Encoded

$$X = \begin{bmatrix} 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 & d_{11} \\ 0 & \cdots & 0 & 0 & d_{11} & d_{12} \\ 0 & \cdots & 0 & d_{11} & d_{12} & d_{13} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ d_{11} & \cdots & d_{1,N-4} & d_{1,N-3} & d_{1,N-2} & d_{1,N-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}, y = \begin{bmatrix} d_{11} \\ d_{12} \\ d_{13} \\ d_{14} \\ \vdots \\ d_{1,N} \\ \vdots \end{bmatrix} \tag{3.13}$$

For each row of X, the model is expected to produce a probability vector

$$\hat{y} = [\hat{y}_0, \cdots, \hat{y}_k, \cdots, \hat{y}_K] \tag{3.14}$$

like a histogram for the next discrete phase. Each floating number in the vector $\hat{y}_k$ then represents the likelihood for the next discrete phase being k. Obviously, all floating numbers in $\hat{y}$ have properties: 1) $\hat{y}_k \geq 0$; and 2) $\sum_{k=0}^{K} \hat{y}_k = 1$. For a well-trained model, the predicted probability vector $\hat{y}$ should have large values at indices that correspond to the digitized phases that are observed in y.

Since the dummy value is set to zero and it does not infer any relation with other angles, the integer-encoded matrix X and y are suggested to convert to One-Hot-Encoded as illustrated in Table 3.4 and Figure 3.6. Therefore, the dataset consists of a One-Hot-Encoded matrix X with shape $(MN, N-1, K+1)$ and a One-Hot-Encoded vector y with shape $(MN, K+1)$.

Before sending the one-hot-encoded X and y to the model training process, the dataset has to be split by rows into a training dataset $(X_{\text{train}}, y_{\text{train}})$ and a validation dataset $(X_{\text{val}}, y_{\text{val}})$. The training process iteratively decreases the loss on the training dataset and stops when the performance on the validation dataset worsens. This training strategy is called *early stopping* as mentioned in Chapter II. For more details,

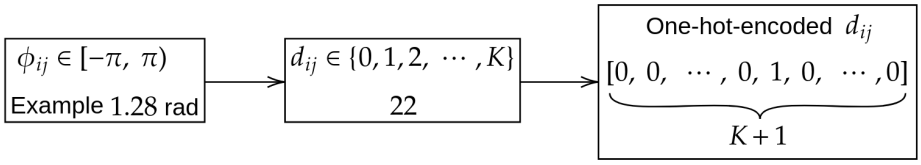

Figure 3.6: One-hot-encoding process to convert a continuous phase angle to a binary vector of length $K + 1$

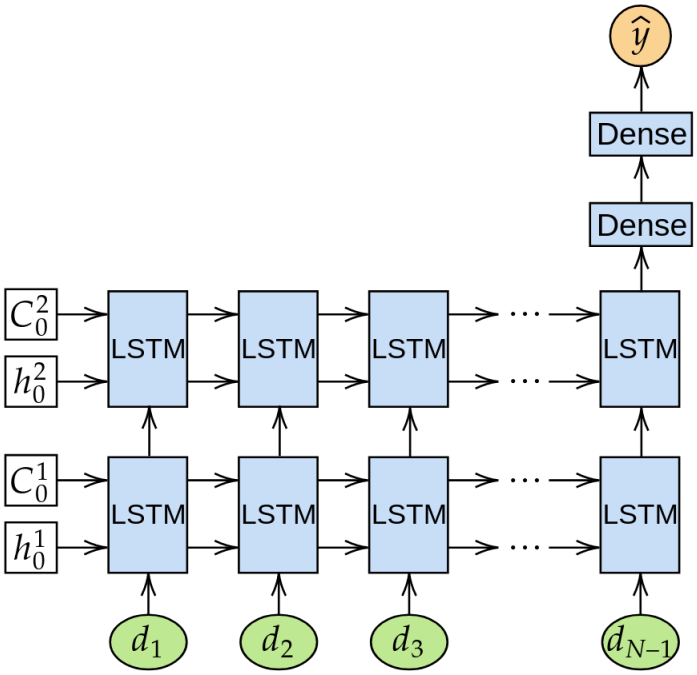

Figure 3.7: TEG Architecture

refer to the subsection 3.2.5. A common split ratio for training and validation is listed in Table 2.3.

3.2.4 Model Architecture

To recognize the complex nonlinear mapping from the generation history to the probability vector, multi-layer LSTMs together with multiple dense layers are designed. Figure 3.7 shows the concrete architecture of the TEG model.

The very bottom layer is the first LSTM layer that takes the generation history as the feature sequence of length $N - 1$, where N is the number of Fourier components. This LSTM layer produces an output sequence with the same length of $N - 1$ and each output element in the sequence is a vector with the same size of the hidden units.

Layer (type)	Output Shape
lstm_1 (LSTM)	$(N-1, H)$
lstm_2 (LSTM)	$(H,)$
dense_1 (Dense)	$(D1,)$
dense_2 (Dense)	$(D2,)$
dense_3 (Dense)	$(K+1,)$

Table 3.5: Model architecture and output tensor shape

The output sequence from the first LSTM layer is taken as an input sequence for the second LSTM layer. The hidden state vector at the very last element in the sequence is then connected to multi-stage Dense (a.k.a Fully-Connected) layers similar to the vanilla layer of the shallow nets shown in Figure 2.7.

The output vector produced by the last Dense layer is then converted to a probability vector of length $K+1$ via the following *softmax* function

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{k=0}^{K} e^{z_k}} \tag{3.15}$$

where $\boldsymbol{z} = (z_0, \cdots, z_K) \in \mathbb{R}^{K+1}$ is the output vector produced by the last Dense layer. One property of the *softmax* function is it converts a real-valued vector into a probability vector $\hat{y}$ with properties: 1) $\hat{y}_k \geq 0$; and 2) $\sum_{k=0}^{K} \hat{y}_k = 1$.

Table 3.5 lists the output shape after each layer in the model, where N is the number of Fourier components, H is the number of cell states or hidden states in the LSTM layers.

3.2.5 Model Training

Training a model is an optimization problem on model parameters such that the loss (or cost) objective is minimized. As reviewed in Chapter II, forward propagation calculates in the direction from the input side to the output side with the current model parameters. Once the output (or prediction) is available, the loss is then calculated by a loss (or cost) function between the prediction and the ground-truth label. Once the loss is evaluated, backpropagation calculates the derivatives of the loss with respect to the model parameters from the output side to the input side. The derivatives calculated in the backpropagation step are then used by the optimization algorithm to update the model parameters to decrease the loss.

For the TEG model, the input is the generated sequence of length $N-1$, where N is the number of Fourier components and the output is a vector representing the estimated probabilities for the next element to be generated. The loss function

between the probability vector and the label in the dataset is defined by a categorical cross-entropy function

$$L(\hat{y}, y) = -\sum_{k=0}^{K} y_k \log(\hat{y}_k) \tag{3.16}$$

where $y = [y_0, y_1, \cdots, y_K]$ is the one-hot-encoded label for one record in the training dataset, and $\hat{y} = [\hat{y}_0, \hat{y}_1, \cdots, \hat{y}_K]$ is the model output in a form of a probability vector. A low probability at the index of the label therefore penalizes the prediction, leading to a large loss.

The process going through forward propagation, loss calculation, backpropagation, and model parameters update is called one iteration. For each iteration, a subset of the whole dataset called a batch is passed through. Too small of a batch size makes the training loss decrease non-smoothly and loses the performance speedup from vectorization. Too large of a batch size takes a long time for calculation of one iteration and utilizes more memory. The typical batch size for the TEG model is from 10^2 to 10^3. One epoch is when the whole dataset is passed through one time. Therefore, the number of iterations for one epoch of training data is calculated as follows.

$$\text{number of iterations/epoch} = \frac{\text{number of records in the training dataset}}{\text{batch size}} \tag{3.17}$$

The training process may need multiple epochs depending on the size of the training data and the batch size to see the loss convergence or reach the early stopping condition. As mentioned earlier, the early stopping condition is often set when the performance on the validation dataset becomes worse (in other words, the validation loss increases). A safer condition is having a patience parameter p to stop training when the performance on the validation dataset has not improved for p epochs.

The model parameters are updated once the derivatives of loss with respect to these parameters are calculated. The general form of the gradient descent algorithms is

$$w \leftarrow w - \alpha f_w(\frac{\partial L}{\partial w}) \tag{3.18}$$

$$b \leftarrow b - \alpha f_b(\frac{\partial L}{\partial b}) \tag{3.19}$$

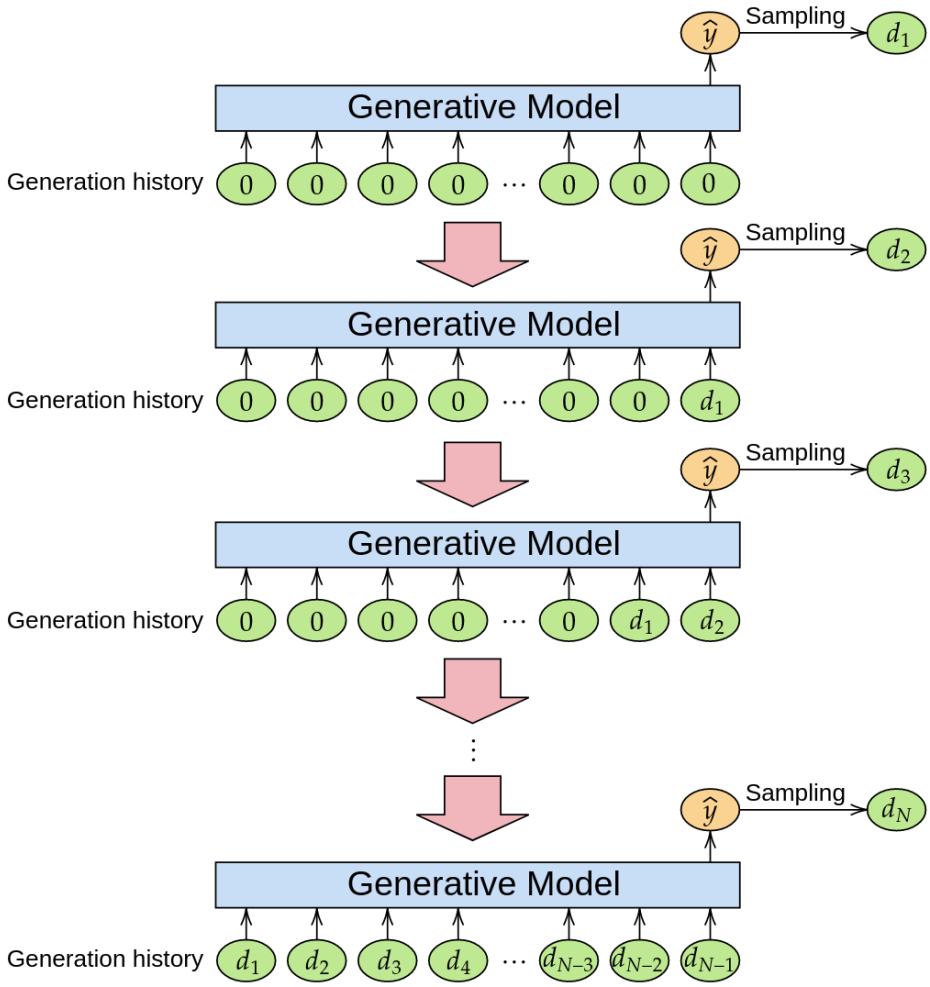

Figure 3.8: Sample the TEG model for one phase vector $(d_1, d_2, d_3, \cdots, d_N)$

where w and b are weights and biases parameters respectively, $\frac{\partial L}{\partial w}$ and $\frac{\partial L}{\partial b}$ are the derivatives of the loss with respect to weights and biases parameters respectively, and α is called the learning rate. Different optimization algorithms use different functions f_w and f_b, and the discussion and comparison on the performance of the algorithms are out of scope for this dissertation. The proposed models use the *Adam* algorithm (*Kingma and Ba*, 2014) which is known for its fast and smooth convergence behavior.

3.2.6 Model Sampling

Once the model is trained, new samples in a format of a phase sequence can be generated. The sampling process is similar to the preparation of sequential training data using a sliding window. Recall that the model is able to produce the distribution of the next element given the generation history of length $N-1$, where N is the

number of Fourier components. To sample one sequence, the generation history is first initialized with all zeros. Therefore the model produces a probability vector of length $K+1$, representing the probability of the "to-be-generated" discrete phase. Then the next discrete phase is randomly sampled according to the produced probability vector. The sampled discrete phase is then added to the tail of the generation history for the model to sample the next phase. Once the phase sequence of length N is sampled, the sampled sequence is finally returned to the user and the generation history is reset with all zeros for the next sequence sampling. This process is illustrated in Figure 3.8.

3.2.7 Bootstrapping

By using the TEG model, the phases from different Fourier components can be generated with their correlation similar to the given dataset. However, to train the TEG model, the data for a high failure threshold is extremely limited as described by the small probability of exceedance $\Pr(\eta_r > \zeta)$. For example, one 4σ failure threshold situation of a zero-mean Gaussian process requires conducting $1/(3.156 \times 10^{-5}) = 31576$ MCS, not to mention that many situations are required to train the generative model. Therefore, most of the phase sequences from the MCS lead to a response smaller than the user-defined threshold and therefore are filtered out of the training dataset for the TEG model resulting in a very high cost to obtain training data.

To address this lack-of-data challenge, an iterative bootstrapping strategy is proposed. The idea is to train a series of models at different failure thresholds, and the latter model is trained with filtered samples generated from the previous model. This bootstrapping process is illustrated in Figure 3.9. Let $\mathcal{D}_0$ be the original phase dataset. The resultant responses at the design time are then used to rank each phase sequence in $\mathcal{D}_0$ and the top $1/q$ phase sequences form the dataset $\mathcal{D}_1$, where q is defined as the bootstrapping stepsize. Then the phase dataset $\mathcal{D}_1$ is used to train a TEG model and new samples from the trained model are denoted by $\mathcal{D}'_1$. The phase sequences of the bootstrapped dataset $\mathcal{D}'_1$ are then ranked by running MCS, and again the top $1/q$ phase sequences form the dataset $\mathcal{D}_2$. This procedure can be conducted multiple times until the quantile boundary of the last dataset $\mathcal{D}'_B$ is close to the user-defined failure threshold. Finally, the phase sequences of $\mathcal{D}'_B$ with responses larger than the threshold are then returned to the user. Since the bootstrapped dataset $\mathcal{D}'_i$ is generated by the model which is trained on the dataset $\mathcal{D}_i$, they are expected to share a similar distribution of phase sequences if the model is trained properly.

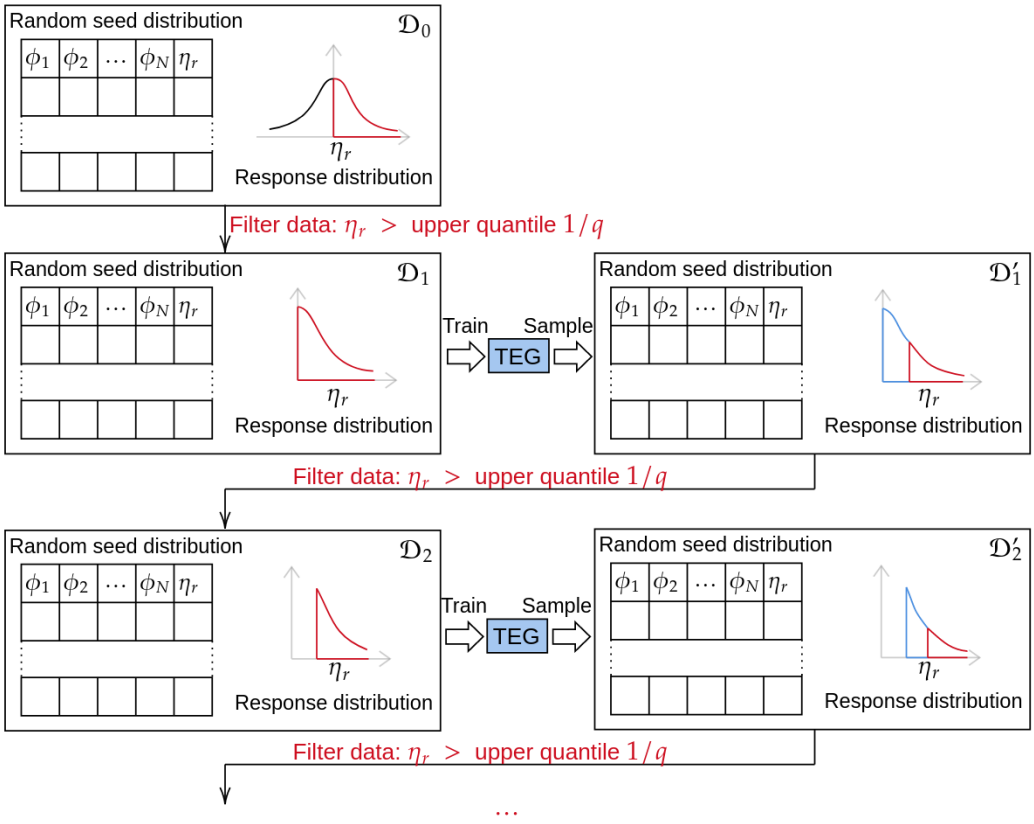

Figure 3.9: Dataset bootstrapping

$$\Pr_{\mathcal{D}_i}(\phi_1, \cdots, \phi_N) \approx \Pr_{\mathcal{D}'_i}(\phi_1, \cdots, \phi_N), i \geq 1 \tag{3.20}$$

Moreover, since the dataset $\mathcal{D}_{i+1}$ is the top $1/q$ of the dataset $\mathcal{D}'_i$ measured by the design-time response, its dataset size is also expected to be $1/q$ of the dataset size of $\mathcal{D}'_i$. To maintain the same dataset size between $\mathcal{D}_i$ and $\mathcal{D}_{i+1}$ for training usage, the model should generate $q|\mathcal{D}_i|$ new samples as the bootstrapped dataset $\mathcal{D}'_i$.

$$|\mathcal{D}_{i+1}| = |\mathcal{D}'_i|/q, i \geq 1 \tag{3.21}$$

A theoretical analysis for a random process can provide the relation between the number of bootstrapping iterations and the lower bound of the response for the bootstrapped dataset. For a random process with the filter quantile q, the theoretical lower bound ζ_i for dataset $\mathcal{D}_i$ can be calculated via the CDF.

$$F(\zeta_i) = 1 - \frac{1}{q^i} \tag{3.22}$$

The number of iterations needed for bootstrapping relates to the failure threshold by

$$B = \left\lfloor -\log_q[1 - F(\zeta)] \right\rfloor \tag{3.23}$$

For the Gaussian CDF, $F(\cdot)$, is expressed in terms of the error function erf$(\cdot)$

$$F(x) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right] \tag{3.24}$$

where μ, and σ are the mean and the standard deviation of the Gaussian process respectively. Table 3.6 lists the lower bound ζ_i change in each iteration for $q = 2$ and $q = 3$ cases.

3.3 Computation Complexity

In this section, the cost of the proposed TEG model is calculated and compared against the brute-force MCS for the response process. Let N_d be the desired number of realizations associated with the failure threshold ζ. In other words, the user requests N_d phase sequences for the large response environment setup. Therefore, for the brute-force MCS, the total number of realizations N_t is calculated by

ITERATION	$q = 2$	$q = 3$
1	0.0	0.43
2	0.67	1.22
3	1.15	1.79
4	1.53	2.25
5	1.86	2.64
6	2.15	3.0
7	2.42	3.32
8	2.66	3.61
9	2.89	3.89
10	3.1	4.15
11	3.3	4.39
12	3.49	4.62
13	3.67	4.85
14	3.84	5.06
15	4.01	5.27
16	4.17	5.46
17	4.32	5.66
18	4.48	5.84
19	4.62	6.02
20	4.76	6.2

Table 3.6: $\frac{\zeta_i - \mu}{\sigma}$ VS. bootstrapping iteration

$$\frac{N_d}{N_t} = 1 - F(\zeta) \tag{3.25}$$

where $F(\cdot)$ is the CDF for the process. Therefore the total cost for the brute-force MCS denoted by C_1 is

$$C_1 = N_t C = \frac{N_d C}{1 - F(\zeta)} \tag{3.26}$$

where C is the simulation cost for one realization.

For the proposed TEG model, let N_{tr} be the number of realizations needed to successfully train the TEG model. Hence, preparation of $\mathcal{D}_1$ requires qN_{tr} realizations, where q is the bootstrapping constant. Let C_{tr} and C_{sp} be the training cost and the sampling cost to prepare the dataset $\mathcal{D}_1'$, where $|\mathcal{D}_1'| = qN_{tr}$. To rank the phase sequences in $\mathcal{D}_1'$ and filter the sequences to form $\mathcal{D}_2$, again qN_{tr} MCS are conducted. Therefore the cost for one bootstrapping step is as follows.

$$C_B = qN_{tr}C + C_{tr} + C_{sp} \tag{3.27}$$

After B bootstrapping steps, where B is calculated in Eq 3.23, the TEG model

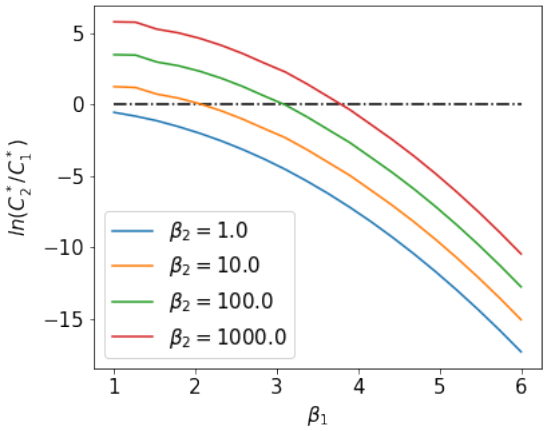

Figure 3.10: Cost comparison between C_1^* and C_2^* (Gaussian, $q = 2$)

is able to generate phase sequences leading to responses larger than ζ_B, listed in Table 3.6. Hence, the expected number of sequence N_s to achieve the desired realizations is

$$\frac{N_d}{N_s} = \frac{1 - F(\zeta)}{1 - F(\zeta_B)} = q^B[1 - F(\zeta)] \tag{3.28}$$

Therefore, the total cost for the proposed bootstrapping TEG method is

$$C_2 = BC_B + N_s C = B(qN_{tr}C + C_{tr} + C_{sp}) + \frac{1}{q^B}\frac{N_d C}{1 - F(\zeta)} \tag{3.29}$$

Define $\beta_1 = \frac{\zeta - \mu}{\sigma}$ and $\beta_2 = \frac{qN_{tr}C + C_{tr} + C_{sp}}{N_d C}$. For a Gaussian process, the non-dimensional cost C_1^* is calculated by

$$C_1^* = \frac{C_1}{N_d C} = \frac{1}{1 - F_{\mathcal{N}}(\beta_1)} \tag{3.30}$$

where $F_{\mathcal{N}}(x) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$ is the marginal CDF for the standardized Gaussian process. The non-dimensional cost C_2^* is calculated by

$$C_2^* = \frac{C_2}{N_d C} = B\beta_2 + \frac{1}{q^B}\frac{1}{1 - F_{\mathcal{N}}(\beta_1)} \tag{3.31}$$

where $B = \left\lfloor -\log_q[1 - F_{\mathcal{N}}(\beta_1)] \right\rfloor$.

Figure 3.10 and Figure 3.11 plot the cost ratio in logarithmic versus non-dimensional parameters β_1 and β_2 for cases $q = 2$ and $q = 3$.

As shown in Figure 3.10 and Figure 3.11, the cost ratio of applying the proposed method compared to the brute-force MCS reduces at least exponentially as

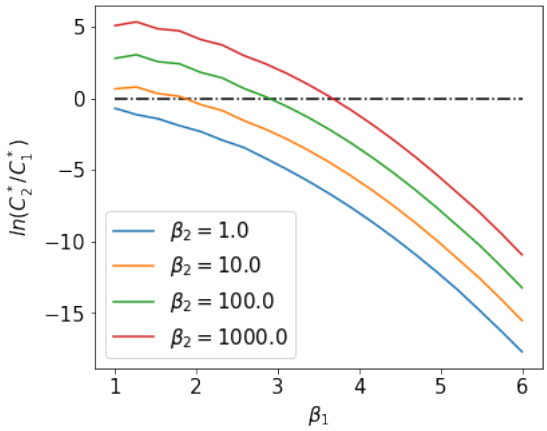

Figure 3.11: Cost comparison between C_1^* and C_2^* (Gaussian, $q = 3$)

the threshold increases. Depending on how large the threshold is set β_1 and how expensive to train and sample the TEG model β_2, users might decide whether to use the proposed method. For a high threshold, for example $\beta_1 = \frac{\zeta - \mu}{\sigma} > 4$, the proposed method can significantly reduce the cost and therefore is suggested to use. However, if the threshold is relatively low and much more training data required for the TEG model compared to the requested number of realizations, the proposed method is more expensive compared to the brute-force method and therefore is not suggested to use. The size of the training dataset is experimented and discussed in Section 3.4. Another observation from the comparison between Figure 3.10 and Figure 3.11 is the bootstrapping constant q plays theoretically little role in affecting the cost reduction. However, a proper value of q is suggested in later sections based on the tests for different example problems.

In the following section, examples including linear waves, nonlinear waves, and a relatively complex ship roll motion are used to illustrate the proposed method, and to compare to MCS results.

3.4 Example: Linear Wave

In this section, the TEG model is applied to linear waves. Specifically, the method is used to generate phase sequences that lead to elevations exceeding a threshold. The linear wave elevation is evaluated at the origin location and zero time. In other words, the method is expected to sample from the distribution

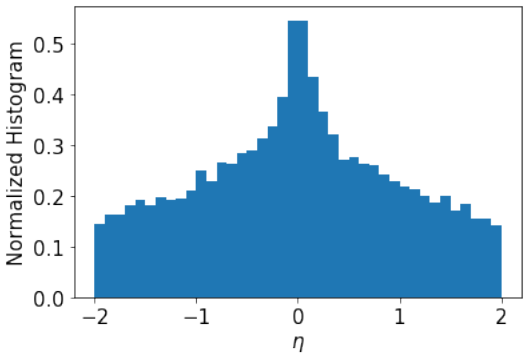

Figure 3.12: Elevation histogram from MCS ($N_{\text{fourier}} = 2$)

$$\Pr\left(\phi_1, \cdots, \phi_N \middle| \eta = \sum_{i=1}^{N} a_i \cos\phi_i > \zeta\right) \tag{3.32}$$

where $\phi_1, \cdots, \phi_N$ are the phase random variables that are independently and uniformly distributed from $-\pi$ to π, a_i is the wave amplitude for the ith Fourier component and ζ is the user-defined threshold. Though the phases are independently and uniformly distributed globally, the threshold-posterior phases are correlated and follow the conditional joint distribution, $\Pr(\cdot|\eta > \zeta)$ as discussed in section 3.1.

3.4.1 Bichromatic Wave

As a beginning of linear wave analysis, the wave is formulated by 2 Fourier components with equal amplitudes and the threshold is set to be zero. Therefore, the objective is to verify that the TEG model can learn and sample from the following distribution.

$$\Pr(\phi_1, \phi_2 | \cos\phi_1 + \cos\phi_2 > 0) \tag{3.33}$$

For the data preparation, the (ϕ_1, ϕ_2) pair is first independently sampled uniformly from 2D space $[-\pi, \pi) \times [-\pi, \pi)$. Then the Fourier sum $\cos\phi_1 + \cos\phi_2$ is evaluated for each phase pair. The pairs resulting in positive Fourier sum (or elevation) are collected as phase vectors of length two in the dataset. Since the number of Fourier components is two, which is much smaller for the Central Limit Theorem to be valid, the resultant elevation is non-Gaussian distributed from boundaries -2 to 2. Figure 3.12 plots the histogram of the resultant elevation (the histogram is non-dimensionalized so that the area of the histogram is one).

The TEG is then trained on the corresponding phase pairs from the positive half

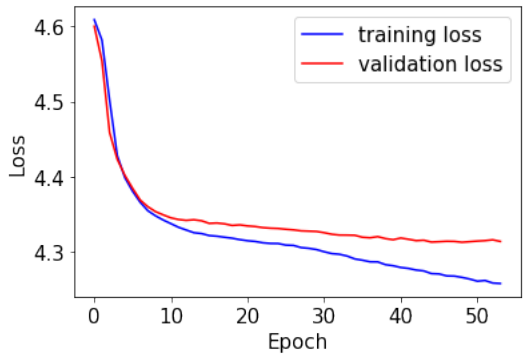

Figure 3.13: Training history ($N_{\text{fourier}} = 2$)

of the histogram. As mentioned in the subsection 3.2.5, the dataset is split into a training dataset and a validation dataset. To visualize the training history, the performance of the model on both datasets is plotted against the number of epochs in Figure 3.13.

After the model is trained, it is able to generate samples in the format of phase pairs that follow the distribution of the provided dataset. Figure 3.14 compares the marginal distribution for each phase component, which are $\Pr(\phi_1 | \cos\phi_1 + \cos\phi_2 > 0)$ and $\Pr(\phi_2 | \cos\phi_1 + \cos\phi_2 > 0)$ between the given dataset (top) and the generated dataset (bottom). As shown in the figure, by learning the joint distribution, the model can generate new samples which also share similar marginal distributions of the given dataset.

Figure 3.15 compares the distribution of resultant elevations between the given dataset (red) and the generated dataset (blue). Though the given dataset has no negative elevation, the generated dataset does have negative resultant elevations. In general, the agreement is good, except near the threshold boundary. This mismatch at the threshold boundary of the given dataset also appears in later examples. One reason is that learning the discontinuity in the probability space is hard for the model. However, this mismatch may not be an issue as long as the right tail matches properly between the given dataset and the generated dataset since the post-filtering of the generated dataset can help the situation.

Figure 3.16 compares the joint distribution of the phases between the given dataset (left) and the generated dataset (right). The mismatch in the Figure 3.15 becomes the blurry boundaries around the edges of the distribution in the Figure 3.16. As shown in the right subplot, the model is able to generate pairs with the majority uniformly distributed inside the square formed by $(\pi, 0), (0, \pi), (-\pi, 0), (0, -\pi)$. Figure 3.17 visualizes the joint distribution in 3-dimensional space by plotting $(\phi_1, \phi_2, \cos\phi_1 +$

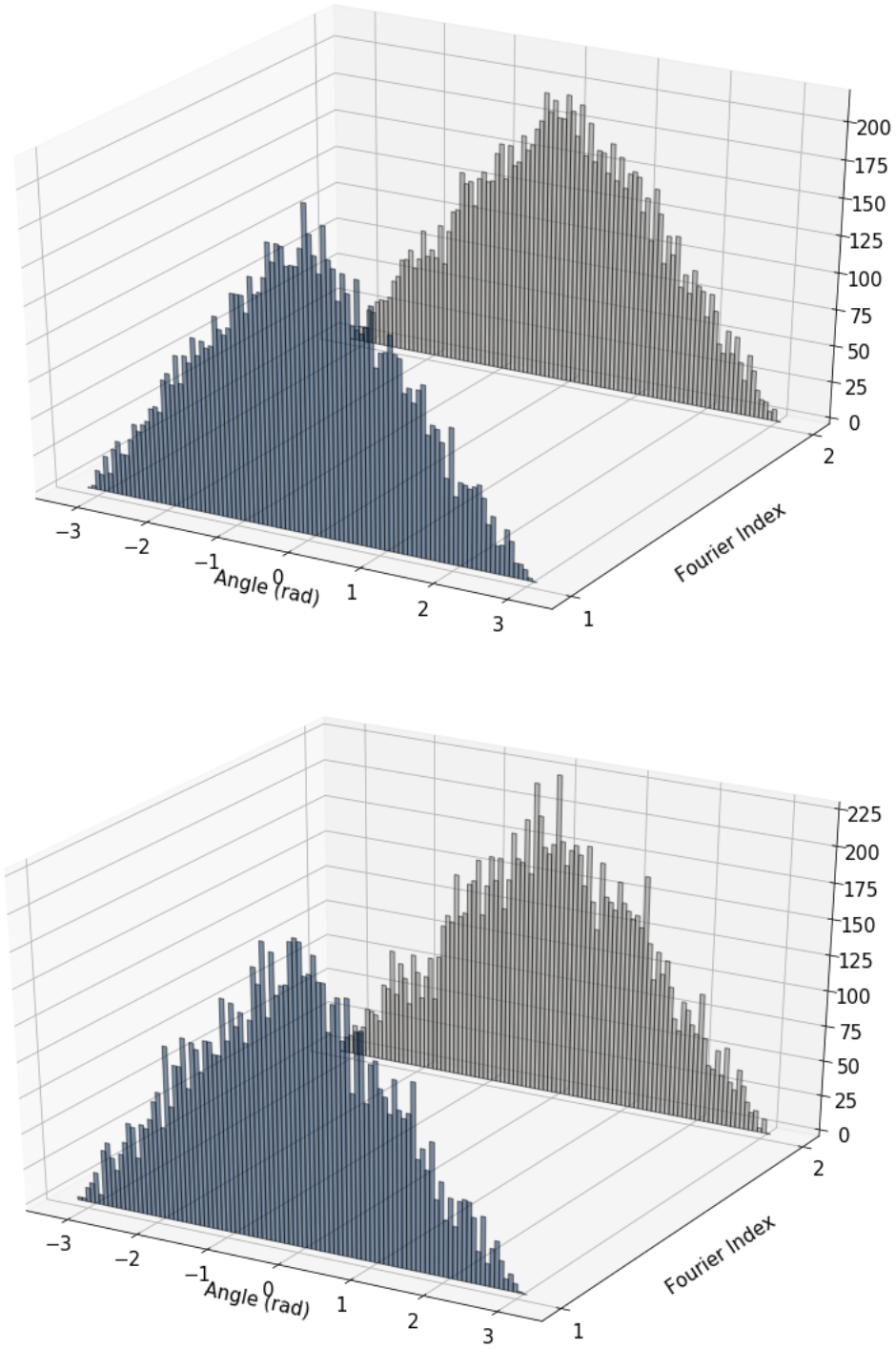

Figure 3.14: Compare the histograms of ϕ_1 and ϕ_2 between the given dataset (top) and the generated dataset (bottom)

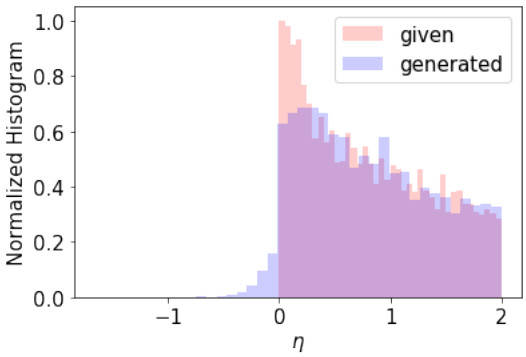

Figure 3.15: Comparison of the histograms of $\cos\phi_1 + \cos\phi_2$ between the given dataset (red) and the generated dataset (blue)

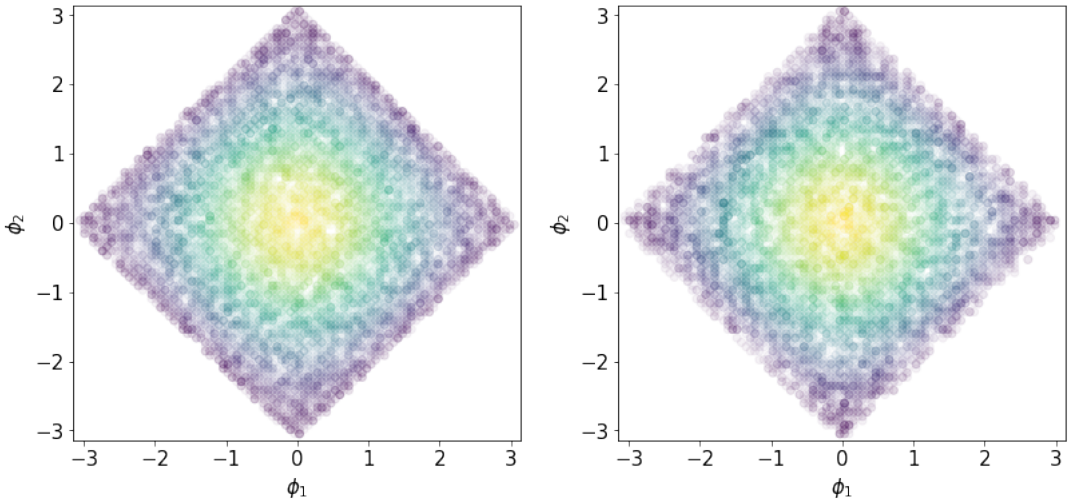

Figure 3.16: Comparison of the joint distribution (ϕ_1, ϕ_2) between the given dataset (left) and the generated dataset (right)

$\cos\phi_2$). As the results show, the TEG model succeeds in learning and sampling the phase pair from the joint distribution. A linear wave consisting of 5 Fourier components is analyzed in the next example, where the distributions of every 2-phase pairs are compared.

3.4.2 Five Component Wave

In this example, a linear wave consisting of five Fourier components is considered. The wave spectrum is a sea-state 8 JONSWAP with a significant wave height H_s of 15 m and the peak period T_p of 17 s. The lower and upper cut-off frequencies are $f_p/1.5$ and $1.5f_p$ respectively, where $f_p = 1/T_p$ is the frequency corresponding to the peak period. The spectral density values are calculated as follows.

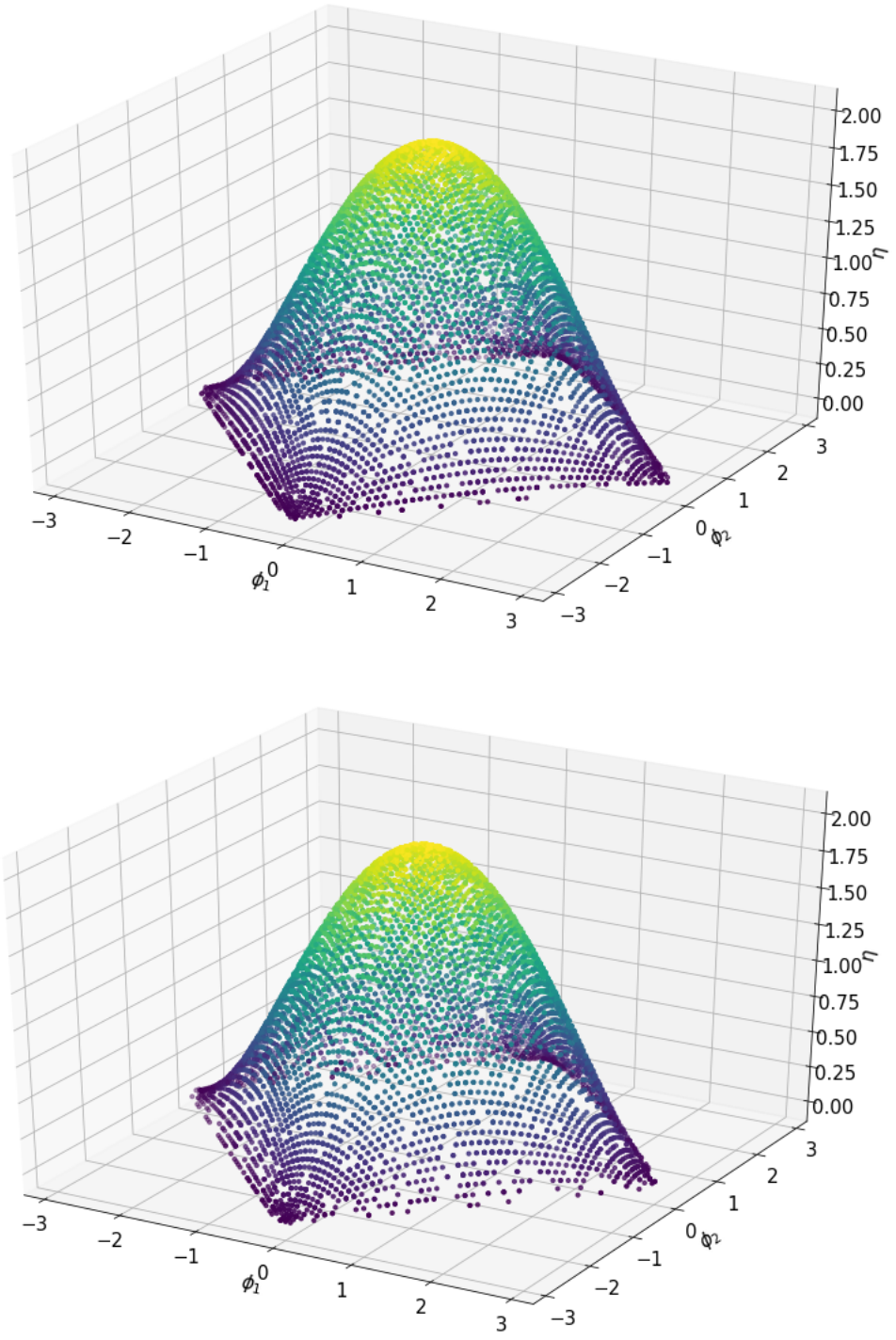

Figure 3.17: Comparison of the joint distribution in 3D $(\phi_1, \phi_2, \cos\phi_1 + \cos\phi_2)$ between the given dataset (top) and the generated dataset(bottom)

$$\alpha = \frac{0.0624}{0.23 + 0.0336\gamma - 0.185/(1.9 + \gamma)} \tag{3.34}$$

$$\sigma = \begin{cases} 0.07 & f < f_p \\ 0.09 & \text{otherwise} \end{cases} \tag{3.35}$$

$$\beta = \exp\left[-\frac{(f - f_p)^2}{2\sigma^2 f_p^2}\right] \tag{3.36}$$

$$S(f) = \frac{\alpha H_s^2 f_p^4}{f^5}\gamma^\beta \exp\left[-\frac{5}{4}\left(\frac{f_p}{f}\right)^4\right] \tag{3.37}$$

Figure 3.18 shows the discritized single-sided spectrum (left) and 5 Fourier amplitudes (right) evaluated on the discrete spectrum. The objective of this example is to test whether the TEG model is able to learn and sample phase sequences $(\phi_1, \phi_2, \phi_3, \phi_4, \phi_5)$ from the distribution

$$\Pr\left(\phi_1, \phi_2, \phi_3, \phi_4, \phi_5 \middle| \sum_{i=1}^{5} a_i \cos\phi_i > 0\right) \tag{3.38}$$

The data is prepared in a similar way of the previous two-component case by running the MCS. First sample the phases independently and uniformly from $-\pi$ to π. Then the resultant elevations are evaluated by $\sum_{i=1}^{5} a_i \cos\phi_i$. Finally the phase sequence

$$(\phi_1, \phi_2, \phi_3, \phi_4, \phi_5) \tag{3.39}$$

is selected to form the given dataset if the corresponding elevation is positive. Figure 3.19 plots the histogram of the resultant elevation and the histogram is non-dimensionalized so that the area of the histogram is one. The TEG model is trained on the given dataset and sampled to form the generated dataset. The training history is plotted in Figure 3.20. Figure 3.22 compares the marginal distribution for each phase component, which is

$$\Pr(\phi_i | \sum_{i=1}^{5} a_i \cos\phi_i > 0), i = 1, 2, 3, 4, 5 \tag{3.40}$$

between the given dataset (top) and the generated dataset (bottom). Similar to the two component case, the model can generate new samples which also share similar marginal distributions of the given dataset.

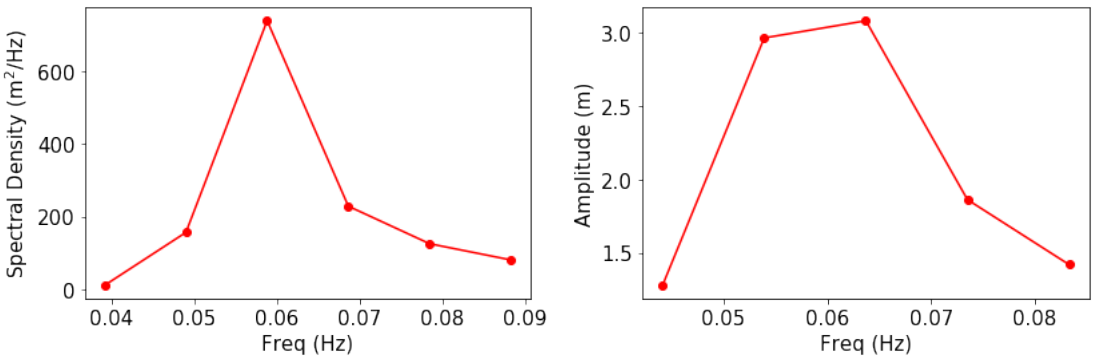

Figure 3.18: Single-sided spectrum and discretized Fourier amplitudes

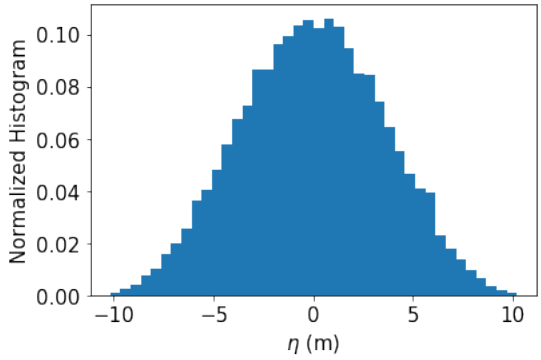

Figure 3.19: Elevation histogram from MCS ($N_{\text{fourier}} = 5$)

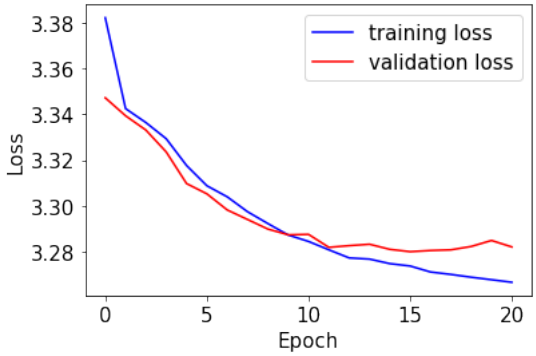

Figure 3.20: Training history ($N_{\text{fourier}} = 5$)

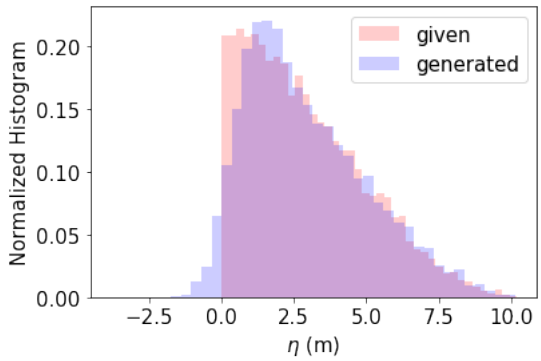

Figure 3.21: Compare the histograms of $\sum_{i=1}^{5} a_i \cos\phi_i$ between the given dataset (red) and the generated dataset (blue)

Figure 3.21 compares the histograms of the resultant elevation between the given dataset (red) and the generated dataset (blue). The mismatch at the threshold still appears in this case and the reasons are discussed in the previous example. The tail distribution matches between the two datasets.

Since visualizing the joint distribution in 5D space is impossible, instead, the pairwise joint distribution is compared in Figure 3.23. The left (red) 5×5 figure displays the (ϕ_i, ϕ_j) joint distribution given positive elevation for the given dataset at the (i, j) grid and the right (blue) 5×5 figure displays the conditional joint distribution for the generated dataset. By comparing the subplot at (i, j) between the given dataset and the generated dataset, it is verified that the model can learn and sample the correct pairwise joint distribution. Since the second and the third amplitudes a_2 and a_3 are the top 2 high-energy Fourier components, their correlation matters most and is successfully captured by the trained model by comparing sub-figures at grid (2,3).

3.4.3 Thirty Component Wave

This example extends the number of Fourier components to 30. Since 30 is large enough for the CLT to be valid, the resultant Fourier sum follows the Gaussian distribution with mean value $\mu = 0$ and the standard deviation σ calculated by Eq 3.5.

The objective is to verify that the model can learn and sample from the distribution

$$\Pr\left(\phi_1, \cdots, \phi_{30} \middle| \eta = \sum_{i=1}^{30} \cos\phi_i > 3\sigma\right) \tag{3.41}$$

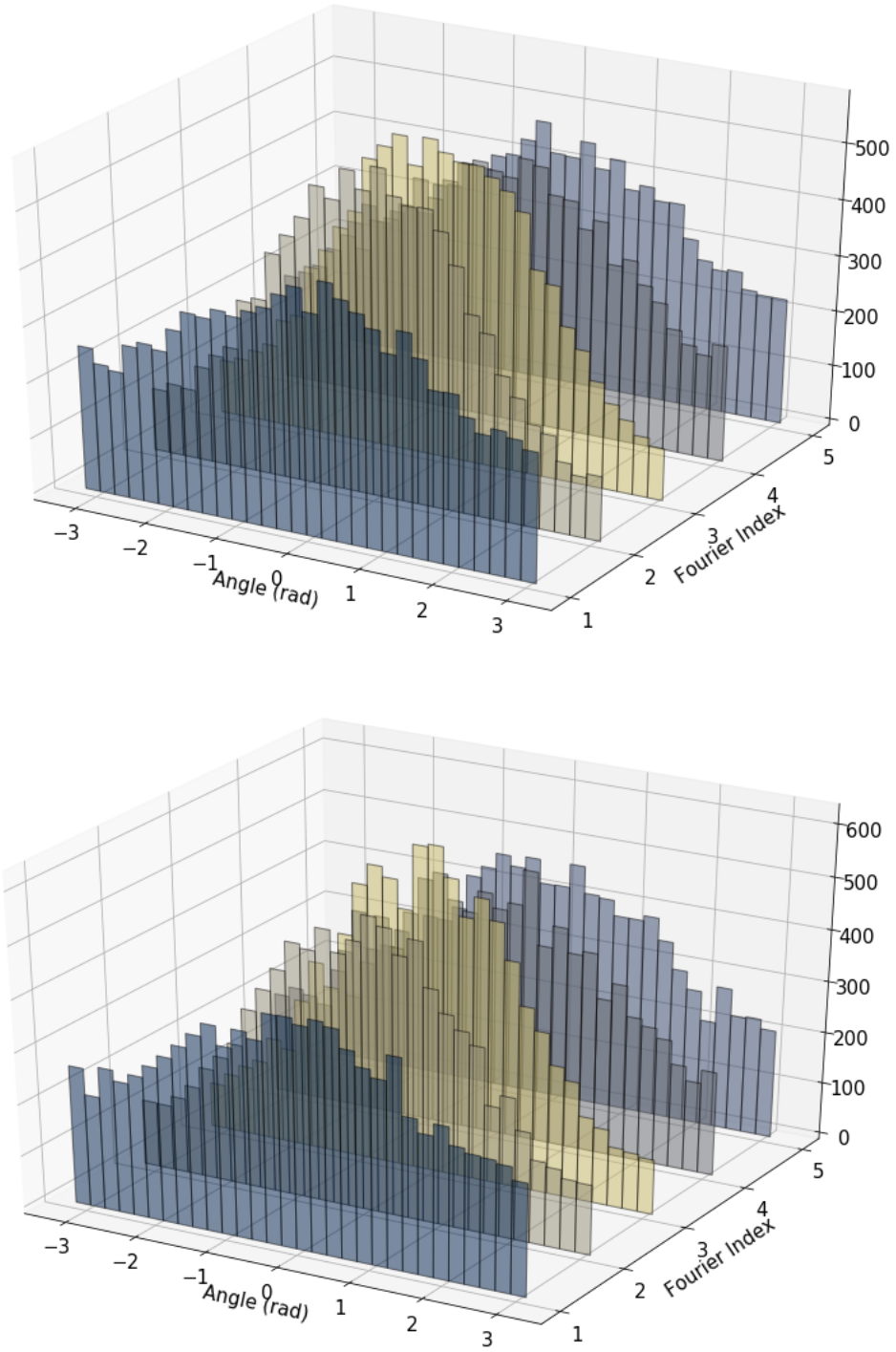

Figure 3.22: Compare the marginal distribution of $\Pr(\phi_i | \sum_{i=1}^{5} a_i \cos\phi_i > 0), i = 1, 2, 3, 4, 5$ between the given dataset (top) and the generated dataset (bottom)

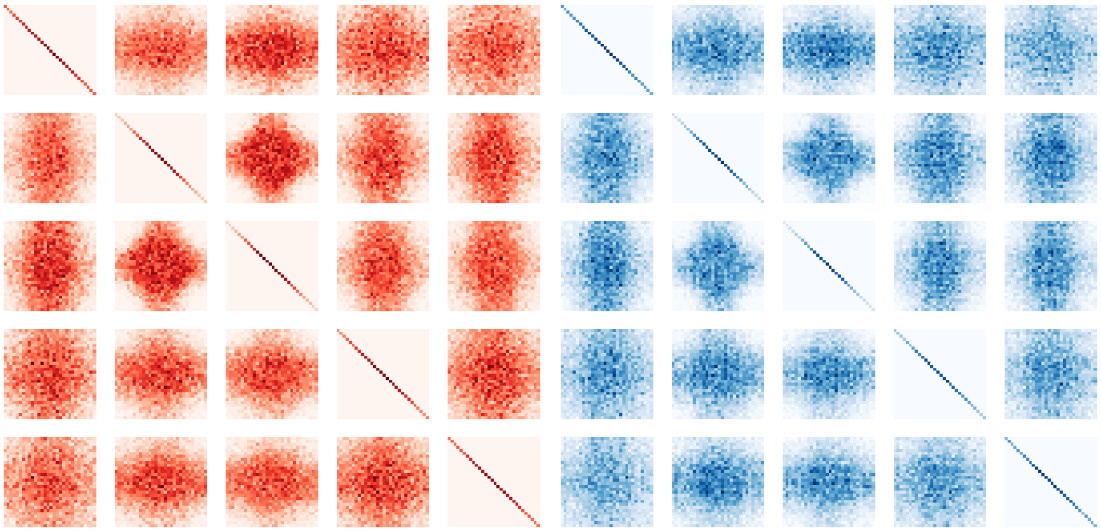

Figure 3.23: Compare pairwise joint distribution $\Pr(\phi_i, \phi_j | \sum_{i=1}^{5} a_i \cos\phi_i > 0)$ between the given dataset (left, red) and the generated dataset (right, blue)

where 3σ is the threshold. The threshold is chosen such that the MCS can be conducted with affordable cost to compare with the model results. Figure 3.24 plots the discretized spectrum (left) and evaluated amplitudes of each Fourier components (right). Again, by sampling the phases independently and uniformly from $-\pi$ to π, the resultant elevation $\eta = \sum_{i=1}^{30} a_i \cos\phi_i$ approximately follows a Gaussian distribution, as shown in Figure 3.25. The elevation in the histogram is non-dimensionalized by the standard deviation σ and the histogram is normalized so that the area of the histogram is one. As shown in Figure 3.25, the phase sequence $(\phi_1, \cdots, \phi_{30})$ leading to large elevation $\zeta = 3\sigma$ occurs rarely so that the TEG model has limited data to directly learn the distribution in Eq 3.41. Therefore, the bootstrapping algorithm discussed in section 3.2.7 is applied with bootstrapping constant $q = 2$.

As mentioned, the first dataset $\mathcal{D}_1$ consists of the phase sequences from the MCS with resultant elevations ranked in the top half. The model learns the distribution of the phase sequences and generates new phase sequences as dataset $\mathcal{D}_1'$. The sequence in $\mathcal{D}_1'$ is then selected into the dataset $\mathcal{D}_2$ if the corresponding resultant elevation is above the mean. To maintain the same size between $\mathcal{D}_1$ and $\mathcal{D}_2$, the number of sampled sequences, $|\mathcal{D}_1'|$, is double the size of $\mathcal{D}_1$. Figure 3.26 compares the given dataset $\mathcal{D}_1$ and the generated dataset $\mathcal{D}_1'$. Again, due to the high dimensional space of the dataset, only the marginal distribution is compared in the figure. Note that the symmetry of the distribution around zero is learned by the model and the height of each bar in the histogram is nearly doubled since the model returns the new dataset

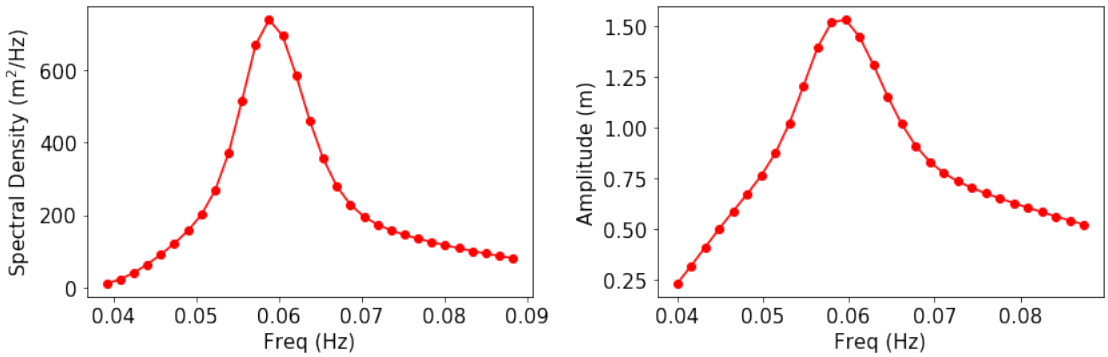

Figure 3.24: Single-sided spectrum and discretized Fourier amplitudes

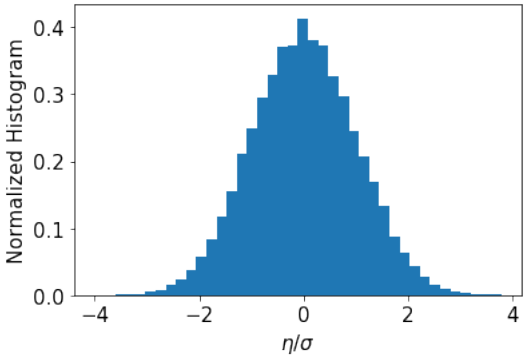

Figure 3.25: Elevation histogram from MCS ($N_{\text{fourier}} = 30$)

with twice the size.

The bootstrapping procedure is conducted seven times in total. After each bootstrapping step, the phase distribution is compared between the given dataset $\mathcal{D}_i$ and the generated dataset $\mathcal{D}_i'$ shown in Figures from 3.26 to 3.32. As shown in the figures, the phases become more clustered around the origin during the bootstrapping iterations, resulting in larger resultant elevation.

The distributions of resultant elevations are compared between the given dataset (red) $\mathcal{D}_i$ and the generated dataset (blue) $\mathcal{D}_i'$ for each bootstrapping iteration in Figures from 3.33 to 3.39. The mismatch is observed at the lower boundary of the red histograms since the dataset $\mathcal{D}_i$ is prepared by filtering the dataset $\mathcal{D}_{i-1}$. However, the tail behavior of the elevation histograms matches between the given dataset and the generated dataset.

After seven iterations of bootstrapping, the model is able to efficiently generate phase sequences leading to elevations greater than 3σ. The generated dataset is selected if the elevation is above 3σ. The histogram of the filtered elevations is plotted in blue in Figure 3.40. A brute-force MCS is conducted to generate 100 times

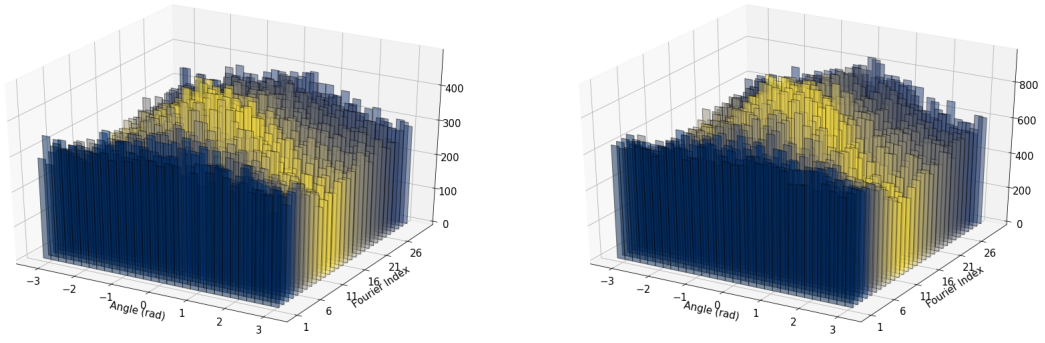

Figure 3.26: Compare the marginal distribution of phases between the given dataset (left) $\mathcal{D}_1$ and the generated dataset (right) $\mathcal{D}'_1$

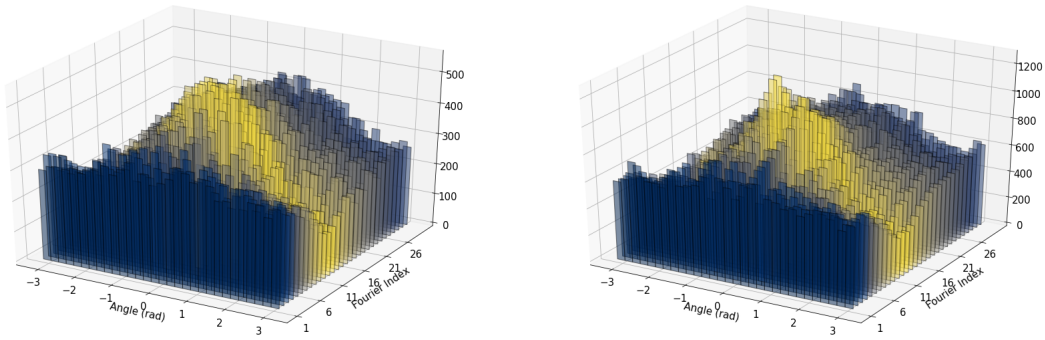

Figure 3.27: Compare the marginal distribution of phases between the given dataset (left) $\mathcal{D}_2$ and the generated dataset (right) $\mathcal{D}'_2$

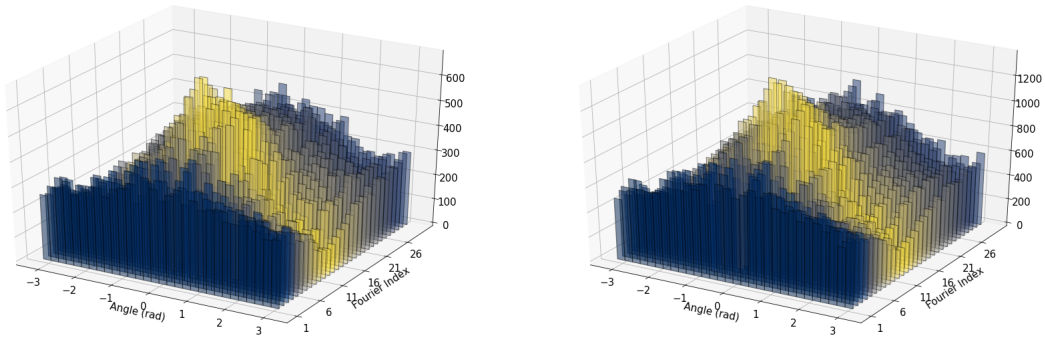

Figure 3.28: Compare the marginal distribution of phases between the given dataset (left) $\mathcal{D}_3$ and the generated dataset (right) $\mathcal{D}'_3$

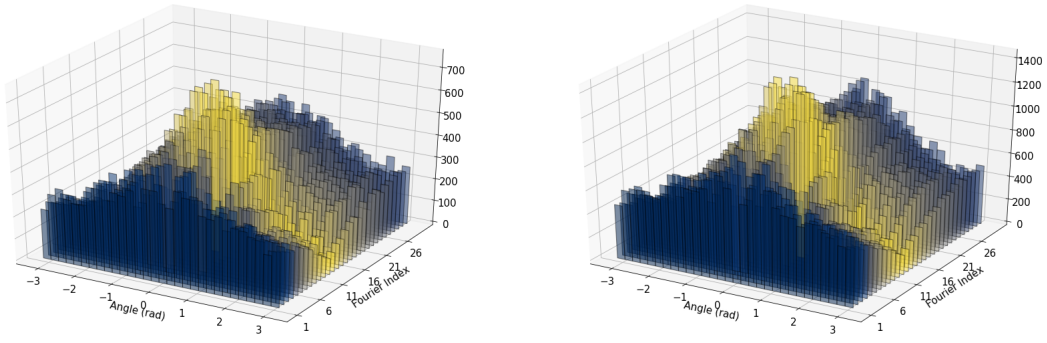

Figure 3.29: Compare the marginal distribution of phases between the given dataset (left) $\mathcal{D}_4$ and the generated dataset (right) $\mathcal{D}'_4$

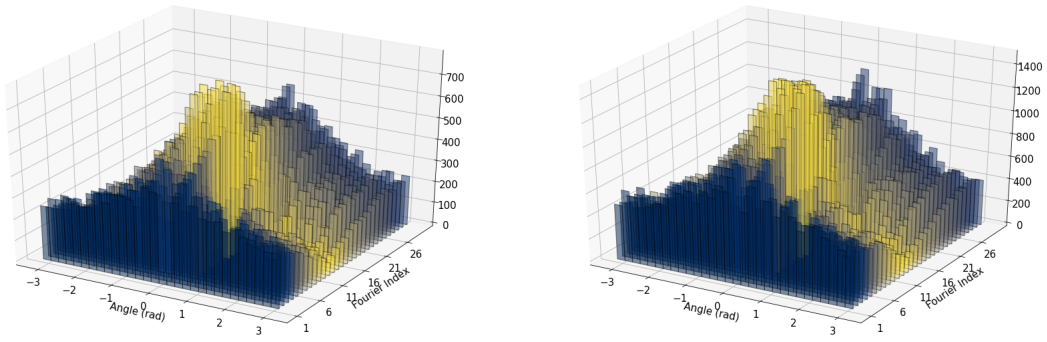

Figure 3.30: Compare the marginal distribution of phases between the given dataset (left) $\mathcal{D}_5$ and the generated dataset (right) $\mathcal{D}'_5$

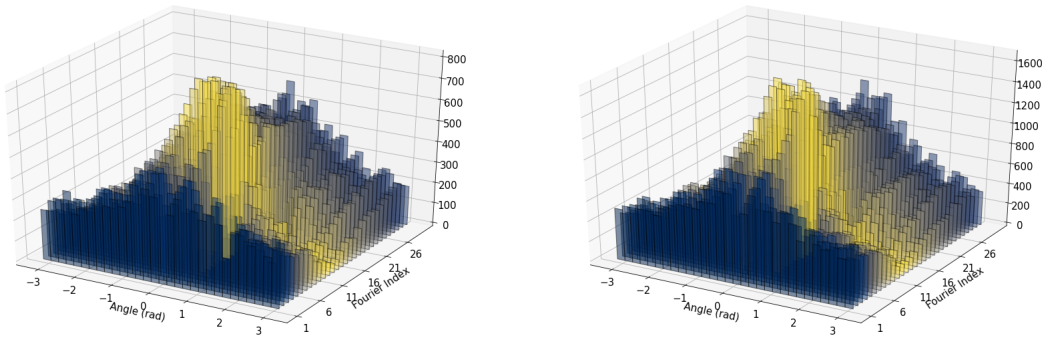

Figure 3.31: Compare the marginal distribution of phases between the given dataset (left) $\mathcal{D}_6$ and the generated dataset (right) $\mathcal{D}'_6$

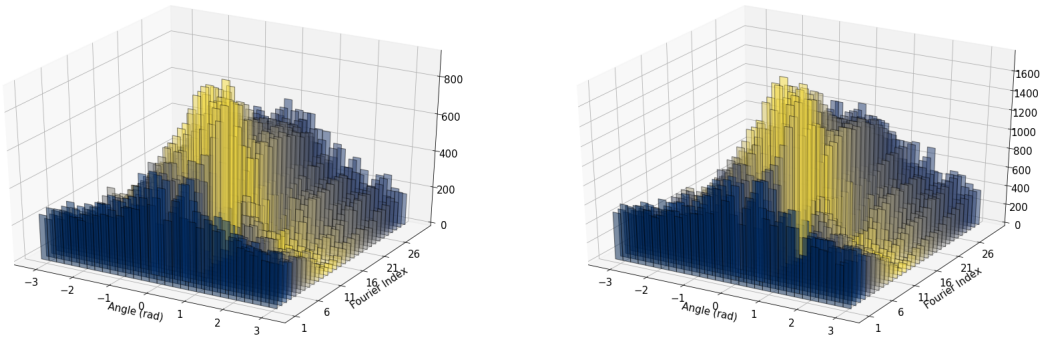

Figure 3.32: Compare the marginal distribution of phases between the given dataset (left) $\mathcal{D}_7$ and the generated dataset (right) $\mathcal{D}'_7$

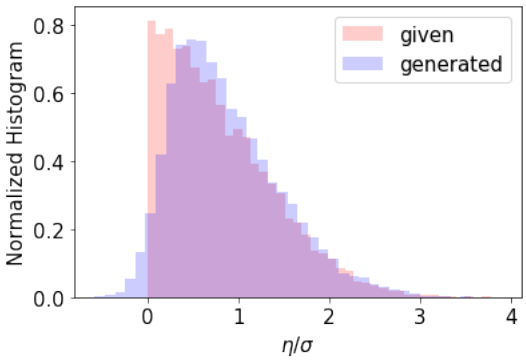

Figure 3.33: Compare the distribution of resultant elevations between the given dataset (red) $\mathcal{D}_1$ and the generated dataset (blue) $\mathcal{D}'_1$

more samples than $\mathcal{D}_0$. To compare, the histogram of the $> 3\sigma$ elevations from the MCS is plotted in red. As the figure shows, the tail decays exponentially since the linear wave with thirty Fourier components follows the Gaussian distribution well.

Since the elevation follows the Gaussian distribution when enough Fourier components are used, the tail behavior is also compared with the theoretical conditional Gaussian PDF in Figure 3.41, where the conditional Gaussian PDF is calculated by

$$\Pr(\eta|\eta > 3\sigma) = \frac{\Pr(\eta, \eta > 3\sigma)}{\Pr(\eta > 3\sigma)} \tag{3.42}$$

The good match between the generated phase sequences and the theoretical elevation distribution verifies the good performance of the model.

The generated phase sequences can be used to evaluate the wave elevation in a window around the design time $t_d = 0$. The elevation time series are evaluated by

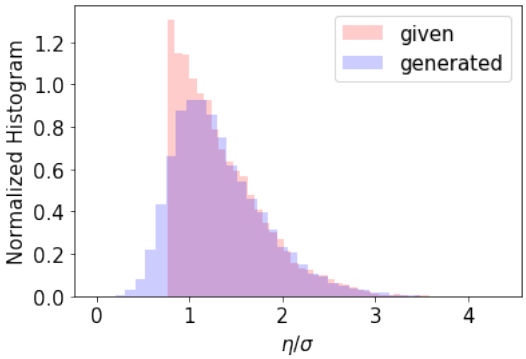

Figure 3.34: Compare the distribution of resultant elevations between the given dataset (red) $\mathcal{D}_2$ and the generated dataset (blue) $\mathcal{D}'_2$

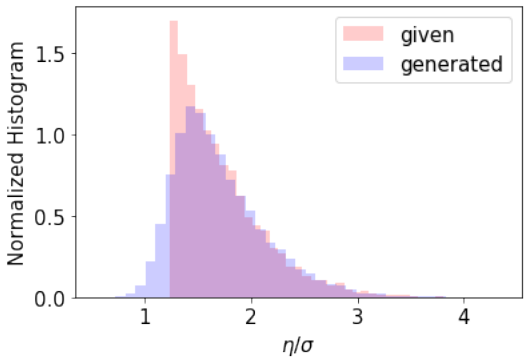

Figure 3.35: Compare the distribution of resultant elevations between the given dataset (red) $\mathcal{D}_3$ and the generated dataset (blue) $\mathcal{D}'_3$

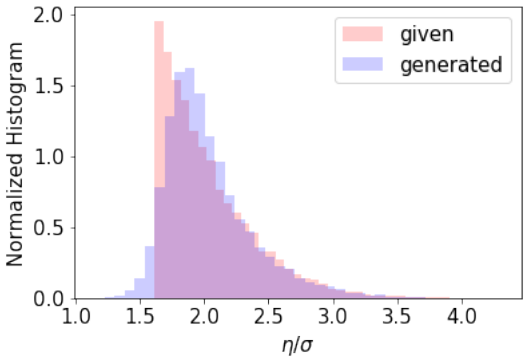

Figure 3.36: Compare the distribution of resultant elevations between the given dataset (red) $\mathcal{D}_4$ and the generated dataset (blue) $\mathcal{D}'_4$

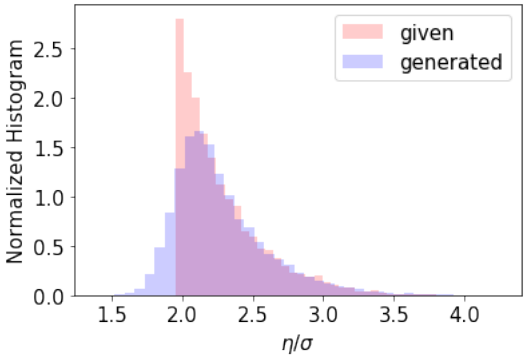

Figure 3.37: Compare the distribution of resultant elevations between the given dataset (red) $\mathcal{D}_5$ and the generated dataset (blue) $\mathcal{D}'_5$

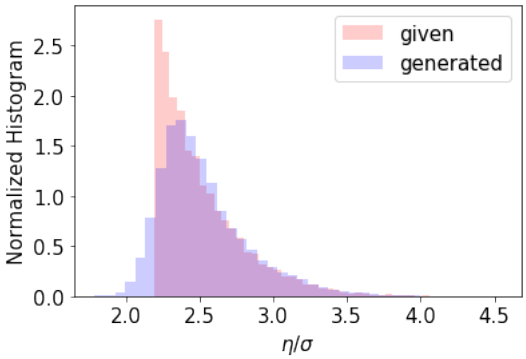

Figure 3.38: Compare the distribution of resultant elevations between the given dataset (red) $\mathcal{D}_6$ and the generated dataset (blue) $\mathcal{D}'_6$

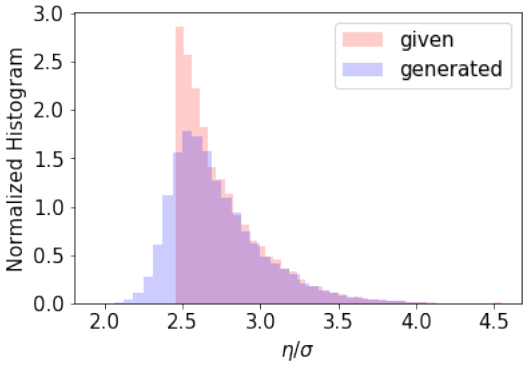

Figure 3.39: Compare the distribution of resultant elevations between the given dataset (red) $\mathcal{D}_7$ and the generated dataset (blue) $\mathcal{D}'_7$

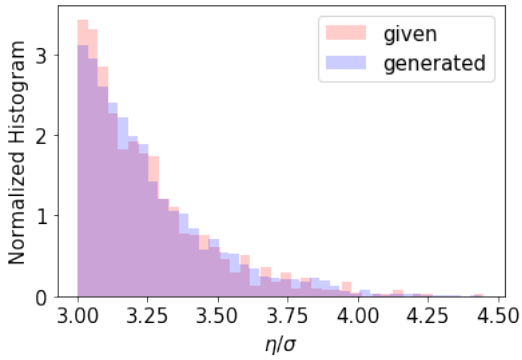

Figure 3.40: Compare tail behavior of the distribution between the generated dataset and the MCS

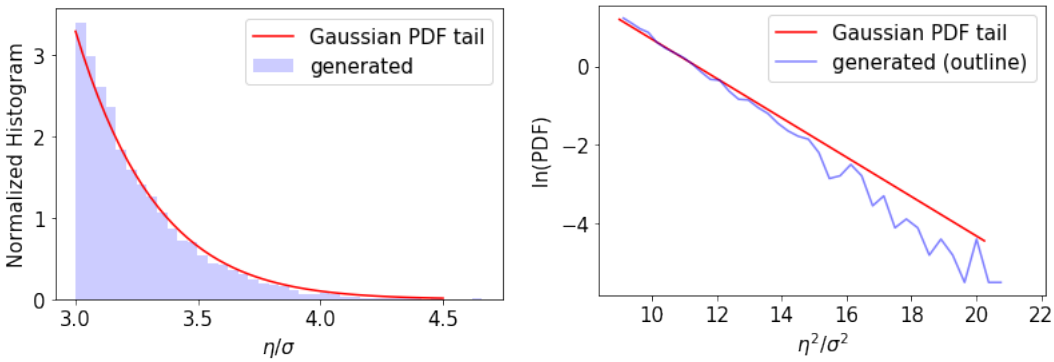

Figure 3.41: Compare tail behavior of the distribution between the generated dataset and the theoretical conditional Gaussian PDF. (left: linear scale, right: log-square scale)

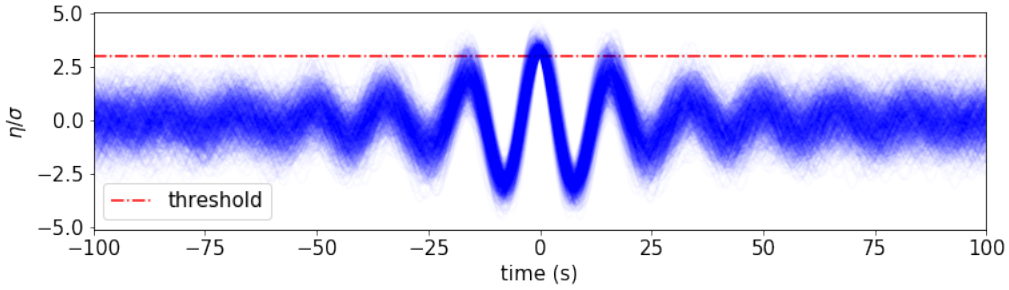

Figure 3.42: 1000 realizations of $\eta(t)$ using generated phase sequences

$$\eta(t) = \sum_{i=1}^{30} a_i \cos(\omega_i t + \phi_i) \tag{3.43}$$

Figure 3.42 shows 1000 phase realizations generated by the model where all the elevation at $t_d = 0$ exceeds the threshold 3σ.

In addition to the wave elevation $\eta(0)$, the crossing velocity $\dot{\eta}(0)$ is also available based on the generated phase sequences.

$$\eta(0) = \sum_{i=1}^{30} a_i \cos\phi_i \tag{3.44}$$

$$\dot{\eta}(0) = -\sum_{i=1}^{30} a_i \omega_i \sin(\phi_i) \tag{3.45}$$

3.4.4 Relation between the Conditional Phase Distribution and the Extreme Value Distribution

The Extreme Value Distribution (EVD) describes the distribution of the extreme value of a random process $X(t)$ during an exposure time window T. In other words, it is defined as the distribution of the random variable $X^T = \max\{X(t), 0 \leq t \leq T\}$. Only knowing the elevation distribution $\Pr(X)$ is not enough to draw conclusions on Extreme Value Distribution (EVD). In fact, knowing the distribution of crossing velocity, $\Pr(\dot{X}|X > \zeta)$ is important to relate the marginal probability $\Pr(X)$ with the EVD. As a trivial example shown in Figure 3.43, let $X_1(t)$ be a stationary random process with the marginal probability $\Pr(X_1)$ and the extreme value during an exposure window is denoted by X_1^T. A derived random process $X_2(t) = X_1(2t)$ has the same marginal probability $\Pr(X_1)$ but a larger extreme value $X_2^T = X_1^{2T} \geq X_1^T$. Therefore, crossing rate information given in the frequency domain also matters to

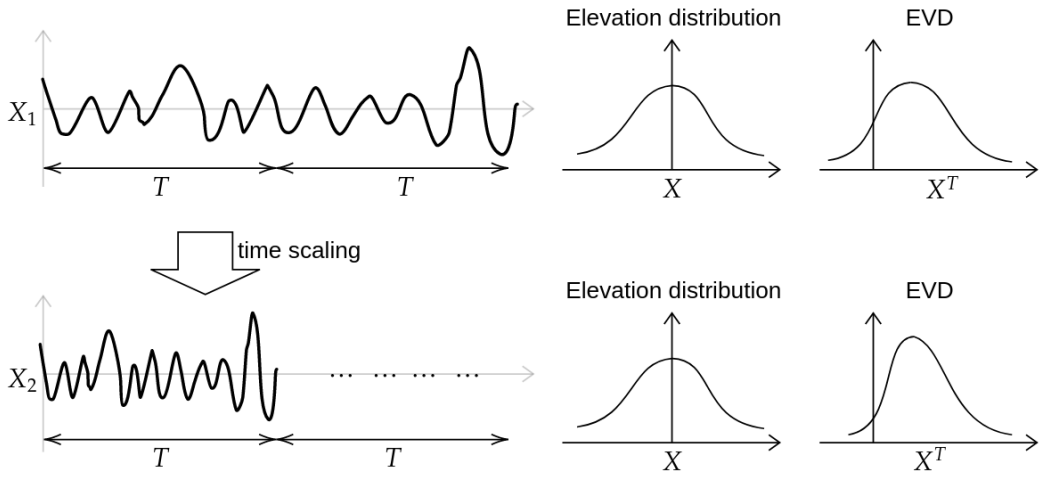

Figure 3.43: Random process $X_1(t)$ and its derived process $X_2(t)$ (time-scaled from X_1) shares the same elevation $f_X(\cdot)$distribution but different Extreme Value distribution $f_{X^T}(\cdot)$

determine the EVD.

The non-exceedance probability of a threshold for a random process $\eta(t)$ can be calculated based on the Poisson assumption with a sufficiently large threshold by

$$F_{\eta^T}(\zeta) = \Pr(\eta^T < \zeta) = e^{-\frac{1}{2}\nu(\zeta)T} \tag{3.46}$$

where $F_{\eta^T}(\cdot)$ is the CDF for the extreme value, ζ is the threshold, T is the exposure time length, and $\nu(\zeta)$ is the mean rate of ζ crossings. The mean rate of ζ crossings counts the average number of times the process crosses the threshold per unit time and it can be calculated for a ergodic stationary random process based on Rice's formula (*Rice*, 1945)

$$\nu(\zeta) = \int_{-\infty}^{\infty} |s| f_{\eta\dot{\eta}}(\zeta, s)\, \mathrm{d}s \tag{3.47}$$

where $f_{\eta,\dot{\eta}}(\cdot,\cdot)$ is the joint PDF of the elevation η and its derivative $\dot{\eta}$.

Using the definition of conditional probability, Rice's formula can be transformed to relate to the marginal probability.

$$\nu(\zeta) = \int_{-\infty}^{\infty} |s| f_{\dot{\eta}|\eta}(s|\zeta)\, \mathrm{d}s f_\eta(\zeta) = \mathbb{E}[|\dot{\eta}|\ |\eta = \zeta] f_\eta(\zeta) \tag{3.48}$$

where $f_\eta(\cdot)$ is the marginal probability.

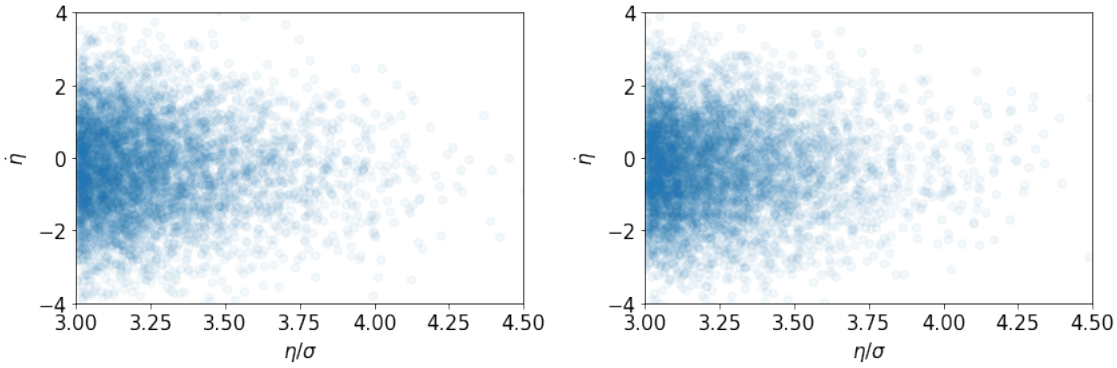

Figure 3.44: Distributions of $(\eta, \dot{\eta})$ for $\eta > 3\sigma$ for the MCS results (left) and the generated dataset (right)

Figure 3.44 compares the joint distribution of $(\eta, \dot{\eta})$ for $\eta > 3\sigma$ between the MCS results (left) and the TEG generated dataset (right) from the thirty component wave example.

The velocities $\dot{\eta}$ with corresponding $\eta \in [3\sigma, 3.05\sigma]$ are collected to estimate the conditional expected absolute velocities $\mathbb{E}[|\dot{\eta}| \ |\eta = \zeta] \approx 1.038$ m/s.

As a comparison, the theoretical conditional velocities can be calculated for the Gaussian process. For a Gaussian process, the elevation and its derivative are indepedent, $\Pr(\eta, \dot{\eta}) = \Pr(\eta)\Pr(\dot{\eta})$ and both follow the zero-mean Gaussian distribution with different variance σ_η^2 and $\sigma_{\dot{\eta}}^2$. Hence, for a Gaussian process

$$\mathbb{E}[|\dot{\eta}| \ |\eta = \zeta] = \mathbb{E}[|\dot{\eta}|] = \sqrt{\frac{2}{\pi}}\sigma_{\dot{\eta}} \tag{3.49}$$

where $\sigma_{\dot{\eta}} = \frac{1}{2}\sum_{i=1}^{\infty}(a_i\omega_i)^2$. The theoretical value is calculated to be 1.094 m/s, which is close to the estimated value calculated above.

Continuing with the mean crossing rate estimated from the generated dataset, the non-exceedance probability of 3σ and window length $T = 500$ s is calculated using Eq 3.46 to be 71.74%.

The non-exceedance probability calculated above can also be verified theoretically and numerically as shown in Figure 3.45. In Appendix A, the distribution of the positive maxima for the Gaussian process can be calculated based on its energy spectrum by Eqs A.2 and A.3, as plotted in blue solid line in the figure. The Extreme Value PDF of the process corresponding to $T = 500$ s is also calculated based on the order statistics of the positive maxima by Eqs A.5 and A.4, and is plotted in orange solid line. These theoretical calculations can be verified by the MCS results that are plotted as the green histogram in the figure. All realizations of the MCS have a

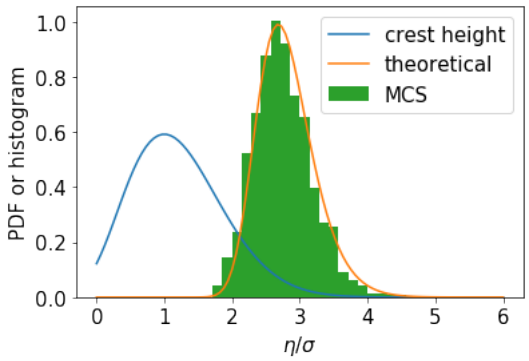

Figure 3.45: Compare the theoretical Extreme Value PDF and the MCS results

time length of 500 s, which is smaller than the self-repeating period (*Belenky*, 2011) $T_{\text{repeat}} = 612$ s of the discretized spectrum. Based on the theoretical calculation of EVD, the non-exceedance probability is calculated to be 70.28%, which is close to the estimation reported above.

Therefore, the TEG model can be used to successfully generate seaways that can be used to explain the full time-domain story of Gaussian linear waves.

3.4.5 Experiment on the Size of Training Dataset

Since the data-driven model is trained on a dataset, the quantity of data matters to the training process, performance, and cost of the model. The user needs to consider how much data should be collected to train the proposed model. More data tells a more complete story of the dynamical behavior but costs more. Fewer data eases the collection cost but reduces the information that the model can learn from. Therefore, users need to consider the tradeoff between the model performance and data collection cost. Unfortunately, the tradeoff point in terms of amounts of collected data is different from case to case. Generally speaking, more complex systems require more training data to characterize their dynamics. As a starting point, the linear wave example is analyzed to provide a general guide to determine the quantity of data needed.

Multiple bootstrapping experiments are carried out with different sizes of datasets. The bootstrapping approach is used in all experiments with seven steps to model large ($> 3\sigma$) linear waves. Table 3.7 lists the different sizes of the initial dataset $\mathcal{D}_0$.

Once the bootstrapping steps are finished, the model generates large seaways that are kept if the resultant elevation is larger than 3σ. The distributions of the resultant elevation from different experiments are compared. Figure 3.46 shows the normalized

Case ID	Dataset size $\lvert\mathcal{D}_0\rvert$
1	2,000
2	5,000
3	6,000
4	7,000
5	8,000
6	9,000
7	10,000
8	20,000

Table 3.7: Experiment setting: different sizes of training data

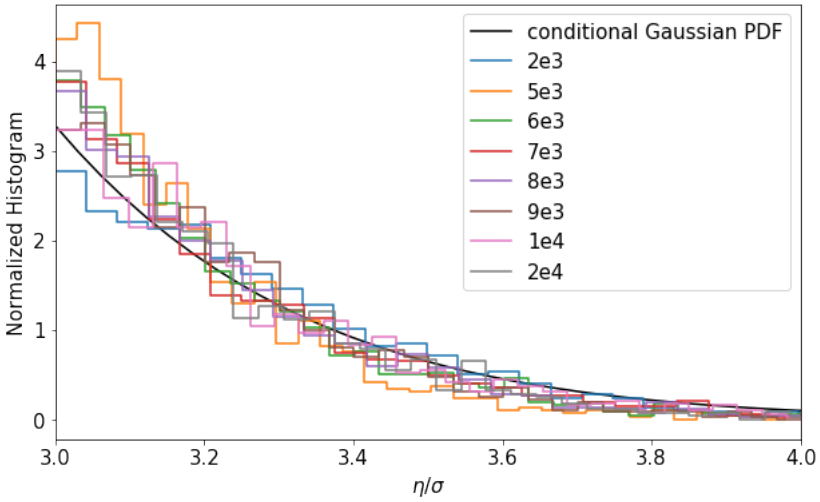

Figure 3.46: Compare the histograms of the generated elevation from different experiments, along with conditional Gaussian PDF.

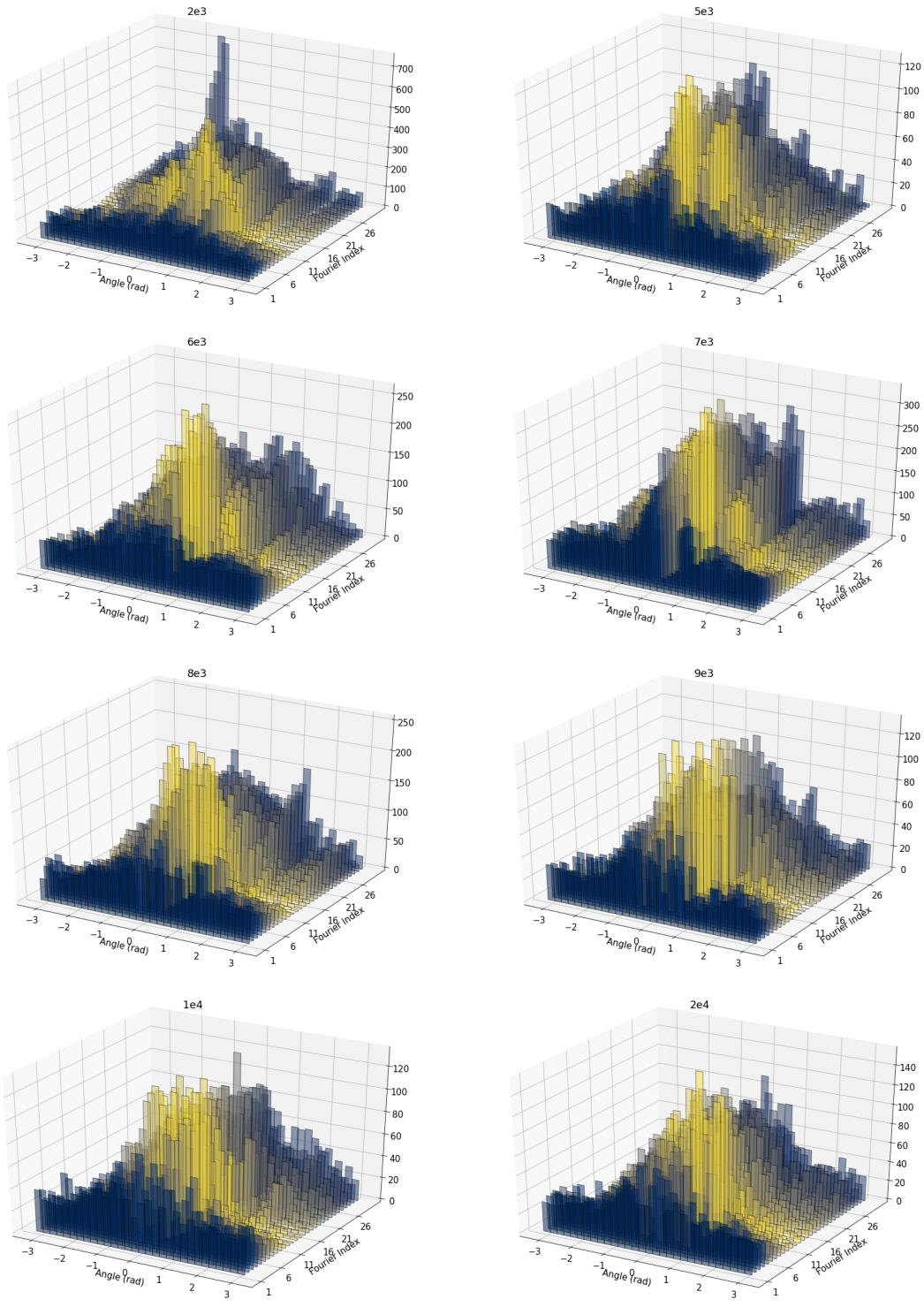

Figure 3.47: Compare the marginal distribution of phases across different experiments

histograms from these experiments along with the conditional Gaussian PDF as a comparison. The generated phase sequences from these experiments can also be compared in terms of marginal probability, shown in Figure 3.47. The experiment with 2,000 phase sequences produces a distribution that is considerably different when compared to results from other experiments. As shown in the figure, less training data produces a less accurate model. Note that the required training data should be described relative to the number of Fourier components, or the number of random parameters of the environment. For the current setting which is $N = 30$ Fourier components and random phases discretized by $K = 30$ bins, a training dataset consists of 10,000 to 20,000 phase sequences is enough for the TEG model to capture the joint distribution and produce good bootstrapping results.

3.5 Example: Second Order Nonlinear Wave

In the previous example, the Gaussian irregular wave represented by a Fourier series is studied. In this example, the second-order nonlinear wave is analyzed. Moreover, the nonlinear wave propagation is considered by allowing a nonzero design location and design time. Due to the interaction among different Fourier components, the resultant wave elevation for the current nonlinear wave is non-Gaussian.

Though the second-order nonlinear wave elevation is not directly represented as a Fourier sum, the randomness of the irregular wave can still be specified by the phase sequence of its linear dominant term. Let $(\phi_1, \cdots, \phi_N)$ be the phase sequence of the linear dominant term. The linear elevation profile is calculated by

$$\eta^{(1)} = \sum_{n=1}^{N} a_n \cos(k_n x - \omega_n t + \phi_n) \tag{3.50}$$

where k_n is the wave number for the n^{th} Fourier component and the wave numbers and their corresponding frequencies are related by the dispersion relation

$$\omega_n^2 = g|k_n| \tanh(|k_n|d) \tag{3.51}$$

where d is the water depth and g is the gravitational constant.

Forristall (2000) gives the second-order correction $\eta^{(2)}$ to the irregular wave elevation by considering the interactions between different Fourier frequencies. Equations from 3.52 to 3.60 calculate the second-order correction so that the final nonlinear elevation is evaluated as $\eta = \eta^{(1)} + \eta^{(2)}$. The correction term is calculated by

$$\eta^{(2)} = \frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j \{K^- \cos(\psi_i - \psi_j) + K^+ \cos(\psi_i + \psi_j)\} \tag{3.52}$$

where

$$K^- = [D_{ij}^- - (k_i k_j + R_i R_j)](R_i R_j)^{-1/2} + (R_i + R_j) \tag{3.53}$$

$$K^+ = [D_{ij}^+ - (k_i k_j - R_i R_j)](R_i R_j)^{-1/2} + (R_i + R_j) \tag{3.54}$$

$$\begin{aligned} D_{ij}^- =& \frac{(\sqrt{R_i} - \sqrt{R_j})\{\sqrt{R_j}(k_i^2 - R_i^2) - \sqrt{R_i}(k_j^2 - R_j^2)\}}{(\sqrt{R_i} - \sqrt{R_j})^2 - k_{ij}^- \tanh k_{ij}^- d} \\ &+ \frac{2(\sqrt{R_i} - \sqrt{R_j})^2 (k_i k_j + R_i R_j)}{(\sqrt{R_i} - \sqrt{R_j})^2 - k_{ij}^- \tanh k_{ij}^- d} \end{aligned} \tag{3.55}$$

$$\begin{aligned} D_{ij}^+ =& \frac{(\sqrt{R_i} + \sqrt{R_j})\{\sqrt{R_i}(k_j^2 - R_j^2) - \sqrt{R_j}(k_i^2 - R_i^2)\}}{(\sqrt{R_i} + \sqrt{R_j})^2 - k_{ij}^+ \tanh k_{ij}^+ d} \\ &+ \frac{2(\sqrt{R_i} + \sqrt{R_j})^2 (k_i k_j - R_i R_j)}{(\sqrt{R_i} + \sqrt{R_j})^2 - k_{ij}^+ \tanh k_{ij}^+ d} \end{aligned} \tag{3.56}$$

and

$$k_{ij}^- = |k_i - k_j| \tag{3.57}$$

$$k_{ij}^+ = |k_i + k_j| \tag{3.58}$$

$$R_i = |k_i| \tanh(|k_i| d) = \omega_i^2 / g \tag{3.59}$$

$$\psi_i = k_i x - \omega_i t + \phi_i \tag{3.60}$$

The second-order correction makes the crest steeper and the trough flatter. Moreover, the correction becomes less important at large water depth. Hence, to illustrate whether the model accounts for the nonlinearity, the water depth of 30 meters is used in the example.

Different from the linear wave example where the design time and the design

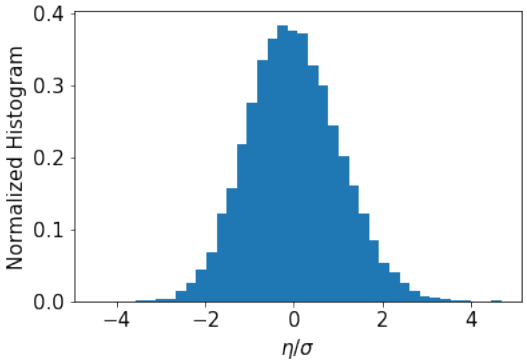

Figure 3.48: Histogram of the wave elevation at the design location and the design time normalized by RMS of the linear components

location are set to zero, this example uses a more general setting. The design location is set at $x_d = 2/k_{\min}$, where $k_{\min}$ is the smallest wave number among all Fourier components. The design time is set at the time needed for the wave component with the smallest group velocity to reach the design location, $t_d = x_d/(\omega/k)_{\min}$. For the discretized spectrum used in the linear wave example, the design location and the design time are calculated to be $x_d = 90.03$ m, and $t_d = 7.45$ s. A MCS is conducted to collect the elevations at the design time and the design location. Figure 3.48 is a histogram of the collected elevations normalized by the standard deviation calculated using the linear components. As shown in the histogram, the second-order correction has tilted the histogram to be asymmetrical and non-Gaussian. Though the mean elevation is still zero, a longer tail is observed for the positive elevation (or crest) compared to the negative elevation (or trough).

The objective of the model is to generate phase vector $(\phi_1, \cdots, \phi_N)$ that the user can specify at the upstream so that a large elevation greater than 2σ will be observed at the design location downstream and the design time. In other words, the model generates phase vectors from the following distribution.

$$\Pr(\phi_1, \cdots, \phi_N | \eta(t_d, x_d) > 2\sigma) \tag{3.61}$$

As before, the bootstrapping method is used to train the TEG model at a large threshold. The bootstrapping procedures are conducted 3 times to make the model generate phase sequences leading to elevations exceeding 2σ. Figures 3.49, 3.50, and 3.51 compare the marginal distribution of the phase sequences between the given dataset (left) $\mathcal{D}_i$ and their corresponding generated dataset $\mathcal{D}_i'$ after each bootstrapping step. The TEG model is again successful to recognize the joint distribution of the

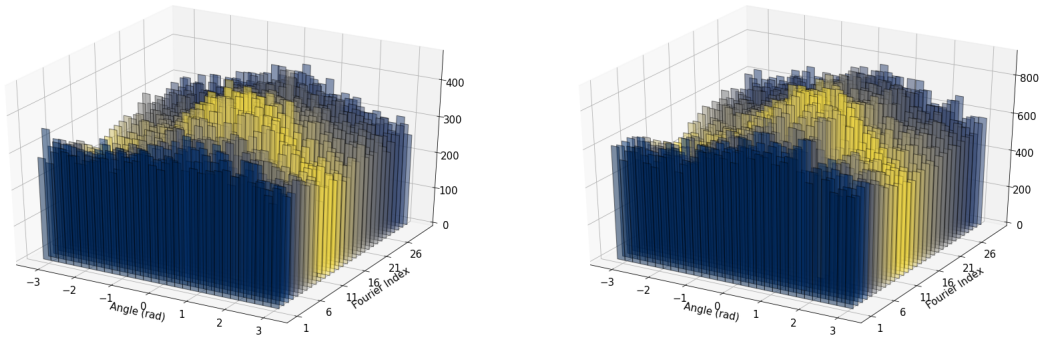

Figure 3.49: Compare the marginal distribution of phases between the given dataset (left) $\mathcal{D}_1$ and the generated dataset (right) $\mathcal{D}_1'$

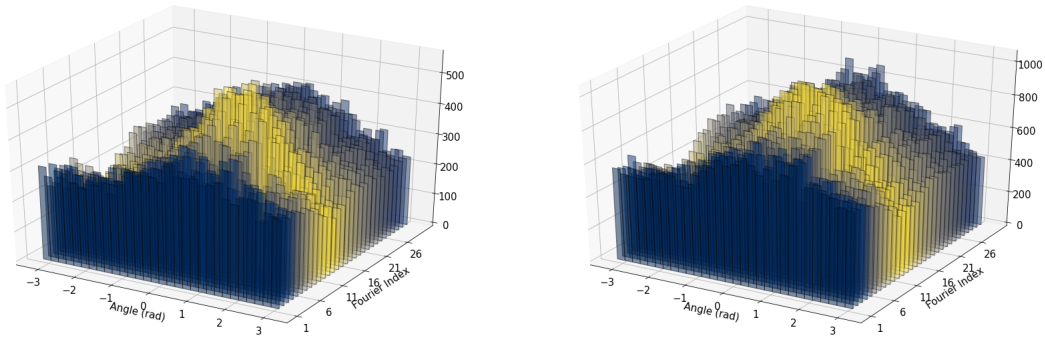

Figure 3.50: Compare the marginal distribution of phases between the given dataset (left) $\mathcal{D}_2$ and the generated dataset (right) $\mathcal{D}_2'$

phases and present a matched marginal distribution. The marginal distribution is no longer symmetric around zero for each phase since the wave propagates downstream.

Figures 3.52, 3.53, and 3.54 compare the histograms of the resultant elevations $\eta(t_d, x_d)$ between the given phases (red) and generated phases (blue) after each bootstrapping step. As before, the histograms are matched in the tail region and the mismatch at the lower boundaries comes from the discontinuity of the filtered dataset.

After 3 bootstrapping steps, the model can generate phase sequences leading to large enough elevations at the design time and the design location. The generated phase sequences with elevations smaller than 2σ are removed, and the remaining histogram is compared against the MCS result in Figure 3.55. Figure 3.56 compares the marginal distribution of phases leading to $\eta(t_d, x_d)$ greater than 2σ between the MCS (left) and the generated dataset (right) after being filtered. Again, all thirty

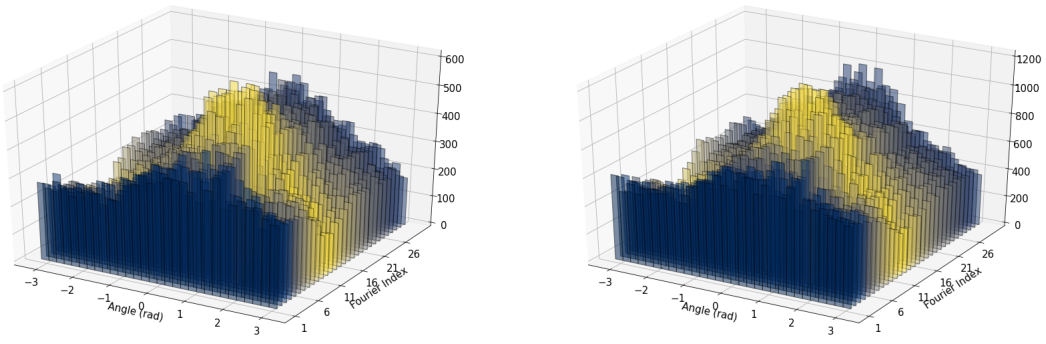

Figure 3.51: Compare the marginal distribution of phases between the given dataset (left) $\mathcal{D}_3$ and the generated dataset (right) $\mathcal{D}'_3$

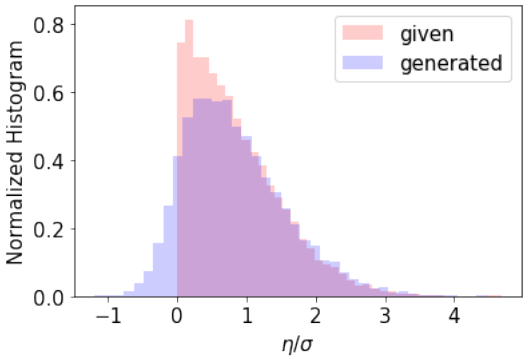

Figure 3.52: Compare the distribution of resultant elevations between the given dataset (red) $\mathcal{D}_1$ and the generated dataset (blue) $\mathcal{D}'_1$

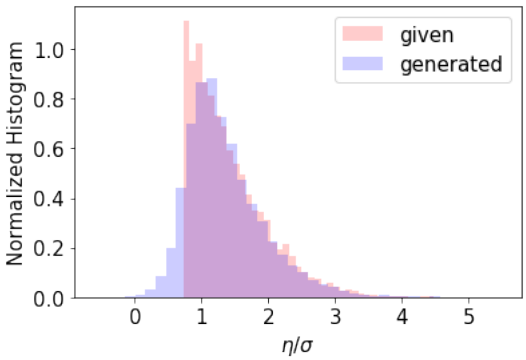

Figure 3.53: Compare the distribution of resultant elevations between the given dataset (red) $\mathcal{D}_2$ and the generated dataset (blue) $\mathcal{D}'_2$

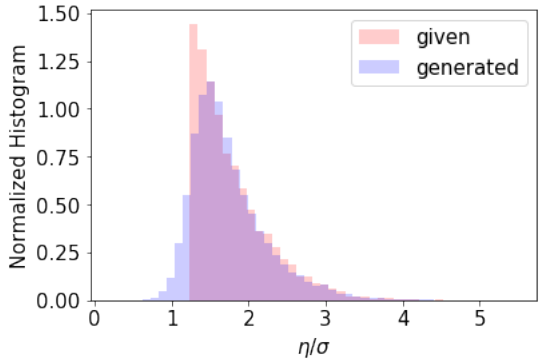

Figure 3.54: Compare the distribution of resultant elevations between the given dataset (red) $\mathcal{D}_3$ and the generated dataset (blue) $\mathcal{D}_3'$

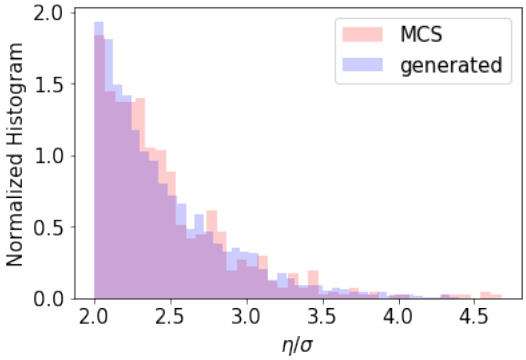

Figure 3.55: Compare tail behavior of the distribution between the generated dataset and the MCS

histograms representing the marginal distribution are colored based on their first-order amplitudes evaluated from the discrete spectrum.

One thousand phase sequences are simulated and their elevation time windows at the design location are plotted in Figure 3.57. The realizations achieve large elevation ($> 2\sigma$) at the design location $t_d = 7.45$ s. As the envelope shows, the nonlinear wave has flatter troughs compared to the linear Gaussian wave.

Therefore, the proposed method succeeds in generating threshold exceedance non-linear and non-Gaussian seaways.

3.6 Example: Nonlinear Ship Roll in Beam Wind and Waves

In this example, the roll motion of a large passenger ship is studied as an example of applying the proposed method on a complex dynamical system under random loading environments. The current example uses a similar setting from papers (*Paroka*

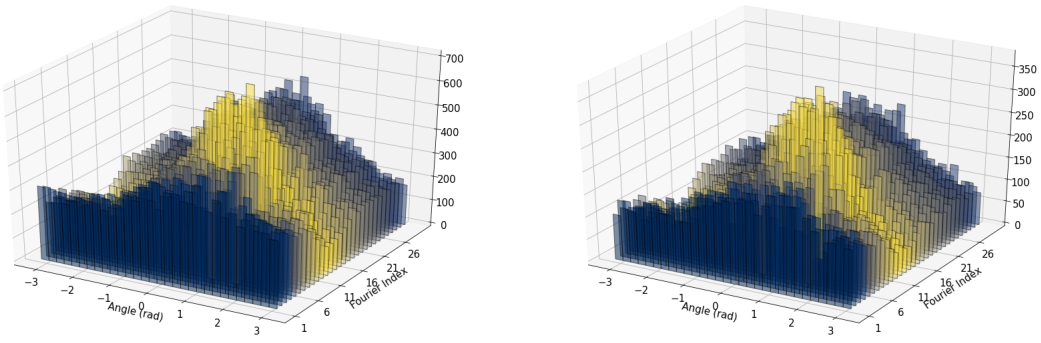

Figure 3.56: Compare the marginal distribution of phases leading to $\eta(t_d, x_d) > 2\sigma$ between the MCS (left) and the generated dataset (right) after being filtered

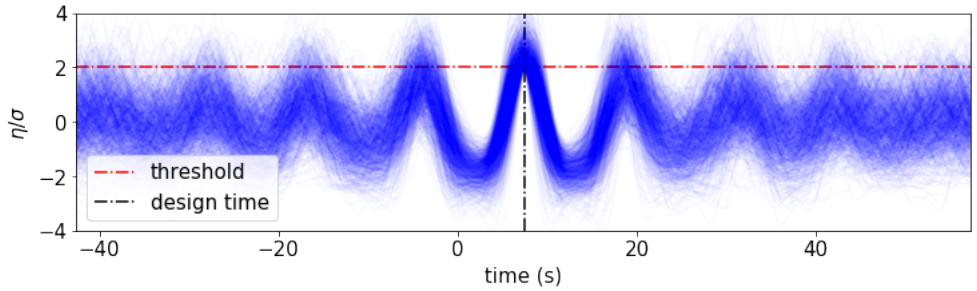

Figure 3.57: 1000 realizations of $\eta(t)$ using generated phase sequences

and Umeda, 2006; *Kogiso and Murotsu*, 2008), where the ship roll is considered to be a single degree of freedom described the following Ordinary Differential Equation (ODE)

$$(I_{xx} + A_{xx})\ddot{\theta} + D(\dot{\theta}) + \Delta GZ(\theta) = M_{\text{air}}(t) + M_{\text{wave}}(t) \tag{3.62}$$

where I_{xx} is the moment of inertia of the ship, A_{xx} is the hydrodynamic coefficient of the added inertia, $D(\dot{\theta})$ is the rolling damping moment, Δ is the ship displacement, and $GZ(\theta)$ is the righting arm, $M_{\text{air}}(t)$ is the wind-induced moment, $M_{\text{wave}}(t)$ is the wave-induced moment, and $\theta(t)$ is the rolling angle of the ship.

The GZ curve of the ship is fit using a 7th-order polynomial with only odd power of roll angle θ,

$$GZ(\theta) = c_1\theta + c_3\theta^3 + c_5\theta^5 + c_7\theta^7 \tag{3.63}$$

where c_1, c_3, c_5, and c_7 are the coefficients of the polynomial fit.

The damping moment is nonlinear by introducing an quadratic term $\dot{\theta}|\dot{\theta}|$

$$D(\dot{\theta}) = (I_{xx} + A_{xx})(2\mu\dot{\theta} + \beta\dot{\theta}|\dot{\theta}|) \tag{3.64}$$

where μ and β are linear and nonlinear coefficients of the damping moments respectively.

The external moment comes from both wave and wind and each is assumed to be a Gaussian process. For the wind-induced moment, the moment time series is evaluated by

$$M_{\text{wind}}(t) = \frac{1}{2}\rho_{\text{air}} C_m(\bar{U}_w^2 + 2\bar{U}_w U(t)) A_L H \tag{3.65}$$

where ρ_{air} is the air density, C_m is the aerodynamic drag coefficent, $\bar{U}_w$ is the mean velocity of the wind, $U(t)$ is the time-varying velocity of the gusty wind, A_L is the lateral windage area, and H is the height distance between the center of the wind force and the center of the hydrodynamic force. The time-varying component of the wind-induced moment is modeled as a Gaussian process with its spectrum S_{am} calculated by

$$S_{\text{am}}(\omega) = (\rho_{\text{air}} C_m \bar{U}_w A_L H)^2 \chi^2(\omega) S_{\text{wind}}(\omega) \tag{3.66}$$

where the aerodynamic admittance of wind turbulence $\chi(\omega)$ and the Davenport spectrum S_{wind} are calculated below.

$$\chi(\omega) = \frac{1}{1 + \left(\frac{\omega\sqrt{A_L}}{\pi \bar{U}_w}\right)^{4/3}} \tag{3.67}$$

$$S_{\text{wind}}(\omega) = 4K\frac{\bar{U}_w^2}{\omega}\frac{X_D^2}{(1+X_D^2)^{4/3}} \tag{3.68}$$

where $K = 0.003$ and $X_D = 600\frac{\omega}{\pi \bar{U}_w}$.

For the wave-induced moment, the Froude-Krylov exciting moment is calculated by

$$M_{\text{wave}}(t) = \Delta G M \gamma \Theta(t) \tag{3.69}$$

where γ is the effective wave slope coefficient and $\Theta(t)$ is the wave slope time series. The wave-induced moment is modeled by a Gaussian process with its spectrum S_{mw} calculated by

$$S_{\text{mw}}(\omega) = (\Delta G M \gamma)^2 S_\alpha(\omega) \tag{3.70}$$

where the effective wave slope coefficient γ and the wave slope spectrum $S_\alpha(\omega)$ are calculated as follows.

$$\gamma = \begin{cases} 0.777 & \omega \leq \sqrt{4\pi g/B_s} \\ 0.0 & \text{otherwise} \end{cases} \tag{3.71}$$

$$S_\alpha(\omega) = \frac{\omega^4}{g^2} S_{\text{wave}}(\omega) = \frac{A}{g^2\omega}\exp(-\frac{B}{\omega^4}) \tag{3.72}$$

The Pierson-Moskowitz spectrum is used with the model coefficients calculated as follows.

$$A = 172.75\frac{H_{1/3}^2}{T_{01}^4} \tag{3.73}$$

$$B = \frac{691}{T_{01}^4} \tag{3.74}$$

$$H_{1/3}(\bar{U}_w) = -2 \times 10^{-5}\bar{U}_w^3 + 8.2 \times 10^{-3}\bar{U}_w + 0.1456\bar{U}_w - 0.1599 \tag{3.75}$$

Symbols	Units	Values
ρ_{air}	kg/m^3	1.225
ρ_{sw}	kg/m^3	1.025×10^3
V	m^3	4.241×10^4
B_s	m	32.25
A_L	m^2	8814
H	m	20.86
C_m		1.1137
K		0.003
ω_0	rad/s	0.1794
T_0	s	35.02
γ		0.777
GM	m	1.06
$\bar{U}_w$	m/s	29.50
μ	1/s	8.5×10^{-3}
β	1/rad	0.385
$I_{xx} + A_{xx}$	kg m^2	1.4033×10^{10}
Δ	kg	4.2608×10^8
c_1		1.06
c_3		0.174
c_5		-6.4269
c_7		5.3323

Table 3.8: Environmental parameters and specification of the ship roll example (*Paroka and Umeda*, 2006; *Kogiso and Murotsu*, 2008)

$$T_{01} = 3.86\sqrt{H_{1/3}} \tag{3.76}$$

Table 3.8 lists the specifications of the current example. With the current setting, Figure 3.58 shows the GZ curve versus the roll angle of the large passenger ship. The angle of vanishing stability θ_v is evaluated as 0.7755 rad and the linear natural frequency $\omega_0 = \sqrt{c_1\Delta/(I_{xx} + A_{xx})} = 0.1794$ rad/s.

The wind-induced spectrum and the wave-induced spectrum are discretized in log scale and plotted in Figure 3.59. Twenty Fourier components are used to discretize the wind-induced moment spectrum and ten Fourier components are used to discretized the wave-induced moment spectrum. More components are used for the wind-induced moment spectrum since the natural frequency of the ship roll is in its effective range. Therefore, the external moment $M(t) = M_{\text{air}}(t) + M_{\text{wave}}(t)$ is described by

$$M_{\text{air}}(t) = M_{\text{static}} + \sum_{i=1}^{N_{\text{wind}}} a_{\text{air},i} \cos(\omega_{\text{air},i} t + \phi_{\text{air},i}) \tag{3.77}$$

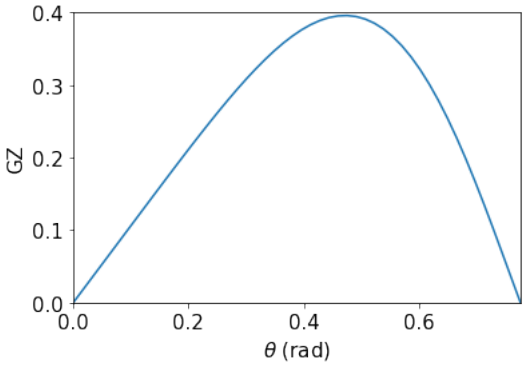

Figure 3.58: GZ curve of the large passenger ship

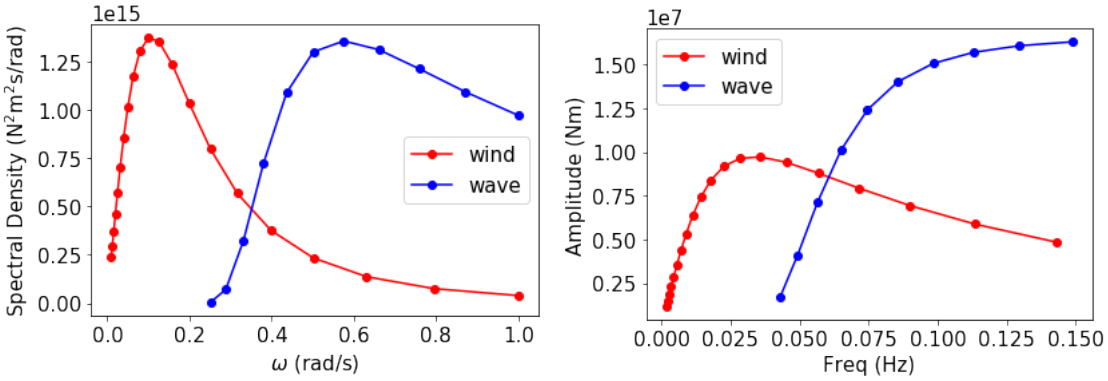

Figure 3.59: Spectral density (left) and the corresponding amplitudes (right) for the wind-induced (red) and wave-induced moment (blue)

where $M_{\text{static}} = \frac{1}{2}\rho_{\text{air}} C_m \bar{U}_w^2 A_L H$, and

$$M_{\text{wave}}(t) = \sum_{i=1}^{N_{\text{wave}}} a_{\text{wave},i} \cos(\omega_{\text{wave},i} t + \phi_{\text{wave},i}). \tag{3.78}$$

The random phase sequence $(\phi_{\text{air},1}, \cdots, \phi_{\text{air},N_{\text{wind}}}, \phi_{\text{wave},1}, \cdots, \phi_{\text{wave},N_{\text{wave}}})$ are sampled independently and uniformly from $-\pi$ to π to simuate a Gaussian loading environment. To carry out the MCS, the dynamical roll has the initial condition $(\theta_0, \dot{\theta}_0) = (\theta_s, 0)$, where θ_s is the roll angle to balance the static wind moment.

$$c_1\theta_s + c_3\theta_s^3 + c_5\theta_s^5 + c_7\theta_s^7 = M_{\text{static}} \tag{3.79}$$

The MCS is carried out by solving the governing ODE starting with the same initial condition and under different Gaussian loading environments. Each realization has the same time length of 16 natural periods and is solved by a 4th-order Ruger-Kutta numerical integration method.

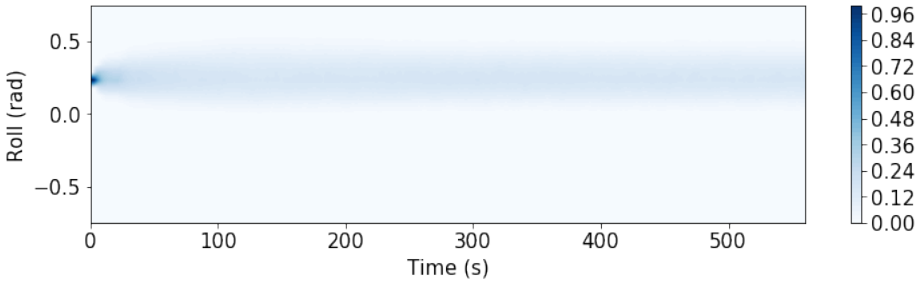

Figure 3.60: Time evolution of the distribution of $\theta(t)$

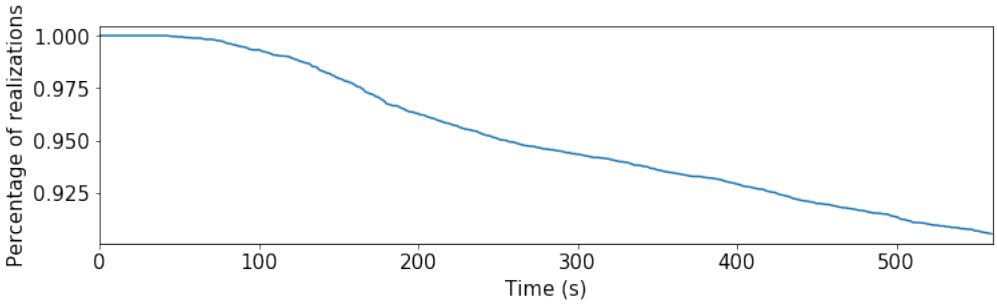

Figure 3.61: Percentage of realizations that are stable versus time

To verify the design time $t_d = 10T_0$ is chosen properly, the roll angle histogram representing its distribution evolution versus time, $\theta(t)$, is plotted in Figure 3.60. Since the stiffness moment approximation is not accurate for $|\theta| > \theta_v$, the simulation is terminated once capsizing occurs and the timestamp when capsizing occurs is recorded. Figure 3.61 shows the percentage of realizations that are stable (not capsized) versus the simulated time length. As shown in the figure, the decay rate of stable realizations is steady at the design time $t_d = 10T_0 = 350$ s.

The number of Fourier components used to discretize the spectrum of the wind-induced moment and wave-induced moment are also validated to balance the accuracy of the marginal distribution of the roll angle and the computational cost. Table 3.9 lists 6 experiments with different discretization strategies $(N_{\text{wind}}, N_{\text{wave}})$.

Both the time-evolved distribution of $\theta(t)$ and the percentage of stable realizations are found to be steady at the design time $t_d = 10T_0$ for all 6 experiments. Figure 3.62 shows the convergence test on statistics $\theta(t_d)$ from experiment 1 to experiment 6. As shown in the Figure, too few number of Fourier components results in no capsizing in the MCS, and ends up with a low-variance distribution since the moment upper bound, $M_{\text{upper}} = M_{\text{static}} + \sum_{i=1}^{N_{\text{wind}}} a_{\text{air},i} + \sum_{i=1}^{N_{\text{wave}}} a_{\text{wave},i}$, is not large enough to excite large roll. Inspired by the experiment results, the setting $(N_{\text{wind}}, N_{\text{wave}}) = (20, 10)$ is affordable and large enough to have converged statistics. Therefore, this setting is

Case ID	N_{wind}	N_{wave}
1	5	5
2	10	5
3	10	10
4	20	10
5	30	15
6	40	20

Table 3.9: Determine the number of Fourier components used to discretize S_{am} and S_{mw}

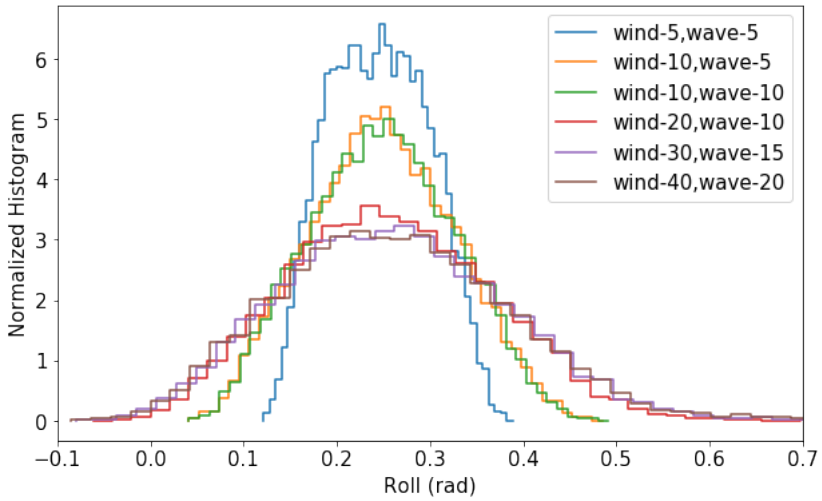

Figure 3.62: Convergence test on the histogram of $\theta(t_d)$

used throughout the example.

The external moment time series and the corresponding roll response for 20,000 realizations are plotted in Figure 3.63. The realizations which present roll angle larger than the angle of vanishing stability during the simulation window are removed since the polynomial fit of the GZ curve is no longer accurate.

The design time is set to be long enough with 10 natural periods ($t_d = 10T_0$) so that the statistical summary is fully developed at t_d. The roll angle at the design time is a random variable with its distribution estimated by evaluating the MCS results in Figure 3.64. The histogram is asymmetrical due to the static wind-induced moment and non-Gaussian due to the nonlinearity of the dynamical system. Based on the distribution, a roll angle of 0.4 or 0.5 rad is considered to be a large response. Therefore, the objective of the proposed model is to provide a loading environment by sampling from the distribution

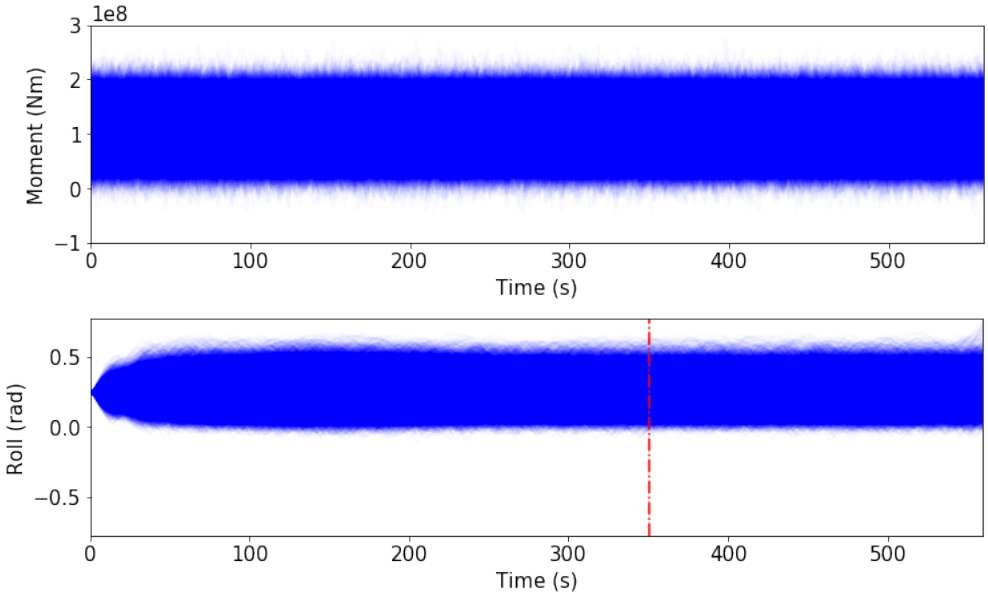

Figure 3.63: 20,000 realizations of the external moment $M(t)$ (top) and corresponding roll response $\theta(t)$ (bottom)

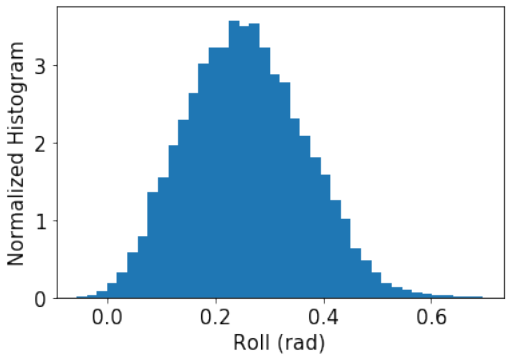

Figure 3.64: Histogram of the roll angle at the design time, $\theta(t_d)$

$$\Pr(\phi_{\text{air},1}, \cdots, \phi_{\text{air},20}, \phi_{\text{wave},1}, \cdots, \phi_{\text{wave},10} | \theta(t_d; \phi_{\text{air},1}, \cdots, \phi_{\text{air},20}, \phi_{\text{wave},1}, \cdots, \phi_{\text{wave},10}) > \zeta) \tag{3.80}$$

Among all 20,000 realizations at the design time, there are about 10.41% realizations having roll angle larger than 0.4 rad and there are about 1.32% realizations having roll angle larger than 0.5 rad. Figure 3.65 shows the realizations with $\theta(t_d) > 0.4$ rad and Figure 3.66 shows the realizations with $\theta(t_d) > 0.5$ rad.

Though the phase angles come from two different environment sources, they are regarded as a sequence of 30 elements that are sampled together $(\phi_1, \cdots, \phi_{30})$ to

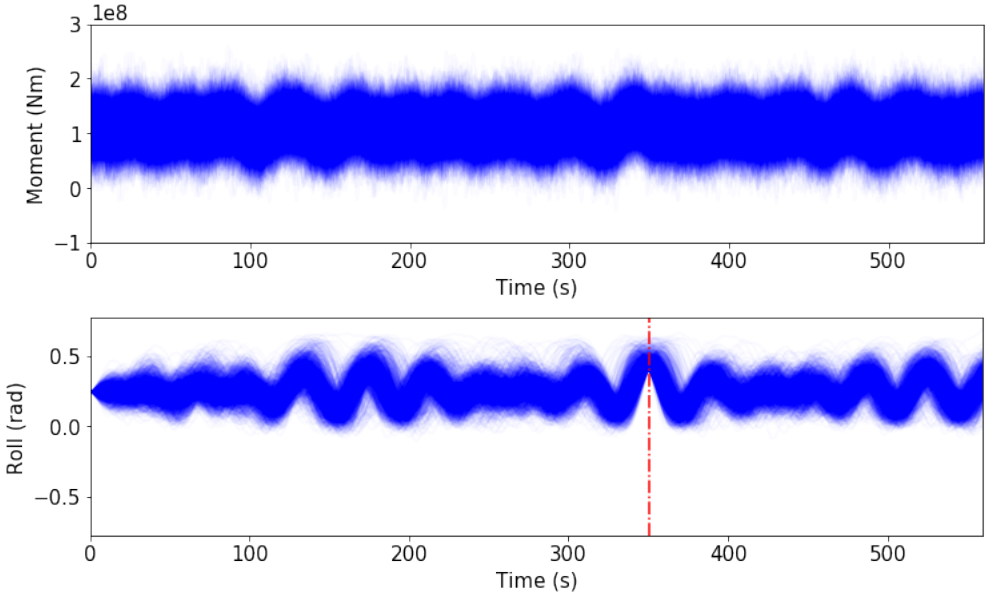

Figure 3.65: MCS realizations external moment (top) and roll response (bottom) with $\theta(t_d) > 0.4$ rad

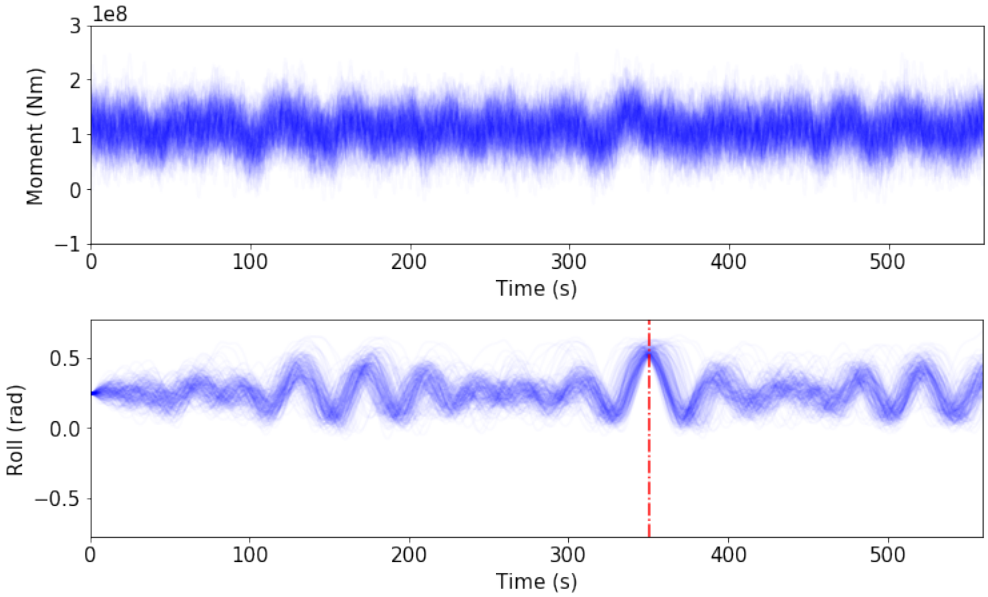

Figure 3.66: MCS realizations external moment (top) and roll response (bottom) with $\theta(t_d) > 0.5$ rad

account for any possible correlation between two sources in large roll situations. The dataset $\mathcal{D}_0$ consists of all the phase sequences where each element in the sequence is independently and uniformly distributed from $-\pi$ to π. The sequences with $\theta(t_d) > \mu_{\theta(t_d)}$ are then selected to form dataset $\mathcal{D}_1$. As before, the TEG model takes the dataset as input and learns the joint distribution among all Fourier components. Once the model is trained, the phase sequence can then be sampled as many times as the user desires. Figure 3.68 compares the marginal distribution of phases between the given dataset (left) $\mathcal{D}_1$ and the generated dataset (right) $\mathcal{D}_1'$. As shown in the comparison, the TEG model succeeds in recognizing the joint distribution among phase angles in the measurement of the marginal distribution. Other observations can also be summarized from both histograms. The phase angles for the wave component, which has indices from 21 to 30 are almost uniformly distributed, and the most nonuniform phase distribution comes from indices from 9 to 17. These phase angles from the wind component play a more important role in roll angles at the design time. This observation is reasonable since the Fourier components with indices from 9 to 17 have large amplitudes and frequencies closer to the natural frequency of the ship roll compared to the wave Fourier components and the other wind Fourier components. Therefore, though the wave components have larger amplitudes compared to the wind components as they are colored in brighter yellow, the wind components in this example produce more impact on the roll, which can be observed in many dynamical systems.

As before, the bootstrapping method is used to gradually increase the lower bound of the resultant roll angle of the dataset. Figures from 3.68 to 3.73 compare the marginal distribution of the phases between the given dataset (left) $\mathcal{D}_i$ and the generated dataset (right) $\mathcal{D}_i'$ after each bootstrapping step. As shown, the TEG model successfully learns the correlation among the Fourier components and generates new phase sequences that follow a similar distribution. As the bootstrapping is carried out, the observation above becomes more apparent. The phases from Fourier indices from 9 to 17 deviate more from the uniform distribution, which also matches the intuition that a larger roll angle is rarer.

In additional to the comparison of the marginal distribution of phases between the given dataset and the generated dataset, Figures from 3.74 to 3.79 compare the resultant roll angle at the design time between the given dataset (red) $\mathcal{D}_i$ and the generated dataset (blue) $\mathcal{D}_i'$ after each bootstrapping step. Again, the match at the tail of the distribution verifies the application of the bootstrapping method. The lower bound of the given dataset gradually increases after each bootstrapping step,

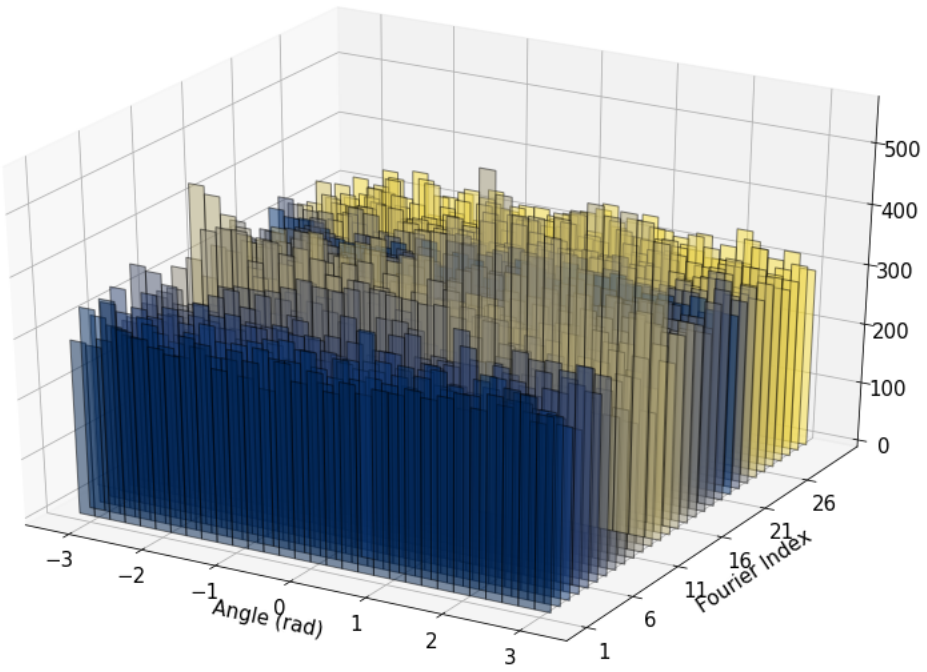

Figure 3.67: The marginal distribution of phases for the given dataset $\mathcal{D}_1$

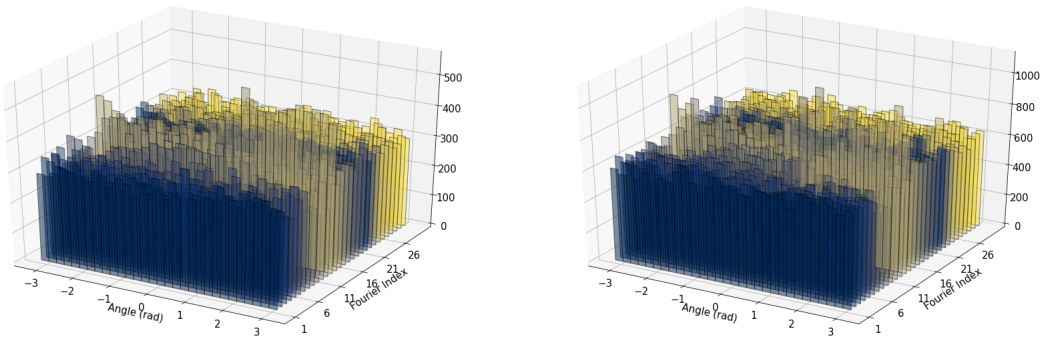

Figure 3.68: Compare the marginal distribution of phases between the given dataset (left) $\mathcal{D}_1$ and the generated dataset (right) $\mathcal{D}'_1$

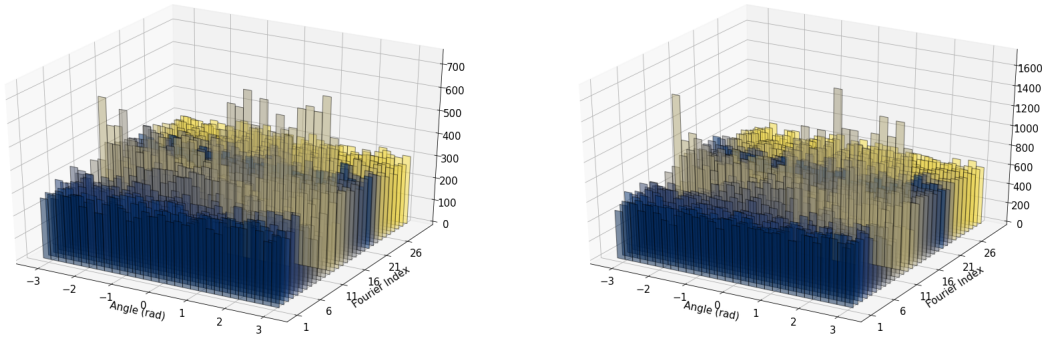

Figure 3.69: Compare the marginal distribution of phases between the given dataset (left) $\mathcal{D}_2$ and the generated dataset (right) $\mathcal{D}'_2$

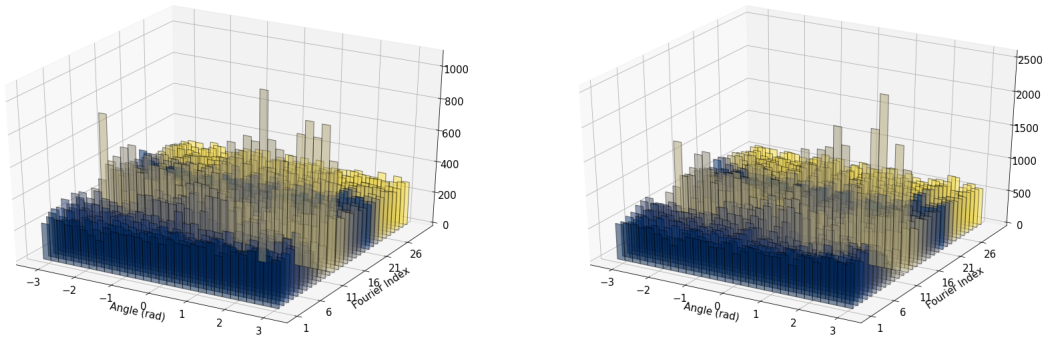

Figure 3.70: Compare the marginal distribution of phases between the given dataset (left) $\mathcal{D}_3$ and the generated dataset (right) $\mathcal{D}'_3$

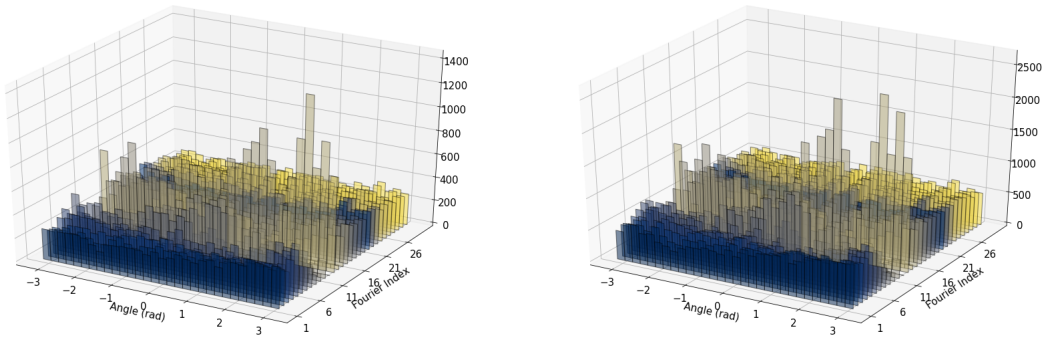

Figure 3.71: Compare the marginal distribution of phases between the given dataset (left) $\mathcal{D}_4$ and the generated dataset (right) $\mathcal{D}'_4$

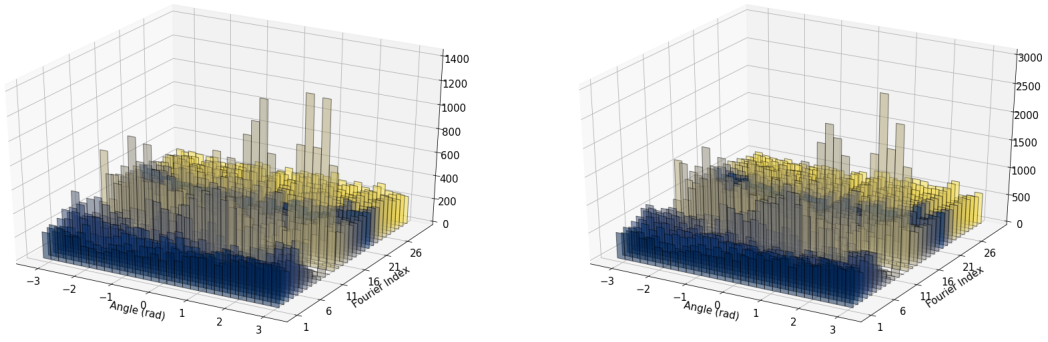

Figure 3.72: Compare the marginal distribution of phases between the given dataset (left) $\mathcal{D}_5$ and the generated dataset (right) $\mathcal{D}'_5$

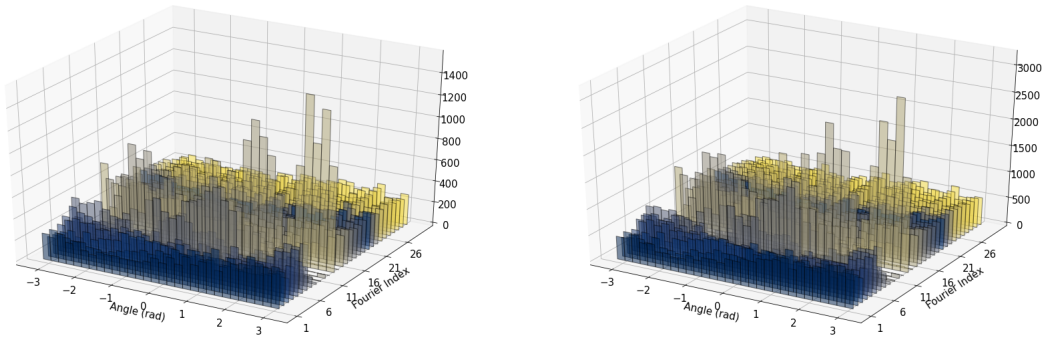

Figure 3.73: Compare the marginal distribution of phases between the given dataset (left) $\mathcal{D}_6$ and the generated dataset (right) $\mathcal{D}'_6$

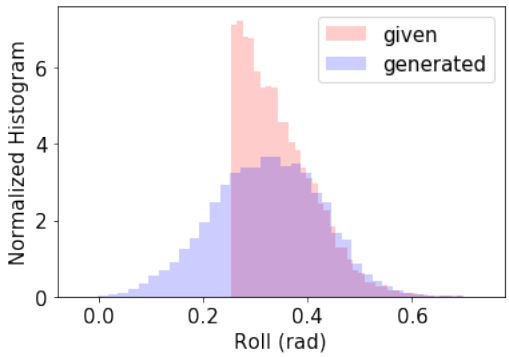

Figure 3.74: Compare the distribution of resultant rolls between the given dataset (red) $\mathcal{D}_1$ and the generated dataset (blue) $\mathcal{D}_1'$

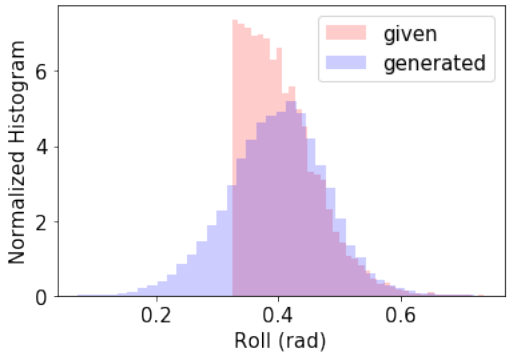

Figure 3.75: Compare the distribution of resultant rolls between the given dataset (red) $\mathcal{D}_2$ and the generated dataset (blue) $\mathcal{D}_2'$

which means the model becomes more likely to generate large resultant roll angles at the design time.

The generated datasets during the bootstrapping procedure are then filtered and compared against the MCS result. For the generated dataset $\mathcal{D}_1'$, the phase sequences leading to $\theta(t_d)$ larger than 0.4 rad are kept and compared against the MCS result in Figure 3.80. The generated loading environments are then used to simulate the roll motion. Figure 3.81 shows the time ensemble of the exciting moment (top) and the corresponding roll angle (bottom). Comparing to the MCS time-domain results in Figure 3.65, both the generated loading environments the corresponding roll response have a good match and large roll which exceeds the threshold $\theta(t_d) > 0.4$ occurs at the design time labeled by a red dash line.

Similarly, the generated dataset $\mathcal{D}_3'$ is filtered by roll angle threshold $\zeta_{\theta(t_d)} = 0.5$ rad and compared against the MCS result in Figure 3.82. As shown in the comparison, the filtered generated dataset has a good match with the MCS result. The

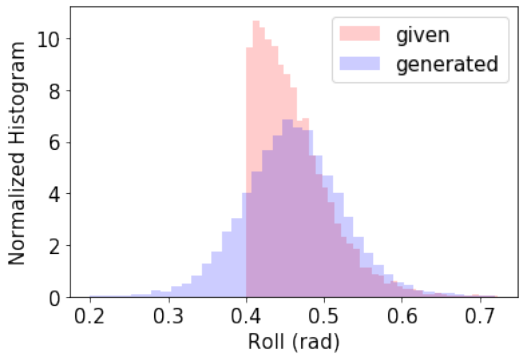

Figure 3.76: Compare the distribution of resultant rolls between the given dataset (red) $\mathcal{D}_3$ and the generated dataset (blue) $\mathcal{D}'_3$

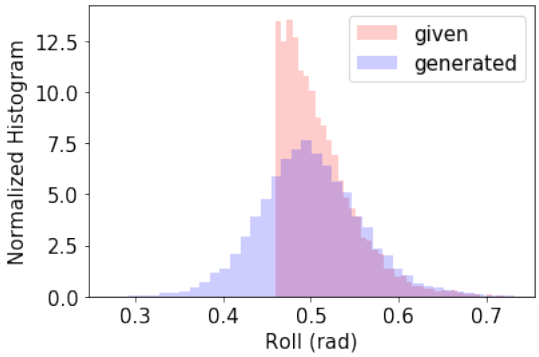

Figure 3.77: Compare the distribution of resultant rolls between the given dataset (red) $\mathcal{D}_4$ and the generated dataset (blue) $\mathcal{D}'_4$

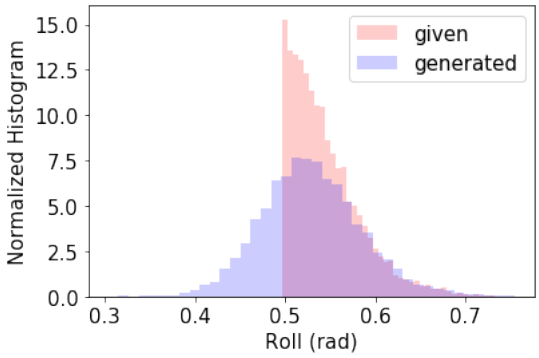

Figure 3.78: Compare the distribution of resultant rolls between the given dataset (red) $\mathcal{D}_5$ and the generated dataset (blue) $\mathcal{D}'_5$

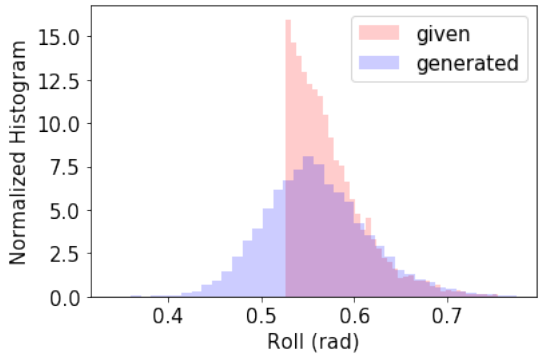

Figure 3.79: Compare the distribution of resultant rolls between the given dataset (red) $\mathcal{D}_6$ and the generated dataset (blue) $\mathcal{D}'_6$

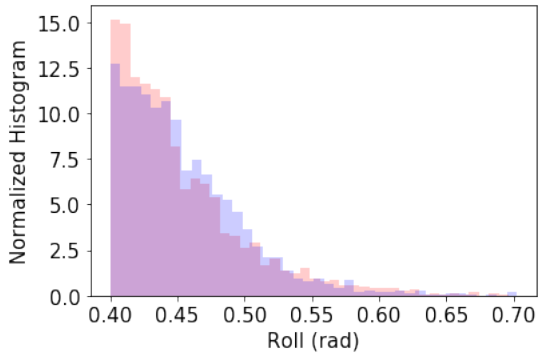

Figure 3.80: Compare the distribution of $\Pr(\theta(t_d)|\theta(t_d) > 0.4)$ between the filtered generated dataset (blue) and the MCS result (red)

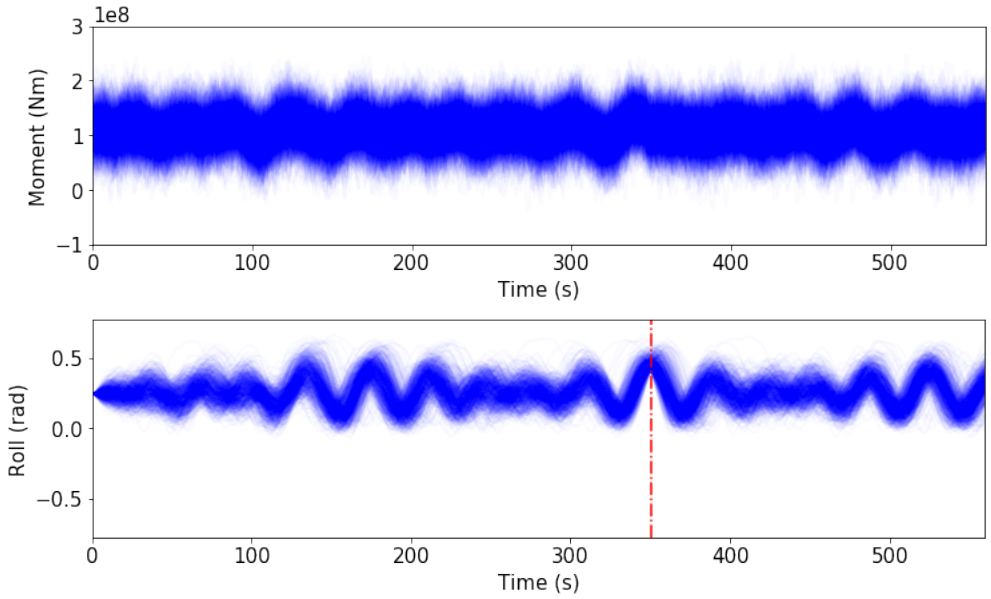

Figure 3.81: Simulation with the generated loading environment, external moment (top) and roll response (bottom) filtered with $\theta(t_d) > 0.4$ rad

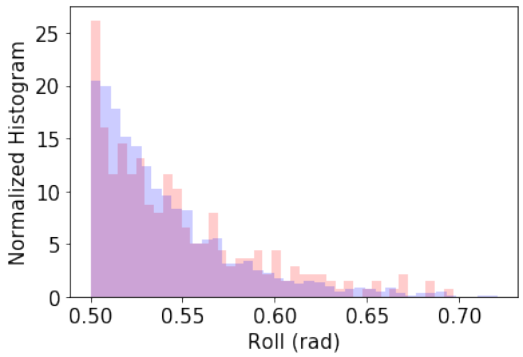

Figure 3.82: Compare the distribution of $\Pr(\theta(t_d)|\theta(t_d) > 0.5)$ between the filtered generated dataset (blue) and the MCS result (red)

generated loading environments are then also used to simulate the roll motion. Figure 3.83 shows the time ensemble of the exciting moment (top) and their roll angle (bottom). Comparing to the MCS results in Figure 3.66, the generated environments and corresponding rolls have similar patterns and large roll ($\theta(t_d) > 0.5$ rad). Moreover, the model can efficiently generate conditional loadings. Compared to the MCS results where a lighter time ensemble is plotted due to limited data, more realizations become possible for the proposed model, and therefore a denser and richer time ensemble is plotted in the figure.

For dataset $\mathcal{D}_6'$, the phase sequences leading to $\theta(t_d) > 0.6$ are kept and their corresponding $\theta(t_d)$ are plotted as a histogram in Figure 3.84. Since the threhold $\theta(t_d) > 0.6$ is high and very few of simulation ends up roll angle larger than the threshold, the histogram is not compared against the MCS results.

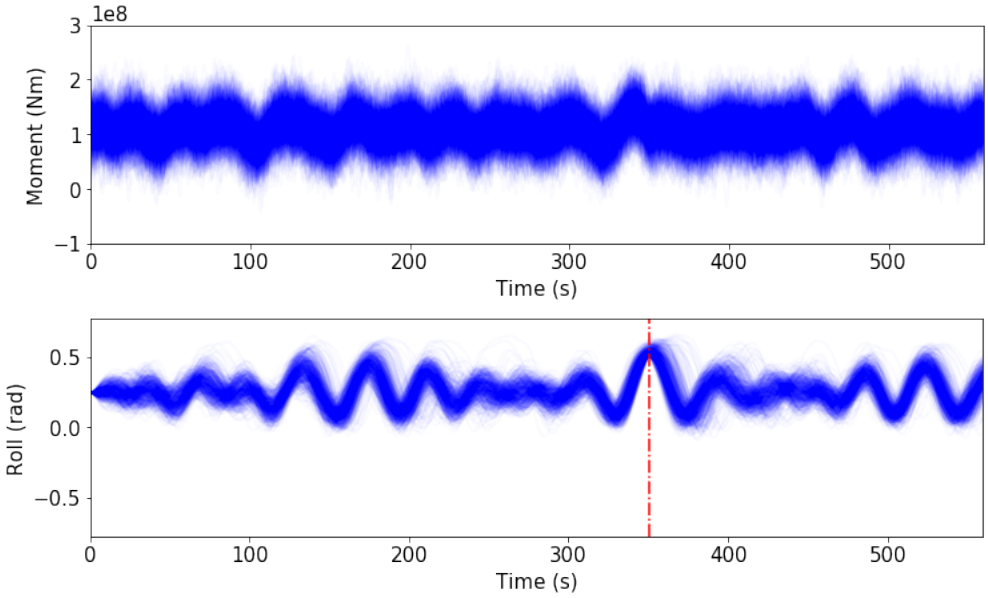

Figure 3.83: Simulation with the generated loading environment, external moment (top) and roll response (bottom) filtered with $\theta(t_d) > 0.5$ rad

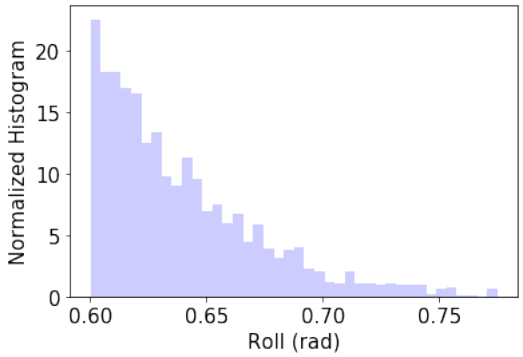

Figure 3.84: Histogram of resultant roll angle at the design time, $\Pr(\theta(t_d)|\theta(t_d) > 0.6)$

CHAPTER IV

Module II - Design Response Estimator

In this chapter, a data-driven module named Design Response Estimator (DRE) is proposed to quickly predict the system output based on the system input. The module is useful when the user lacks mathematical models that can describe the complex nonlinear relationship in the marine systems. Thanks to the DRE module, large amounts of data for a complex marine system can be quickly produced for TEG use. Machine learning methods, especially a LSTM architecture, are used in the DRE module. The example marine systems discussed in Chapter III are successfully identified by the DRE module. Moreover, another two Computational Fluid Dynamics (CFD) examples, tank sloshing and a floating object in waves, are also used to illustrate how the DRE model can be applied.

4.1 Motivation

Data-driven models usually require a certain quantity of data to produce a model of the desired accuracy. Generative models like TEG usually requires more data since the distributions that they learn often have large dimensions. Enough training data for the TEG module has to be prepared by evaluating the system response under various ocean environments. For simple systems whose dynamics have been described often by their state-space equations, evaluation of the response under prescribed loading environments is often fast. However, for complex systems where high-fidelity simulations are often involved, evaluating the response may be expensive. For example, solving the response of a ship in waves via CFD can take days for a simulation time window of minutes. Hence the preparation of enough scenarios with different environments for model training becomes unaffordable. On the other side, high-fidelity simulations are valuable because modeling extreme events of nonlinear systems requires accurate information about the response. Therefore, for complex

systems, building models that approximate their dynamical behavior is valuable, as long as the approximation properly represents the nonlinearity in the extreme state. Moreover, it helps to better understand the statistical behavior of the system by running long-time simulations using the faster-running model, rather than being limited to the quantity of data.

Building models for complex systems is often challenging. Reduced-order models are often achieved by leveraging many assumptions to simplify the complex dynamical behavior of the systems. For example, quasi-static or linear assumptions are often used when the studied response is typically small so that the higher-order terms are negligible. Due to the strong assumptions leading to simplified equations, closed-form conclusions are sometimes available. For highly nonlinear systems, such assumptions might fail to present important nonlinear behaviors including but not limited to the dependency of initial conditions and chaos. Hence, more advanced mathematical models like nonlinear ODEs or Partial Differential Equation (PDE)s are often selected, and numerical simulations of mathematical models are used to explore the response. However, there exist many real-world examples where mathematical models are often unaccessible and only observation of the system response is possible. Successful characterization of the system based on observation becomes crucial to system analysis and design. A data-driven model can also be helpful when the existing method is computationally expensive and the user is only interested in partial results from the simulation. As an example, when the user would like to know the ship response under various wave environments, a CFD simulation is often used to calculate the whole flow field, including pressure, velocity for all discretized fluid cells to predict its response. However, if the user is only interested in the ship's motion, most of the computation is unused. A more efficient approach is to directly map the wave environments to the ship response without calculation of the entire fluid domain.

The process to characterize the system given its inputs and outputs is called System Identification (SI). The objective of SI is to build a model that can be used to predict the system outputs under new inputs. Figure 4.1 shows the flowchart for a SI problem. Many studies have been developed in nearly all engineering fields. Semi-experimental models combine parameters and existing knowledge about the system. The model parameters are estimated based on the observed inputs and outputs and determined by solving an optimization problem. Models built from SI have been successfully applied to real-world systems for decades. However, many models require users' intuition, domain knowledge, and intrinsic familiarity about the system. A new

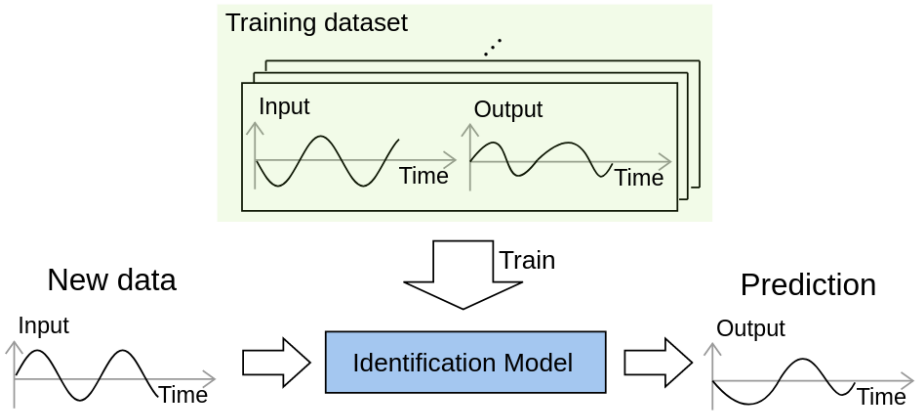

Figure 4.1: System Identification. Train a parametric model based on observed data and predict on new inputs

family of models that utilize deep learning has recently emerged to accomplish this task since their powerful and efficient model architectures outperform other existing methods in solving the SI problems. Moreover, new methods are so general that users need much less information about the systems to build the model.

In the module presented in this chapter, the LSTM model is introduced to solve the SI problem. Many seakeeping dynamical and fluid-related problems are found to be solved successfully with little cost. Once the model is built, it can be used as an alternative to the original system for simulation usage. The cost of conducting simulations with the new model is found to be much less than compared to the original system. In the next section, details about the proposed method are discussed including problem formulation, data preparation, model architecture, training, and inference. The proposed method is supported by many examples from simple surrogate problems like linear waves to more complicated scenarios like CFD simulations. Table 4.1 gives more details about what input signals and output signals are for each example system.

4.2 Methodology

4.2.1 Regression Model

The objective of this module is to build a model that can predict the system output given the system input. For a causal system, the current output depends only on the input up to and including the current timestamp. Similar to the discrete state-space representation, let x_t and y_t be the input and output signals at time index

Example	Input	Output
Linear wave propagation	upstream wave elevation	downstream wave elevation
Nonlinear wave propagation	upstream wave elevation	downstream wave elevation
Nonlinear ship roll	external moment	roll angle
Sloshing tank (CFD)	tank roll angle	resultant force and moment
Floating body in irregular waves (CFD)	upstream wave elevation	pitch angle

Table 4.1: Examples of applying the proposed method to marine systems

t respectively. A simple general equation that governs a discrete dynamical system is

$$s_t = f(s_{t-1}, s_{t-2}, \cdots ; x_t, x_{t-1}, \cdots) \tag{4.1}$$

$$y_t = g(s_t, s_{t-1}, s_{t-2}, \cdots) \tag{4.2}$$

where f and g are nonlinear mappings and s is the state variable.

The state variable for a 1-DoF simple spring-mass-damper system can be position and velocity. However, a more general example can be a ship with 6-DoF motion in waves. When only the heave response is collected, recorded, and asked to be predicted, the state variable can be far more complex for the coupled nonlinear system. Possible states may include input or response history, and any function of the history (polynomial, sinusoidal, etc.). It is the machine learning model's job to learn what measurements to include in s, how to transfer s from time $t-1$ to t, and the best nonlinear mappings f and g. Hence, the machine learning algorithm takes in the input series $(x_1, \cdots, x_T)$ and the output series $(y_1, \cdots, y_T)$, and produces out the trained mapping f, g, and s. Once the model training is finished, it can predict the system output series based on given new input series even with time length different from that used in the training time series.

Predicting output time series in the form of a real number sequence $(y_1, \cdots, y_T)$ is a regression problem. The model parameters are learned to minimize the difference between the ground-truth sequence $(y_1, \cdots, y_T)$ and the sequence prediction $(\hat{y}_1, \cdots, \hat{y}_T)$. As seen in the TEG module, the LSTM network is a reasonable choice to conduct sequence modeling, though necessary changes are needed.

4.2.2 Data Preparation

To train the model, both input and corresponding output in the format of discrete-time series are prepared. It is suggested to have the training data in a similar sea state with the test data. Otherwise, response behavior under one loading condition might be different from another loading condition. As a result, the given output in the training dataset provides limited and less information about the dynamical behavior under the test condition. Some simulations or experiments are conducted to collect the input-output pairs of time series. The input matrix and the output matrix therefore have the shape of (M, T) where M is the number of collected time series and T is the sequence length of the discrete-time series. For many cases, the input and output signals are stationary time series, which makes data normalization easier. Let μ_x and σ_x be the mean and standard deviation for the input series and let μ_y and σ_y be the mean and standard deviation for the output series. After standardizing each time series using the respective μ and σ, the training dataset consisting of the input matrix X and the output matrix y is formed as

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1T} \\ x_{21} & x_{22} & \cdots & x_{2T} \\ \vdots & \vdots & \vdots & \vdots \\ x_{M1} & x_{M2} & \cdots & x_{MT} \end{bmatrix} \tag{4.3}$$

$$y = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1T} \\ y_{21} & y_{22} & \cdots & y_{2T} \\ \vdots & \vdots & \vdots & \vdots \\ y_{M1} & y_{M2} & \cdots & y_{MT} \end{bmatrix} \tag{4.4}$$

As a general reference, Table 4.2 lists the dimensions of X or y for the example cases. All time series are discretized uniformly with constant sampling frequency. The length of the discrete sequence is given in the third column, labeled by T. Each simulation is conducted with time length measured in terms of the peak period of the input environment, T_p.

The collected dataset is split into training (80%) and test (20%) datasets. The model is then trained using the training dataset and the model performance on the test dataset is evaluated and reported.

Example	#Time Series	#Time Steps	time length (s)
Linear wave propagation	100	400	240
Nonlinear wave propagation	100	400	240
Nonlinear ship roll	10	800	560
Sloshing tank (CFD)	50	800	40
Floating object in irregular waves (CFD)	50	400	20

Table 4.2: Dimensions of X or y dataset for various examples

Layer (type)	Output Shape
lstm_1 (LSTM)	(None, H)
lstm_2 (LSTM)	(None, H)
lstm_3 (LSTM)	(None, H)
lstm_4 (LSTM)	(None, H)
lstm_5 (LSTM)	(None, H)
dense_1 (Dense)	(None, 1)

Table 4.3: Model architecture and output tensor shape

4.2.3 Model Architecture

The model is built by stacking multiple LSTM layers vertically. Figure 4.2 shows a network consisting of 5 layers in folded and unfolded view. The length of the input sequence equals the length of the output sequence. The output from the top LSTM layer is then connected to a dense layer with linear activation functions. Table 4.3 lists the output shape after each layer, where H is the number of hidden state units. "None" in the output shape allows an arbitrary number of time steps in the time series. A typical number of hidden units, H, is from 20 to 100 and $H = 50$ is used for all examples discussed later.

4.2.4 Model Training

Model training is an optimization problem, which means finding the model parameters that minimize the loss (or cost) function between the prediction and the ground-truth reference. Since each element in the output sequence y is real-valued, the loss function for this regression task can be means of squared error measure averaged over all time steps.

$$L(\hat{y}, y) = \frac{1}{T}\sum_{t=1}^{T}(\hat{y}_t - y_t)^2 \tag{4.5}$$

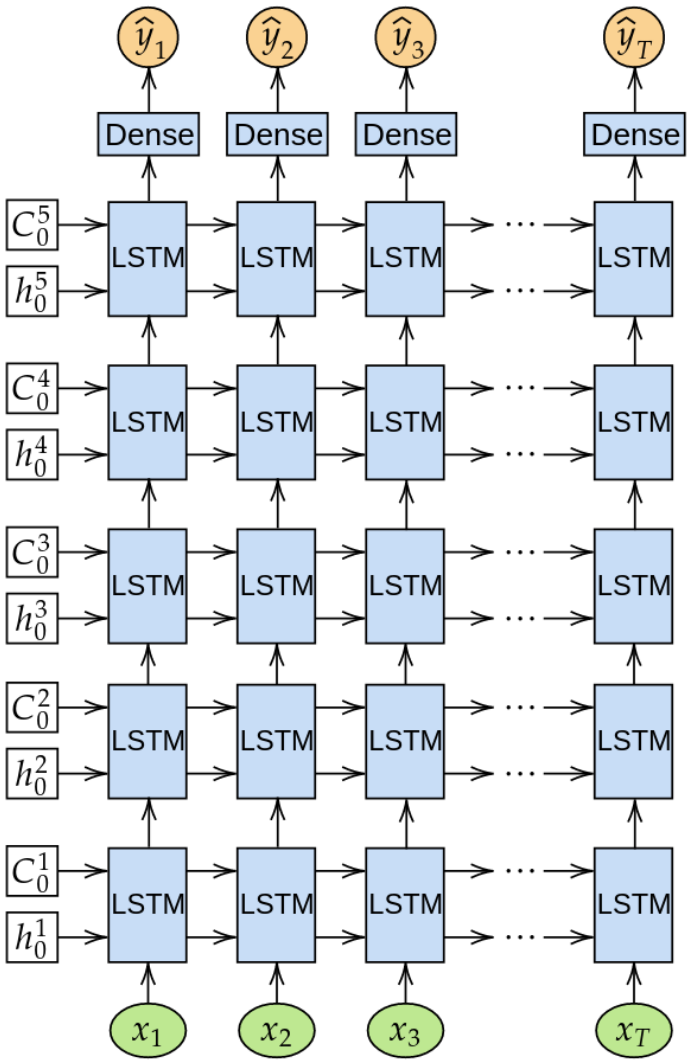

Figure 4.2: DRE Architecture

where $\hat{y}$ is the prediction of the output sequence and y is the ground-truth sequence.

Therefore, the training process consists of the following steps to update the parameters of the model.

1) Forward propagation. The model computes from the input layer towards the output layer for a batch of time series based on the current parameters.

2) Compute loss. The outputs from the model are compared to the provided ground-truth sequences and the loss for the batch is calculated.

3) Backward propagation. The model computes the loss derivative with respect to each parameter of the model. This computation is from the output layer to the input layer.

4) Update parameters. The model updates its parameters based on the loss derivatives calculated above.

The parameters are updated in form of gradient descent,

$$w \leftarrow w - \alpha f_w(\frac{\partial L}{\partial w}) \tag{4.6}$$

$$b \leftarrow b - \alpha f_b(\frac{\partial L}{\partial b}) \tag{4.7}$$

where w and b are weights and biases parameters respectively, $\frac{\partial L}{\partial w}$ and $\frac{\partial L}{\partial b}$ are the derivatives of the loss with respect to the parameters, and α is the learning rate. Specific forms of f_w and f_b depends on the optimizer and again, the *Adam* optimizer (*Kingma and Ba*, 2014) is used for the current examples.

Since the number of time series in the training dataset M is relatively small from 10 to 100, the batch size is selected to be M, which is also called Batch Gradient Descent. With batch size M, the number of epochs equals to the number of iterations during training. For all current examples, 500 iterations with the learning rate $\alpha = 0.001$ are typically enough for the loss to converge.

4.2.5 Model Inference

Once the model is trained, it can predict the output sequence given any new input sequence, following the same procedure from the forward propagation. The prediction and the ground-truth output sequences are compared and error is computed. The model performance is then evaluated based on the error for both the training dataset and the test dataset.

Symbol	Description	Value
H_s	significant wave height	11.2 m
T_p	peak period	12 s
$T_{\min}$	smallest wave period after cutoff	8.075 s
$T_{\max}$	largest wave period after cutoff	17.63 s
D	water depth	30 m
$\lambda_{\min}$	shortest wave length after cutoff	97.61 m
$\lambda_{\max}$	longest wave length after cutoff	282.83 m
$c_{p,\min}$	smallest phase velocity	12.09 m/s
$c_{p,\max}$	largest phase velocity	16.04 m/s
$x_f = 2\lambda_{\max}$	distance between probes	565 m
$T_s = T_p/20$	sampling period	0.6 s
$T = 20T_p$	simulation window length	240 s

Table 4.4: Linear Wave Environment Specifications

4.3 Example: Linear Wave

The first example shows applying the LSTM network to characterize the 2D linear wave propagation system as a surrogate problem. Two wave probes are placed a distance apart to record the wave elevation time series. The upstream wave elevation is regarded as the system input and the downstream wave elevation is regarded as the system output. The elevation of the irregular wave is a Gaussian process and evaluated by

$$\eta(x,t) = \sum_{i=1}^{N} a_i \cos(k_i x - \omega_i t + \phi_i) \tag{4.8}$$

where N is the number of Fourier components and, a_i, k_i, ω_i and ϕ_i are wave amplitudes, wavenumbers, wave angular frequencies, and wave phases respectively. Table 4.4 lists the wave environment specifications.

The distance between two probes is $2\lambda_{\max} = 565$ m. By uniformly and identically sampling the phase angle ϕ_i from $-\pi$ to π, a 100 wave scenarios are simulated. Therefore, 100 input-output pairs of time series are collected. Each time series has time window length of 240 s and is sampled by period of 0.6 s. The input dataset X and output dataset y has the shape of $(100, 400)$. Figure 4.3 shows one pair of input (upstream elevation) and output (downstream elevation) time series. Both input and output elevation series are then normalized by the mean and standard deviation, and then are split into a training dataset $(X_{\text{train}}, y_{\text{train}})$ and a test dataset $(X_{\text{test}}, y_{\text{test}})$. The model is trained on the training dataset for 500 iterations with the training history plotted in Figure 4.4.

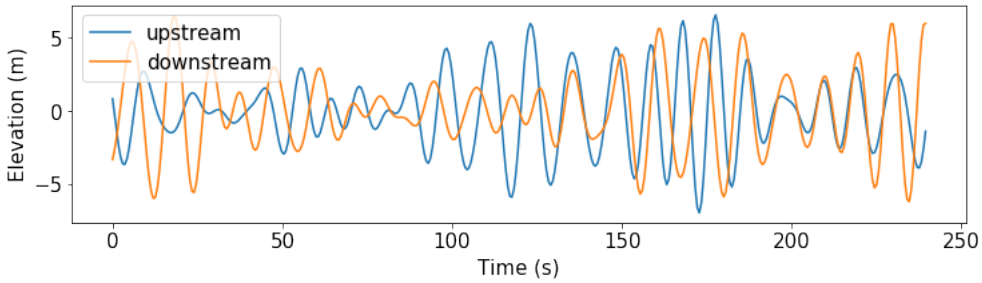

Figure 4.3: Linear wave: one pair of input (upstream elevation) and output (downstream elevation) time series

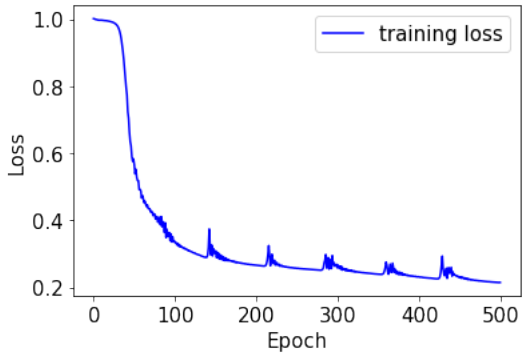

Figure 4.4: Linear wave: loss on the training dataset decreases during model training

The performance of the trained model is evaluated by asking the model to predict outputs based on inputs for both the training and test datasets. Three input examples are selected from both the training dataset and the test dataset, and the model predictions are compared against the ground-truth in Figure 4.5 (performance on the training dataset) and Figure 4.6 (performance on the test dataset). As shown in the Figure 4.6, the trained model succeeds in predicting the system behavior for new wave scenarios except for the transient at the beginning.

Generally speaking, the prediction of the output is close to the ground-truth after $t = 45$ s or so. The mismatch between the prediction and the ground-truth is expected and can be explained in two perspectives. From the time domain point of view, the situation happening at the upstream takes time to propagate to the downstream location. Therefore, the current elevation at the upstream affects the downstream elevation in the future. Moreover, the time spent to propagate the fresh information from the upstream to the downstream is relative to the phase velocity, which is calculated in the Table 4.4. The time delay is estimated by $x_f/c_{p,\min} = 46.7$ s for the slowest phase velocity to travel the distance between the probes. The mismatch at the beginning can also be explained from the frequency point of view. Since the wave

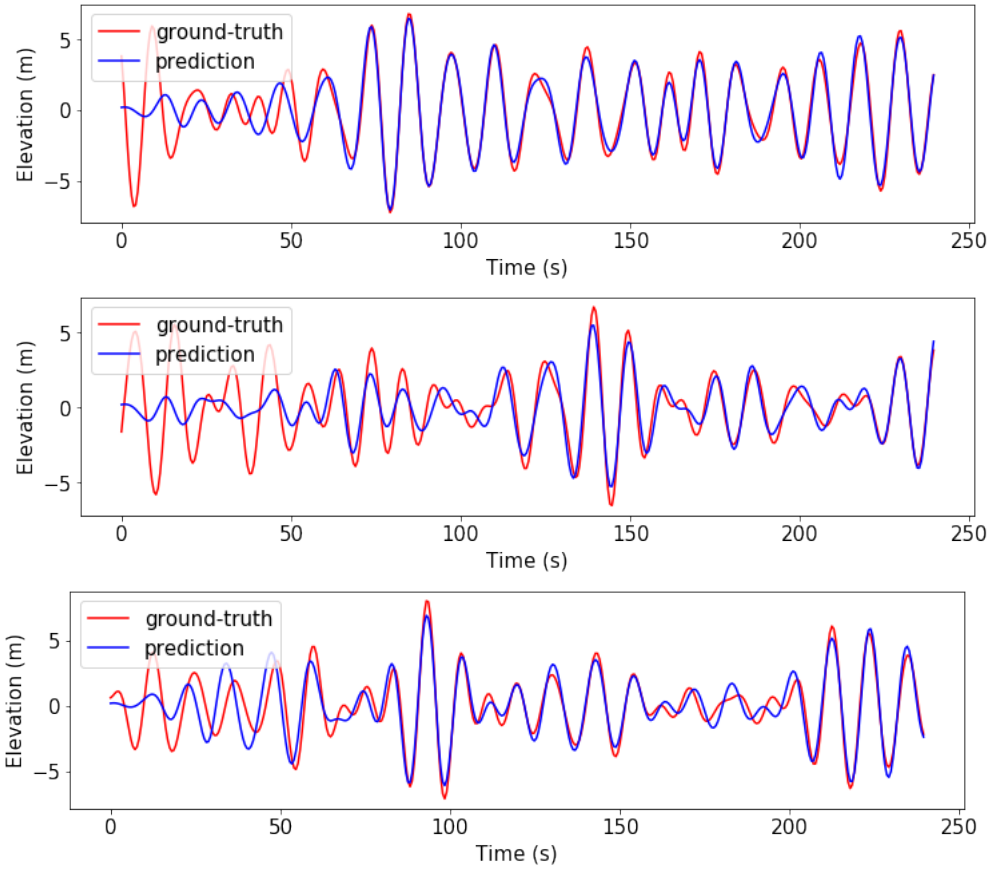

Figure 4.5: Linear wave: compare predictions (blue) of 3 training outputs and their ground-truth (red)

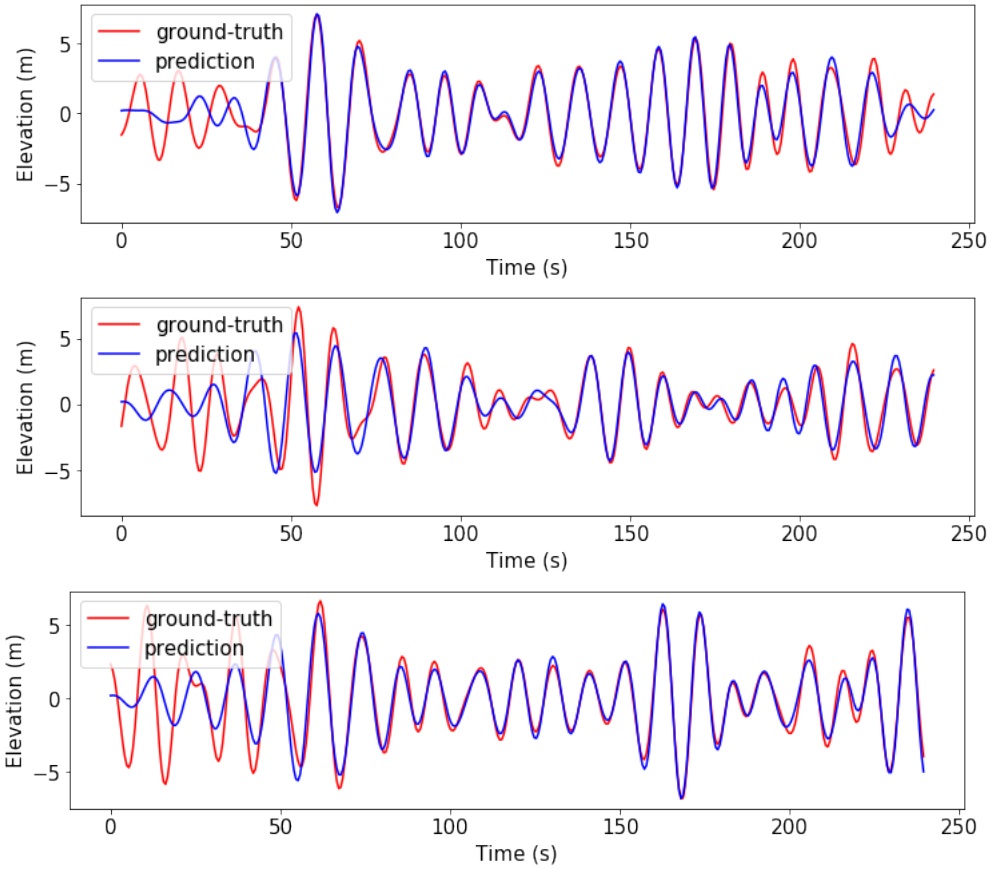

Figure 4.6: Linear wave: compare predictions (blue) of 3 test outputs and their ground-truth (red)

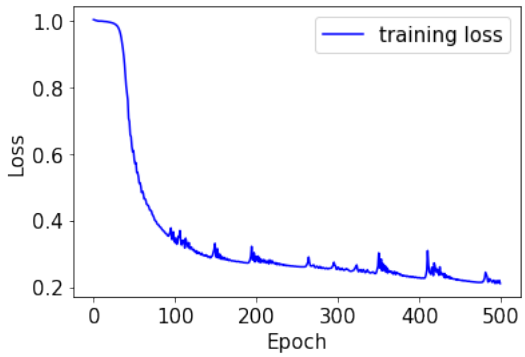

Figure 4.7: Nonlinear wave: loss on the training dataset decreases during model training

elevation is evaluated by the Fourier series, the output prediction requires a specific length of input clip before gathering frequency-domain information. This means the model is useful after the transient window.

4.4 Example: Second-Order Nonlinear Wave

The same environment in the linear wave example is reused with the only difference being the system is now a second-order nonlinear wave. In the Section 3.5, the second order-nonlinear correction of the elevation is presented considering the interaction among various wave frequencies. The probe distance is the same as the linear wave example. Again, the system input is the upstream elevation and the system output is the downstream elevation.

With the same architecture, the model can predict the output after 500 training steps. Figure 4.7 shows the loss on the training data decreases during the training process. The performance of the trained model is evaluated for both training and test datasets. Figure 4.8 compares predictions of 3 training outputs and their ground-truth. Figure 4.9 compares predictions of 3 test outputs and their ground-truth. As shown in the figures, a similar mismatch is observed at the beginning of the output series and it can be explained with the same analysis presented in the linear wave example. The general matching between the prediction and the ground-truth validates that the nonlinear wave propagation is successfully identified.

In the linear and nonlinear wave propagation examples, the downstream elevation is nonzero when $t = 0$, which is equivalent to a nonzero initial condition of the system. Hence, a difference between the predicted output and the ground-truth is observed at the beginning of time. For a system starting from rest or with known initial

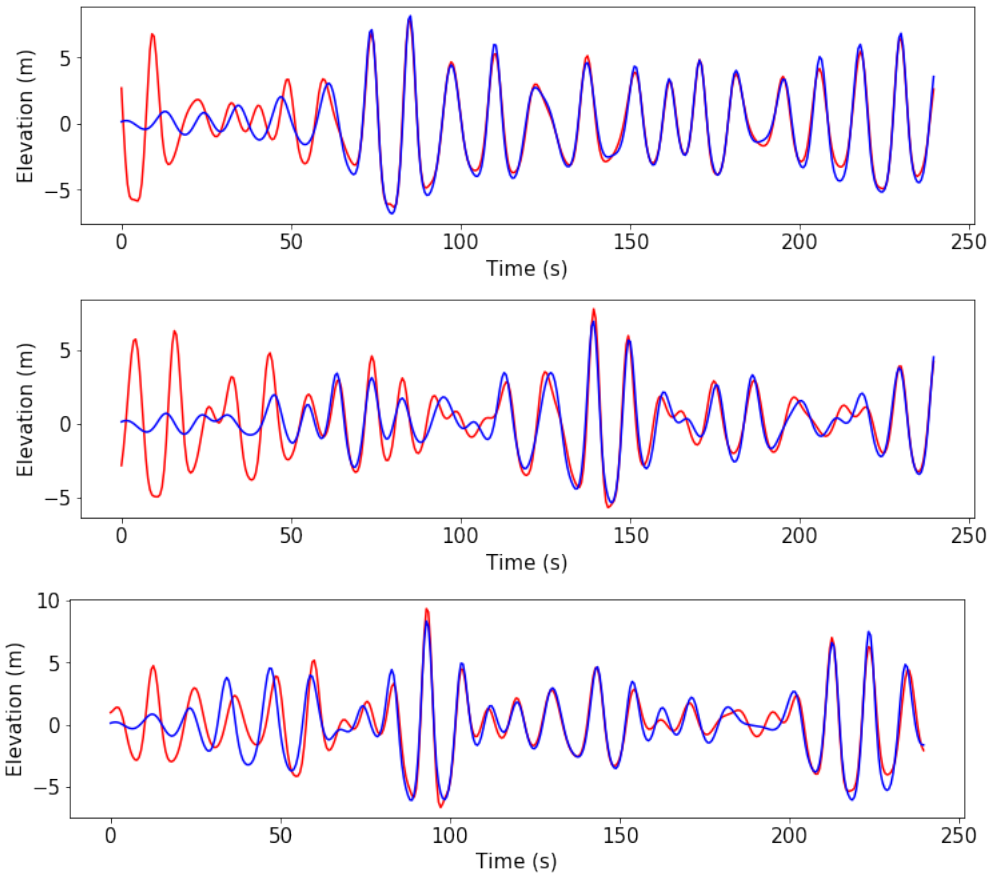

Figure 4.8: Nonlinear wave: compare predictions (blue) of 3 training outputs and their ground-truth (red)

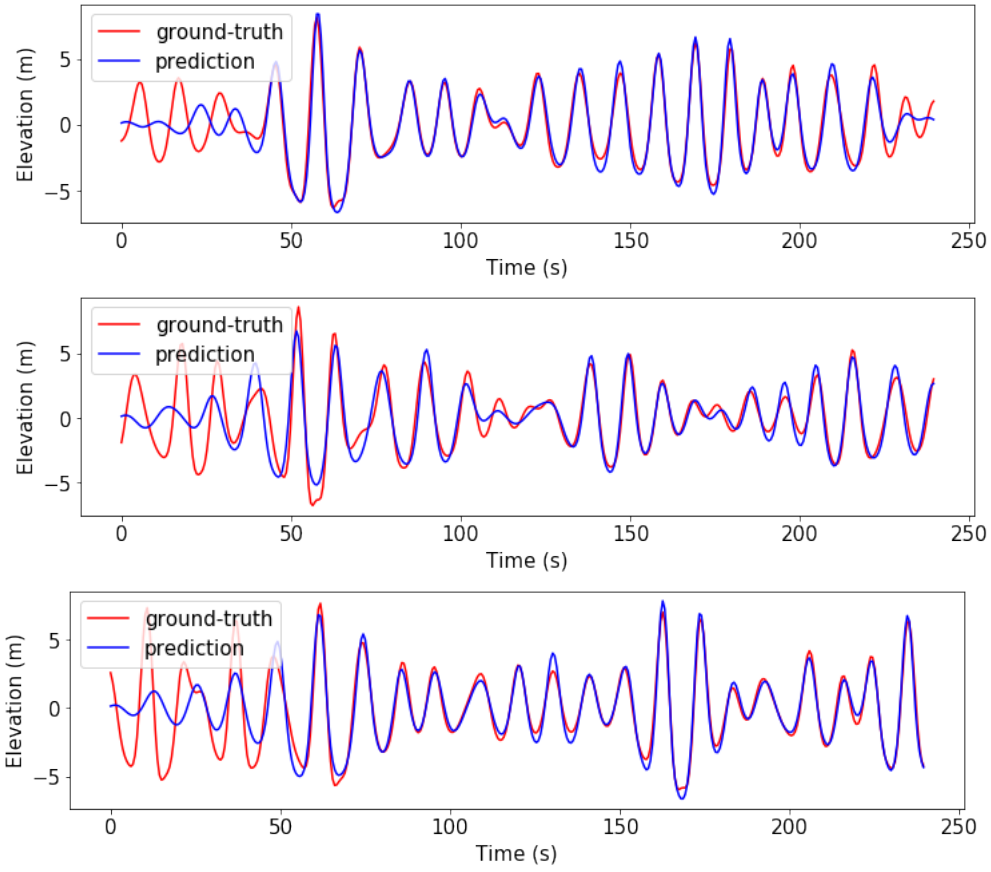

Figure 4.9: Nonlinear wave: compare predictions (blue) of 3 test outputs and their ground-truth (red)

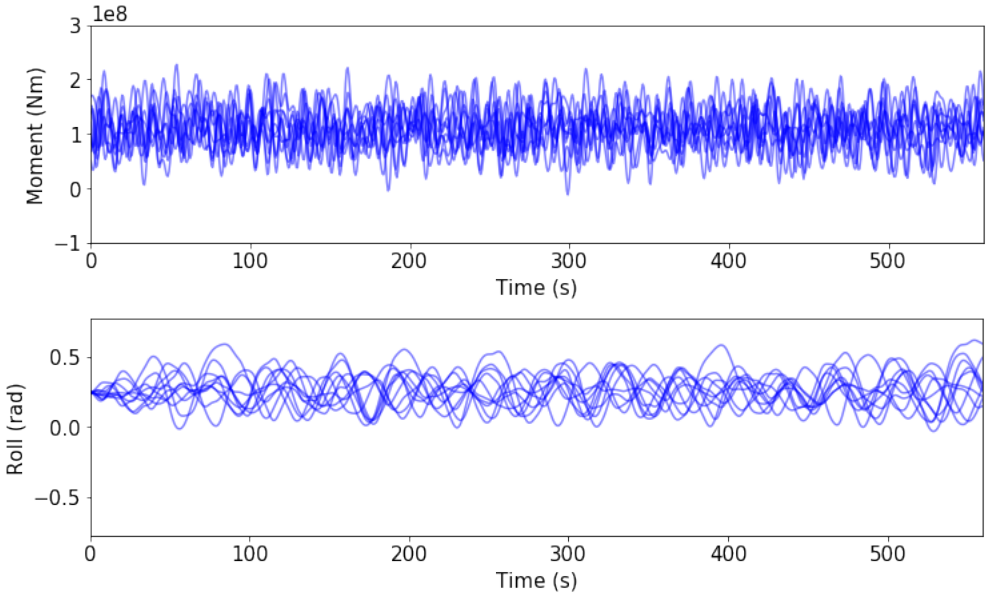

Figure 4.10: Simulation results: 10 external moment series (top) and 10 roll response series (bottom)

conditions, the performance of prediction at the transient time should be improved. An example with known initial conditions can be a ship roll used in Section 3.6 and is identified in Section 4.5.

4.5 Example: Nonlinear Ship Roll

The Ship roll angle is governed by the ODE in Equation 3.62. From the system dynamic perspective, the system input is the external rolling moment $M(t) = M_{\text{air}}(t) + M_{\text{wave}}(t)$ and the system output is the roll angle of the ship, $\theta(t)$. A random seed $(\phi_{\text{wind}}, \phi_{\text{wave}})$ is sampled uniformly and identically from $-\pi$ to π, producing a Gaussian external moment series with nonzero mean. The roll angle is simulated by integrating the ODE using a Runge-Kutta scheme. There are 10 realizations simulated, producing 10 pairs of $M(t), \theta(t)$. Refer to Section 3.6 for the specification of the ship. The simulation window is 560 s, or 16 natural periods T_0 of roll, and is sampled with sampling period $T_0/50$. Figure 4.10 shows the 10 external moment series (top) and their 10 roll response series (bottom).

The external moment and roll response are normalized by their mean values and standard deviations. For the dataset split, 8 input-output pairs are used as the training dataset and 2 input-output pairs are used as the test dataset. Figure 4.11 plots the loss on the training dataset versus training epochs during 500 iterations.

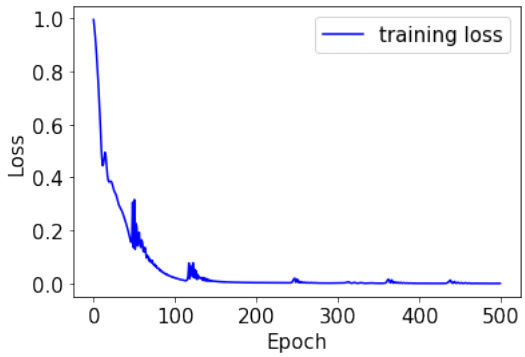

Figure 4.11: Ship roll: loss on the training dataset decreases during model training

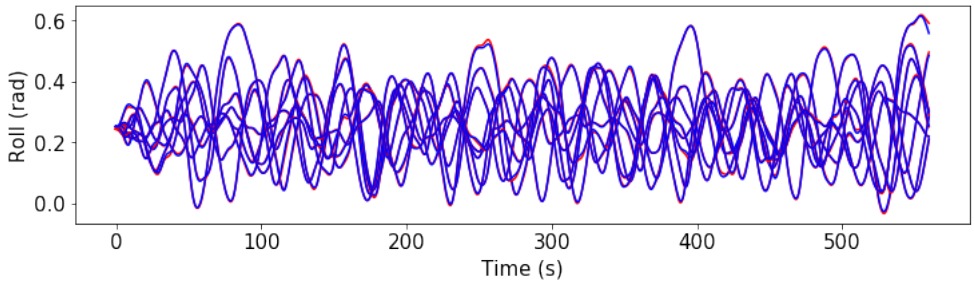

Figure 4.12: Ship roll: compare predictions (blue) of rolls and their ground-truths (red) for the training series

The trained model is asked to predict all 10 roll series given their external moment series. For the training data, Figure 4.12 compares the prediction of roll and the ground-truth. For the test data, Figure 4.13 compares the prediction of roll and the ground-truths.

As shown in the comparison, the prediction is almost identical to the ground-truth when predicting the system output. It is also worth noting in this example only a small amount of data is needed to train the model with good prediction performance.

4.6 Example: Sloshing Tank

This example uses a 2D sloshing tank to show the effectiveness of the model in characterizing complex systems. A water tank in Figure 4.14 is rolling with user-defined motion and the forces produced by the water on the tank wall are collected. Hence, the system input is the time series of the roll angle and the output is the hydrodynamic force and moment. The roll angle $\theta(t)$ is generated by sampling a zero-mean Gaussian process with a JONSWAP spectrum shown in Figure 4.15. The peak period of the process is 2 s and the height of the spectrum is set such that rolling

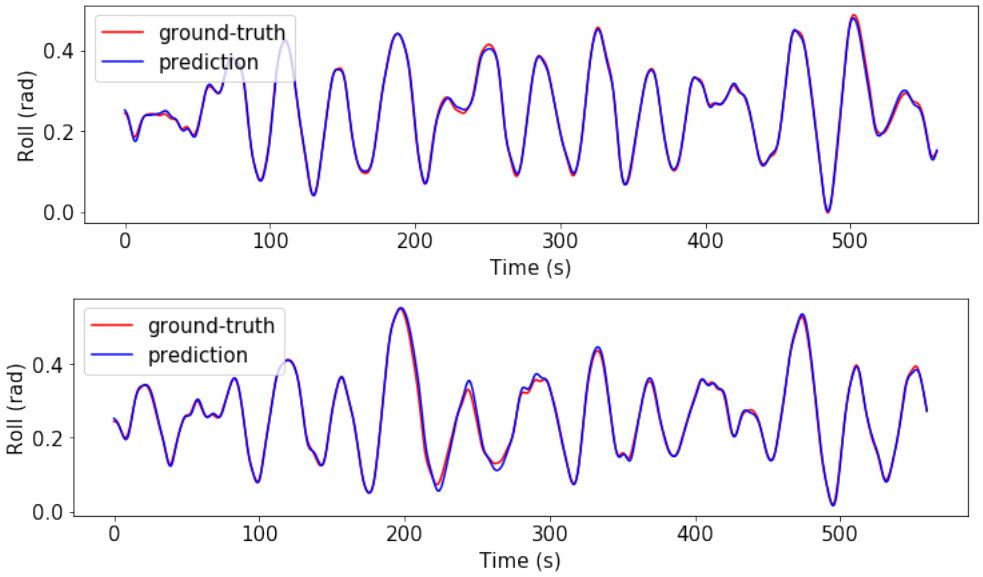

Figure 4.13: Ship roll: compare predictions of rolls and their ground-truths for test series

standard deviation σ_θ is about 10 deg. Figure 4.16 shows one sampled roll motion $\theta(t)$ with length of 40 s.

The system is complex due to the complex air-water interaction phenomenon inside the tank. OpenFOAM, an open-source CFD simulation tool, is used to simulate the fluid motion inside the tank. The water flow is assumed to be laminar and its fields are solved by the provided multiphase solver, interFoam. A coarse grid is used and grid refinement study is neglected since the current grid is enough to represent irregularity and nonlinearity of the system. Figure 4.17 shows 4 snapshots during its motion. As shown in the snapshots, complex fluid motion happens inside the tank and makes its dynamics complicated. Postprocessing is conducted to calculate hydrodynamic forces and moment on the wall, F_Y, F_Z, M_X. Figure 4.18 shows the hydrodynamic forces and moment versus time for the roll input in Figure 4.16.

Fifty simulations under different roll time histories are conducted. The simulation results $\theta(t); F_Y(t), F_Z(t), M_X(t)$ then form the dataset for the modeling task. Each time series is normalized by $(\mu_\theta, \sigma_\theta)$, $(\mu_{F_Y}, \sigma_{F_Y})$, $(\mu_{F_Z}, \sigma_{F_Z})$, $(\mu_{M_X}, \sigma_{M_X})$. The dataset is split with a ratio of 80%:20% to form the training dataset and the test dataset, respectively. Each output dimension is treated separately and three models are built independently. Hence, three models predict the force series in Y direction, the force series in Z direction, and the moment series in X direction. All three models are trained using the training dataset for 500 epochs (or iterations). Figure 4.19 shows

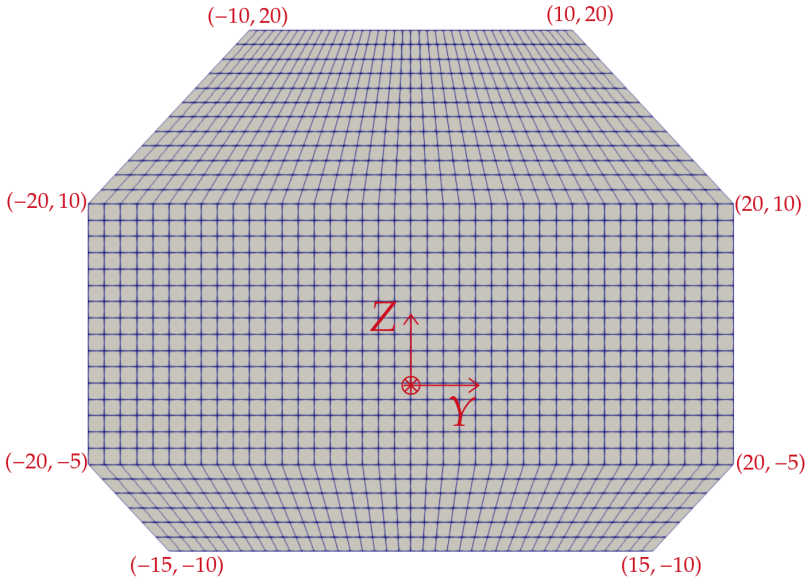

Figure 4.14: Water tank geometry and computation grid

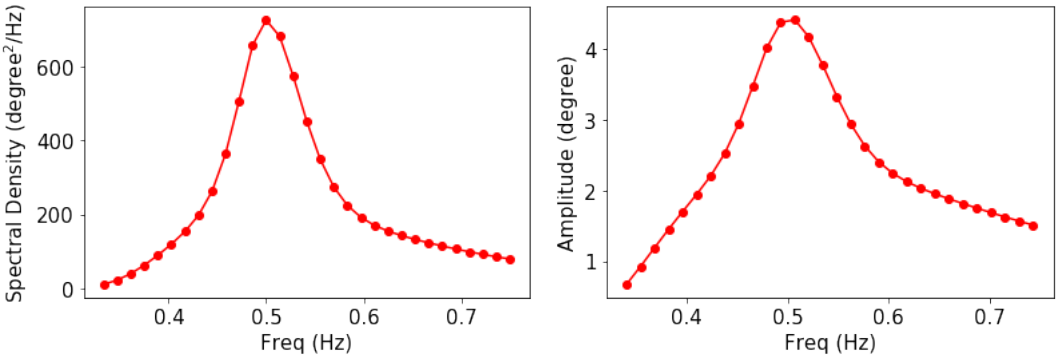

Figure 4.15: Roll angle spectrum

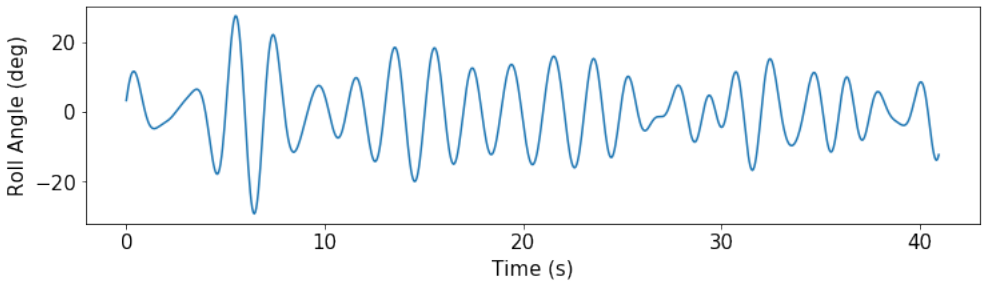

Figure 4.16: One sampled roll series $\theta(t)$

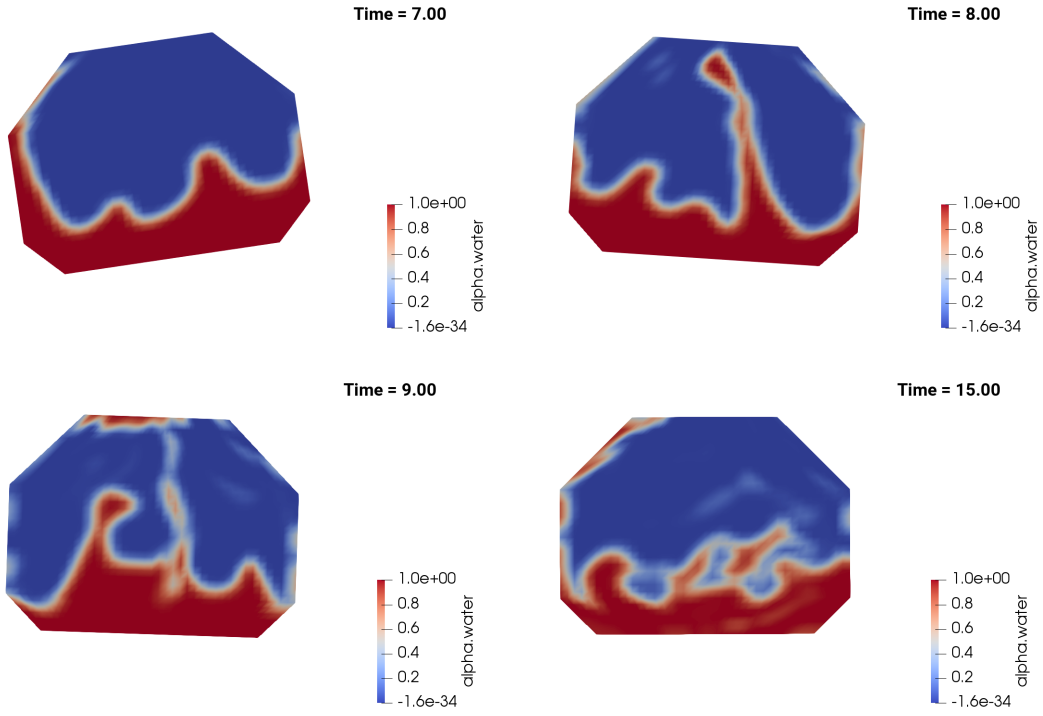

Figure 4.17: Snapshots of water in tank at time = 7, 8, 9, 15 s

their training histories.

After the loss on the training data converges after 500 iterations, all three models are asked to predict the outputs using new input series from the test dataset. Therefore, each model performs 10 (20% of 50 simulations) predictions, and the predictions are compared with the simulation results, which are regarded as the ground-truth. As an example, predictions of $F_Y(t)$, $F_Z(t)$, and $M_X(t)$ for one test input are compared with their ground-truth in Figure 4.20. As shown in the figure, all three models give a reasonable prediction for the new inputs. Generally speaking, the model of $M_X(t)$ has better performance in predicting the ground-truth than the other two models. This might be due to the more smooth response of $M_X(t)$ compared to $F_Y(t)$ and $F_Z(t)$.

It is also possible to build a combined model to forecast $F_Y(t), F_Z(t), M_X(t)$ at one time instead of three separate models. The only difference between the combined model and the separate model is the final output tensor of the response is a 3-component vector of (F_Y, F_Z, M_X). The same data are prepared and used to train the model. Figure 4.21 shows the training history during 500 iterations. Figure 4.22 compares the predictions of the combined model and the ground-truths for one test scenario. No significant difference is found between the performances of the combined

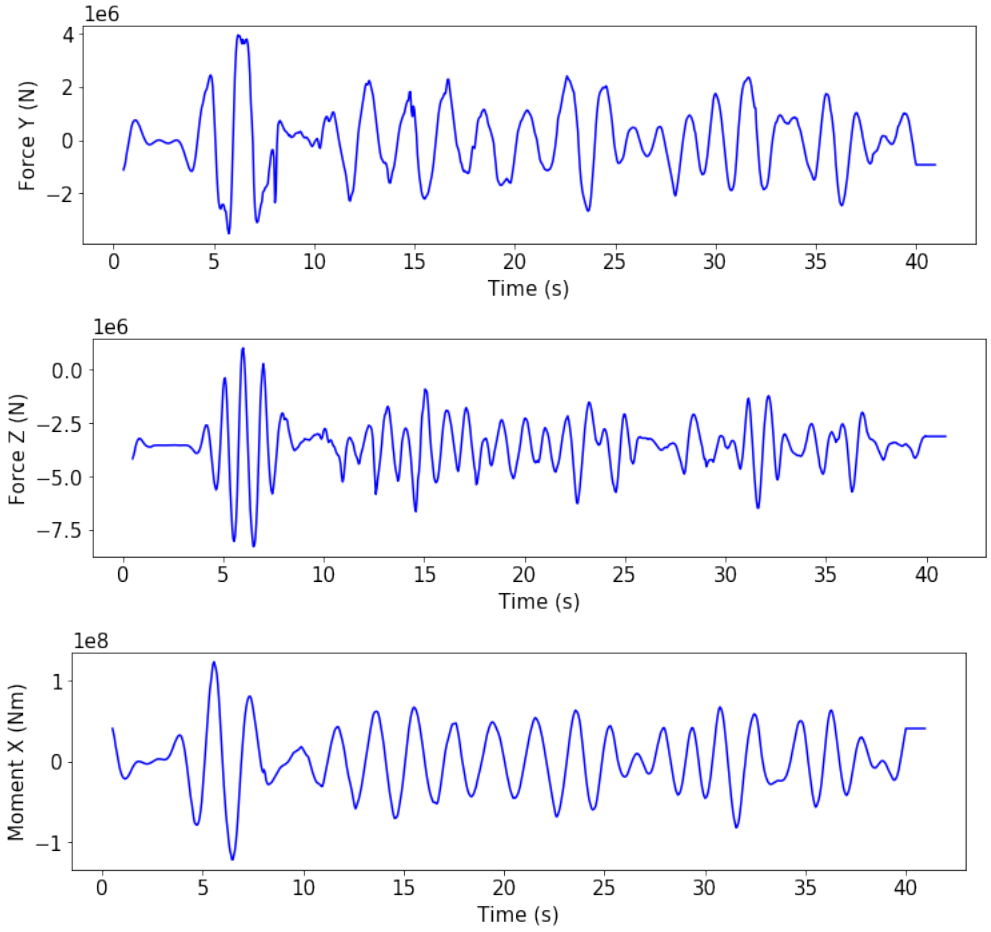

Figure 4.18: Hydrodynamic forces F_Y, F_Z and moment M_X for the given roll input in Figure 4.16

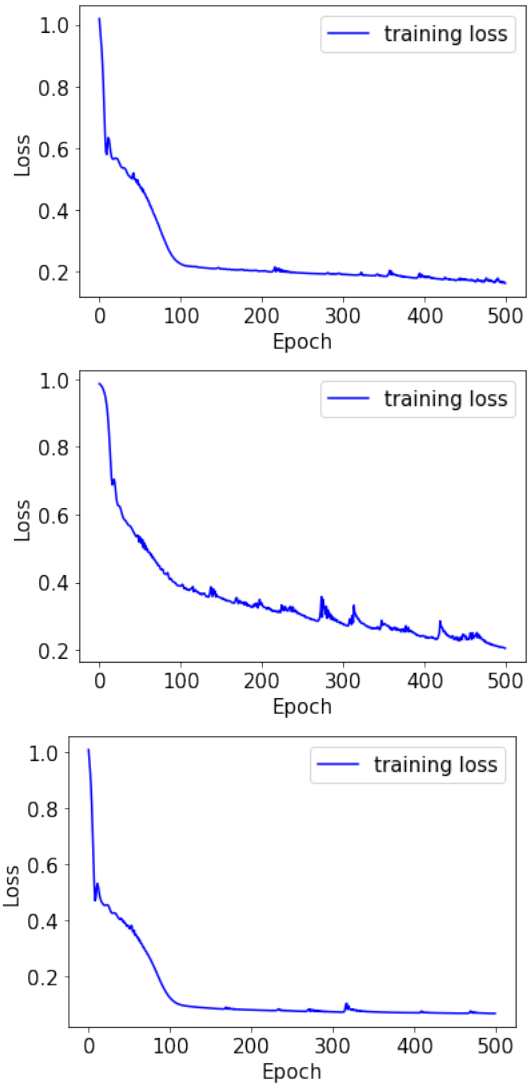

Figure 4.19: Training history during 500 iterations for F_Y (top), F_Z (middle), and M_X (bottom)

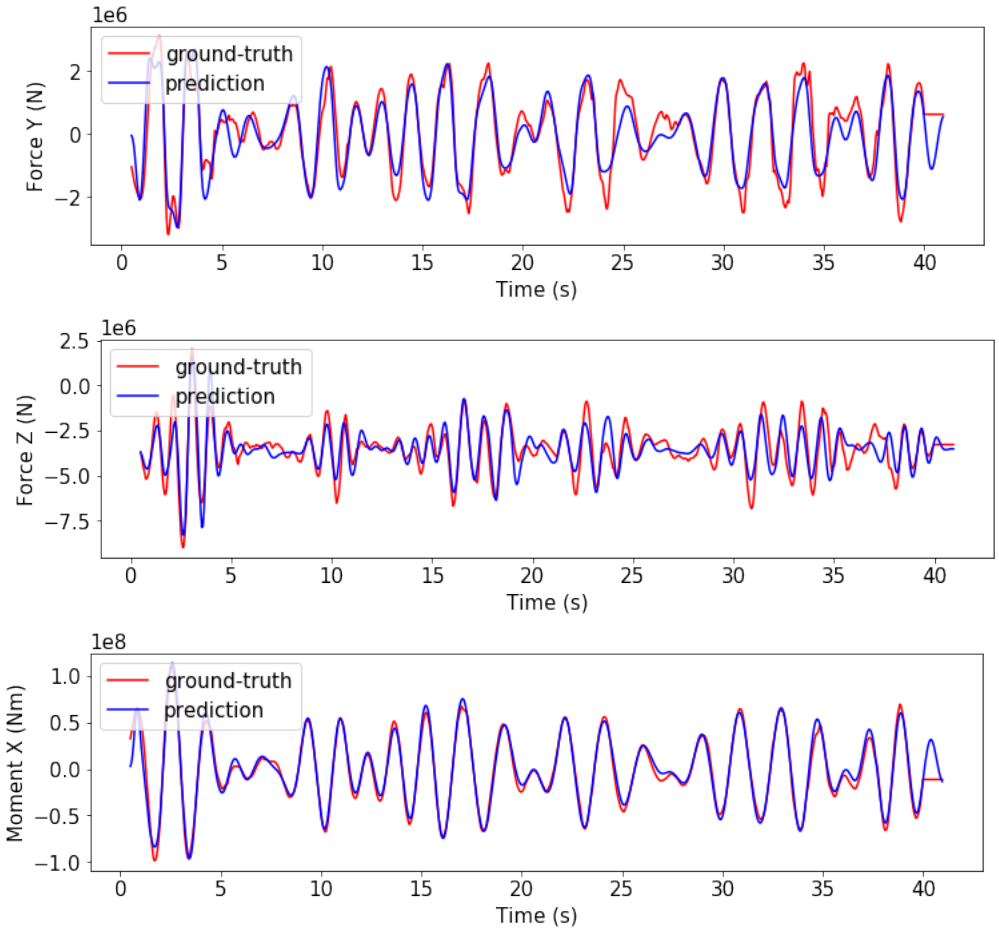

Figure 4.20: Compare predictions of $F_Y(t), F_Z(t), M_X(t)$ series and their ground-truths

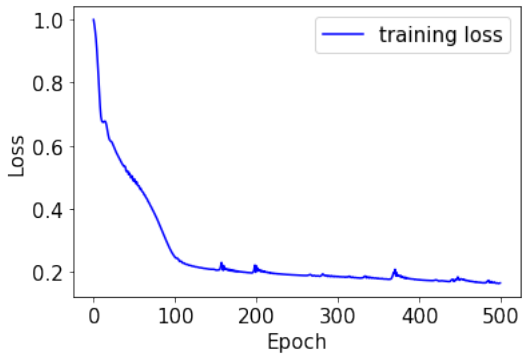

Figure 4.21: Training history during 500 iterations for the combined model

model and three separate models since the number of parameters of the combined model is close to the number of parameters of three separate models put together.

4.7 Example: Floating Object in Irregular Waves

In the last example of the chapter, the proposed identification method is applied to predict the motion of a floating object in irregular waves. The long-crest irregular waves are generated in a 3D tank and induce a floating cube to move. The floating object is constrained to 1-DoF rotation around the center of its bottom panel. The system input is the upstream wave elevation and the system output is the pitch angle of the object. To simulate this complex Fluid Structure Interaction (FSI) problem, olaFlow (*Higuera et al.*, 2014, 2015), an extension of OpenFOAM (*Greenshields*, 2015), is used. Figure 4.23 shows a snapshot during one simulation. The computation domain consists of inlet, outlet, front, back, bottom, atmosphere, and floating object. Figure 4.24 shows the computation grid with the total number of cells being 36,000. The grid is coarse but enough to represent the nonlinearity of the system. Table 4.5 gives the specifications of the numerical tank and the floating object.

The waves are generated at the inlet by setting the domain boundary condition based on wave theories. For irregular waves, the linear summation of Stokes components is used, which is

$$\eta(x,t) = \sum_{i=1}^{N} a_i \cos(k_i x - \omega_i t + \phi_i) \tag{4.9}$$

where N is the number of Fourier components, and a_i, k_i, ω_i, ϕ_i are the amplitude, wave number, angular frequency, and phase angle for ith component. The JONSWAP spectrum is selected to generate the waves. Table 4.6 lists the environmental

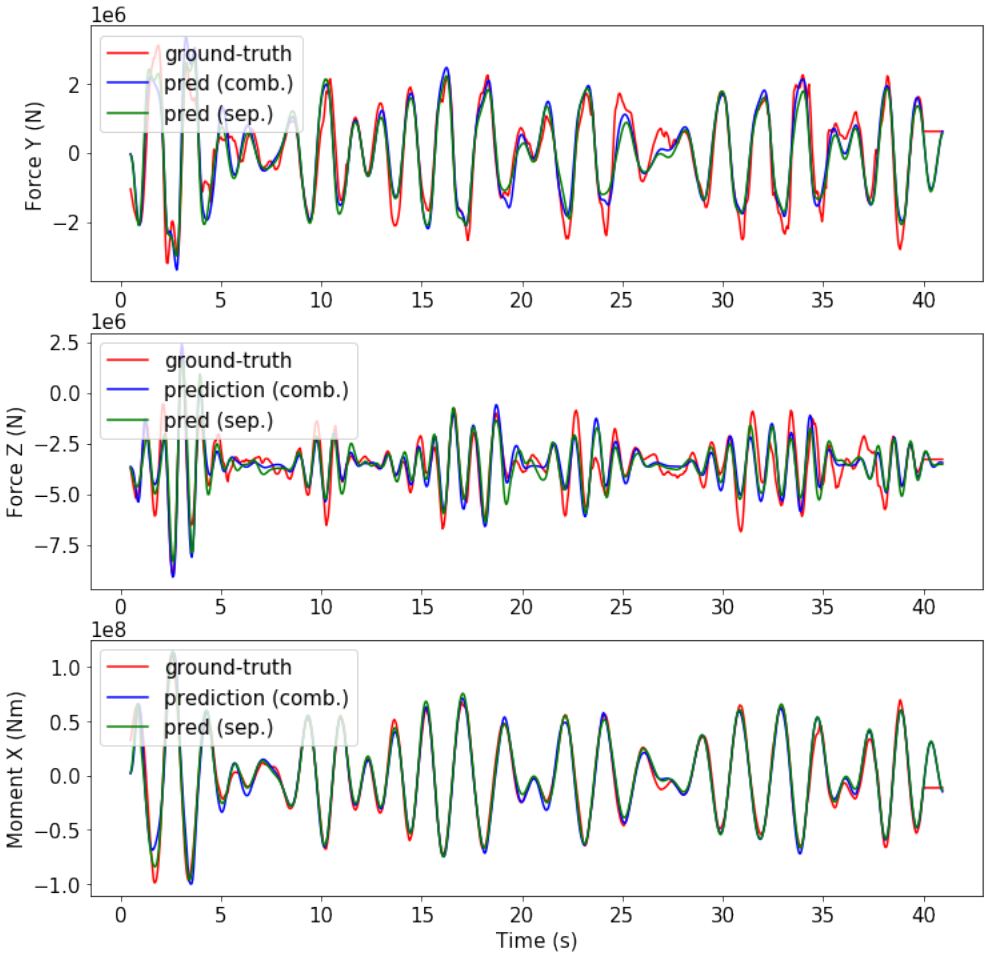

Figure 4.22: Compare predictions of $F_Y(t), F_Z(t), M_X(t)$ series (the combined model and 3 separate models) and their ground-truths

Dimension	Value
tank length (x direction)	3 m
tank width (y direction)	1 m
tank height (z direction)	1.5m
water depth	1.0368 m
object length (x direction)	0.3 m
object width (y direction)	0.2 m
object height (z direction)	0.6 m
object draft	0.4368 m
object freeboard	0.1632 m
upstream distance	1.35 m
downstream distance	1.35 m
#cells (x direction)	40
#cells (y direction)	20
#cells (z direection)	45

Table 4.5: Wave Tank Dimension

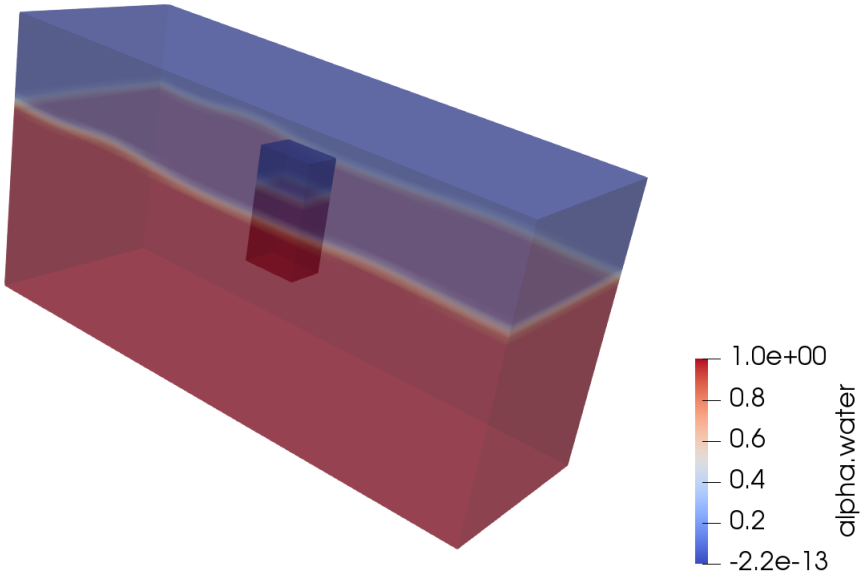

Figure 4.23: One snapshot during one simulation

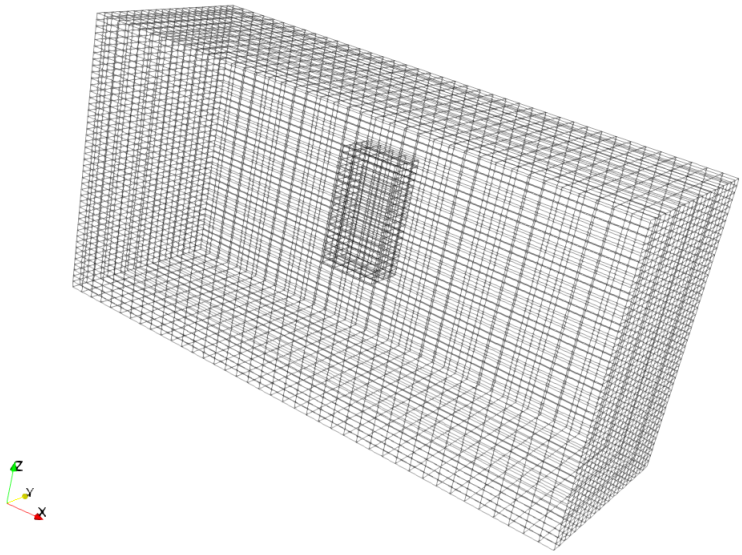

Figure 4.24: CFD flow domain grid

Symbol	Description	Value
H_s	significant wave height	0.1 m
T_p	peak period	1.0 s
N	number of Fourier components	30
f_{lower}	lower cutoff frequency	0.68 Hz
f_{upper}	upper cutoff frequency	1.49 Hz

Table 4.6: Specifications of Irregular Wave Spectrum (JONSWAP)

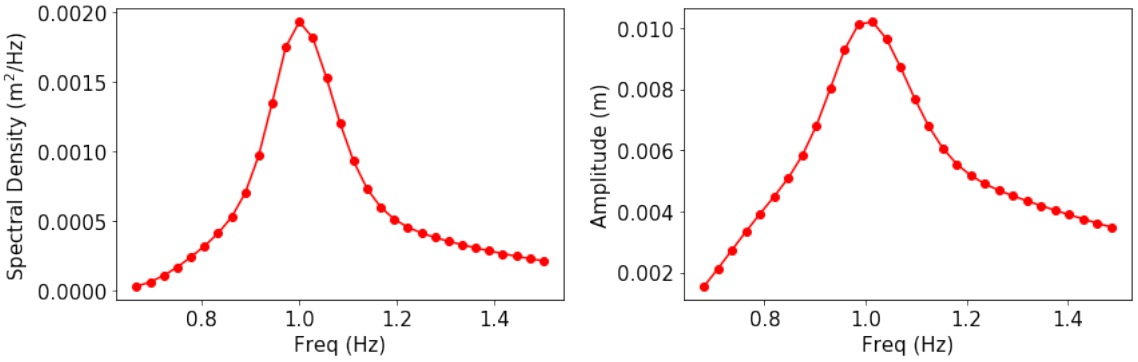

Figure 4.25: Discretized JONSWAP spectrum (left) and $a(f)$ amplidue curve (right)

specifications for the spectrum, and Figure 4.25 plots the discretized spectrum and amplitudes for the Fourier components.

Fifty wave environments are generated by sampling the phase angle from $-\pi$ to π. Figure 4.26 plots these fifty wave elevations $\eta(t)$ at the inlet of the tank calculated by Equation 4.9. All environment realizations are simulated with olaFlow for 20 s and 50 pitch angle time series of the floating object are plotted in Figure 4.27. The floating object starts from an upright position and the wave surface starts from the calm water condition. The system under study consists of wave propagation and rigid body motion. Therefore, to characterize the system from end to end, the wave elevations at the inlet $\eta(t)$ are regarded as system input and the simulated pitch angles of the floating object $\eta_5(t)$ are regarded as system output.

Both the system input $\eta(t)$ and the system output $\eta_5(t)$ are normalized by their mean and their standard deviation (μ_η, σ_η) and $(\mu_{\eta_5}, \sigma_{\eta_5})$, respectively. Forty input-output pairs are used as the training dataset and the remaining ten are used as the test dataset. The model is trained for 500 iterations, and the training history shows the loss decreases with respect to iterations as plotted in Figure 4.28.

Once the model is trained, the performance of the trained model is evaluated on the training dataset and the test dataset. Figure 4.29 compares the model prediction

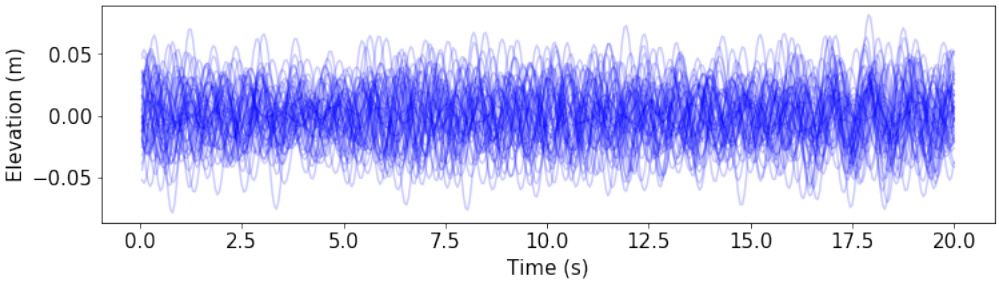

Figure 4.26: Wave elevation at the inlet of the tank, $\eta(t)$

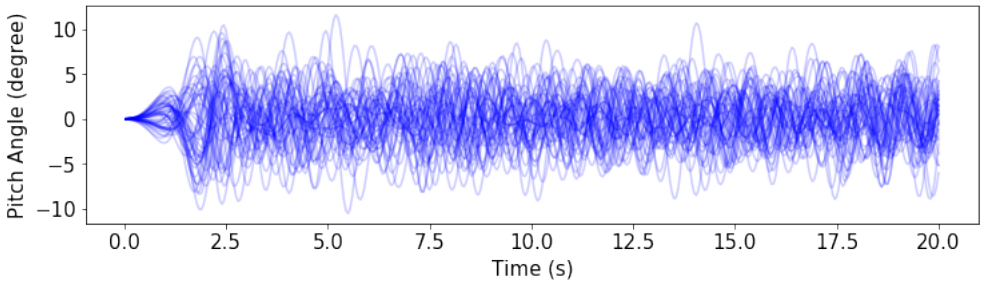

Figure 4.27: Pitch angle time series of the floating object under wave environments in Figure 4.26

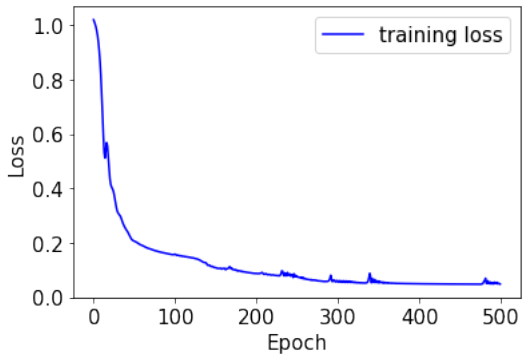

Figure 4.28: Training history of the model that characterizes the floating object under waves

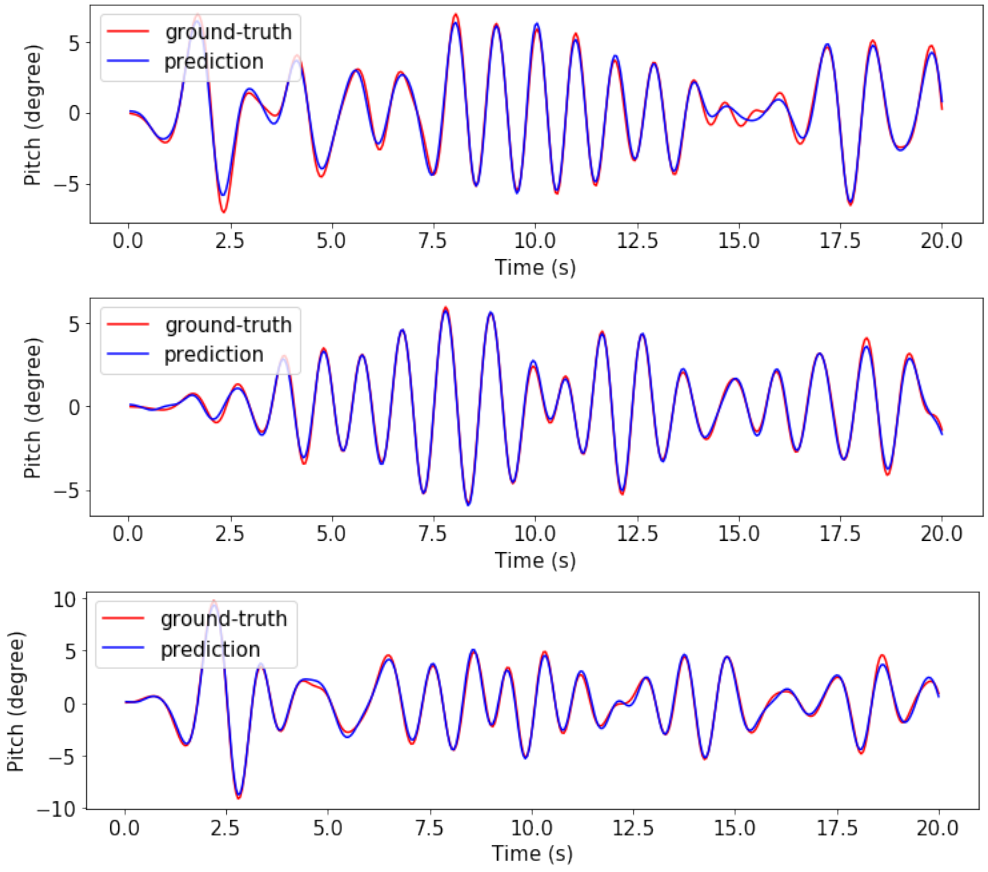

Figure 4.29: Compare pitch prediction against the ground-truth for the training dataset (3 of 40 time series shown)

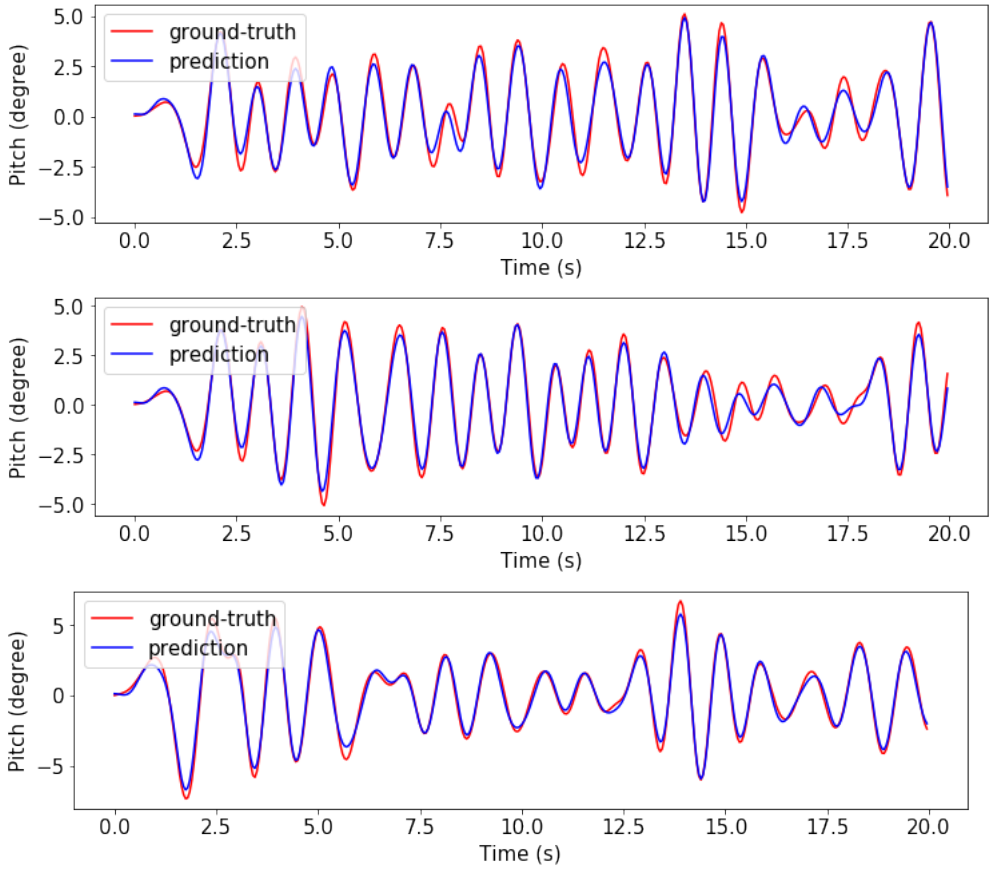

Figure 4.30: Compare pitch prediction against the ground-truth for the test dataset (3 of 10 time series shown)

with the CFD ground-truth for the training dataset. Figure 4.30 compares the model prediction with the CFD ground-truth for the test dataset. As shown in the figures, the model predicts an accurate pitch series based on the wave elevation series.

4.7.1 Training Data Size

An advantage of applying the DRE model to identify the system is the model does not need large amounts of training data. In the sloshing tank and floating object examples, forty pairs of input and output series are used as the training dataset for these CFD applications. Generally speaking, more data is helpful for the model to identify the system though it costs more. Hence, the user needs to balance between the model performance and data collection cost. Unfortunately, there are no universal guidelines on the size of the training data in need since the required size of the training dataset depends on the complexity of the studied system.

In order to provide an intuition on the required size, an experiment with the floating object example is conducted. The experiment uses different sizes of training dataset (input-output pairs) ranging from 10 to 40 and 500 iterations are used during the training process. The trained model is then evaluated on both training and test dataset via the Root Mean Square Error (RMSE) measurement

$$\mathrm{RMSE}(\hat{\eta}_5, \eta_5) = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(\hat{\eta}_{5,t} - \eta_{5,t})^2} \tag{4.10}$$

where η_5 is the CFD ground-truth of the pitch angle, $\hat{\eta}_5$ is the model prediction of the pitch angle, and T is the number of time steps in the time series. To summarize the whole performance on both datasets, the RMSE measure is averaged across all pitch series within the dataset, that is

$$\overline{\mathrm{RMSE}} = \frac{1}{N}\sum_{n=1}^{N}\mathrm{RMSE}(\hat{\eta}_5^{(n)}, \eta_5^{(n)}) \tag{4.11}$$

where N is the number of realizations in the dataset (40 for the training dataset, 10 for the test dataset), and $\hat{\eta}_5^{(n)}, \eta_5^{(n)}$ are the nth pitch prediction and ground-truth respectively in the dataset.

Figure 4.31 shows the performance of these models on one of the training series. Figure 4.32 shows the performance of these models on one of the test series. Figure 4.33 plots the $\overline{\mathrm{RMSE}}$ scores against the data quantity. As shown in the figure,

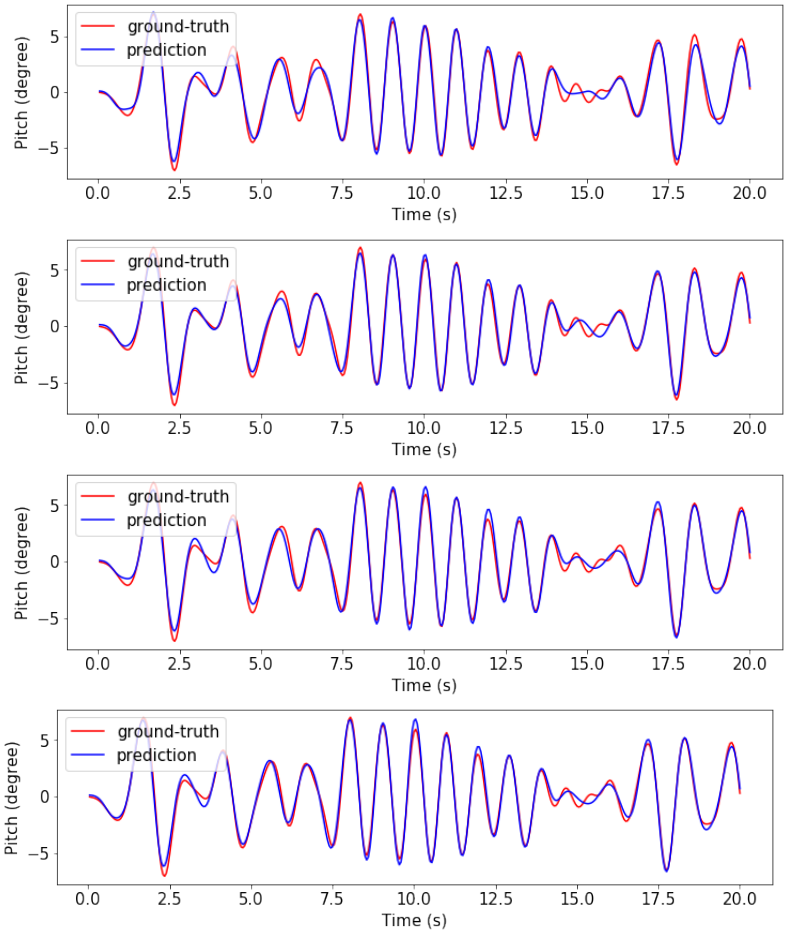

Figure 4.31: Performance on one training sample by the models trained with different quantity of training data (10,20,30, and 40 from top to bottom)

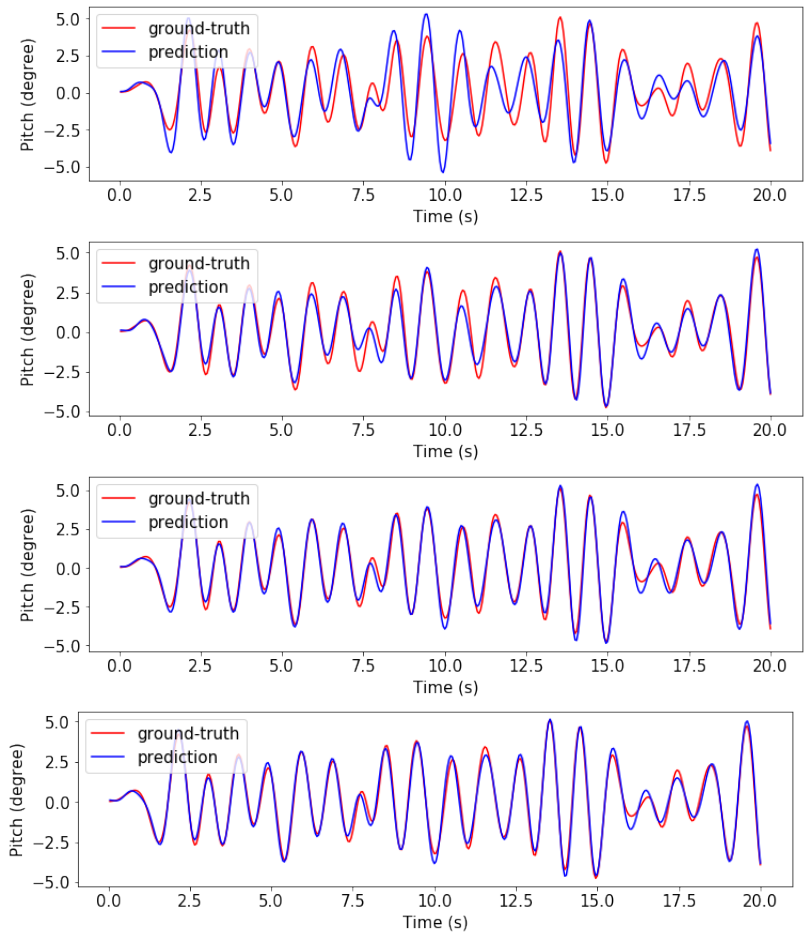

Figure 4.32: Performance on one test sample by the models trained with different quantity of training data (10,20,30, and 40 from top to bottom)

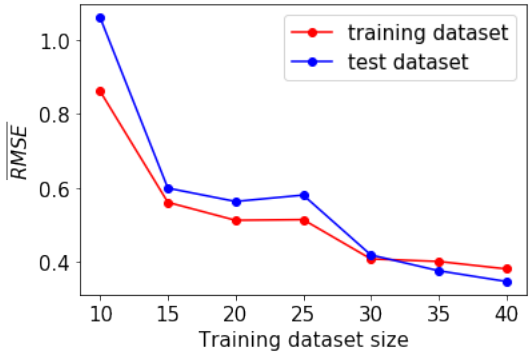

Figure 4.33: More collected data gives better identification of the system.

models trained by more data usually lead to lower $\overline{\text{RMSE}}$ score and perform better on both training and test dataset. Therefore, collecting more data is helpful to identify the system more successfully.

CHAPTER V

An Example with both TEG and DRE

In this chapter, a floating block in irregular waves is discussed as a concrete example to show how the proposed TEG and DRE models play roles and can be used together in extreme pitch modeling. Specifically, the chapter solves the problem: For a mild wave environment where the floating object undergoes small pitch ($<< 5°$) in most of the time, how are the wave scenarios that lead to large pitch angle ($> 7°$) distributed, and how to generate these dangerous cases?

5.1 Motivation

Since the TEG model is a generative model that often requires large amounts of training data, efficiently scoring each wave seed in terms of system response is needed. When the studied system is properly represented by the simplified model, efficient simulation of the response behavior is therefore accessible, and the TEG model can be easily applied as shown by the examples in Chapter III. However, when the simplified model is unavailable due to system complexity or lack of insight, the DRE model can be used as a data-driven alternative in the simulations and help to score the wave seed.

Consider the example in Section 4.7 where a floating object in a long-crest irregular wave is studied. CFD is used to simulate the flow domain and the object is free to pitch under the wave-induced loadings. In order to apply the TEG model to generate wave environments that lead to dangerous conditions, a large amount of training data is needed. Specifically, a mapping from the wave random seeds (in the format of phase angles) to the pitch angle at the design time must be established, which is

$$\theta = \eta_5(t_d; \phi_1, \cdots, \phi_N) \tag{5.1}$$

ϕ_1	ϕ_2	$\cdots$	ϕ_N	θ
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 5.1: Training data needed by the TEG model

where t_d is the design-time specified by the user and $(\phi_1, \cdots, \phi_N)$ is the wave seed vector that consists of N Fourier phases at the upstream boundary.

Direct CFD simulation of pitching in waves is expensive in this FSI problem. Therefore, filling the table of training data for the TEG model via direct CFD is cost-prohibitive, as shown in Table 5.1. A more efficient and fast approach is to identify the system first via the DRE model, and then ask the trained DRE model to estimate the function of θ column.

5.2 Procedure

Figure 5.1 shows how to integrate the DRE model and the TEG model. First, a training dataset that consists of the time-domain input-output time series pairs is collected to identify the system via the DRE model. After the DRE model is trained, for any new input series (or environments), corresponding output series (or response) can be predicted. Then the system output at the design time is regarded as the 'score' for the system input environment. This scoring process can be conducted multiple times until enough training data is collected for the TEG model. Since the trained TEG model is generative, new environments can be sampled from the high-dimensional (N) probability space in order to produce environments that lead to critical response.

Starting from the Section 4.7, the pitch motion of a floating object in an irregular wave has been successfully identified by establishing the map from the upstream wave elevation $\eta(t)$ to pitch $\eta_5(t)$. Refer to Section 4.7 for the environmental parameters and flow domain setup. The objective of this chapter is to generate seaways of the JONSWAP spectrum that will lead the object to a large pitch angle at a design-time. Specifically, seaways sampled from the following distribution are desired,

$$\Pr(\phi_1, \cdots, \phi_N | \theta > \zeta) \tag{5.2}$$

where θ is the pitch angle of the object at the design-time defined in Equation 5.1 and ζ is a threshold specified by the user. For the current example, design time is selected to be 15 s, and the critical pitch threshold is 7 degrees.

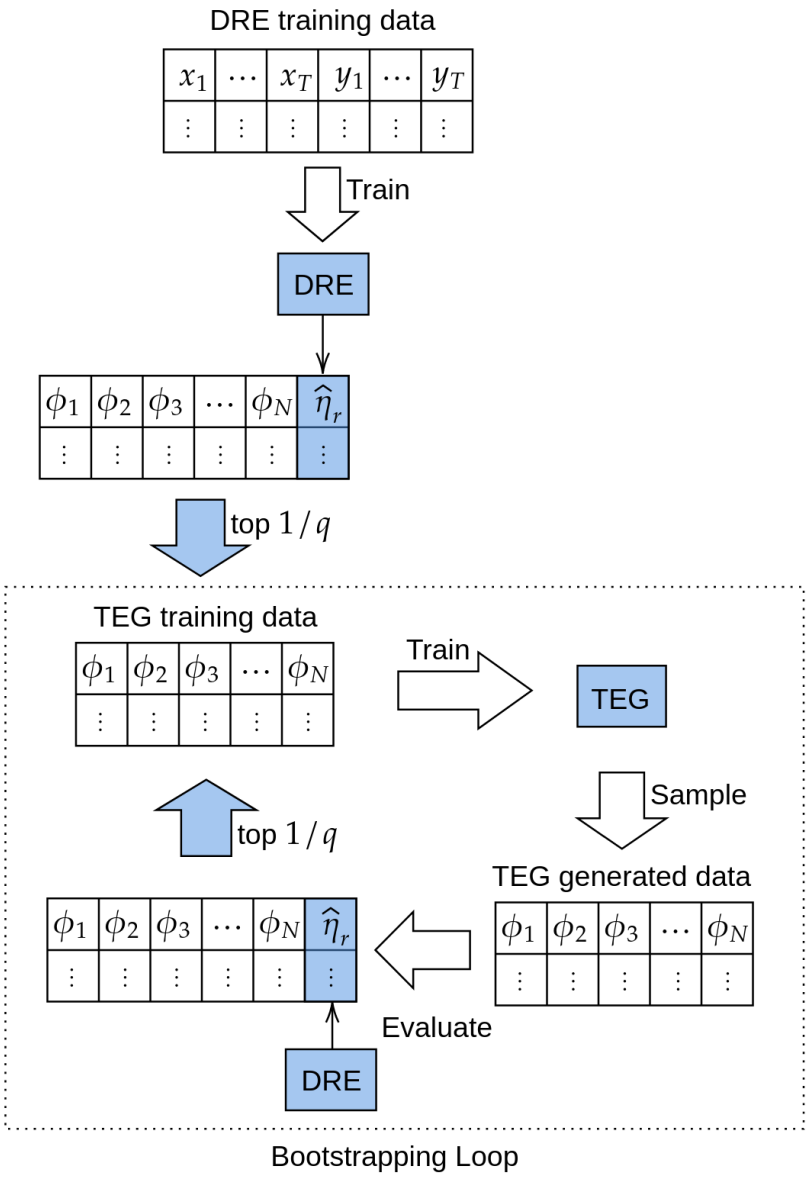

Figure 5.1: A diagram shows how to integrate both the DRE model and the TEG model

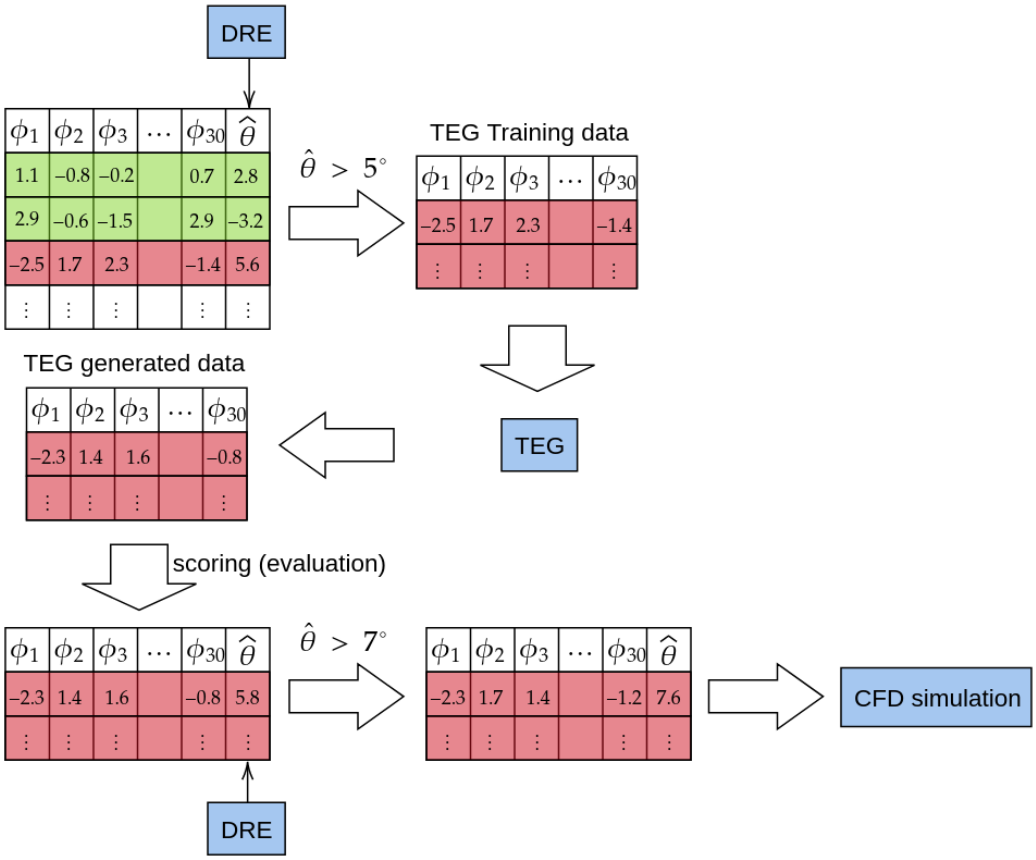

Figure 5.2: Recipe to generate desired seaways ($t_d = 15$ s, $\zeta = 7^\circ$) using DRE and TEG

Figure 5.2 shows the procedure of generating the desired seaways. First, all wave phases are sampled uniformly and identically across Fourier components from $-\pi$ to π since no prior on dynamical behavior is given. The DRE model is then used to score each wave scenario by estimating the pitch angle at the design-time. Once all phase sets are scored, the phase sets ending with an estimated pitch angle greater than 5 degrees are selected as the training dataset for the TEG model. The TEG model learns the joint distribution among the phases in the training dataset and generates new phase vectors that follow a similar distribution. The newly generated phase vectors are then scored by the DRE model and those scenarios leading to large pitch angles (7 degrees) are then available for use in CFD for simulation.

5.3 Results

In Section 4.7, the pitch angle for a given wave elevation has been identified by the DRE model using fifty input and output $(\eta(t), \eta_5(t))$ pairs. By simulating the DRE as

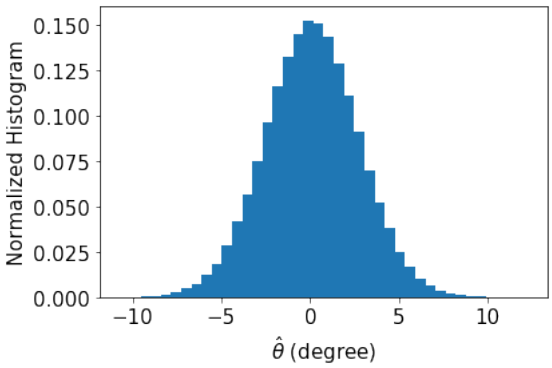

Figure 5.3: Histogram of the estimated pitch $\hat{\theta}$ by the DRE model for all 200,000 scenarios

the reduced-order model, the pitch angle of the floating object at design-time $t_d = 15$ s can be estimated for any given wave scenarios $(\phi_1, \cdots, \phi_{30})$.

As mentioned in the section above, the plain seed space is scored by the DRE model. For the experiment, 200,000 phase vectors are sampled from $[-\pi, \pi)^{30}$. All 200,000 scenarios are scored by the DRE model by returning the pitch angle at 15 s. Figure 5.3 shows the normalized histogram of the estimated pitch angles. Among 200,000 estimated pitch angles, 6,423 phase vectors have scores higher than 5 degrees. Figure 5.4 shows the marginal distribution $\Pr(\phi_i|\theta > 5°)$ from the 6,423 filtered phase vectors. Note that it is not possible to perform 200,000 CFD simulations without a great cost.

These phase vectors serve as the training data for the TEG model. Figure 5.5 plots the training history in terms of performance loss. The small increase of the loss on the validation dataset after 30 epochs is due to the *patience* parameter during training. A plain *early stopping* strategy is to stop the training process once the validation loss increases. With the patience parameter, a delay is added so that the training keeps going until no improvement is observed for a number of epochs. More details are discussed in Section 3.2.5. After the model is trained, new phase vectors can be sampled from the TEG model. Figure 5.6 shows the marginal distribution of the new phases is well-maintained by comparing to Figure 5.4. The histograms of the resultant pitch angle can also be compared between the given dataset and the generated dataset in Figure 5.7. As shown in the figure, the difference occurs at the boundary $\theta = 5°$ but the tail behavior is kept. The tail part of the histogram ($> 7°$) is also compared in Figure 5.8.

Ten phase vectors in the tail ($\hat{\theta} > 7°$) are randomly selected for further CFD investigation. These ten phase vectors produce wave elevation series at the upstream

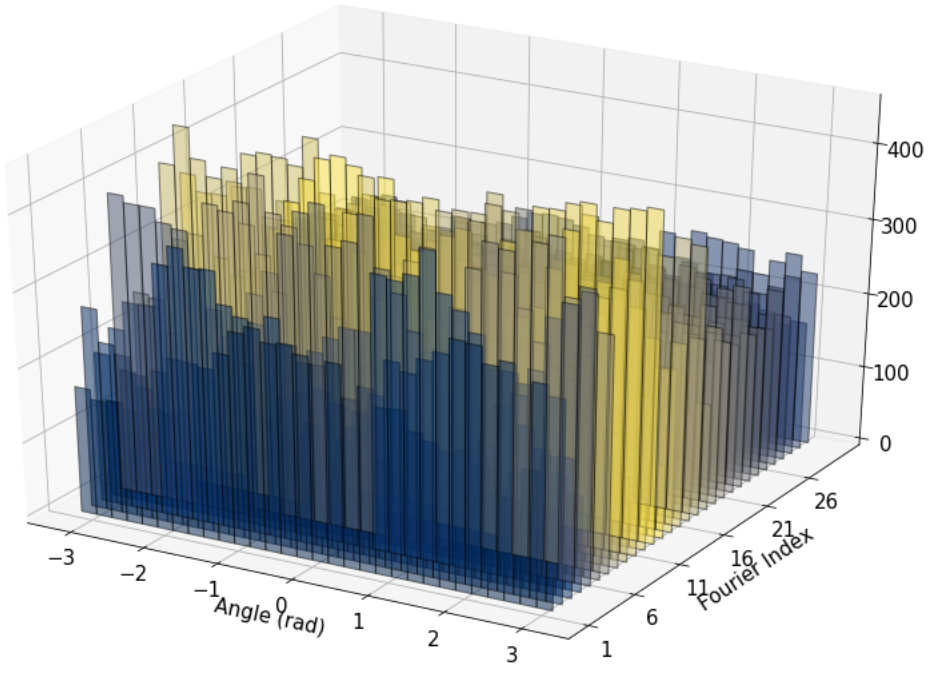

Figure 5.4: Marginal distribution $\Pr(\phi_i|\theta > 5°)$ for the given dataset

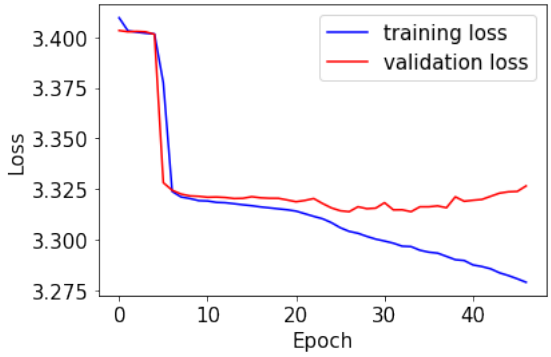

Figure 5.5: TEG model training history for the floating object example

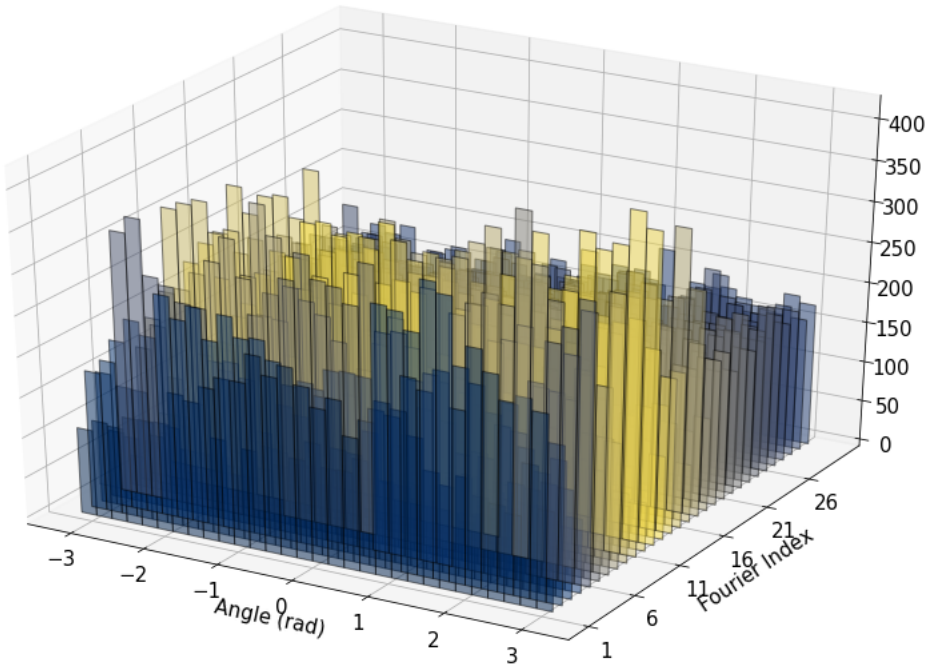

Figure 5.6: Marginal distribution $\Pr(\phi_i|\theta > 5°)$ for the newly generated dataset

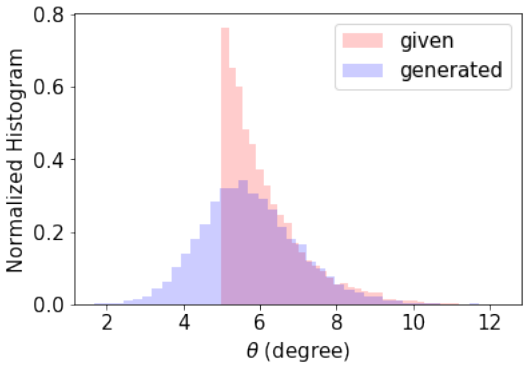

Figure 5.7: Compare histograms of the resultant pitch angle between the given dataset (red) and the generated dataset (blue)

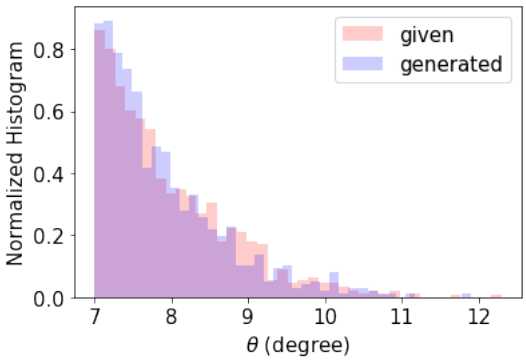

Figure 5.8: Compare tail of the histograms ($> 7°$) between the given dataset (red) and the generated dataset (blue)

boundary for ten CFD scenarios. Figure 5.9 plots all ten series of upstream wave elevations together and the corresponding predicted pitch angle by the DRE model. For the simulation results, Figure 5.10 compares the CFD simulated pitch angle for all ten scenarios against the DRE predictions one by one.

As shown in the comparison, both predicted and the simulated pitch series achieve a large angle around the design-time $t_d = 15$ s. It is also worth noting that DRE model underpredicts the large absolute value of the pitch angle, especially at the response crest and trough. The error can be explained by looking at the training dataset for the DRE model in section 4.7. Most pitch angles in the dataset range from $0°$ to $\pm 5°$, which provides limited data on predictions on extreme scenarios in this chapter, which are over $\pm 7°$. It is worth noting that in the current setup the wave spectrum used to train the DRE model is exactly the same with the desired spectrum under extreme event study. However, these two spectrums can be different to improve the results. The model trained with more severe wave scenarios should yield better estimation when predicting large and rare response in mild seaways. Therefore, it is suggested to use a more severe sea state to more accurately characterize the system behavior in large response in order to reduce the gap when generating wave scenarios in the TEG model. In the next section, the distribution within the training dataset for the DRE module is discussed to improve the observed gap.

5.4 Effect of the Dataset Distribution on System Identification

As shown in Figure 5.9, the DRE module underpredicts the pitch angle when the pitch is large, compared to the CFD ground-truth. The under-estimation is

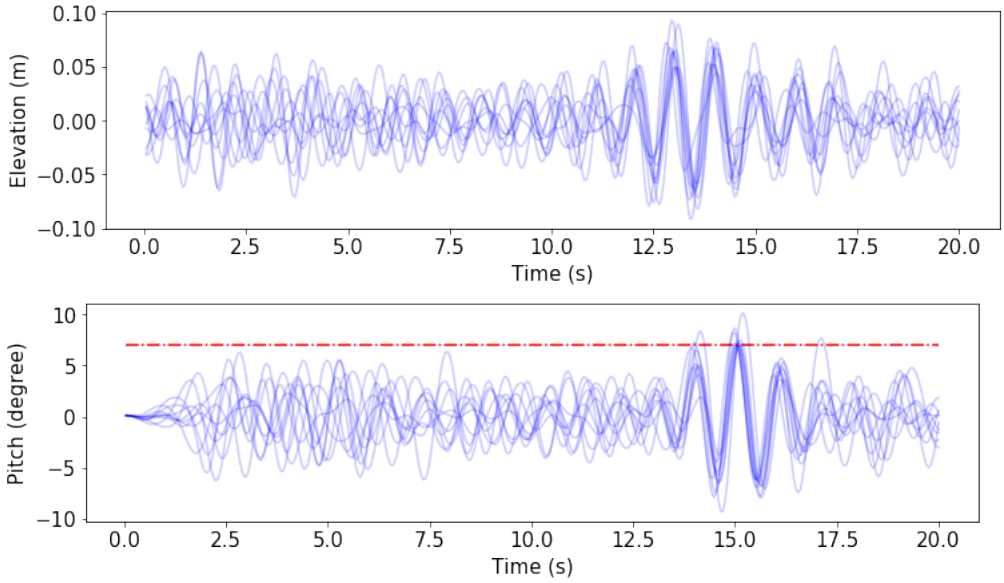

Figure 5.9: Randomly selected ten series of upsteam wave elevation (input) and corresponding predicted pitch angle (output)

due to the distribution of the collected data for the DRE module. The nonlinear system dynamics may not be characterized well at the extreme region ($\approx 7°$) if most collected data comes from a mild operating condition ($<< 5°$). Therefore, in order to accurately characterize the system for the extreme cases that are generated by the TEG module, a more severe environment needs to be included in the DRE training dataset.

To demonstrate the effect of the dataset distribution on the system identification, the wave input is simply amplified in this section. Three experiments are conducted with different scaling factors. Specifically, the amplitudes in the wave elevation are increased by 10%, 20%, and 50%. For each experiment, fifty CFD simulations are carried out to collect the training and test datasets for the DRE module. Each DRE module is trained with 500 iterations. Figure 5.11 plots the performance of the trained DRE modules on one test sample. As shown in the figure, the pitch increases as the wave elevation is amplified. The performance on the unseen test dataset is good for all three DRE modules, though the models identify the systems that operate in different wave severities.

All three models are then asked to predict the extreme pitch events in Figure 5.9. Figure 5.12, 5.13, and 5.14 show their predictions on the TEG generated extreme scenarios. By comparing the three plots, the DRE model trained with the severe wave produces a larger pitch since the distribution of their characterization data is biased

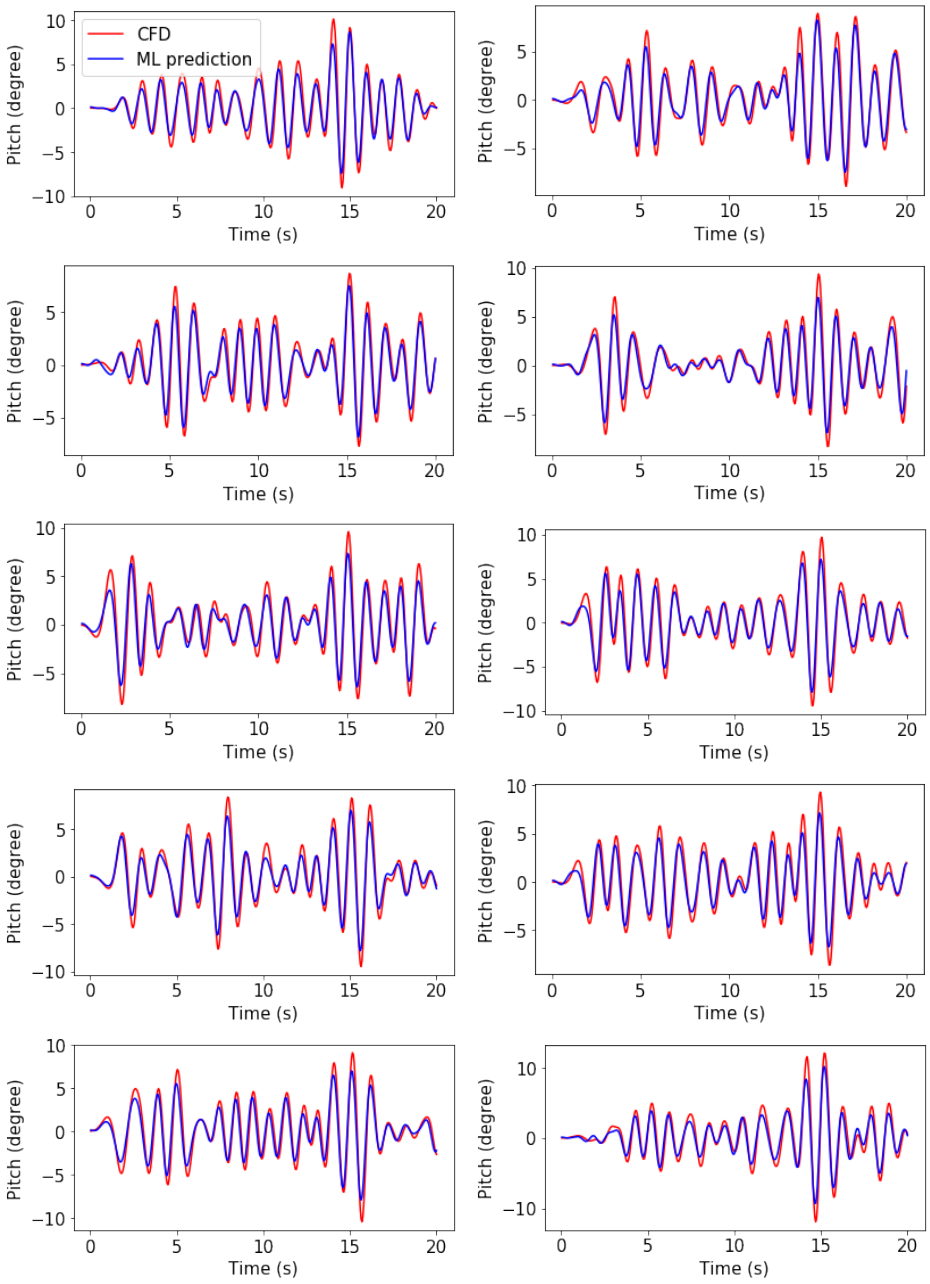

Figure 5.10: Compare the resultant pitch angle between the CFD simulation result (red) and the DRE prediction for each wave scenario

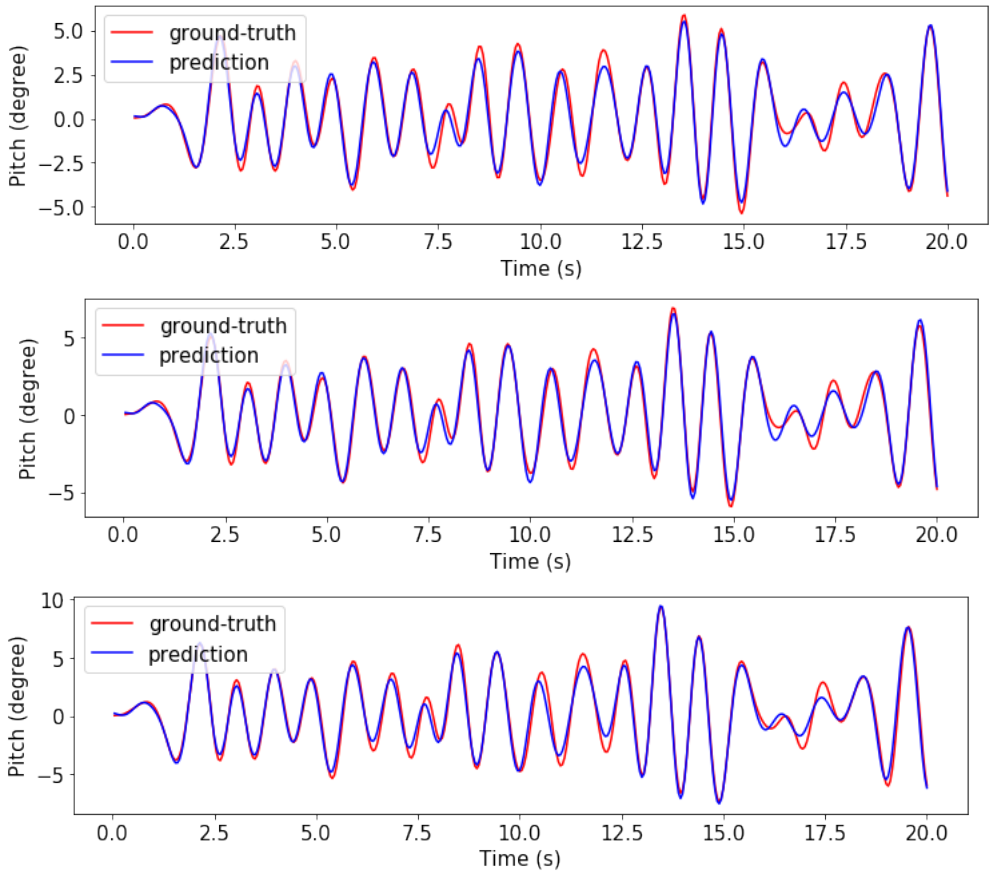

Figure 5.11: Compare the performance of DRE model trained with different wave severities (top: 10%, middle: 20%, bottom: 50%)

towards a larger response region. As shown in the figure, the DRE model trained in the 20% amplified waves produces the closest prediction to the CFD ground-truth, while the DRE models trained in the 10% and 50% amplified waves underpredict and overpredict the pitch respectively. The comparison is better visualized in Figure 5.15, where the predictions from all three models and the original DRE model are compared against the CFD ground-truth for the first extreme test scenario.

A blended dataset is produced by combining all fifty original (without amplified) scenarios, fifty 10%-amplified scenarios, and fifty 20%-amplified scenarios. A new DRE model is trained using the blended dataset. The averaged RMSE scores from all five DRE models are calculated by Equation 4.11 and plotted in Figure 5.16. As shown in the figure, the DRE model characterizing the 20% amplified waves produces the best prediction for the TEG generated extreme pitch.

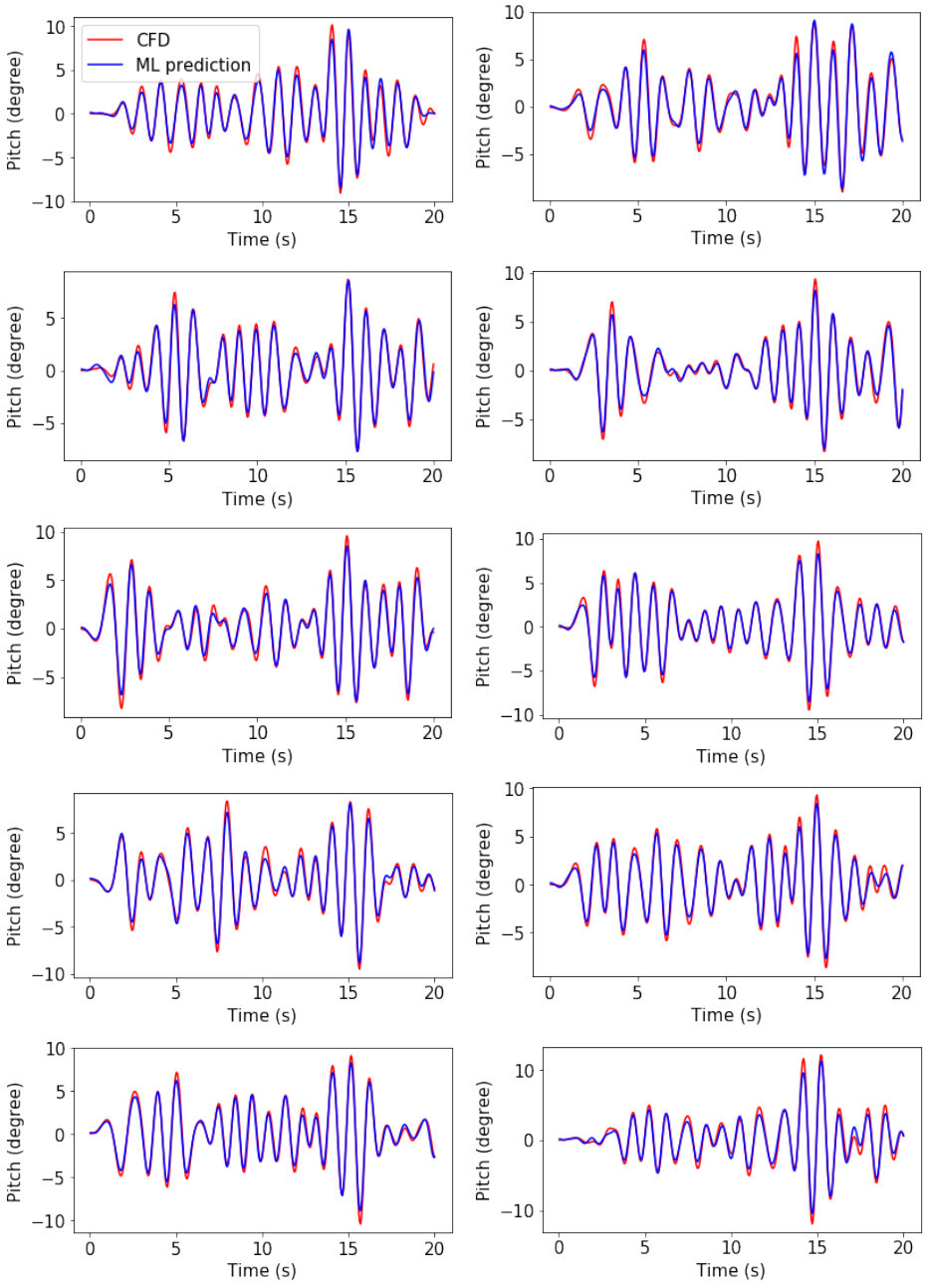

Figure 5.12: Compare the resultant pitch angle between the CFD simulation result (red) and the DRE-10 prediction (blue) for each wave scenario

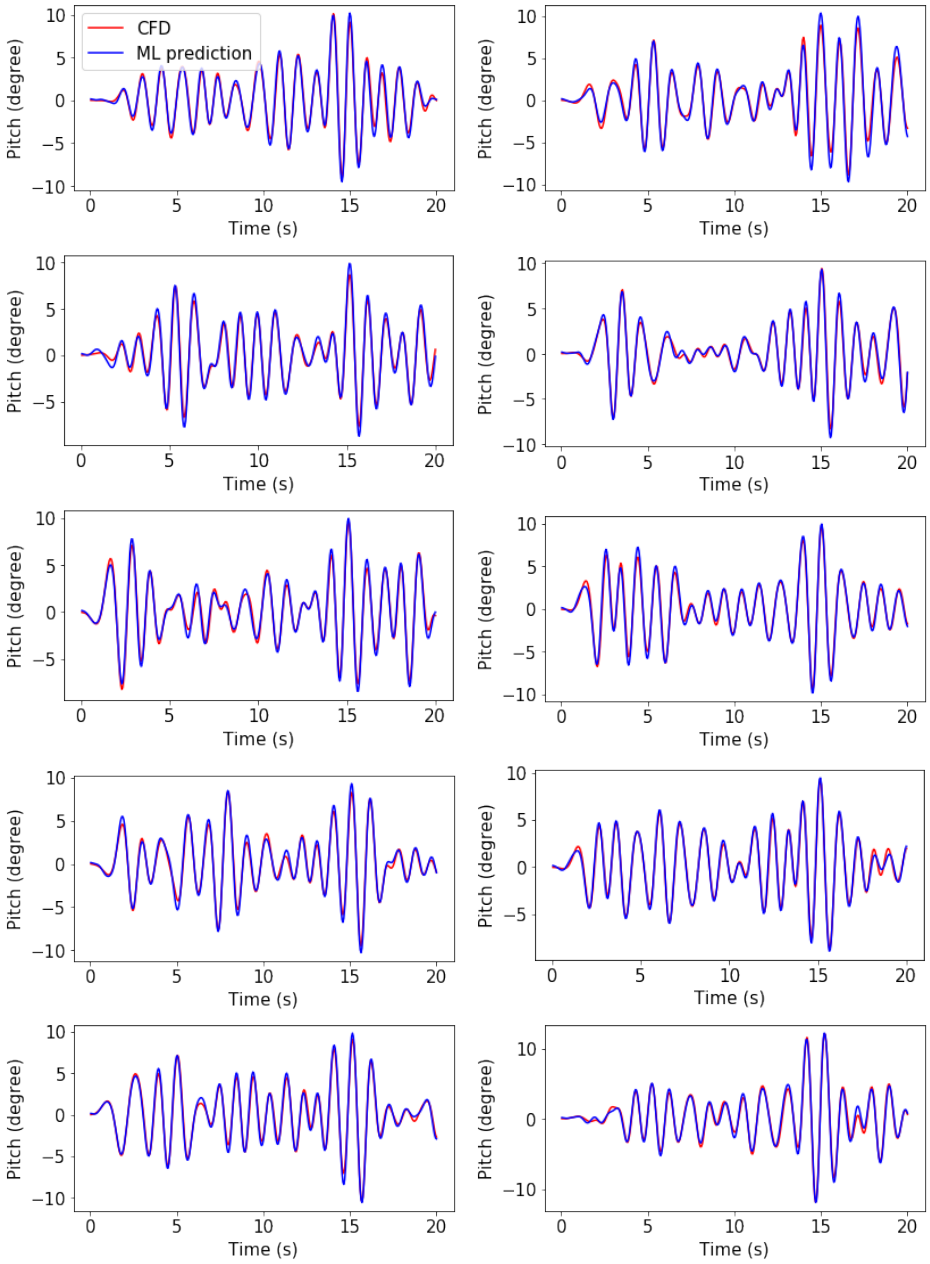

Figure 5.13: Compare the resultant pitch angle between the CFD simulation result (red) and the DRE-20 prediction (blue) for each wave scenario

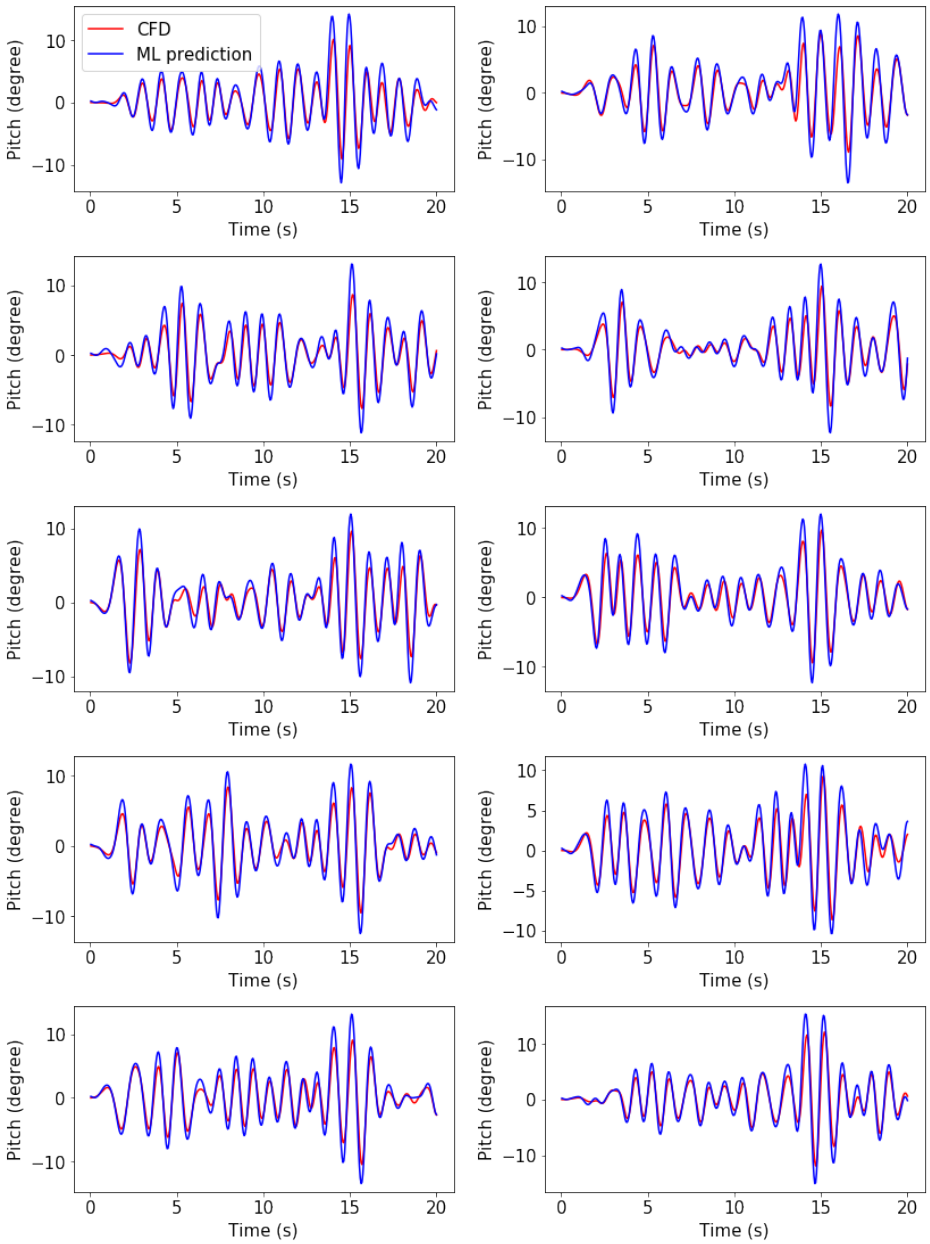

Figure 5.14: Compare the resultant pitch angle between the CFD simulation result (red) and the DRE-50 prediction (blue) for each wave scenario

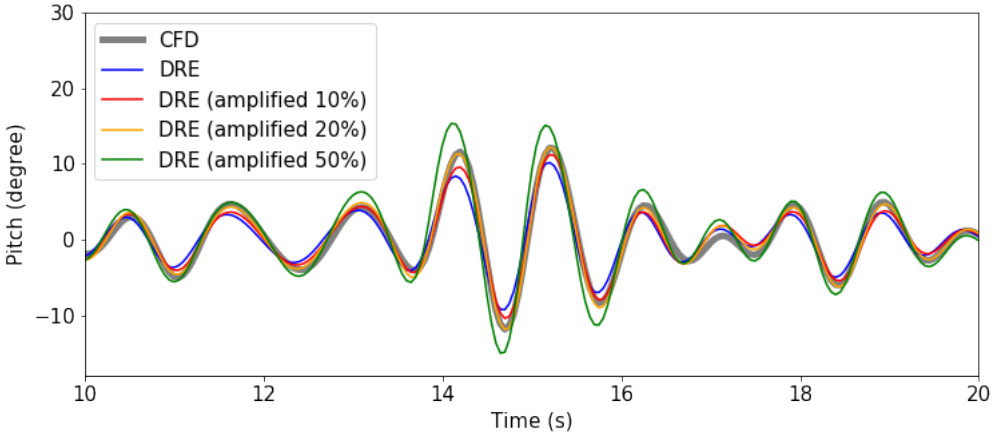

Figure 5.15: Compare the predictions from all four DRE models (DRE, DRE-10, DRE-20, DRE-50) against the CFD simulation result for one wave scenario

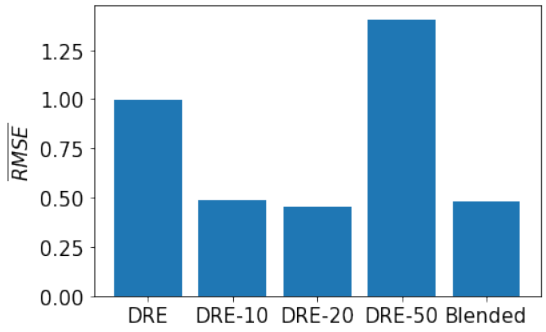

Figure 5.16: Averaged RMSE score of the TEG generated pitch vs. characterization wave condition used in DRE

CHAPTER VI

Conclusion

In this thesis, a purely data-driven framework is proposed to study ship dynamics problems suitable for extreme event analysis. In this chapter, the current research work and the contributions are summarized, the future work is suggested.

6.1 Summary

For the conventional design of ships and marine units, rule-based approaches have long been used by the designers and ship classification societies. Most of these approaches are developed based on simplified models. Strong assumptions are often made during the analysis such that the closed-form or semi-experimental solutions are available. However, these results may not accurately capture the response of the ship, and this is especially true for the extreme (max) response. For example, marine systems are often assumed to be linear and exposed to a Gaussian wave environment. In fact, many marine systems are either weakly or even strongly nonlinear. Therefore, to shorten the gap between the results of simplified models and real conditions, safety factors, for example, dynamic loading factors, are often used. However, such rule-based estimates are usually rough and lack of more completed case-by-case analysis. Moreover, this situation becomes even worse for the extreme-related analysis since the extreme events occur rarely and very few observations can be collected.

Due to the development of computational approaches, large-scale time-domain simulations become more accurate to characterize the system dynamics than the traditional methods. The geometry of the marine devices and complex interaction between the fluid and structure can be better represented via computer-based methods. Nowadays, designers have access to high-performance cloud computing that allows them to simulate the response of the device in various ocean environments. The ocean environments that rarely occur in real life can now be well established in

numerical simulations. Compared to the experimental methods, numerical methods also generate large amounts of data for analysis. Due to the advantages of numerical simulations, designers and classification societies tend to validate the design and make rules based on simulations. However, more severe ocean environments do not necessarily produce a larger system response. Setting up ocean environments that lead to extreme system response is usually difficult. Moreover, these environments need to be sampled from the conditional distribution.

A data-driven framework is proposed to address the above challenge. Unlike other methods found in the literature, the proposed framework contains the following features:

- The framework can generate wave environments that lead to large system response. By simulating the generated environments, the user can observe system responses that exceed the user-defined threshold at the design time. With high-fidelity simulations, a time-domain context for the designed extreme events can be better understood. The user can closely investigate the design by checking behaviors of different parts when the extreme event occurs. Necessary modification can then be made to mitigate the response of interest. The framework can generate as many environments as the user wants, and they follow the threshold exceedance conditional probability. Therefore, an extreme event that is more likely to occur appears in the generated scenarios with a larger chance.

- The framework can be used with few assumptions. The dynamical system of interest can be nonlinear, and the prior ocean environments can be non-Gaussian. To fully utilize the information embedded in the data, machine learning methods are heavily used in the proposed framework. As a result, the framework is extremely useful when a complex system has no mathematical model or when very limited insight about the system is available.

- For the extreme event modeling, observation of the events are usually rare, which produces a challenge of limited data for many data-driven methods. Since the framework is generative, it uses a novel "bootstrapping" strategy to address the challenge as much as possible.

The proposed framework consists of two modules which are named Threshold Exceedance Generator (TEG) and Design Response Estimator (DRE). They can work together or separately to return a collection of environment scenarios to the user.

By simulating the design in the generated environments, the user can observe various extreme responses that exceed a user-specified threshold of ζ at a user-specified timestamp t_d. Mathematically, a random environment is specified by a seed vector $(\phi_1, \cdots, \phi_N)$. The proposed framework can generate a collection of seed vectors from the distribution.

$$\Pr(\underbrace{\phi_1, \cdots, \phi_N}_{\text{TEG}} \,|\, \underbrace{\eta_r(t_d; \phi_1, \cdots, \phi_N)}_{\text{DRE}} > \zeta). \tag{6.1}$$

where $\eta_r(\cdot; \phi_1, \cdots, \phi_N)$ is the time series of the system response according to the environment seed. The TEG module focuses on the generation aspect of the framework, while the DRE module addresses the system identification aspect of the framework. Intuitively, the TEG module comes up with a collection of environments that could lead to large system responses, and the DRE module judges each TEG-generated environment by estimating the system response and return the feedback to the TEG for a next-round generation.

Many examples are used in the thesis to validate both TEG and DRE modules. The examples of linear wave propagation, nonlinear wave propagation, nonlinear ship roll, sloshing tanks, and floating object in irregular waves cover a wide range of applications for the developed modules. The successful application of these examples shows that the proposed framework can be used to linear or nonlinear, Gaussian or non-Gaussian, environment or dynamical systems, and single-DOF or multi-DOF dynamical systems. The efficient utilization of the data allows the framework to model extreme events as accurately as possible.

6.2 Contribution

The proposed framework is unique and first of its kind in generating environments for the marine extreme simulations from a machine learning perspective. The key contribution of current research is listed as follows.

- The framework re-formulates the problem of generating extreme marine events in a machine-learning perspective. The concepts in the marine dynamic field are related to a data science task. Examples are abstracting the ocean environment randomness into the seed vector, converting the environment generation problems into a generative machine-learning family.
- The framework can learn a conditional distribution that is spanned by the Fourier phases in a high dimension space. Once the distribution is learned, the

framework can sample the Fourier phases as a vector to establish the environment to model nonlinear extreme events.

- The framework relies on very few assumptions about the dynamical system and the loading environment, which allows the user to apply it to a wide range of problems. The framework is designed to be data-driven, and the data quantity required to use the framework is also designed to be as little as possible to reduce the cost. In the sloshing tank and floating object examples, the framework achieves good results even when the collected CFD data is limited.

- Many methods and models are developed for the first time to address the challenge occurring in the marine extreme context. For example, limited observation of extreme system response is addressed by a novel "bootstrapping" approach. Model architectures are designed and implemented to achieve relatively accurate results.

- By carrying out many experiments, the performance of the machine learning models is measured and compared, which provides the user a guideline on various choices of setup. The cost of applying the proposed models is estimated theoretically and numerically.

6.3 Future Work

Though the novel framework can produce useful results for several marine examples, there are still open questions that have not been answered in the current work. Interesting questions are listed as follows.

- The DRE module is designed to identify a marine system. Though the results of the examples are generally good, it may require more data and a more complex model design to identify other system behavior. In the sloshing tank example, predicting the hydrodynamic force is more difficult than predicting the hydrodynamic moment with the same model design. Hence, determining the amounts of data required is still a case-by-case question.

- The proposed framework is designed without assuming the systems are linear. Nonlinear systems have many complex phenomena that linear systems do not usually have, like chaos and sensitivity of initial conditions. In the most examples discussed in the thesis, the extreme response is designed at a user-specified

timestamp, called design-time t_d. In the current framework, the statistics of the system response at the design time is assumed to be converged and steady. Based on the assumption, all the large responses that the system may undergo during its lifetime are expected to be possible to reconstruct within a short time window $[0, t_d]$. However, such an assumption may not valid for all nonlinear systems.

- The Fourier phases for the wave environment are considered to be correlated as a seed vector in the proposed framework. Learning the joint distribution for the phases in a high-dimensional space is challenging. In most examples of this thesis, the dimension of the seed vector (or the number of Fourier components) is less than fifty. As the dimension increases, the joint distribution becomes more complicated. Therefore, more data needs to be collected and the training process takes more time. With the current framework design, a suggestion to reduce this additional cost is to assume that there exists no correlation among groups of phases. Based on the assumption, only the correlation within each group is considered and the phases from different groups are assumed to be uncorrelated. To determine the partition of the phase groups, statistical analysis is needed.

- Since the framework is data-driven, the collected data and the learning process can be random. Hence, the confidence intervals for the estimated quantities are desired. A typical approach to measuring the confidence intervals is to conduct the estimation process multiple times and the empirical distribution of the estimated quantities yields the confidence intervals. However, Monte Carlo tests for the proposed framework is computationally expensive since each experiment involves data collection, model training, and performance evaluation.

- As shown in the floating object example, different system operating conditions can produce different distributions of the collected data. Therefore, a model that successfully identifies the system in one operating condition may produce errors when predicts the system response at another operating condition. To solve the problem, a dynamic or adaptive system identification process is suggested to be built. When the TEG is generating environments that lead to larger response during the bootstrapping process, a subset of the environments are selected and the DRE model is updated by re-training with these environments.

APPENDICES

APPENDIX A

Gaussian Process Results

Lindgren (1970) and *Tromans et al.* (1991) showed that the most likely wave profile around an a priori maximum crest height is the autocorrelation curve scaled by the crest height. Specifically, on the condition that the crest height a occurs at $t = 0$, the expected wave elevation is given by

$$E[\eta(t)|\eta(0) = a, \dot{\eta}(0) = 0] = a\rho(t) \tag{A.1}$$

where $\eta(t)$ is the Gaussian elevation stochastic process, $\rho(t)$ is the normalized autocorrelation function of $\eta(t)$, which relates to its energy density function by Wiener-Khinchin theorem.

Cartwright and Longuet-Higgins (1956) gave the distribution of positive maxima for narrowband or non-narrowband Gaussian process, normalized by the 0^{th} spectral moment

$$f_H(h) = \frac{2}{1+\sqrt{1-\epsilon^2}}\left[\frac{\epsilon}{\sqrt{2\pi}}e^{-h^2/2\epsilon^2} + \sqrt{1-\epsilon^2}he^{-h^2/2}\Phi\left(\frac{\sqrt{1-\epsilon^2}}{\epsilon}h\right)\right] \tag{A.2}$$

and the corresponding cumulative distribution function (CDF) is

$$\begin{aligned} F_H(h) = \frac{2}{1+\sqrt{1-\epsilon^2}}\Bigg[&-\frac{1}{2}(1-\sqrt{1-\epsilon^2}) + \Phi(h/\epsilon) \\ &- \sqrt{1-\epsilon^2}e^{-h^2/2}\Phi\left(\frac{\sqrt{1-\epsilon^2}}{\epsilon}h\right)\Bigg]. \end{aligned} \tag{A.3}$$

where H is the non-dimensional positive maxima ($0 \leq H < \infty$) and $\epsilon = \sqrt{1 - m_2^2/(m_0 m_4)}$ is the bandwidth parameter of Gaussian spectrum, and m_i is the i^{th} moment of the spectrum.

Ochi et al. (1973) gave the expected number of positive maxima per unit time as

$$n = \frac{1}{4\pi}\left(\frac{1+\sqrt{1-\epsilon^2}}{\sqrt{1-\epsilon^2}}\right)\sqrt{\frac{m_2}{m_0}} \tag{A.4}$$

and applied Order Statistics Theory on local positive maxima to produce the Extreme Value PDF associated with the exposure window.

$$f_{X(T)}(x) = n f_H(x) F_H^{n-1}(x) \tag{A.5}$$

As a result, the most probable extreme maxima in an exposure window T is given by

$$\bar{H}(T) = \sqrt{2\ln\left(\frac{T}{2\pi}\sqrt{m_2/m_0}\right)} \tag{A.6}$$

Equation A.6 is beautiful in connecting time domain, frequency domain, and probability domain.

APPENDIX B

Code Snippet

Python is selected as the major programming language for the current research work. With the help of machine learning libraries, many building blocks in the proposed framework can be implemented efficiently with good readability and performance. Among many modern machine learning libraries, **TensorFlow** (*Abadi et al.*, 2015), powered by Google Inc, is widely used in academia and industry. On top of TensorFlow, an open-source neural network library **Keras** (*Chollet et al.*, 2015) provides a user-friendly upper-level wrapper language for rapid prototyping, and is selected for the modeling tasks. Other used libraries for data processing includes **Numpy** (*Oliphant*, 2006), **scikit-learn** (*Pedregosa et al.*, 2011), **Pandas** (*McKinney et al.*, 2010). The development of current research work is written in an interactive format called **Jupyter-Notebook** (*Kluyver et al.*, 2016). The results in the thesis are plotted using **Matplotlib** (*Hunter*, 2007) and the illustrative diagrams are made using **Mathcha**.

In this appendix, important code snippets are attached for illustrative use. A repository of the notebooks is hosted on Github with complete implementation.

B.1 Threshold Exceedance Generator

B.1.1 Discretize the Fourier phases

```
def phase_cont2disc(Phi):
    num_samples, seq_len = Phi.shape[0], Phi.shape[1]
    res = np.digitize(Phi, phase_bin_edge) - 1
    return res

def phase_disc2cont(Phi):
    res = np.zeros(Phi.shape)
    for i in range(len(Phi)):
        res[i,:] = phase_bin_center[Phi[i,:]]
    return res
```

B.1.2 Sequence padding and One-Hot-Encoding

```
def gen_padded_sequences(dataset):
    res = []
    for seq in dataset:
        for i in range(len(seq)):
            res.append(seq[:i+1])
    integer_encoded = np.array(pad_sequences(res, maxlen=n_fourier, padding='pre'))
    one_hot_encoded = keras.utils.to_categorical(integer_encoded,
     num_classes=len(phase_bin_center)+1)
    return one_hot_encoded
```

B.1.3 Model Architecture

```
model = Sequential()
model.add(LSTM(100, input_shape=(n_fourier-1, len(phase_bin_center)+1), return_sequences=True))
model.add(Dropout(0.1))
model.add(LSTM(100))
model.add(Dense(80, activation='relu'))
model.add(Dense(len(phase_bin_center)+1, activation='softmax'))
print(model.summary())

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

B.1.4 Model Sampling

```
def sample_seqs_batch(model, batch_size):
    res = np.zeros((batch_size, n_fourier))
    seed = np.zeros((batch_size, n_fourier-1))

    def sample_row(p):
        return np.random.choice(np.arange(len(phase_bin_center)+1), p=p)

    for idx in range(n_fourier):
        encoded = keras.utils.to_categorical(seed, num_classes=
         len(phase_bin_center)+1).reshape((batch_size, n_fourier-1, len(phase_bin_center)+1))
        y_proba = model.predict_proba(encoded, verbose=0)
        sampled_class = np.apply_along_axis(sample_row, 1, y_proba)
        seed = np.roll(seed, -1, axis=1)
        seed[:,-1] = sampled_class
        res[:, idx] = sampled_class
    return res
```

B.1.5 Model Training and Bootstrapping

```
def train_and_sample(dataset):

    padded_seqs = gen_padded_sequences(dataset)
    X, y = padded_seqs[:,:-1,:], padded_seqs[:,-1, :]

    usualCallback = EarlyStopping(monitor='val_loss', min_delta=0, patience=5)
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=2020)
    history = model.fit(X_train, y_train,
```

```
                        validation_data=(X_val, y_val),
                        batch_size=int(1000),
                        epochs=3000,
                        callbacks=[usualCallback],
                        verbose=1)

    bootstrapping_quantile = 0.5

    n_samples = int(1.0/bootstrapping_quantile*dataset.shape[0])
    samples_from_model = sample_seqs_batch(model, n_samples)

    # assign all zero idx to 1
    samples_from_model[samples_from_model == 0] = 1

    dataset_cont = phase_disc2cont((dataset-1).astype(int))
    dataset_evaluation = eval_phase(dataset_cont)
    samples_from_model_cont = phase_disc2cont((samples_from_model-1).astype(int))
    samples_evaluation = eval_phase(samples_from_model_cont)

    quantile_boundary = np.quantile(samples_evaluation, 1 - bootstrapping_quantile)
    return samples_from_model[samples_evaluation >= quantile_boundary, :].astype(int)
```

B.2 Design Response Estimator

B.2.1 Dataset Scaling

```
class MyScaler:
    def __init__(self):
        self.mean, self.std = None, None

    def transform(self, dataset):
        self.mean, self.std = np.mean(dataset), np.std(dataset)
        return (dataset - self.mean)/self.std
    def inverse_transform(self, dataset):
        return dataset*self.std + self.mean
```

B.2.2 Dataset Reshaping

```
def reshape_into_input(dataset):
    # reshape into [samples, time steps, features]
    return dataset.reshape((dataset.shape[0], dataset.shape[1], 1))
def reshape_from_input(dataset):
    return dataset.reshape((dataset.shape[0], dataset.shape[1]))
```

B.2.3 Model Architecture

```
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(None, 1)))
model.add(LSTM(50, return_sequences=True))
model.add(LSTM(50, return_sequences=True))
model.add(LSTM(50, return_sequences=True))
```

```
model.add(LSTM(50, return_sequences=True))
model.add(TimeDistributed(Dense(1, activation='linear')))
print(model.summary())

model.compile(loss='mean_squared_error', optimizer='adam')
```

B.2.4 Model Training

```
history = model.fit(X_train, y_train, epochs=500, batch_size=X_train.shape[0], verbose=1)
```

B.3 Dynamic-related

B.3.1 JONSWAP spectrum

```
class Jonswap:

    def __init__(self, Hs, Tp):
        fp = 1.0/Tp
        f_lower = fp/1.5
        f_upper = fp*1.5
        nodefreq = np.linspace(f_lower, f_upper, n_fourier + 1)

        # JONSWAP
        gamma = 3.3
        alpha = 0.0624 / (0.23 + 0.0336 * gamma - 0.185 / (1.9 + gamma))
        sigma = np.ones((len(nodefreq))) * 0.07
        sigma[nodefreq >= fp] = 0.09
        beta = np.exp(-(nodefreq - fp)**2 / (2*sigma**2*fp**2))
        nodeS = alpha * Hs**2 * fp**4 * nodefreq**(-5) * gamma**beta
         * np.exp(-5.0/4.0*(fp/nodefreq)**4)

        # bandwidth
        m0 = np.trapz(nodeS, nodefreq)
        m2 = np.trapz(nodeS*nodefreq**2, nodefreq)
        m4 = np.trapz(nodeS*nodefreq**4, nodefreq)
        bandwidth = np.sqrt(1 - m2**2/(m0*m4))

        # return
        self.freq = 0.5*(nodefreq[1:] + nodefreq[:-1])
        self.amp = np.sqrt((nodeS[1:] + nodeS[0:-1]) * (nodefreq[1:] - nodefreq[0:-1]))
        self.k = (self.freq*2*np.pi)**2 / 9.81
        self.Hs = Hs
        self.Tp = Tp
        self.bandwidth = bandwidth
        self.m0, self.m2, self.m4 = m0, m2, m4
```

B.3.2 Second-order nonlinear wave

```
def eval_phase(Phi, xf, time=[]):

    R = (2*np.pi*jonswap.freq)**2 / 9.81
    N = n_fourier
```

```
K_plus = np.zeros((N, N))
K_minus = np.zeros((N, N))

freq, amp, k, depth = jonswap.freq, jonswap.amp, jonswap.k, jonswap.depth

for i in range(N):
    for j in range(N):
        # plus term
        k_plus = np.abs(jonswap.k[i] + jonswap.k[j])

        tmp1 = np.sqrt(R[j])*(k[i]**2 - R[i]**2)
        tmp2 = np.sqrt(R[i])*(k[j]**2 - R[j]**2)
        tmp3 = np.sqrt(R[i]) + np.sqrt(R[j])
        tmp4 = np.sqrt(R[i]) - np.sqrt(R[j])
        tmp5 = R[i]*R[j]
        tmp6 = R[i] + R[j]
        tmp7 = k[i]*k[j]

        D_plus = tmp3 * (tmp1 + tmp2) + 2*tmp3**2*(tmp7 - tmp5)
        if depth is None:
            D_plus /= tmp3**2 - k_plus
        else:
            D_plus /= tmp3**2 - k_plus*np.tanh(k_plus * depth)

        K_plus[i, j] = (D_plus - (tmp7 - tmp5))/np.sqrt(tmp5) + tmp6

        if i == j:
            continue
        # minus term
        k_minus = np.abs(k[i] - k[j])
        D_minus = tmp4 * (tmp1 - tmp2) + 2*tmp4**2*(tmp7 + tmp5)
        if depth is None:
            D_minus /= tmp4**2 - k_minus
        else:
            D_minus /= tmp4**2 - k_minus*np.tanh(k_minus * depth)

        K_minus[i, j] = (D_minus - (tmp7 + tmp5))/np.sqrt(tmp5) + tmp6

res = np.zeros((Phi.shape[0], len(time)))
time = np.array(time).reshape((1, -1))
k, freq, amp = k.reshape((N,1)), freq.reshape((N,1)), amp.reshape((N,1))

def score_phase(phase):
    phase = phase.reshape((N,1))
    psi = k * xf - 2*np.pi*freq * time + phase
    eta1 = np.sum(amp * np.cos(psi), axis=0)
    eta2 = np.zeros(eta1.shape)
    for i in range(N):
        for j in range(N):
            eta2 += amp[i]*amp[j]*(K_minus[i,j]*np.cos(psi[i,:]-psi[j,:])
             + K_plus[i, j]*np.cos(psi[i,:]+psi[j,:]))
    eta2 /= 4.0
    return eta1 + eta2
```

```
    for idx in range(len(Phi)):
        if idx > 0 and idx % (Phi.shape[0]/10) == 0:
            print(f'evaluating dataset: {idx *100.0/Phi.shape[0]}%')
        res[idx,:] = score_phase(Phi[idx,:])

    if res.shape[1] == 1:
        return res[:,0]
    else:
        return res
```

B.3.3 Ship Roll Environment

```
params = {
    'rho_air': 1.225,
    'rho_sw': 1.025*10**3,
    'V': 4.241*10**4,
    'Bs': 32.25,
    'AL': 8814,
    'H': 20.86,
    'Cm': 1.1137,
    'K': 0.003,
    'w0': 0.1794,
    'gamma': 0.777,
    'GM': 1.06,
    'Uw': 29.50, # wind velocity
    'mu': 8.5*10**(-3),
    'beta': 0.385,
    'alpha_e': 8.607*10**(-2),
    'zeta_e': 0.2399,
    'IAxx': 1.4033*10**10,
    'Delta': 4.2608*10**8,
    'c1': 1.06,
    'c3': 0.174,
    'c5': -6.4269,
    'c7': 5.3323
}

class WaveMomentSpec:

    def __init__(self):
        nodew = np.logspace(-0.6, 0, n_fourier_wave + 1)
        nodefreq = nodew/(2*np.pi)

        Uw = params['Uw']
        H13 = -2*10**(-5)*Uw**3 + 8.2*10**(-3)*Uw**2 + 0.1456*Uw - 0.1599
        T01 = 3.86*np.sqrt(H13)
        A = 172.75*H13**2/T01**4
        B = 691/T01**4
        g = 9.81
        gamma = np.ones(nodefreq.shape)*params['gamma']
        gamma[nodew > np.sqrt(4*np.pi*g/params['Bs'])] = 0.0

        Salpha = A/(g**2*nodew)*np.exp(-B/nodew**4)
        Smw = (params['Delta']*params['GM']*gamma)**2*Salpha
```

```
        # return
        self.freq = 0.5*(nodefreq[1:] + nodefreq[:-1])
        self.amp = np.sqrt((Smw[1:] + Smw[:-1]) * (nodew[1:] - nodew[:-1]))
        self.nodew = nodew
        self.Smw = Smw

class WindMomentSpec:

    def __init__(self):
        nodew = np.logspace(-2, 0, n_fourier_wind + 1)
        nodefreq = nodew/(2*np.pi)

        Uw, AL, rho_air = params['Uw'], params['AL'], params['rho_air']
        Cm, H = params['Cm'], params['H']
        K = 0.003
        XD = 600*nodew/(np.pi*Uw)
        Swind = 4*K*Uw**2/nodew*XD**2/(1 + XD**2)**(4.0/3.0)
        Chi = 1.0/(1 + (nodew*np.sqrt(AL)/np.pi/Uw)**(4.0/3.0))
        Sam = (rho_air*Cm*Uw*AL*H)**2*Chi**2*Swind

        # return
        self.freq = 0.5*(nodefreq[1:] + nodefreq[:-1])
        self.amp = np.sqrt((Sam[1:] + Sam[:-1]) * (nodew[1:] - nodew[:-1]))
        self.nodew = nodew
        self.Sam = Sam

def cal_angle_of_vanishing_stability():
    c1, c3, c5, c7 = params['c1'], params['c3'], params['c5'], params['c7']
    vanishing_angle = np.roots([c7, 0, c5, 0, c3, 0, c1, 0])
    vanishing_angle = np.real(vanishing_angle[np.isreal(vanishing_angle) & (vanishing_angle > 0)])
    return np.min(vanishing_angle)

rho_air, Cm, Uw, AL, H = params['rho_air'], params['Cm'], params['Uw'], params['AL'], params['H']
Mwind_static = 0.5*rho_air*Cm*AL*H*Uw**2

def cal_angle_of_equilibrium():
    c1, c3, c5, c7 = params['c1'], params['c3'], params['c5'], params['c7']
    equilibrium_angle =
     np.roots([c7, 0, c5, 0, c3, 0, c1, -Mwind_static/params['Delta']])
    equilibrium_angle =
     np.real(equilibrium_angle[np.isreal(equilibrium_angle) & (equilibrium_angle > 0)])
    return np.min(equilibrium_angle)
```

B.3.4 Ship Roll Solver

```
def score_phase(phase_time_tuple):
    phase, time = phase_time_tuple[0], phase_time_tuple[1]
    c1, c3, c5, c7 = params['c1'], params['c3'], params['c5'], params['c7']
    mu, beta, IAxx, Delta = params['mu'], params['beta'], params['IAxx'], params['Delta']
    initial_disp, initial_velo = equilibrium_angle, 0.0

    def dy(t, y):
```

```
    damping = ( 2*mu*y[1] + beta*y[1]*np.abs(y[1]) ) * IAxx
    spring = ( c1*y[0] + c3*y[0]**3 + c5*y[0]**5 + c7*y[0]**7 ) * Delta
    Mwind = np.sum(wind_spec.amp*np.cos(2*np.pi*wind_spec.freq*t + phase[:n_fourier_wind]))
    Mwave = np.sum(wave_spec.amp*np.cos(2*np.pi*wave_spec.freq*t + phase[n_fourier_wind:]))
    return [y[1], (Mwind_static + Mwind + Mwave - damping - spring)/IAxx]
sol = solve_ivp(dy, [0, time_length], [initial_disp, initial_velo], t_eval=time)
return sol.y[0,:]
```

BIBLIOGRAPHY

BIBLIOGRAPHY

Abadi, M., et al. (2015), TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org.

Alford, L. K. (2008), Estimating extreme responses using a non-uniform phase distribution., Ph.D. thesis.

Belenky, V. (2011), On self-repeating effect in reconstruction of irregular waves, in *Contemporary Ideas on Ship Stability and Capsizing in Waves*, pp. 589–597, Springer.

Belenky, V., C. Bassler, M. Dipper, B. Campbell, K. Weems, and K. Spyrou (2010), Direct assessment methods for nonlinear ship response in severe seas, in *Proc. of ITTC Workshop on Seakeeping, Seoul, Korea*.

Belenky, V., K. M. Weems, C. C. Bassler, M. J. Dipper, B. L. Campbell, and K. J. Spyrou (2012), Approaches to rare events in stochastic dynamics of ships, *Probabilistic Engineering Mechanics*, *28*, 30–38.

Campbell, B., V. Belenky, and V. Pipiras (2014), On the application of the generalized pareto distribution for statistical extrapolation in the assessment of dynamic stability in irregular waves, in *14th International Ship Stability Workshop. Kuala Lumpur, Malaysia*.

Cartwright, D., and M. S. Longuet-Higgins (1956), The statistical distribution of the maxima of a random function, *Proc. R. Soc. Lond. A*, *237*(1209), 212–232.

Chollet, F., et al. (2015), Keras, https://keras.io.

Davison, A. C., and R. L. Smith (1990), Models for exceedances over high thresholds, *Journal of the Royal Statistical Society: Series B (Methodological)*, *52*(3), 393–425.

Duchi, J., E. Hazan, and Y. Singer (2011), Adaptive subgradient methods for online learning and stochastic optimization, *Journal of machine learning research*, *12*(Jul), 2121–2159.

Filip, G., W. Xu, and K. Maki (2017), Urans predictions of resistance and motions of the kcs in head waves, *Tech. rep.*

Filip, G. P., W. Xu, and K. J. Maki (2018), Prediction of extreme wave slamming loads on a fixed platform, in *ASME 2018 37th International Conference on Ocean, Offshore and Arctic Engineering*, American Society of Mechanical Engineers Digital Collection.

Forristall, G. Z. (2000), Wave crest distributions: Observations and second-order theory, *Journal of physical oceanography*, *30*(8), 1931–1943.

Gaidai, O., G. Storhaug, and A. Naess (2016), Extreme value statistics of large container ship roll, *Journal of ship research*, *60*(2), 92–100.

Greenshields, C. J. (2015), Openfoam user guide, *OpenFOAM Foundation Ltd, version*, *3*(1), e2888.

Gumbel, E. (), Statistics of extremes. columbia university press, 1958.

Hebb, D. O. (2005), *The organization of behavior: A neuropsychological theory*, Psychology Press.

Higuera, P., J. L. Lara, and I. J. Losada (2014), Three-dimensional interaction of waves and porous coastal structures using openfoam®. part i: Formulation and validation, *Coastal Engineering*, *83*, 243–258.

Higuera, P., I. J. Losada, and J. L. Lara (2015), Three-dimensional numerical wave generation with moving boundaries, *Coastal Engineering*, *101*, 35–47.

Hinton, G., N. Srivastava, and K. Swersky (2012), Neural networks for machine learning lecture 6a overview of mini-batch gradient descent, *Cited on*, *14*(8).

Hochreiter, S., and J. Schmidhuber (1997), Long short-term memory, *Neural computation*, *9*(8), 1735–1780.

Hopfield, J. J. (1982), Neural networks and physical systems with emergent collective computational abilities, *Proceedings of the national academy of sciences*, *79*(8), 2554–2558.

Hunter, J. D. (2007), Matplotlib: A 2d graphics environment, *Computing in science & engineering*, *9*(3), 90–95.

Jensen, J. J. (1996), Second-order wave kinematics conditional on a given wave crest, *Applied Ocean Research*, *18*(2-3), 119–128.

Jensen, J. J. (2008), Extreme value predictions and critical wave episodes for marine structures by form, *Ships and Offshore Structures*, *3*(4), 325–333.

Jensen, J. J. (2009), Stochastic procedures for extreme wave load predictions–wave bending moment in ships, *Marine Structures*, *22*(2), 194–208.

Kim, D. H. (2012), Design loads generator: Estimation of extreme environmental loadings for ship and offshore applications., Ph.D. thesis.

Kingma, D. P., and J. Ba (2014), Adam: A method for stochastic optimization, *arXiv preprint arXiv:1412.6980*.

Kluyver, T., et al. (2016), Jupyter notebooks – a publishing format for reproducible computational workflows, in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, edited by F. Loizides and B. Schmidt, pp. 87 – 90, IOS Press.

Kogiso, N., and Y. Murotsu (2008), Application of first order reliability method to ship stability–final report of scape committee (part 5)–, in *Proceedings of the 6th Osaka Colloquium on Seakeeping and Stability of Ships*.

Leadbetter, M. R. (1991), On a basis for peaks over thresholdmodeling, *Statistics & Probability Letters*, *12*(4), 357–362.

Lindgren, G. (1970), Some properties of a normal process near a local maximum, *The Annals of Mathematical Statistics*, pp. 1870–1883.

McCulloch, W. S., and W. Pitts (1943), A logical calculus of the ideas immanent in nervous activity, *The bulletin of mathematical biophysics*, *5*(4), 115–133.

McKinney, W., et al. (2010), Data structures for statistical computing in python, in *Proceedings of the 9th Python in Science Conference*, vol. 445, pp. 51–56, Austin, TX.

Naess, A., and O. Gaidai (2008), Monte carlo methods for estimating the extreme response of dynamical systems, *Journal of Engineering Mechanics*, *134*(8), 628–636.

Næss, A., and O. Gaidai (2009), Estimation of extreme values from sampled time series, *Structural safety*, *31*(4), 325–334.

Naess, A., O. Gaidai, and O. Batsevych (2010), Prediction of extreme response statistics of narrow-band random vibrations, *Journal of engineering mechanics*, *136*(3), 290–298.

Ochi, M. K. (1990), *Applied probability and stochastic processes*, Wiley New York.

Ochi, M. K., et al. (1973), On prediction of extreme values, *Journal of Ship Research*, *17*(01), 29–37.

Olah, C. (2015), Understanding lstm networks.

Oliphant, T. E. (2006), *A guide to NumPy*, vol. 1, Trelgol Publishing USA.

Paroka, D., and N. Umeda (2006), Capsizing probability prediction for a large passenger ship in irregular beam wind and waves: comparison of analytical and numerical methods, *Journal of Ship Research*, *50*(4), 371–377.

Pedregosa, F., et al. (2011), Scikit-learn: Machine learning in Python, *Journal of Machine Learning Research*, *12*, 2825–2830.

Rice, S. O. (1945), Mathematical analysis of random noise, *Bell system technical journal*, *24*(1), 46–156.

Rosenblatt, F. (1958), The perceptron: a probabilistic model for information storage and organization in the brain., *Psychological review*, *65*(6), 386.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986), Learning representations by back-propagating errors, *nature*, *323*(6088), 533–536.

Shinozuka, M., and G. Deodatis (1991), Simulation of stochastic processes by spectral representation.

Troesch, A. W. (1997), Study of the use of nonlinear time domain simulation in a design loads generator (dlg), *Final Report to the American Bureau of Shipping*.

Tromans, P. S., A. R. Anaturk, P. Hagemeijer, et al. (1991), A new model for the kinematics of large ocean waves-application as a design wave, in *The First International Offshore and Polar Engineering Conference*, International Society of Offshore and Polar Engineers.

Weihull, W. (1951), A statistical distribution function of wide applicability, *J Appl Mech*, *18*, 290–293.

Werbos, P. J. (1990), Backpropagation through time: what it does and how to do it, *Proceedings of the IEEE*, *78*(10), 1550–1560.

Widrow, B., and M. E. Hoff (1960), Adaptive switching circuits, *Tech. rep.*, Stanford Univ Ca Stanford Electronics Labs.

Xu, W., G. Filip, and K. J. Maki (2019), A method for the prediction of extreme ship responses using design–event theory and computational fluid dynamics, *Journal of Ship Research*.