MASTER OF SCIENCE THESIS

Budapest University of Technology and Economics

Improving Wave Generation in CFD-based Numerical Wave tank using Machine Learning

IRFAN UL HASSAN

May 12, 2022

BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS
FACULTY OF MECHANICAL ENGINEERING
DEPARTMENT OF APPLIED MECHANICS

BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS
FACULTY OF MECHANICAL ENGINEERING
DEPARTMENT OF APPLIED MECHANICS

Irfan Ul Hassan

Master Thesis

Improving wave generation in CFD-based numerical wave tank using Machine Learning

Consultant:
Dr. Davidson Josh
Assistant Professor

Supervisor:
Dr. Habib Giuseppe
Assistant Professor

Budapest, 2022

DECLARATION

Declaration of individual work

I, *Irfan Ul Hassan* (KN04N2), the undersigned, student of the Budapest University of Technology and Economics hereby declare that the present thesis has been prepared by myself without any unauthorized help or assistance such that only the specified sources (references, tools, etc.) were used. All parts taken from other sources word by word or after rephrasing but with identical meaning were unambiguously identified with explicit reference to the sources utilized.

Budapest, 2022

Irfan ul Hassan

Contents

Abstract

The oceans are one of the largest resources of mankind, covering more than 70% of the Earth's surface and maintaining huge industries such as fishing, shipping, tourism, offshore oil, gas and renewable energies. Numerical wave tanks (NWTs) can be a vital tool for understanding, designing and optimizing systems in the ocean. A key element of the NWT is the wavemaker, which is responsible for generating NWT waves. For the wavemaker to generate desired water waves for modelling of nonlinear free surface waves, hydrodynamic forces and floating body motions, fine-tuning and iterative calibration are required, which is computationally intensive and thereby increasing the number of CFD simulations to run and the user's work. This study reports on a numerical wavetank implemented in OpenFOAM and improving the wave generation capabilities of impulse source wavemaker method by employing data-driven system identification and machine learning techniques to identify models mapping a desired wave surface-elevation time-trace at a probe to the wavemaker input required to create it. A comparative study of performances of various model structures is presented for both forward and inverse NWT problems based on their modal accuracy, complexity and computational cost. It was demonstrated that Autoregressive with exogeneous inputs emerged as the best method to model the dynamic behaviour of NWT satisfactorily when trained for a single training example of medium sized amplitude. ARX was able to validate well over the datasets with amplitude variations of almost 10 times the amplitude of the actual training dataset.

Keywords : *OpenFOAM, Numerical Wavetanks (NWTs), Nonlinear System Identification, Neural Networks, Machine Learning, Dynamic systems*

* * *

Acknowledgement

First and foremost, praises to God, the Almighty for his showers of blessings throughout my research work to complete the M.Sc. thesis successfully. I would like to express my deep and sincere gratitude to my thesis supervisor Dr. Habib Giuseppe, assistant professor, Department of Applied Mechanics, and co-supervisor Dr. Josh Davidson, assistant professor, Department of Fluid Mechanics, Faculty of Mechanical Engineering, Budapest University of Technology and Economics for giving me an opportunity to carry out my research work under their kind supervision and providing invaluable guidance throughout my thesis work. I couldn't have imagined completing my thesis work without their continuous support. It was a great privilege and honor to work and study under their supervision. I would like to thank Dr. Josh specially for providing me with the necessary tools and guidance to carry out simulation part of my thesis work. Finally, I would like to thank my family and friends, who continuously supported me through my two years of master studies, thesis work and academically throughout my life.

Budapest, 2022

Nomenclature

Latin symbols

Notation	Name, comment, value	Dimension
g	gravitational acceleration	m/s^2
p	pressure	bar
v	velocity	m/s

Greek symbols

Notation	Name, comment, value	Dimension
φ	angle	rad
μ	dynamic viscosity	Pa.s
α	volume of fraction	No units
∇	del operator	No units
ω	angular frequency	rad/s
ρ	density	kg/m^3

Abbreviations

Notation	Name, comment, value
AIC	Akaike Information Criterion
ARX	Autoregressive with External Input.
CFD	Computational Fluid Dynamics
DL	Deep Learning
FFT	Fast Fourier Transformation
HW	Hammerstein Wiener
LS	Least Square
NARX	Nonlinear Autoregressive with External Input
NN	Neural Networks
NWT	Numerical Wave Tanks
OpenFOAM	Open Source Field Operation and Manipulation.
OE	Output Error.
SISO	Single Input Single Output
SYSID	System Identfication
VOF	Volume of Fluid method

List of Figures

List of Tables

Chapter 1

1 Introduction

A Numerical wave tank (NWT) which is the generic name of numerical simulators, has been one of the most important topics of interest to researchers and scholars working in the areas of the marine industries for modelling nonlinear free surface waves, hydrodynamic forces and floating body motions. Because of improved methodology and increasing computational capacity, NWTs are becoming a viable supplement, even an alternative, for physical model testing in the exploration of fluid-structure interaction. However, all wavemakers are required to obtain desired surface waves or time trace of surface elevation at some location in the tank which is usually achieved by some methods that control wave generation process in a desired fashion.

Although different types of numerical wavemakers (NWM) exist to create waves in NWT, in this study only impulse source method is used. The difficulty in utilising an impulse source as an NWM lies in calculating the required source function to obtain a desired target wave series. Since the free surface is not a variable in the Reynolds-averaged Navier–Stokes equations (RANS) (see equation 2.41), there is no direct expression relating the impulse source function to the resulting generated wave series [4]. Furthermore, in many practical CFD applications, the accurate wave trace description is required at some distance away from the wave maker, typically in the middle of the domain [5]. This requires fine-tuning and iterative calibrations to achieve the desired free surface modelling, thereby increasing the number of CFD simulations to run and the user's work. The goal is to explore data-driven system identification and deep learning techniques and identify the models mapping a desired surface wave-elevation at the probe location in the tank to the wavemaker input required to create it.

1.1 Objectives

Numerical wave tanks (NWTs) are playing a significant role in ocean engineering and energy wave industries. NWTs are used to generate water waves in a controlled manner for modelling of other marine and coastal related phenomena. Unlike many other dynamic systems, governed by linear input-output relationships, NWT which is implemented in computational fluid dynamics (CFD) solver, OpenFOAM (v2012), exhibits nonlinear relationship between the generated free surface waves and the wavemaker input required to create these waves. With the impulse wavemaker method, it is not possible to generate desired waves in NWTs as prior knowledge of exact description of impulse as input to the OpenFOAM is required which is not known. The iterative approach of running many CFD simulations for achieving the desired surface wave-elevation at some probe length in NWTs

is inefficient and time consuming. Such type of problems where the effect (free surface waves) is known and cause (wavemaker input) is to be determined are termed as inverse problems. Moreover, the presence of a time varying delay between the excitation applied at the source region in NWT and the free surface waves detected by the probe poses additional complexity to the existing problem. Since these are multi-frequency waves, travelling with different velocities and thus need different amount of finite time to reach to the probe. System identification and deep learning techniques are explored in building mathematical models and thereby establishing a nonlinear input-output relationship from data with no or limited physical insights of the system.

There are various sub-objectives within the primary goal of improving wave generation in NWT using data-driven system identification. The first sub-objective is to generate free surface waves in OpenFOAM for inputs which can captivate most of the dynamics of the system. The data acquisition of surface waves is made after performing multiphase simulations of NWT in a CFD solver. The second objective is to use exciting, challenging and yet largely unexplored application field - system identification with tailor-made model structures and fitting criteria. Once the data is generated, it is then used for developing black box models which can map the unknown nonlinear relationship between surface waves and wavemaker input. The selection of the model form or architecture is a challenging part of this process. Various discrete model structures and deep learning models such as ARX, Neural Network etc. are explored to describe the uncertain components of the dynamics, while retaining structural (physical) knowledge, if available. Once a particular model structure is selected, the most challenging part then is the selection of its model order, which may require some prior knowledge of the system. The final objective is cross validation of the model developed within acceptable accuracy of the fitted criteria. The comparative study of the performance of various models is also demonstrated, based on the accuracy, model complexity and computational costs.

It is noteworthy to mention that the objective here is to solve the NWT problem both in forward and inverse manner. A forward problem is to find a unique effect of a given cause by a data-driven system identification approach. Forward problems are usually well-posed, i.e., they have a unique solution which is insensitive to small changes of the initial values. Inverse problems are the opposite to forward problems, meaning that one is given the effect and the task is to recover the cause . The system identification process in inverse problem is the case 'model is to be identified for known output (surface-wave elevation) and we want to identify input (wavemaker input)'.

Additionally, a case study of 1 degree of freedom linear dynamic system such as Mass-Spring-Damper is also presented, where the same set of tools and strategies as applied to NWT problem are explored in order to check the capabilities of data-driven system identification and deep learning in establishing a cause and effect relationship both in direct and inverse fashion.

Chapter 2

2 Theory and Methods

2.1 System Identification

System identification is of special interest in science and engineering. System identification is concerned with building mathematical models of dynamical systems based on measurements of input and output signals. There are four essential decisions in a system identification problem [6]:

- Design an experiment or setting up simulation environment and collect the data.
- Decide upon a model architecture.
- Estimate the parameters of the model architecture by adjusting it to the data.
- Validate the model.

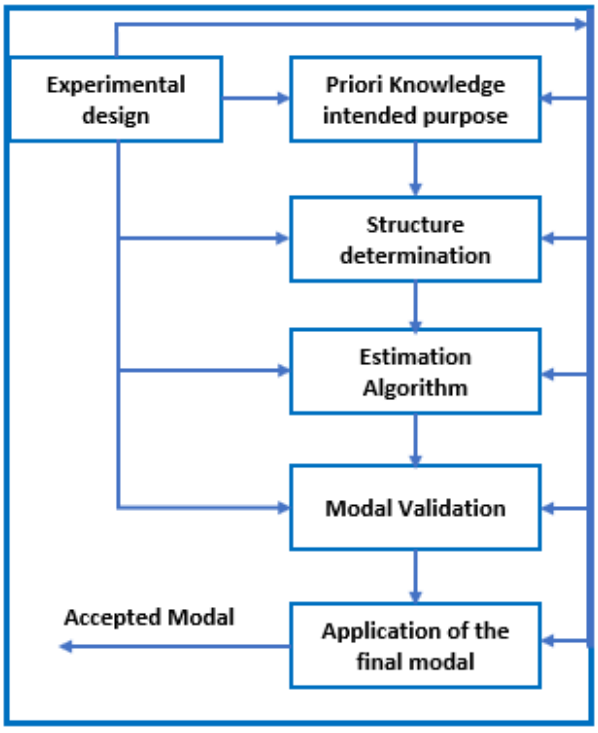


Figure 2.1: System Identification Process

2.2 Modelling approaches in system identification

There are basically three different modeling approaches employed in system identification which are:

White box models, also known as interpretable models, are based on the first principle i.e. physical equations. Equations and parameters are obtained by theoretical modeling. But mostly such models can be complex or inadequate knowledge about the system can restrict their usage.

Black box models: Models that are based exclusively on measurement data are known as black box models. There is no previous model available. Most system identification algorithms are based on this model. There is no direct relationship between the model parameters and first principles.

Grey box Models: Such models represents a trade off between white box and black box. A model which is based on both system knowledge and experimental data. However, there are still a number of unknown free parameters in such a model that can be determined via system identification.

In this thesis black box data driven modeling approach is utilized to identify most suitable model that maps nonlinear relationship between surface wave-elevation to the force input in NWT

2.3 A brief overview of optimization techniques

Finding optimal points of a model is crucial for its better performance in both prediction and simulation. For supervised learning, where for each input data there is a labelled output, various optimization techniques are used, and the objective is to find the minimum of some loss function. Sum of Squared error, also known as Mean Squared error is the most popular choice for the loss functions:

$$F(\underline{\theta}) = \sum_{i=1}^{N} \varepsilon^2(i) \quad \text{with} \quad \varepsilon(i) = y(i) - \hat{y}(i) \tag{2.1}$$

Least squares (LS) and nonlinear least squares (NLS) are the optimization problems that use this special type of loss function. As the gradient of this loss function is equated to zero in order to find minimum, leads to the linear system of equations. Moreover, If some weights are provided to each square error term, then the resulting loss function is called weighted least square and is useful in a situation when data points are of varying quality. In this case noise has independent normal distribution whereas in LS. noise is white with constant variance [7].

2.3.1 Linear Regression and Least Squares

In a linear optimization problem with sum of squared error loss function, the model output $\hat{y}$ depends linearly on n parameters

$$\hat{y} = \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n = \sum_{j=1}^{n} \theta_j x_j \quad \text{with} \quad x_j = f_j(\underline{u}) \tag{2.2}$$

where the coefficient θ_j are the unknown parameters. The goal is to find the model output that best approximates the process output y in the least squares sense, i.e., with the minimal sum of squared error loss function value. Let's assume we have N data sample points so that the difference between the measured output and the model prediction in vector/matrix notation is:

$$\varepsilon = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - \mathbf{X}\boldsymbol{\theta} \tag{2.3}$$

where ε is called the residual.

$$\begin{aligned} \boldsymbol{\varepsilon} &= [\varepsilon(1)\ \varepsilon(2) \dots \varepsilon(N)]^T, \\ \mathbf{y} &= [y(1)\ y(2) \dots y(N)]^T, \\ \hat{\mathbf{y}} &= [\hat{y}(1)\ \hat{y}(2) \dots \hat{y}(N)]^T \end{aligned} \tag{2.4}$$

$$\begin{aligned} \mathbf{X} &= \begin{bmatrix} x_1(1) & x_2(1) & \dots & x_n(1) \\ x_1(2) & x_2(2) & \dots & x_n(2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(N) & x_2(N) & \dots & x_n(N) \end{bmatrix}, \\ \boldsymbol{\theta} &= \begin{bmatrix} \theta_1 & \theta_2 & \dots & \theta_n \end{bmatrix}^T. \end{aligned} \tag{2.5}$$

where $\mathbf{X}$ and $\mathbf{y}$ are called the data matrix and the observation vector respectively. The least squares problem (2.1) becomes:

$$F(\underline{\theta}) = \frac{1}{2}\underline{\varepsilon}^{\mathrm{T}}\ \underline{\varepsilon} \longrightarrow \min_{\underline{\theta}} \tag{2.6}$$

Taking its gradient that leads to the well known orthogonal equation which provides the estimated parameters that minimizes the least square error:

$$\hat{\boldsymbol{\theta}} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} \tag{2.7}$$

The matrix $(\mathbf{X}^{\mathrm{T}}\mathbf{X})$ in (2.7), also known as Hessian Matrix is decisive for the accuracy of a numerical inversion and its direct computation may pose ill conditioned problem (the condition number of $(\mathbf{X}^{\mathrm{T}}\mathbf{X})$ is approximately the square of the condition number of the data matrix $\mathbf{X}$). A QR decomposition method instead can be used which computes the data matrix $\mathbf{X}$ directly without computing $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ [8].

2.3.2 Regularization

It is also termed as Ridge Regression. Increase in regression terms may increase model flexibility but it can also increase additional possibility of ill conditioned problem and increased estimated variance [9]. To avoid this condition a penalty term $\lambda(|\underline{\theta}|^2)$ is introduced to (2.1) and cost function becomes

$$F(\underline{\theta}) = \frac{1}{2}\underline{\varepsilon}^{\mathrm{T}}\ \underline{\varepsilon} + \lambda(|\underline{\theta}|^2) \longrightarrow \min_{\underline{\theta}} \tag{2.8}$$

and the regularized Least Square problem leads to the following parameter estimate:

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\underline{\mathbf{I}})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} \tag{2.9}$$

This additional penalty term takes care to all of those parameters which are not important for solving LS problem. Such redundant terms are pushed towards zero in order to reduce the penalty term.

2.3.3 Gradient Based Algorithm

It is a nonlinear local optimization technique extensively used in machine learning. It works on a principle to change the parameter vector θ_{k-1} in proportion to some learning rate (step size), η into a direction d_{k-1} that points towards a gradient direction g_{k-1} rotated and scaled by some direction matrix R_{k-1}.

$$\underline{\theta}_k = \underline{\theta}_{k-1} - \eta_{k-1}\underline{d}_{k-1} \;\&\; \underline{d}_{k-1} = \underline{R}_{k-1}\underline{g}_{k-1} \tag{2.10}$$

Once the parameters are initialized randomly, the aim of each iteration is to reduce the loss function and find local minimum. How big the steps are, the gradient descent takes into the direction of the local minimum are determined by the learning rate. It is important to set the learning rate to an appropriate value, which is neither too low nor too high. Higher values cause it to bounces back and forth between the convex function of gradient descent whereas smaller values causes algorithm to find the local minimum eventually while taking a higher toll on time[7].

Recently a new stochastic first order gradient based optimization algorithm, 'Adaptive Moment Estimation', (ADAM) has been introduced and is becoming popular in deep learning community. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods. it accelerates the gradient descent algorithm by taking into consideration the 'exponentially weighted average' of the gradients [10].

2.3.4 Nonlinear Least Square Problem

Loss function in nonlinear optimization problem is expressed as:

$$F(\underline{\theta}) = \frac{1}{2}\sum_{i=1}^{m} f^2(i, \underline{\theta}) = \frac{1}{2}\underline{\mathbf{f}}^T\underline{\mathbf{f}} \tag{2.11}$$

This is the nonlinear least square function, equation 2.1 is a special case of this function. It is required to minimize $||f(\theta)||$, or equivalently to find

$$\underline{\theta}^* = \mathrm{argmin}_\theta\{F(\underline{\theta})\} \tag{2.12}$$

Taking the second derivative leads to:

$$\mathbf{F}''(\theta) = \mathbf{J}(\theta)^\top\mathbf{J}(\theta) + \sum_{i=1}^{m} f_i(\theta)\mathbf{f}_i''(\theta) \tag{2.13}$$

where $\mathbf{J} \in \mathbb{R}^{m \times n}$ is the Jacobian. And $\mathbf{f}$ usually represents residuals (error). General optimization methods can solve least squares problems, but there are special approaches that are more efficient and don't require the implementation of second derivatives. Such methods can achieve convergence better than linear, and in some cases even quadratic convergence [11]. Some of the technique used to solve nonlinear least square problems are gradient, conjugate gradient, Guass-Newton and Lebenberg-Marquardt algorithms [12] [13].

2.3.5 Lebenberg-Marquardt

It is one of the most widely used technique to solve nonlinear least squares problem. It can be thought as an extention of Guass-Newton Method or a combination of the steepest descent and the Gauss-Newton methods. Gradient can be expressed as $\underline{g} = \underline{J}^T \underline{f}$ and for small f, Hessian is approximated by $\underline{H} \approx \underline{J}^T \underline{J}$. The additional term $\beta_{k-1}\ \underline{I}$ added to Guass-Newton Method is equivalent to the regularization technique in ridge regression for linear least squares problems [7]. The algorithm can be expressed as:

$$\underline{\theta}_k = \underline{\theta}_{k-1} - \eta_{k-1} \left(\underline{J}_{k-1}^{\mathrm{T}} \underline{J}_{k-1} + \beta_{k-1} \underline{I} \right)^{-1} \underline{J}_{k-1}^{\mathrm{T}} \underline{f}_{k-1} \tag{2.14}$$

The Gauss-Newton algorithm is characterized by its fast convergence, but it is often unstable. The gradient algorithm is generally stable, but it is very slow at the neighbourhood of the optimum. The damping term β_{k-1} in equation 2.14 allows a compromise between the stability of the gradient algorithm and the rapidity of the Newton algorithm [14] [15]. For smaller values of β_{k-1}, Levenberg-Marquardt algorithm approaches to Guass-Newton method while for larger values of β_{k-1}, it approaches the steepest descent method. Far away from the optimum the Guass-Newton method may diverge, and a larger value of β_{k-1} should be chosen. Close to the optimum the second order approximation of the loss function performed by the Guass-Newton method is very good, and a small β_{k-1} should be chosen.

2.4 Discerete time Models Structures

In recent years, the modeling, identification, and control of unknown nonlinear dynamic systems have received a lot of attention. Developing mathematical models based solely on first principles with the derivation of accurate and complete physical equations describing input-output relationships may not be possible due to the complex nonlinear nature of the systems. As a result we frequently need to obtain models based on input-output data with no or limited physical knowledge to represent this unknown nonlinear behaviour of the system. In system identification most of the models are based on discrete time models.

2.4.1 Autoregressive with Exogenous Input (ARX)

ARX modelling was the subject of studies in several fields such as chemical engineering [16] [17], medicine [18], agriculture and biological science [19] [20], energy and the power [21], Energy economics [22]. For linear dynamic systems, ARX is widely used black box model for input/output relationship [6]. In this study, much focus is devoted on the development of a mathematical model representing a nonlinear relationship between wave generation and wavemaker input in NWTs, from only (inputs/outputs) data obtained from CFD simulation using the model parametric ARX. Because of its simplicity and efficiency, ARX is used for modelling of the dynamic behaviour of NWTs. The aim is to analyze the model orders, the time delay and the validation of the identified model.

The ARX structure, as shown in Fig 2.2 describes the input effects $u(t)$ on the process

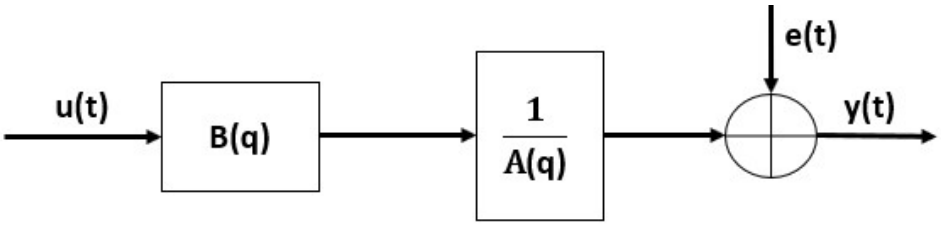


Figure 2.2: The ARX model structure

output $y(t)$. The ARX model is represented by the following expression:

$$y(t) = \sum_{i=1}^{n_y} a_i y(t-i) + \sum_{i=0}^{n_u} b_i u\left(t - n_d - i\right) + e(t) \tag{2.15}$$

where $e(t)$ refers to the noise supposed to be Gaussian. $[a_i\ b_i]$ are the model parameters. $[n_y\ n_u]$ indicate the order of the polynomials of the output $A(q)$ and the input $B(q)$ respectively. The parameter n_d is the time delay between $y(t)$ and $u(t)$ and the number of parameters to be estimated is $N_{par} = n_y + n_u + 1$. The polynomial representation of the equation 2.15 is given as follows:

$$A(q)y(t) = B(q)u(t - n_d) + e(t) \tag{2.16}$$

where $A(q)$ and $B(q)$ are given by:

$$\begin{aligned} A(q) &= 1 + a_1 q^{-1} + \cdots + a_{n_a} q^{-n_y} \\ B(q) &= b_1 q^{-1-n_d} + \cdots + b_{n_b} q^{-n_u - n_d} \end{aligned} \tag{2.17}$$

q^{-1} is the delay operator such as:

$$u(t = 1) = q^{-1} \tag{2.18}$$

$A(q)$ and $B(q)$ are estimated by the least squares identification [23] [24].

2.4.2 Nonlinear Autoregressive with Exogenous Inputs (NARX)

Nonlinear Autoregressive with Exogenous Inputs (NARX) models are a very common class of nonlinear models and can describe a large class of nonlinear systems [25]. The NARX representation has attracted considerable interest in modeling nonlinear systems, and many relevant analysis tools and identification algorithms have been developed in recent years [26]. The NARX model is an extension of the linear ARX model [27] [28]. The Autoregressive (AR) model is used when current output is dependent only on the previous outputs, and the ARX model is used when there is exogenous input given to the AR model [29], as shown in Fig 2.2. The NARX model is expressed in terms of the discrete-time input–output equation as:

$$\hat{y}(t) = f\left[u(t), u(t-1), u\left(t-n_u\right), \cdots, u\left(t-n_u-n_d\right), y(t-1), \cdots,\ y\left(t-n_y\right)\right] + e(t) \tag{2.19}$$

where $y(t)$ and $\hat{y}(t)$ are the target and predicted output variables, respectively; $u(t)$ is the network's input variable; n_u and n_y are the input and output variable's time delays; f is a nonlinear function which needs to be identified from given observed data; and $e(t)$ is the model error between target and prediction. Increasing these values makes the model more flexible and capable of displaying more complicated dynamical behavior, but too high orders can cause overfitting. The number of samples before the output reacts to the input is denoted by input time delay, n_d (in such case $n_d \geq 0$, and system is termed as casual) or the number of future input steps that influence the current value of the output (in that case $n_d < 0$, and thus system is non-casual) [30].

The complex nonlinear behaviour can be modelled using flexible nonlinear functions, such as wavelet and sigmoid networks with the arx model structure. The structure of a nonlinear ARX model allows the following additional flexibility:

- Instead of the weighted sum of the regressors that represents a linear mapping, the nonlinear ARX model has a more flexible nonlinear mapping function, f as shown in equation 2.19. Inputs to f are model regressors. When you specify the nonlinear ARX model structure, you can choose one of several available nonlinear functions. For example, F can represent a weighted sum of wavelets that operate on the distance of the regressors from their means.
- Nonlinear ARX regressors can be both delayed input-output variables and more complex, nonlinear expressions of delayed input and output variables.

2.4.3 Structure of NARX Models

A NARX model consists of model regressors and an output function. The output function contains one or more mapping objects, one for each model output. Each mapping object can include a linear and a nonlinear function that act on the model regressors to give

the model output and a fixed offset for that output. This block diagram represents the structure of a single-output NARX model in a simulation scenario.

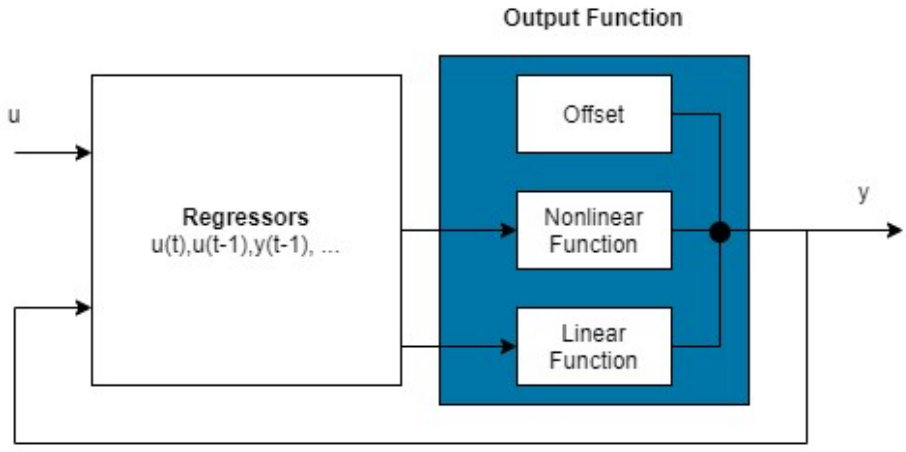


Figure 2.3: NARX Model Structure [1]

In MATLAB, the NARX model output $y(t)$, is computed in two stages:

- In the simplest case, regressors are delayed inputs and outputs, such as $u(t-1)$ and $y(t-3)$. These kind of regressors are called linear regressors. You specify linear regressors using the *linearRegressor* object. The user can also specify linear regressors by using linear ARX model orders as an input argument. However, this second approach constrains your regressor set to linear regressors with consecutive delays. To create polynomial regressors, use the *polynomialRegressor* object. To create periodic regressors that contain the sine and cosine functions of delayed input and output variables , use the *periodicRegressor* object. You can also specify custom regressors, which are nonlinear functions of delayed inputs and outputs. For example, $u(t-1)y(t-3)$ is a custom regressor that multiplies instances of input and output together. Specify custom regressors using the *customRegressor* object [1].

- It maps the regressors to the model output using an output function block. The output function block can include multiple mapping objects linear, nonlinear, and offset blocks in parallel. For example, consider the following equation:

$$F(x) = L^T(x-r) + g(Q(x-r)) + d \tag{2.20}$$

  Here, x is a vector of the regressors, and r is the mean of x, $F(x) = L^T(x-r) + y_0$ is the output of the linear function block, $g(Q(x-r)) + y_0$ represents the output of the nonlinear function block. Q is a projection matrix that makes the calculations well-conditioned. d is a scalar offset that is added to the combined outputs of the linear and nonlinear blocks. The exact form of *F(x)* depends on your choice of output function such as tree-partition networks, wavelet networks, and multilayer neural networks. When estimating a NARX model, the MATLAB computes the model parameter values, such as L, r, d, Q, and other parameters specifying g [1].

2.4.4 Different Mapping Functions for NARX Models

In MATLAB, there are several mapping objects for nonlinear ARX models. Some mapping functions represent the nonlinear function as a summed series of nonlinear units, such as wavelet networks or sigmoid functions. Others use models that draw on machine learning algorithms. One mapping object contains no nonlinear function at all, just a linear function and an offset.

- *idTreePartition* object implements a tree-partitioned nonlinear function, and is a nonlinear mapping function for estimating nonlinear ARX models. The mapping function, which is also referred to as a nonlinearity, uses a combination of linear weights, an offset and a nonlinear function to compute its output. The nonlinear function contains *idTreePartition* unit functions that operate on a radial combination of inputs. Mathematically, *idTreePartition* is a nonlinear function $y = F(x)$

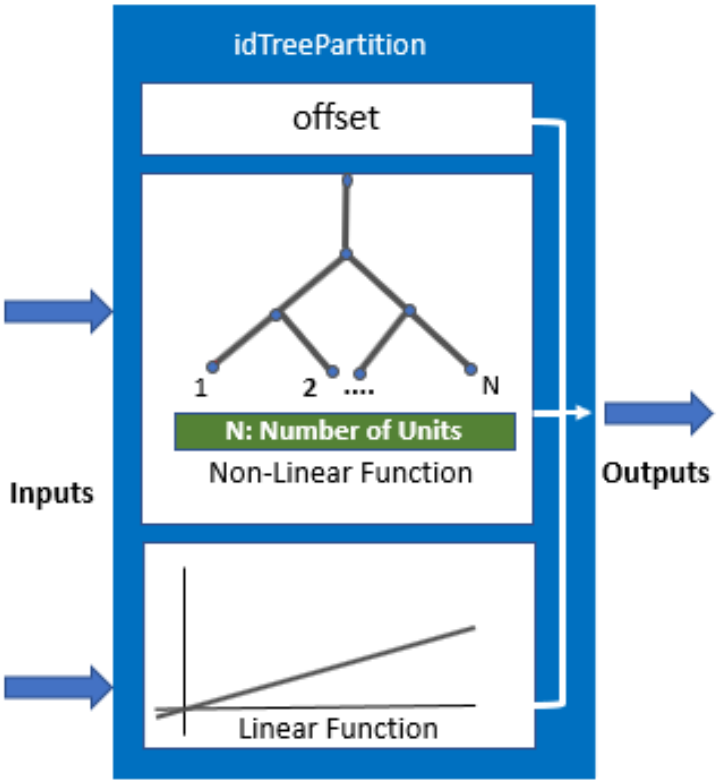


Figure 2.4: Tree Partition Function

that maps n inputs $X(t) = [x_1(t), x_2(t)...x_n(t)]^T$ to a scalar output $y(t)$. F is a piecewise-linear (affine) function of x:

$$F(x) = xL + [1, x]C_k + d \tag{2.21}$$

Here, x belongs to the partition P_k. L is a 1-by-n vector, C_k is a 1-by-n+1 vector, and P_k is a partition of the x-space.

- One layer sigmoid network: An *idSigmoidNetwork* object implements a sigmoid network function, and is a nonlinear mapping function for estimating nonlinear

ARX and Nonlinear Hammerstein-Wiener models. The mapping function, which

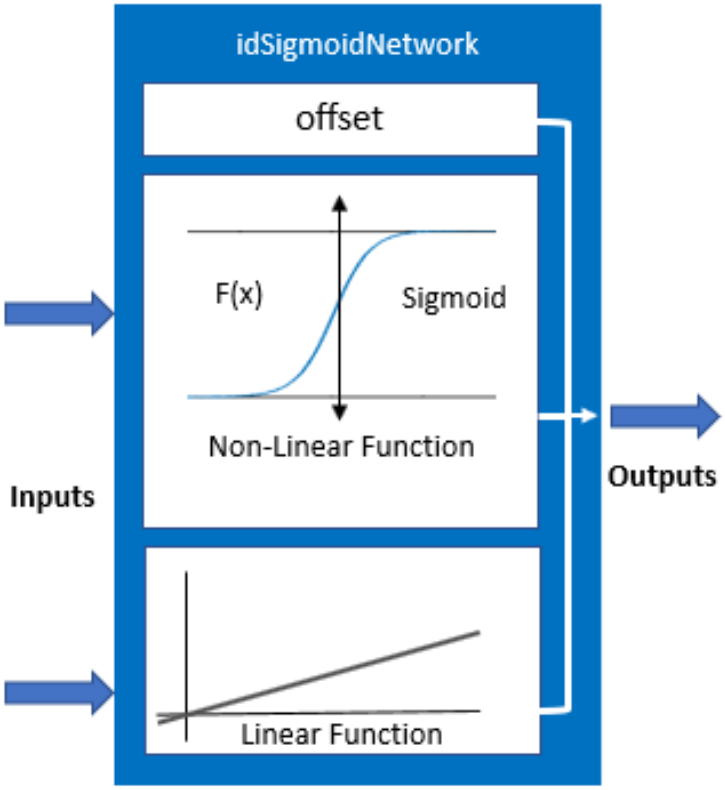


Figure 2.5: One layer Sigmoid network

is also referred to as a nonlinearity, uses a combination of linear weights, an offset and a nonlinear function to compute its output. The nonlinear function contains sigmoid unit functions that operate on a ridge combination (weighted linear sum) of inputs. Mathematically, *idSigmoidNetwork* is a function that maps n inputs $X(t) = [x_1(t), x_2(t)...x_n(t)]^T$ to a scalar output $y(t)$ using the following relationship:

$$y(t) = y_0 + X(t)^T PL + S(X(t)) \tag{2.22}$$

- *X(t)* is an n-by-1 vector of inputs, or regressors.
- y_0 is the output offset, a scalar.
- P is an n-by-p projection matrix, where n is the number of regressors and p is the number of linear weights. n must be greater than or equal to p.
- L is a p-by-1 vector of weights.
- *S(X)* is a sum of dilated and translated sigmoid functions. The total number of sigmoid functions is referred to as the number of units n of the network.

- *idLinear* object implements an affine function, and is a mapping function for estimating NARX models. The mapping function uses a combination of linear weights and an offset. Unlike the other mapping objects for the nonlinear models, *idLinear* object contains no accommodation for a nonlinear component. Mathematically, *idLinear*

is a linear function $y = F(x)$ that maps m inputs $X(t) = [x_1(t), x_2(t)...x_n(t)]^T$ to a scalar output $y(t)$.

$$y(t) = y_0 + X^T PL \tag{2.23}$$

2.4.5 Output Error (OE) Model

Output-Error (OE) models are a special configuration of polynomial models, having only two active polynomials - B and F. OE models represent conventional transfer functions that relate measured inputs to outputs while also including white noise as an additive output disturbance and is represented by:

$$y_M(t) = \frac{B(q)}{F(q)} u(t - n_d) + e(t) \tag{2.24}$$

where

$$\begin{aligned} B(q) &= 1 + a_1 q^{-1} + \ldots + a_{n_a} q^{-n_b} \\ F(q) &= b_1 q^{-1} + \ldots + b_{n_b} q^{-n_f} \end{aligned} \tag{2.25}$$

$y_M(t)$ is the output, $u(t)$ is the input, and $e(t)$ is the error, the elements of $[n_b\ n_f\ n_d]$ are as follows:

- n_b - Order of the $B(q)$ polynomial + 1, which is equivalent to the length of the $B(q)$ polynomial. n_b is an N_y-by-N_u matrix. N_y is the number of outputs and N_u is the number of inputs.
- n_f - Order of the F polynomial. n_f is an N_y-byN_u matrix.
- n_d - Input delay, expressed as the number of samples. n_d is an N_y-by-N_u matrix. The delay appears as leading zeros of the B polynomial.

The model based on the estimation of parameters at the ith iteration (e.g. using Levenberg-Marquardt algorithm) is described by:

$$y_M(t, \hat{\theta}_i) = \hat{\theta}_i^T \varphi_M(t, \hat{\theta}_i) \tag{2.26}$$

where

$$\begin{aligned} &\hat{\theta}_i = \left[\hat{a}_1, \ldots, \hat{a}_{n_a}, \hat{b}_1, \ldots, \hat{b}_{n_b}\right]^T \\ &\varphi_M\left(t, \hat{\theta}_i\right) = \left[-y_M\left(t-1, \hat{\theta}_i\right), \ldots, -y_M\left(t-n_f, \hat{\theta}_i\right), u(t-n_d), \ldots, u\left(t-n_d-n_b+1\right)\right]^T \end{aligned} \tag{2.27}$$

Equation 2.26 has a formulation similar to that of a least squares regression (e.g ARX see equation 2.15). However, there is a fundamental difference since the output values in $\varphi_M(t, \hat{\theta}_i)$ are those of the model simulated output $y_M(t, \hat{\theta}_i)$ which depends on $\hat{\theta}_i$, the parameter vector to be identified. The optimal output error model is basically a simulator type model where no need of measurable process output $y(t)$ of the system is required [7]. Note that $y_M(t, \hat{\theta}_i)$ results from the simulation of the model knowing $u(t)$ and $\hat{\theta}$. OE model can be nonlinear with respect to its parameters. And vector of parameters $\hat{\theta}$ is obtained by minimizing a quadratic criterion using a nonlinear programming algorithm.

2.4.6 Hammerstein-Wiener Model (HW)

When a system output is non linearly dependent on its inputs, the relationship between input and output can sometimes be decomposed into two or more interconnected elements. In this scenario, linear dynamics of the system can be represented by a linear transfer function [31], while the non-linearities of the system can be captured using nonlinear functions such as algebraic polynomial expressions, piecewise-linear, basis function, wavelets, neural networks, look-up tables and fuzzy models [32]. The Hammerstein-Wiener model (HW) achieves this configuration as a series connection of static nonlinear blocks with a dynamic linear block. One or two static nonlinear blocks in series with a linear block are used in Hammerstein-Wiener models to describe the dynamic systems. In this study, non-parametric models, based on HW model with the use of various combinations of non-linear static block functions such as *idTreePartition*, *idLinear* and *sigmoid network* are explored to identify models which can represent the dynamics of NWTs. Moreover, use of *sigmoid network* with HW model can be classified in neural network model category. Figure 2.6 illustrates the block diagram of a Hammerstein-Wiener model structure.

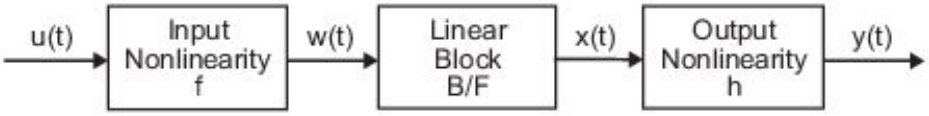


Figure 2.6: Hammerstein-Wiener Model

We could study this model as a combination of three series blocks. To formulate the problem, we have equation 2.28 which is a nonlinear function transforming input data $u(t)$ into an internal variable $w(t)$.

$$w(t) = f(u(t)) \tag{2.28}$$

$w(t)$ has the same dimension as $u(t)$.

For the second block:

$$x(t) = \frac{B}{F} w(t) \tag{2.29}$$

where B/F is a linear transfer function transforming w(t) into $x(t)$. B and F are the polynomials similar to the polynomials (2.27), used in Output-Error model. For n_y and n_u number of outputs and inputs respectively, the linear block is a transfer function matrix containing entries:

$$\frac{B_{ji}(q)}{F_{ji}(q)} \tag{2.30}$$

where:

$$\begin{aligned} j &= 1, 2, \ldots, n_y \\ i &= 1, 2, \ldots, n_u \end{aligned} \tag{2.31}$$

Finally for the third block:

$$y(t) = h(x(t)) \tag{2.32}$$

where h is a nonlinear function that transforms the output of the linear block $x(t)$ to the system output $y(t)$. Because f acts on the input part of the linear block, this function is called the input nonlinearity. Similarly, because h acts on the output part of the linear block, this function is called the output nonlinearity. The nonlinearities f and h are scalar functions, one nonlinear function for each input and output channel. It is not necessary to include both the input and the output nonlinearity in the model structure. When a model contains only the input nonlinearity f, it is called a Hammerstein model. Similarly, when the model contains only the output nonlinearity h, it is called a Wiener model.

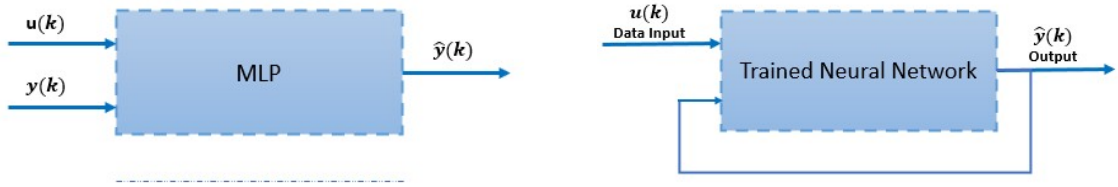
Applications of Hammerstein-Wiener model are in wide areas, for example we can mention modelling electro-mechanical system and radio frequency components, audio and speech processing and predictive control of chemical processes. These models have a useful block representation, transparent relationship to linear systems, and are easier to implement than heavy-duty nonlinear. Therefore, they are very useful. The Hammerstein-Wiener model can be used as a black-box model structure since it prepares a flexible parameterization for nonlinear models. It is possible to estimate a linear model and try to improve its quality by adding an input or output or both nonlinearities to this model [33].

2.5 Deep Learning Model Structures

We can think of the general function approximators for f in equation 2.19 as a sequential construction of several generalized linear regressions, i.e. repetitive use of linear regression and static nonlinearities [34]. This is, loosely, the idea behind neural networks.

2.5.1 NARX Neural Network

The NARX neural network model can realize an overall input/output black-box mapping by the multilayer perceptron (MLP) incorporating time delay unit and output feedback in the input layer [35] [36]. The discrete-time input–output form is expressed in equation 2.19. In the process of identifying with NARX model, serial-parallel structure is usually adopted that has two advantages as below. First, it can make feedback signal more accurate and improve the success rate of identification. Second, serial-parallel structure can easily be converted to parallel structure to proceed system simulation [2]. In order to show NARX structural model visually, the general architecture of the NARX network with three input's tapped delays and four output's tapped delays is shown in Fig. 2.8. Here z^{-1} represents delay operator. Multilayer perceptron network acts as the nonlinear system, and for example, in this figure, *tansig* (Hypertangent) transfer function is used in the hidden layer while *purelin* function (linear) is used in output layer. This structure



(a) NARX NN Series-Parallel model used for training phase

(b) NARX NN Parallel model used for Prediction phase

Figure 2.7: NARX Neural Network Model

can be extended to other network dimensions according to the needs of the user. Usually, the total number of parameters of the NARX Neural network includes tapped delays of input and output, the number of hidden layers and neurons in each of the layers. The parameters are usually obtained through heuristic process or optimization algorithm for example, Levenberg-Marquardt algorithm.

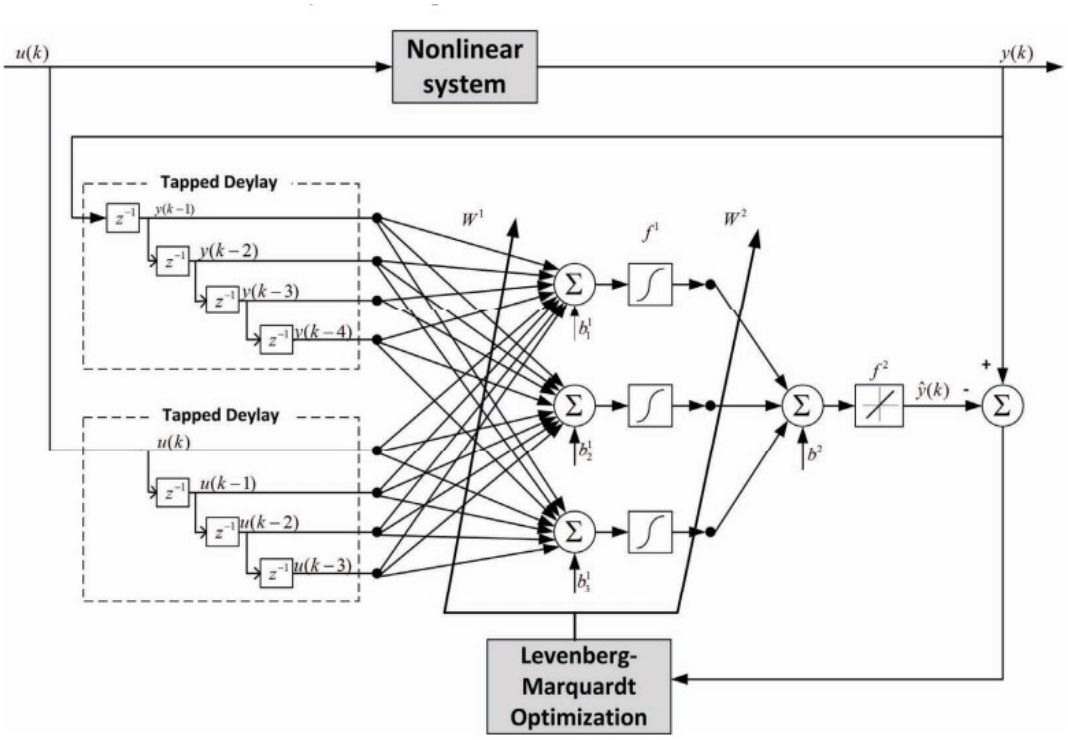


Figure 2.8: NARX Neural Network Model Structure [2]

According to the input variable u(k), the hidden layer output at time k is obtained as:

$$H_i(k) = f_1\left[\sum_{r=0}^{n_u} w_{ir}u(k-r) + \sum_{l=1}^{n_y} w_{il}y(k-l) + a_i\right] \tag{2.33}$$

where w_{ir} is the connection weight between the input neuron $u(k-r)$ and ith hidden neuron. w_{il} is the connection weight between the ith hidden neuron and output feedback

neuron $y(k-l)$; a_i is the bias of the ith hidden neuron; and $f_1(\cdot)$ is the hidden layer activation function.

Combining the hidden layer output, the final prediction can be given by

$$\hat{y}_j(k) = f_2\left[\sum_{i=1}^{n_h} w_{ji} H_i(k) + b_j\right] \tag{2.34}$$

where w_{ji} is the connection weight between the ith hidden neuron and jth predicted output n_h; bj is the bias of the jth predicted output; n_h is the number of hidden neurons; and $f_2(\cdot)$ is the output layer activation function.

The Fig. 2.8 is the classical one-hidden layer feed forward neural network. This is what is called feedforwardnet in the MATLAB Deep Learning Toolbox. This network corresponds to the so called TCN (Temporal Convolutional Network) [37]. The TCN however uses a convolutional neural network instead of a feedforward neural network.

A variant is called cascadeforwardnet in the Deep Learning Toolbox as shown in Fig. 2.9. It corresponds to the case where all previous hidden units are used as regressors in the next layer, not only the ones from the previous layer. That is, f_2 will be a function of u, $f_1(u)$, etc.

2.6 Model performance evaluation metric

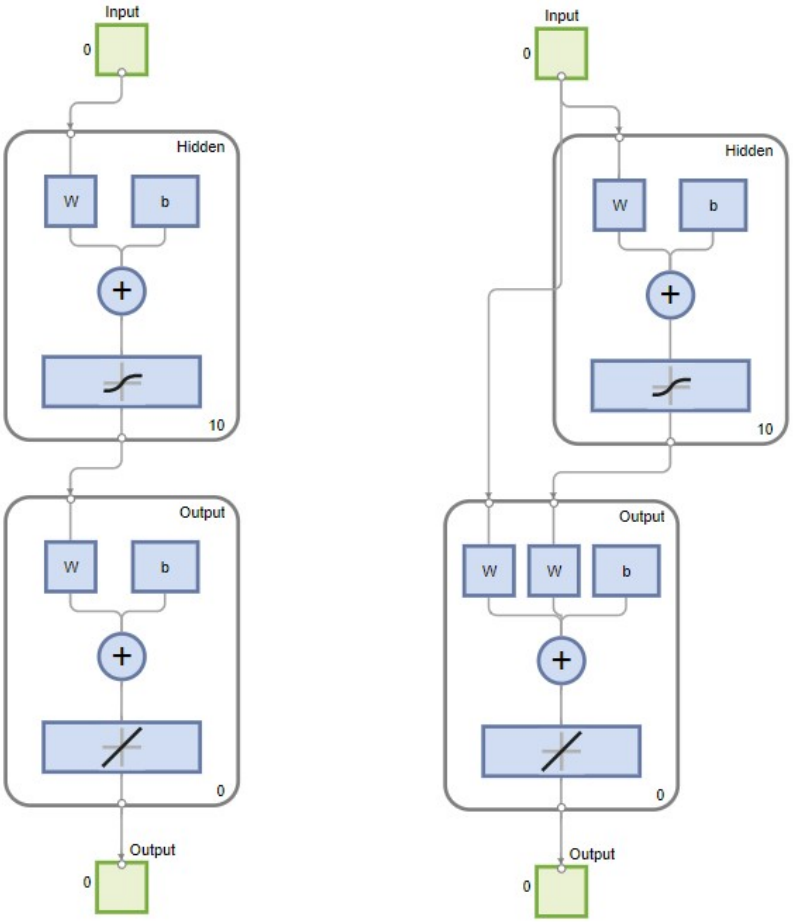
In Matlab, *Goodness of fit* or simply fit is used to evaluate the performance of the model over the validation data which is a form of normalised root mean-squared error (NRMSE) and is expressed as

$$\text{fit} = 100\left(1 - \frac{\sqrt{\sum_k (y(k) - \hat{y}(k))^2}}{\sqrt{(y(k) - \text{mean}(y(k)))^2}}\right) \tag{2.35}$$

where $\hat{y}(k)$ is the model output and $y(k)$ is the actual signal. Fit values may vary between -Inf and 100. A fit value close to 100 means a perfect fit to reference data (zero error) whereas -Inf refers to the worst fit.

2.7 Model Validation

Model validation means that we should gain confidence that the estimated model is capable of covering essential parts of the system behavior - not only for the estimation data used for its estimation. A simple and common way to do this is Cross Validation: Collect a validation data set, that is different from the estimation data set. Simulate the model for the validation input and compare that model output with the measured validation output. Make the comparison by eye inspection or compute a numerical measure of the fit.



(a) 1-hidden layered FeedForwardnet

(b) 1-hidden layered Cascadeforwardnet

Figure 2.9: Variants of MultiLayer Perceptron

2.8 Akaike information criterion (AIC)

AIC estimates the relative amount of information lost by a given model: the less information a model loses, the higher the quality of that model. AIC deals with the trade-off between the goodness of fit of the model and the simplicity of the model. In other words, AIC deals with both the risk of overfitting and the risk of underfitting. According to Akaike's theory, the most accurate model has the smallest AIC. Normalized AIC (nAIC) is expressed as:

$$\text{nAIC} = \log(V) + \frac{2 * n_p}{N} \tag{2.36}$$

where V is the loss function (see 2.1), n_p is the number of estimated parameters, and N is the number of data points used for the estimation. In MATLAB, function Report.Fit is used to access the value of AIC.

2.9 Introduction to OpenFOAM

The OpenFOAM (Open Source Field Operation and Manipulation) is an open-source CFD software package. It is essentially a collection of text files written in C++ language which is being operated entirely by text based commands to create applications. Due to its free licensing, it allows the user to execute as many jobs/tasks as they need on an unlimited number of processors for free. It gives the user complete control over the software, allowing them to customize it to their specific needs. Users are free to modify or contribute towards the development of OpenFOAM by improving the code, creating useful libraries and toolboxes being shared freely in public domain. Solvers and utilities are the intrinsic part of the software. Solvers are employed to solve a particular physical problem whereas utilities are involved for data processing and handling.

OpenFOAM comes with large number of pre-installed solvers and finding its applications over large domains of science and engineering fields like Complex fluid flows, heat transport, chemical processes, and electromagnetics. While OpenFOAM does not have a graphical user interface, its customizability has made it a popular choice for users who want to have some control over the physics and calculation of a problem solution. Many commercial and academic organizations use OpenFOAM, and it has been used in numerous peer-reviewed articles [38].

2.9.1 Governing Equations

In OpenFOAM, the Navier-Stokes equations for an incompressible, constant fluid viscosity which governs the dynamics of fluid in the ocean and describing the conservation of momentum and mass is employed. In cartesian coordinate system, these equations can be expressed as:

$$\begin{aligned}
\rho\left(\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+w\frac{\partial u}{\partial z}\right) &= -\frac{\partial p}{\partial x}+\mu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}\right)+\rho g_x \\
\rho\left(\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+w\frac{\partial v}{\partial z}\right) &= -\frac{\partial p}{\partial y}+\mu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}+\frac{\partial^2 v}{\partial z^2}\right)+\rho g_y \\
\rho\left(\frac{\partial w}{\partial t}+u\frac{\partial w}{\partial x}+v\frac{\partial w}{\partial y}+w\frac{\partial w}{\partial z}\right) &= -\frac{\partial p}{\partial z}+\mu\left(\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}+\frac{\partial^2 w}{\partial z^2}\right)+\rho g_z
\end{aligned} \tag{2.37}$$

where p is the pressure [Pa], $g_x,\ g_y,\ g_z$ are the components of acceleration due to gravity [ms^{-2}], μ is the dynamic viscosity of fluid [Pa.s], ρ is the density of fluid [kgm^{-3}], t represents time [s] and $u,\ v,\ w$ are the components of velocity [ms^{-1}] in x, y and z directions respectively.

And the continuity equation for incompressible fluid, $\rho =$ constant can be expressed as:

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z} \ = 0 \tag{2.38}$$

The above equations are solved by OpenFOAM using an Eulerian finite volume model [39].

2.9.2 Volume of Fluid (VOF)

The most common method to capture the free surface and only one to simulate breaking or overtopping waves is the VOF approach [40]. In OpenFOAM, the proportion of each fluid present in each cell of the computation mesh (also known as volume fraction) is determined using this method. The volume fraction of water, α, is captured by solving the following additional equations:

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\mathbf{U}\alpha) = 0 \tag{2.39}$$

where $\mathbf{U}$ is the velocity, and α is the volume fraction of water. Value of α varies between 0 to 1. For a mesh cell completely full of water, α =1, if it is full of air then α=0.
The density of the water-air mixture inside each cell of the mesh is determined by volume fraction,α. The density of the mixture is expressed as:

$$\rho = \alpha\rho_{\text{water}} + (1-\alpha)\rho_{\text{air}} \tag{2.40}$$

where ρ_{water} is the density of water and ρ_{air} is the density of air.

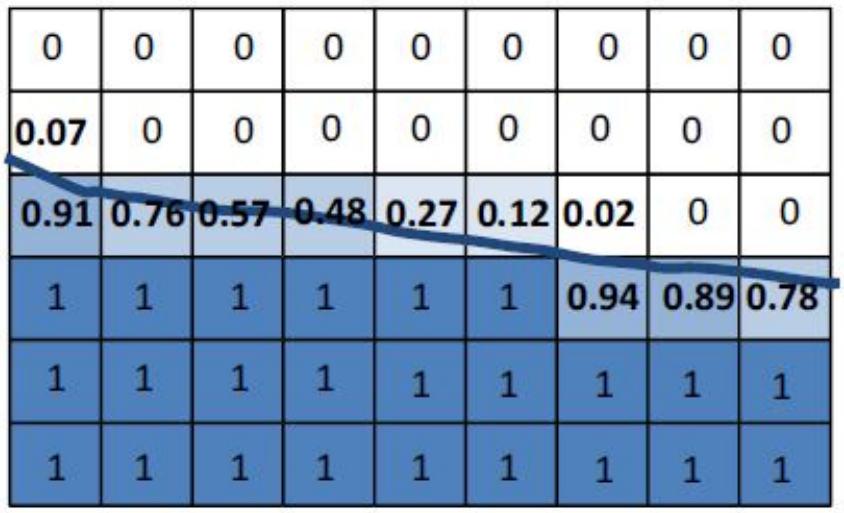


Figure 2.10: Volume of fluid technique for capturing the free surface interface [3]

2.10 Introduction to Numerical Wavetanks

The accurate modeling of water wave behavior is a crucial topic in the field of maritime engineering engineering. Physical wave tanks and flumes are commonly used to study the effects of water waves on coastal structures and other associated coastal phenomena [41] [42]. A paddle with a predefined motion produces the appropriate waves. In recent years, the availability and capabilities of numerical wave tanks has increased dramatically owing to increase in computational power. Numerical models applied to fluid flows are becoming an increasingly significant tool for maritime engineering and can be utilized in studies such as energy converters.

NWT can be implemented in both commercial and opensource software packages where the users are provided with manuals, video tutorials, documentations and graphical user interfaces (GUIs) [3]. In our case CFD solver package OpenFOAM was utilized. The ability to predict wave energy converters efficiently and their response to wave loads is reliant on accurate modeling of wave behavior and its hydrodynamic features. As a result, during the previous decade, researchers worked towards development of numerical wave tanks [43].

To simulate the ocean waves, researchers has used variety of numerical approaches. Based on Boussinesq model, Wei [44] and Chawla [45] implemented a source function approach for generating ocean waves. Numerical models based on Boussinesq-type equations have some drawbacks, such as the inability to model wave breaking and needs additional modifications to account for energy dissipation [46]. To simulate small-amplitude waves and solitary waves, Dong and Huang [47] developed 2D numerical wavetank based on Navier-Stokes equations. In our case NWT uses InterFoamsrc solver, a modified version of InterFoam which is designed to sovle Navier-Stokes and VOF equations for two incompressible fluids.

Impulse source wavemaker is implemented to the Volume of Fluid RANS model for incompressbile flow by introducing two terms to the impulse equation; a source term $r_w \rho \mathbf{a}_{wm}$ and dissipation term sand $\rho \mathbf{U}$ used to implement a numerical beach [5] and is expressed as:

$$\frac{\partial(\rho \mathbf{U})}{\partial t} + \nabla \cdot (\rho \mathbf{U}\mathbf{U}) = -\nabla p + \nabla \cdot \mathbf{T} + \rho \mathbf{F}_b + r_w \rho \mathbf{a}_{wm} + \text{sand}\, \rho \mathbf{U} \tag{2.41}$$

where t denotes time, $\mathbf{U}$ denotes velocity, p is pressure, ρ denotes fluid density, $\mathbf{T} = \mu \nabla^2 \mathbf{U} + \frac{1}{3} \nabla (\nabla \cdot \mathbf{U})$ denotes viscous stress tensor, $\mathbf{F}_b$ is external body forces, $\mathbf{r}_w$ is a scalar, dimensionless variable that defines wavemaker region. Its value is 1 in the wavemaker's operating zone and zero everywhere else in the NWT domain. $\mathbf{a}_{wm}$ is the acceleration input to the wavemaker at each cell centre within $\mathbf{r}_w$ [48].

The strength of the dissipation is controlled by variable field sand, which starts at zero in the central areas of the domain where the working wavefield is needed and gradually

increases towards the boundary over the length of the numerical beach. Its unit is s^{-1}. In the current implementation, the following equation 2.42 is utilized, which has been proved to result in enough wave absorption by coupling with a numerical beach[49]. In OpenFOAM *setExprFieldsDict* toolbox was used to implement such a function, representing numerical beaches in the CFD solver:

$$\text{sand}(x) = -2 \cdot \text{sand}_{\text{Max}} \left(\frac{(l_{\text{beach}} - x)}{l_{\text{beach}}} \right)^3 + 3 \cdot \text{sand}_{\text{Max}} \left(\frac{(l_{\text{beach}} - x)}{l_{\text{beach}}} \right)^2 \tag{2.42}$$

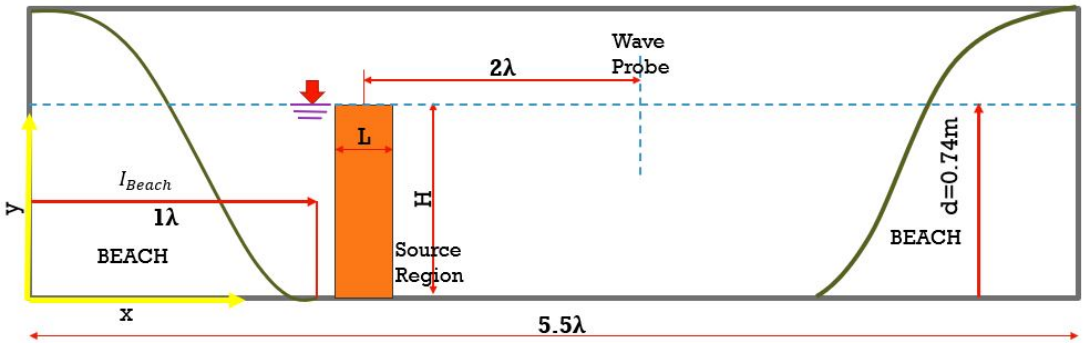


Figure 2.11: Schematic of the numerical wave tank (NWT).

2.10.1 Geometry of Wave Tank

As the input wave is unidirectional, a 2D NWT was implemented to simplify the setup and reduce computational costs. The NWT geometry is shown in Figure 2.11, The NWT depth was set at 0.74 meters for shallow water. The wave probe location was 2λ from the center of the wavemaker source region. The source centre was then positioned at 1λ upwave and 4.25λ downwave from two absorption beaches. The relax zone or source region is rectangular in shape with dimension 4m in length and 0.555m in height.

2.10.2 Background Mesh

Mesh plays an important role for calculation. A trade-off between number of cells and computation costs has to be made. Vertically the mesh is splitted into three regions: The interface region spans the area where the free surface may appear during simulation and is set with a high density of uniform vertical cell length. The air region has low cell density and the mesh stretches with distance from the interface upwards to the atmosphere boundary. The water region has moderate mesh density and the mesh stretches gradually from interface down to the tank floor. The total cell count in this simulation was found to be 385140 using checkMesh utility in OpenFOAM.

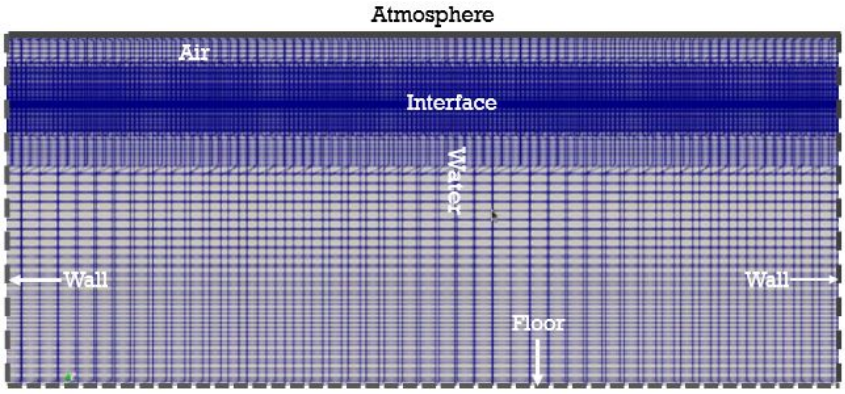


Figure 2.12: Background Mesh

2.10.3 Simulation

In this study NWT is based on InterFoamsrc solver, a modified version of InterFoam, which uses impulse source wavermaker to produce waves. It solves for the fluid pressure, velocity and volume of fraction. The simulation type was set to laminar flow. The solver for the simulation was set in controlDict file of the system directory. Other important simulation parameters, such as the simulation length and libraries to be linked, were also set in the controlDict file. Selection of the finite-volume techniques for the simulation were set in the fvSchemes file. Some of the results obtained are shown as:

Figure 2.13: Surface Wave Variation in NWT

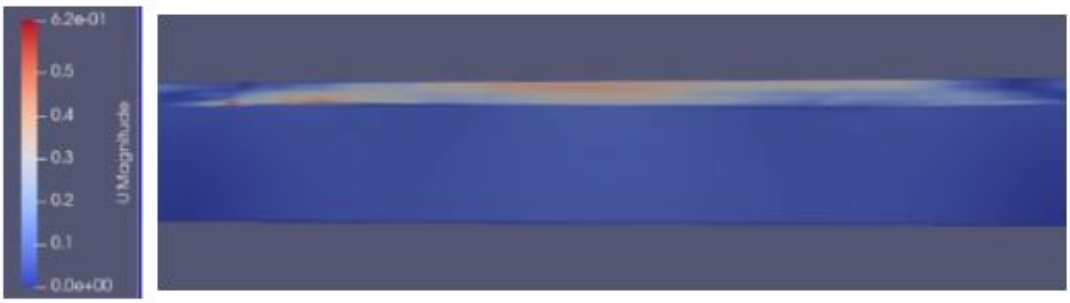


Figure 2.14: Velocity variation in horizontal direction

Chapter 3

3 Numerical Study of Mass Spring Damper System

In this section, capabilities of system identification techniques manily ARX and NARX Nerual Network applied to a linear damping oscillator is demonstrated. Let's consider a mass spring damper system as shown in figure 3.1 subjected to a multi-harmonic ex-

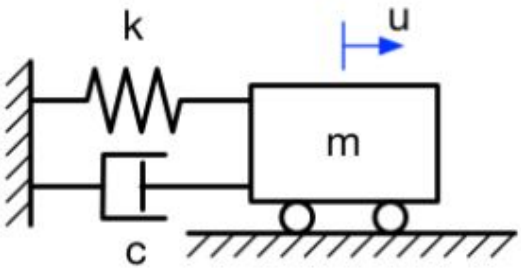


Figure 3.1: Mass Spring Damper System

citation. The mathematical model of such a system is given by the following ordinary differential equation:

$$m\ddot{u}(t) + c\dot{u}(t) + ku(t) = F(t) \tag{3.1}$$

Although equation 3.1 can be solved both analytically and numerically but it is interesting how black box modelling approach is effective in establishing a cause and effect relationship in this case. First the direct problem (in which Force is given and model needs to predict the response of the system, displacement) and followed by inverse problem (where displacement is given and force is predicted) are discussed. Four datasets are taken into consideration, first one is a multi-harmonic excitation input equally spaced over an interval of 0 to 100s with a random phase. All the data has been obtained by solving equation 3.1 in MATLAB using ODE45. The mass, spring and damping constant are taken as unity. Testing sets consists of both single frequency and multi-frequency excitation spread over the same time interval. Only the first 500 time steps are shown in the following figures to get a more closed view of the generated data:

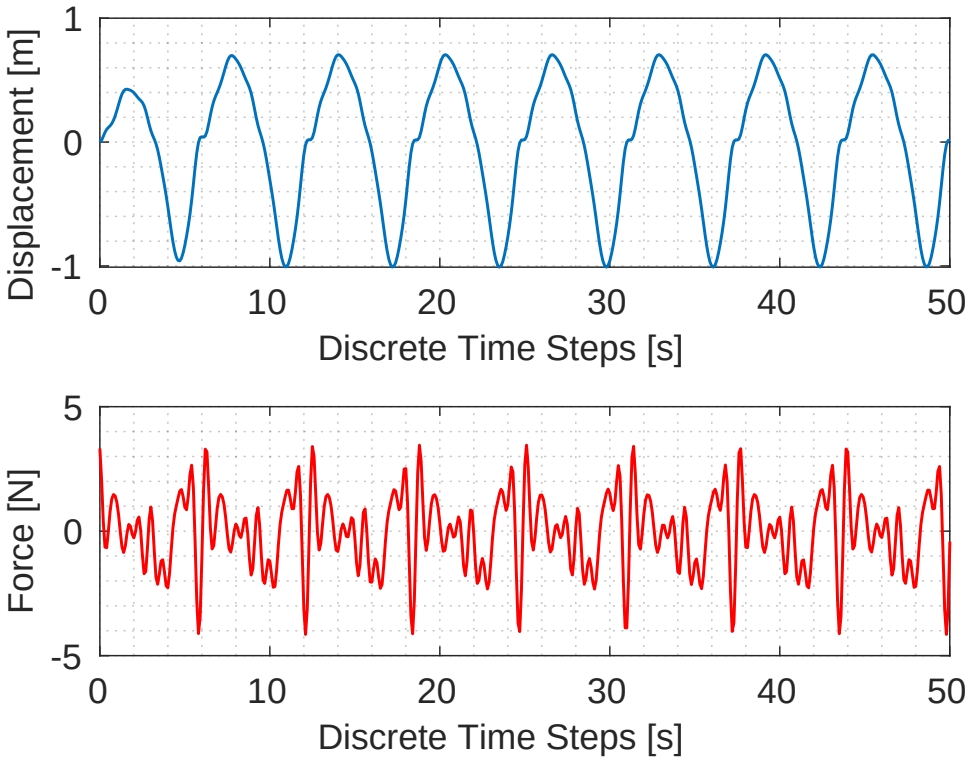


Figure 3.2: Training Dataset (Multi-frequency harmonic signal)

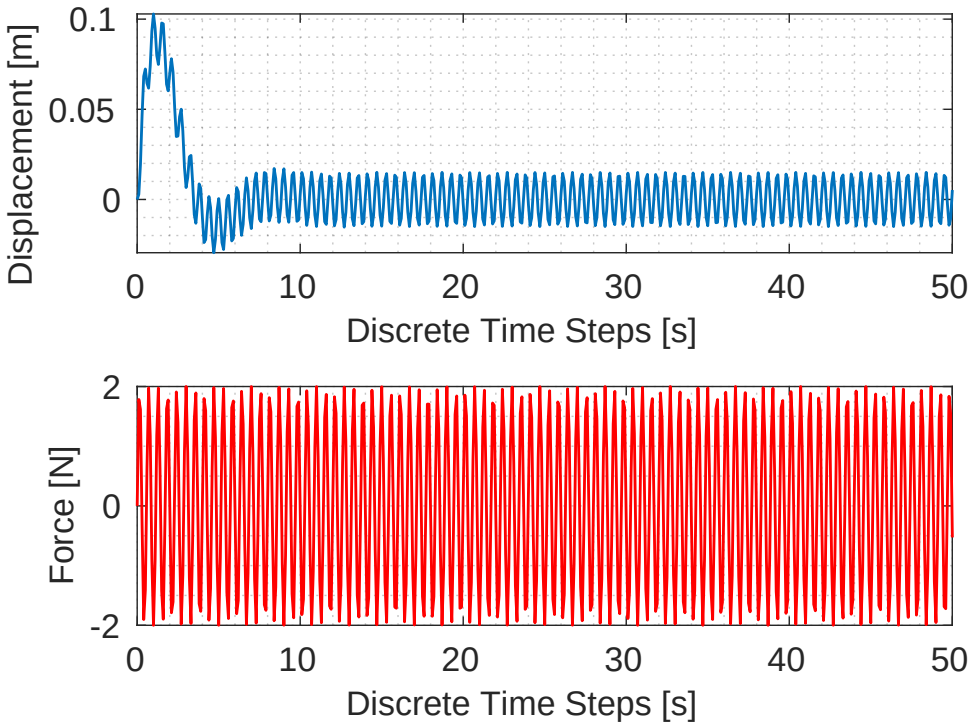


Figure 3.3: Testing Dataset I (Single frequency sinusoidal signal)

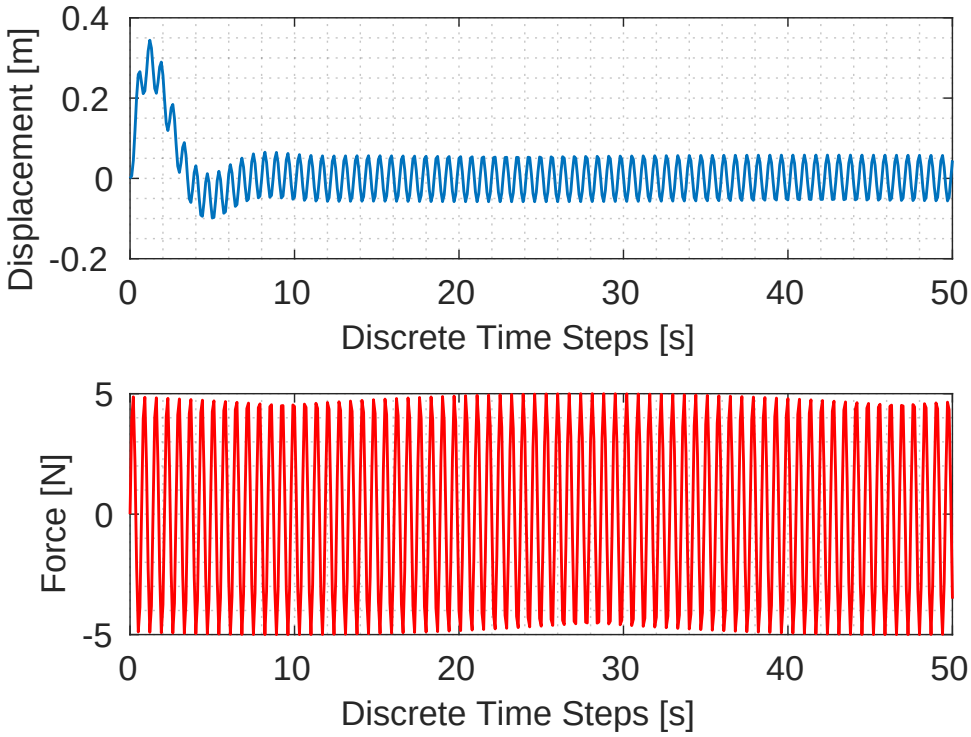


Figure 3.4: Testing Dataset II (Single frequency sinusoidal signal)

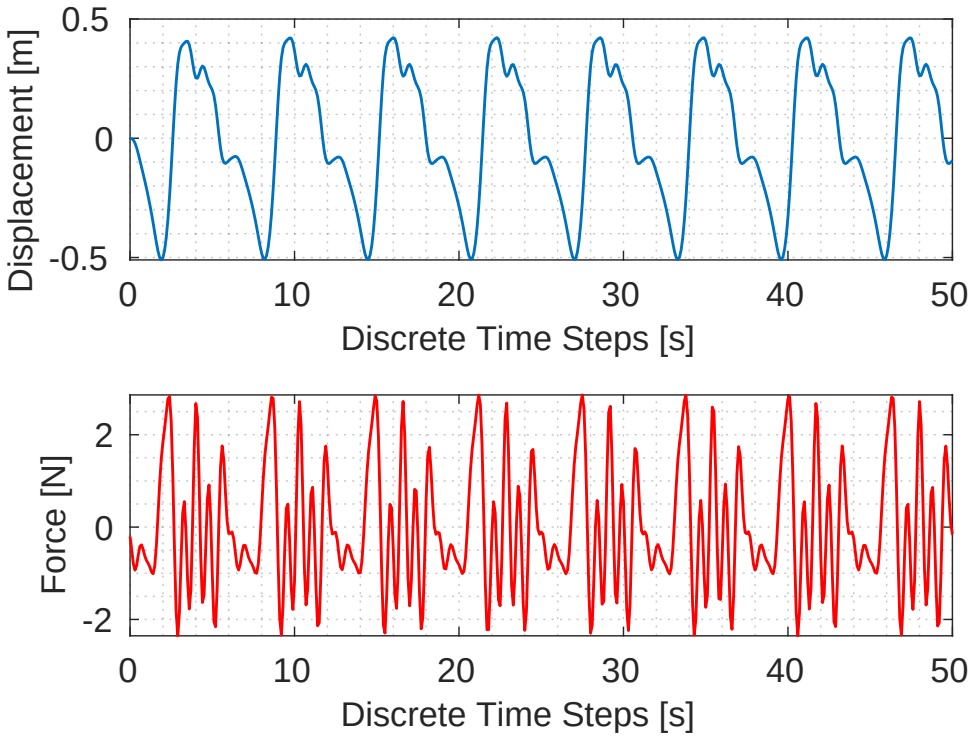


Figure 3.5: Testing Dataset III (Multi-frequency harmonic signal)

3.1 Direct MSD Problem

3.1.1 Autoregressive with Exogenous Input (ARX)

ARX time series model is a linear representation of a dynamic system in discrete time. See Section 2.4.1 for further details. Matlab SYSID toolbox is used to find model order with maximum goodness of fit. Since there is no delay in the system so $n_d = 0$, With the increase in $n_y > 3$ and $n_u > 4$ does not affect much to the loss function 2.1. For model parameter [3 4 0], the training and testing achieved the best fit as is shown in the following figures. The best ARX model obtained can be expressed in discrete form as:

$$\begin{aligned} &A(z)y(t) = B(z)u(t) + e(t) \\ &A(z) = 1 - 2.891z^{-1} + 2.791z^{-2} - 0.9007z^{-3} \\ &B(z) = 0.001591 + 0.004765z^{-1} - 0.004788z^{-2} - 0.00152z^{-3} \end{aligned} \tag{3.2}$$

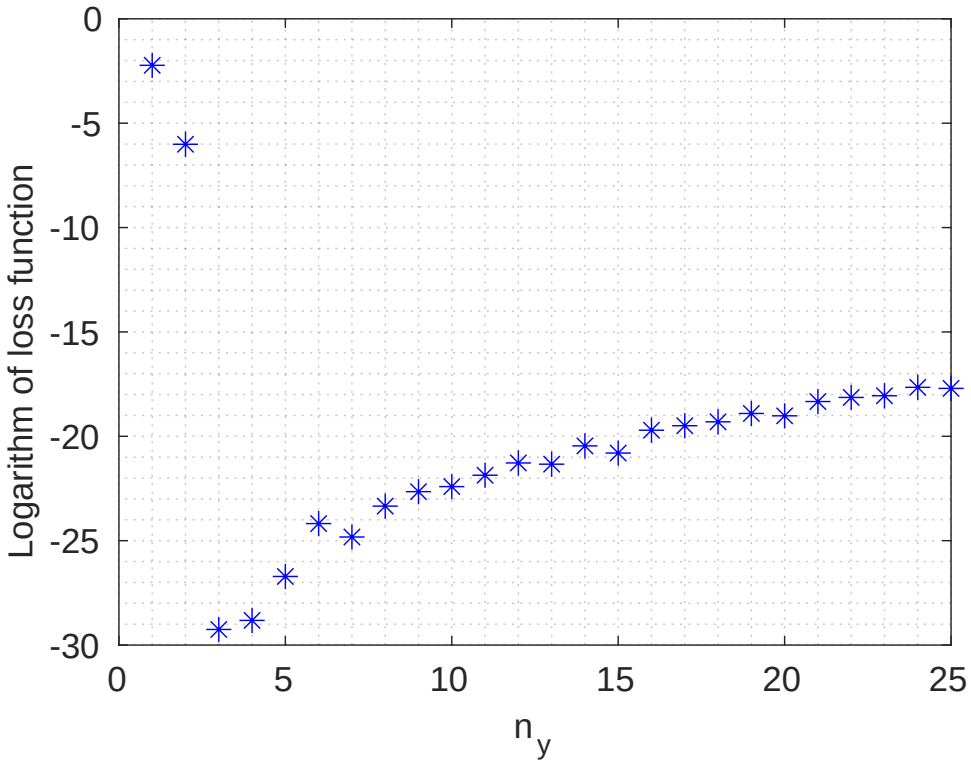


Figure 3.6: Variation of logarithm of loss function with $n_y, for\ n_u = 4,\ n_d = 0$

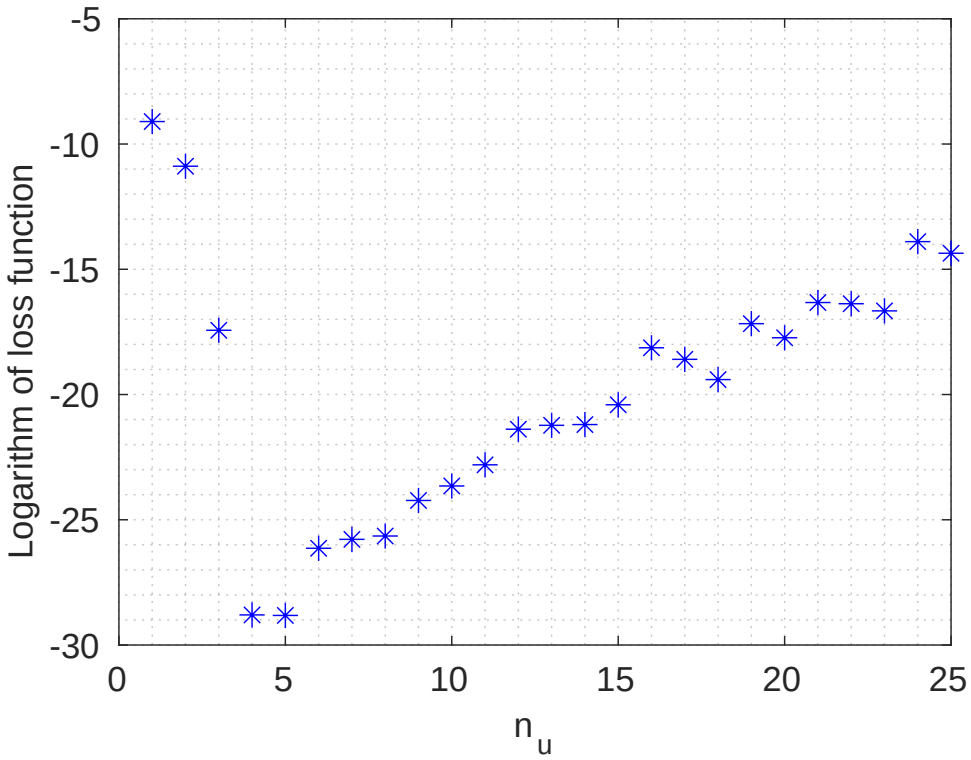


Figure 3.7: Variation of logarithm of loss function with $n_u, for\ n_y = 3,\ n_d = 0$

ARX	Performance				
	Training Dataset		Testing Datasets		
$[n_y\ n_u\ n_d]$	Loss Function	Fit (%)	Testing I Fit(%)	Testing II Fit (%)	Testing III Fit(%)
[1 3 2]	3.083×10^{-2}	70.21	−791.09	−286.6	45.38
[2 1 0]	1.116×10^{-3}	94.43	29.58	38.06	88.85
[2 1 1]	1.174×10^{-4}	99.20	88.41	92.78	98.41
[2 2 3]	9.767×10^{-4}	94.28	−105.64	−52.78	90.39
[3 4 0]	$\mathbf{4.946} \times 10^{-7}$	**99.99**	**99.75**	**99.99**	**99.77**
[3 3 0]	2.084×10^{-6}	99.47	89.39	93.06	99.01
[3 4 1]	1.631×10^{-6}	99.68	91.93	96.67	99.48

Table 3.1: ARX Model Orders Vs Performance for Direct Problem

3.1.2 NARX Neural Network

The performance of a Non-linear Autoregressive with exogeneous variables using Neural Network implemented in MATLAB SYSID is also investigated. In this case, after performing optimization of hyper-parameters such as number of layers, neurons per layer, type of activation function etc., a two layered Feedforward Neural Network with 8 nodes per layer and linear activation function with $n_u = 0$ and $n_y = 8$ as Input and output delays respectively produced desirable results which is shown in the following table:

Nodes per Layers	$[n_u\ n_y]$	Activation Func	Loss Function (refer 2.1)
[8 8]	[2 4]	[Purelin Purelin]	MSE

Table 3.2: NARX Neural Network hyper-parameter selection for Direct MSD problem

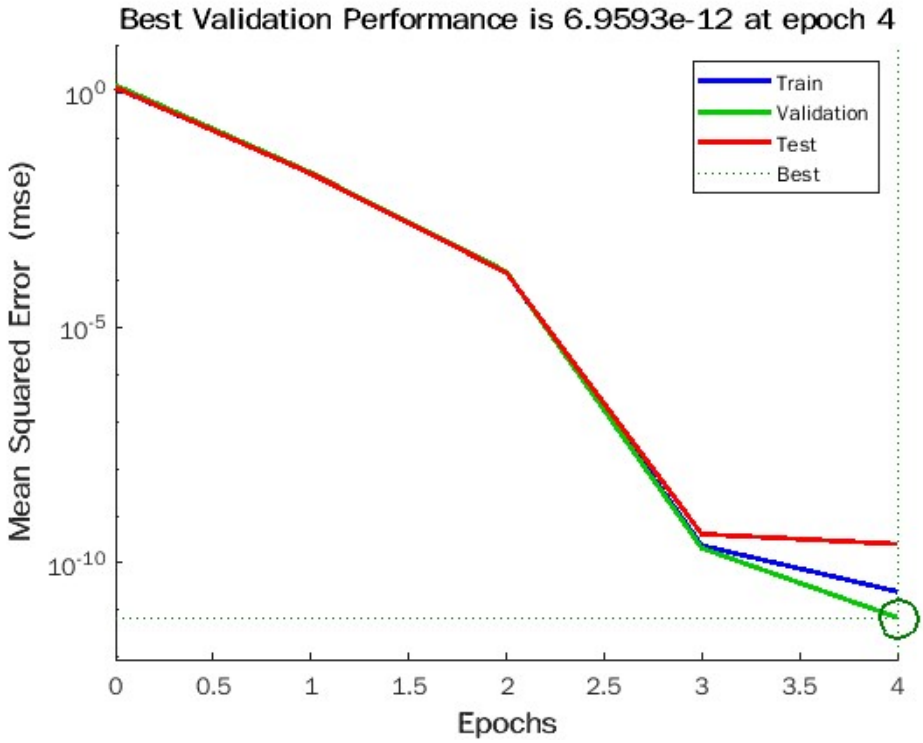


Figure 3.8: Loss function vs Epocs

The training took almost 4 Epocs (number of passes of the entire training dataset the machine learning algorithm has completed) to reach to convergence.

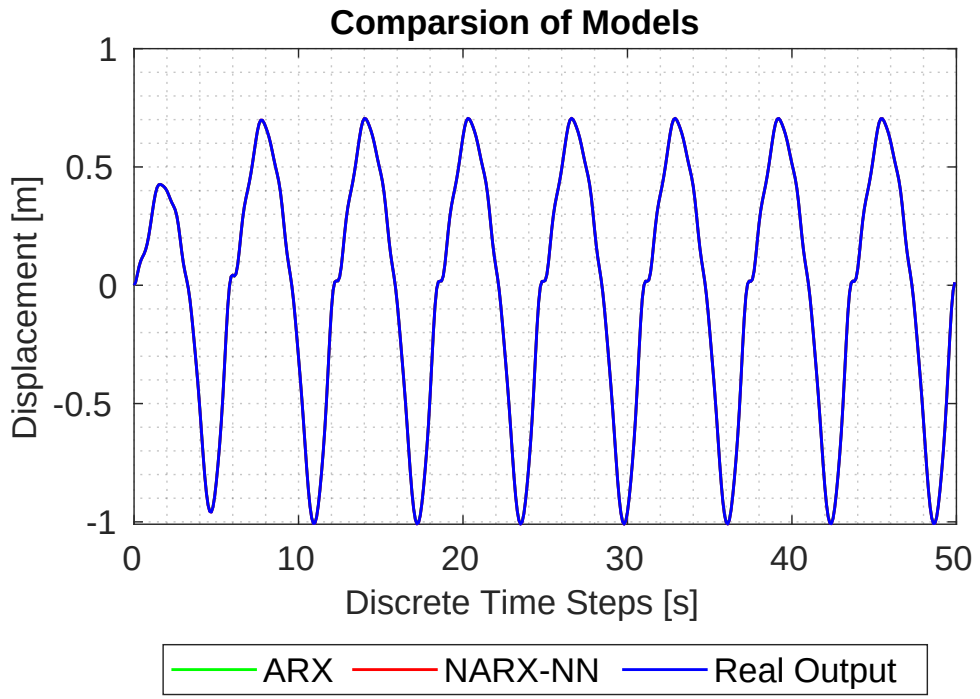


Figure 3.9: Identified model performance comparison on Training dataset

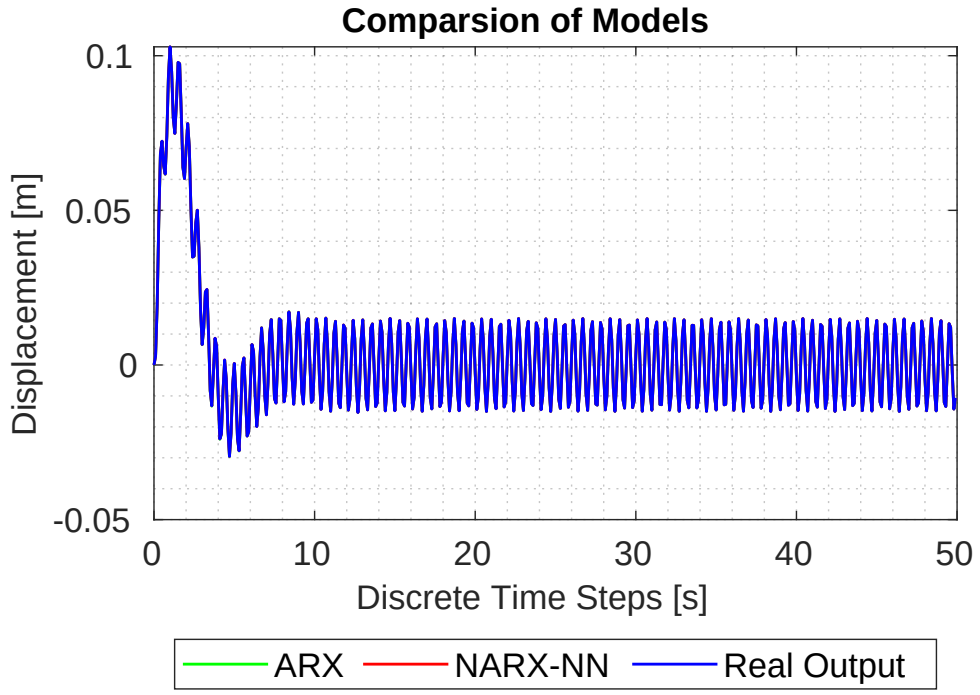


Figure 3.10: Identified model performance comparison on Testing dataset I

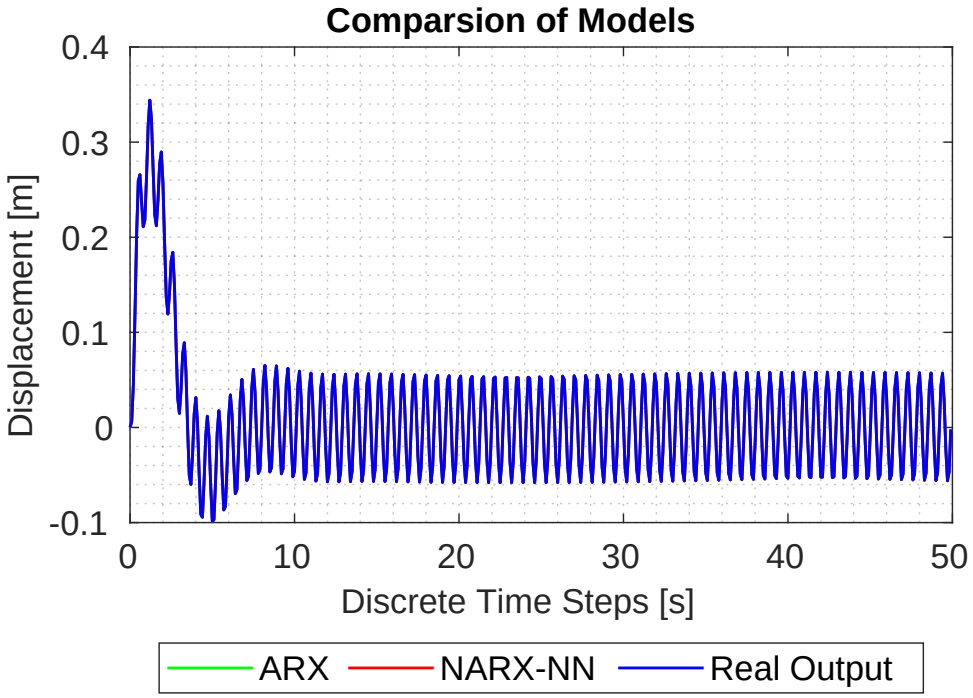


Figure 3.11: Identified model performance comparison on Testing dataset II

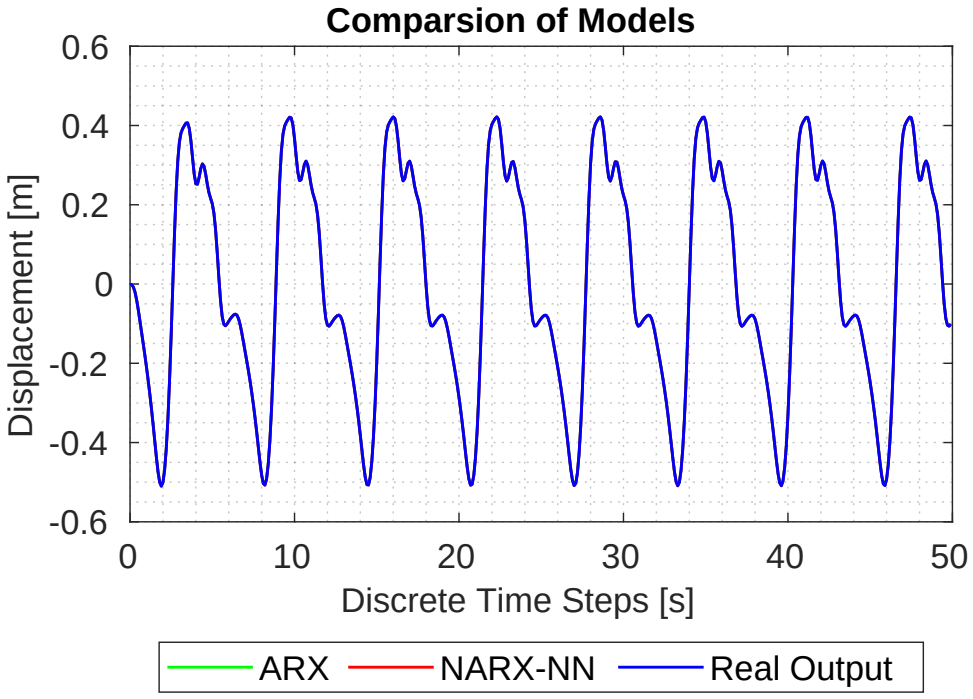


Figure 3.12: Identified model performance comparison on Testing dataset III

Table 3.3 shows the comparison of performances of the models during the training and testing. It can be stated that both the models were capable of simulating the response of MSD system with a very high accuracy. But the training duration for ARX was almost 25 times shorter than NARX Neural Network as shown in Fig 3.19. Since the problem is linear, therefore, It is not surprising to see the performance of ARX is similar to NARX.

Models	Training perform. (%age)	Testing I Perform. (%age)	Testing II Perform. (%age)	Testing III Perform. (%age)
NARX-NN	99.99	99.86	99.96	99.96
ARX	99.84	99.75	99.70	99.99

Table 3.3: Model performance comparison on available data in case of Direct problem

3.2 Inverse MSD Problem

One of the advantages of the black box models is their success in solving inverse problems, which can otherwise be difficult to solve by other means. One of the interesting inverse problem of its kind can be simulation of required excitation subjected to the Mass-Spring-damper system in order to produce a desired displacement. Although this problem can be solved analytically, but it may be a good example to check the abilities of these models. Furthermore, similar set of strategies and methods will be applied in more challenging problems of NWTs. In this particular problem, model needs to map a relationship be-

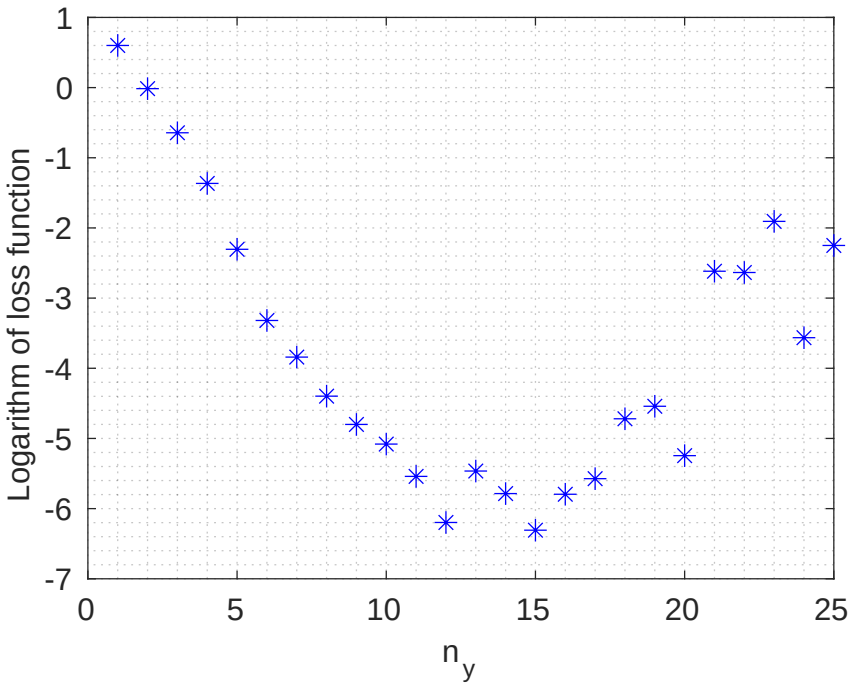


Figure 3.13: Variation of logarithm of loss function with n_y, for $n_u = 8,\ n_d = 0$

tween given displacement and the required force. Application of ARX followed by NARX Neural Network for this inverse problem will be described. Fig 3.13 and Fig 3.14 shows the variation of loss functions w.r.t. model orders n_y and n_u respectively. MATLAB's inbuilt functions such as *arxstruc* and *selstruc* were useful in finding the best ARX model parameters. From the Table 3.4, model with orders such as [3 11 0] and [0 8 0] per-

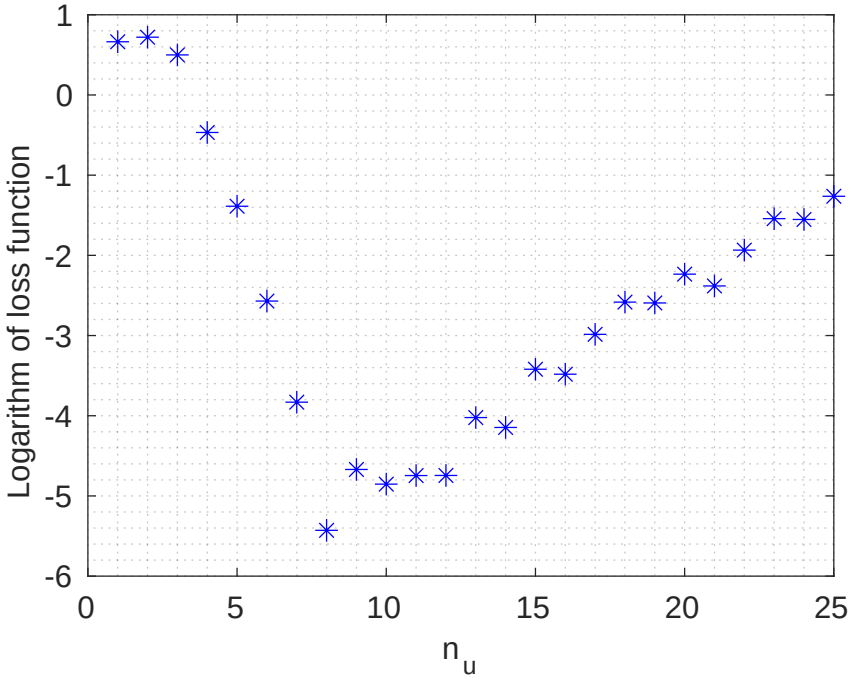


Figure 3.14: Variation of logarithm of loss function with n_u, for $n_y = 0,\ n_d = 0$

formed better during training as well as testing than others with minimum number of parameters required. But due to the lesser model complexity of later one, it was chosen as the best model order. The performance of other model orders can be checked in Table A.1.

ARX	Performance				
	Training Dataset		Testing Datasets		
$[n_y\ n_u\ n_d]$	Loss Function	Fit (%)	Testing Fit I(%)	Testing Fit II (%)	Testing Fit III (%)
[0 7 0]	8.089×10^{-4}	98.11	89.52	99.35	98.26
[0 8 0]	$\mathbf{3.013} \times 10^{-4}$	**98.85**	**95.27**	**99.14**	**99.31**
[1 7 2]	5.544×10^{-1}	52.12	−75.07	69.87	57.10
[2 1 1]	19.05×10^{-1}	2.058	−0.09	−0.28	2.05
[3 4 1]	4.182×10^{-1}	27.58	−43.19	22.18	20.66
[3 11 0]	2.643×10^{-4}	98.80	95.65	99.21	99.35

Table 3.4: ARX Model Orders Vs Performance for Inverse Problem

The discrete ARX model obtained in this case is:

$$y(t) = B(z)u(t) + e(t)$$
$$B(z) = 409.8 - 1514z^{-1} - 2574z^{-2} - 2766z^{-3} + 2766z^{-4} - 1028z^{-5} + 315z^{-6} - 45.04z^{-7} \tag{3.3}$$

Additionally, use of NARX-Neural Network was also investigated in this case, a two hidden layered MLP with 8 nodes per layer and linear activation functions with $n_u = 0$ and $n_y = 8$ produced satisfactory results. The performance was evaluated based on Goodness of Fit criteria (equation 2.35). Table 3.5 shows the comparison of performances of the models for inverse problem during the training and testings.

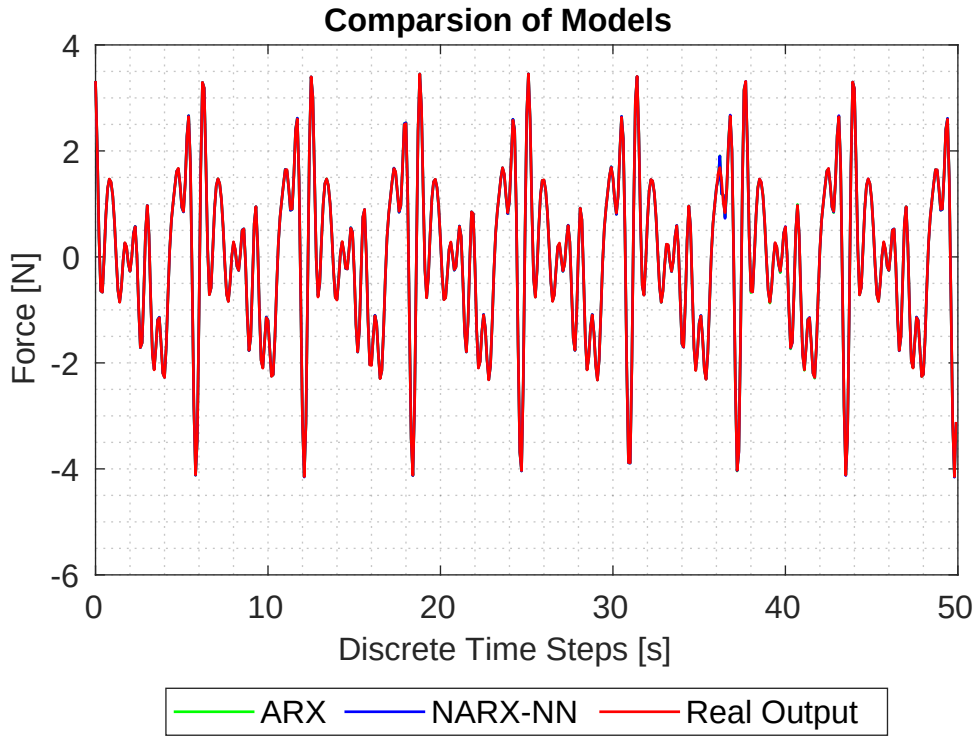


Figure 3.15: Identified model performance comparison during training for Inverse problem

Models	Training perform.(%)	Testing I Perform.(%)	Testing II Perform.(%)	Testing III Perform.(%)
NARX-NN	98.85	95.49	99.10	99.31
ARX	98.85	95.27	99.14	99.35

Table 3.5: Model performance comparison on available data in case of Inverse problem

Because of the excellent agreement between the real output and the model prediction curves obtained from ARX and NARX models are almost perfectly overlapping with each other. But ARX is much faster than NARX Neural Network as depicted in Figure 3.19.

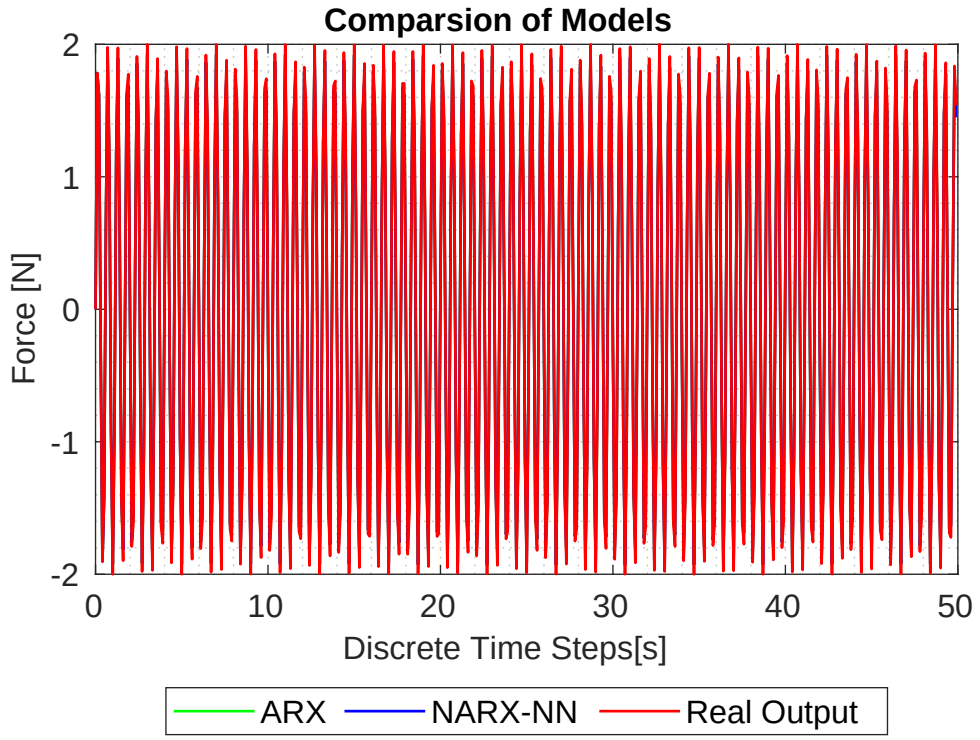


Figure 3.16: Identified model performance comparison during Testing Dataset I for inverse problem

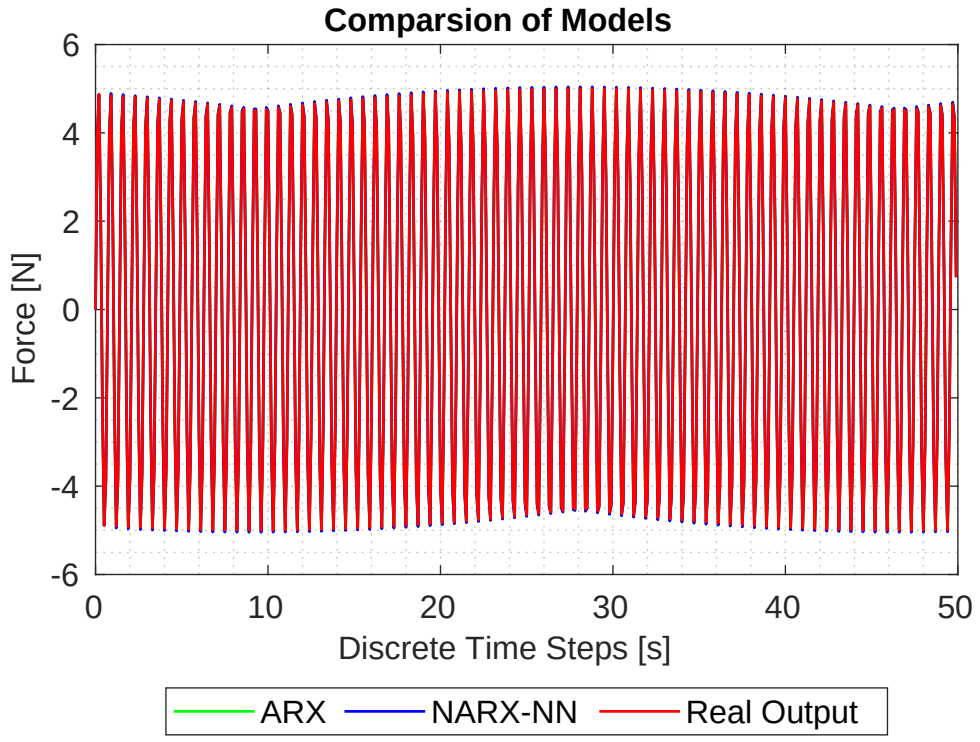


Figure 3.17: Identified model performance comparison during Testing Dataset II for inverse problem

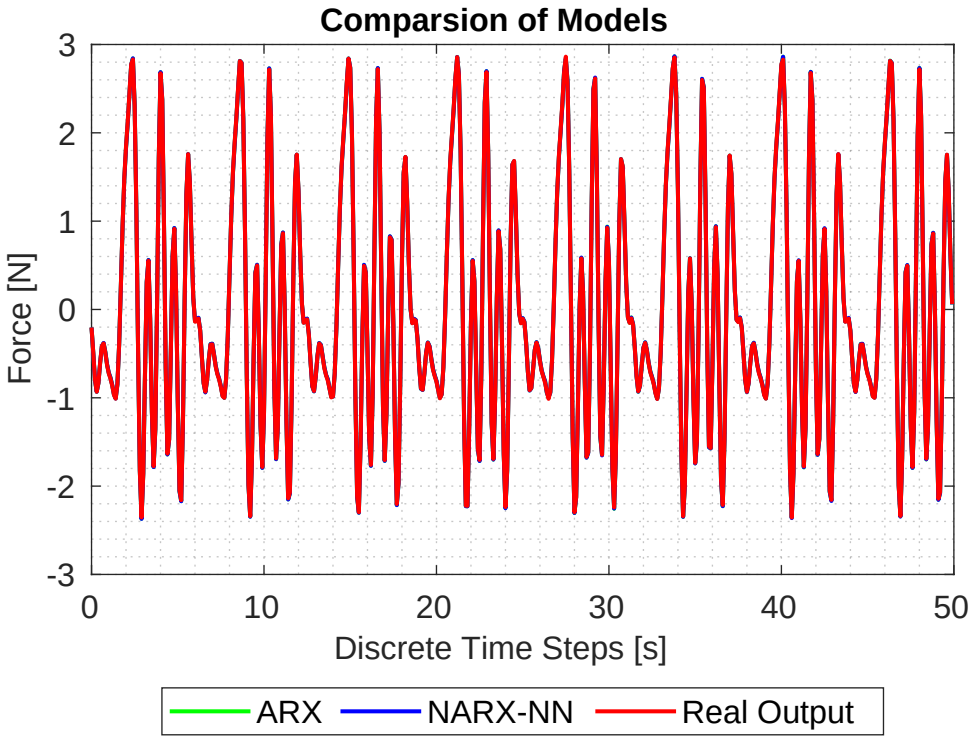


Figure 3.18: Identified model performance comparison during Testing Dataset III for inverse problem

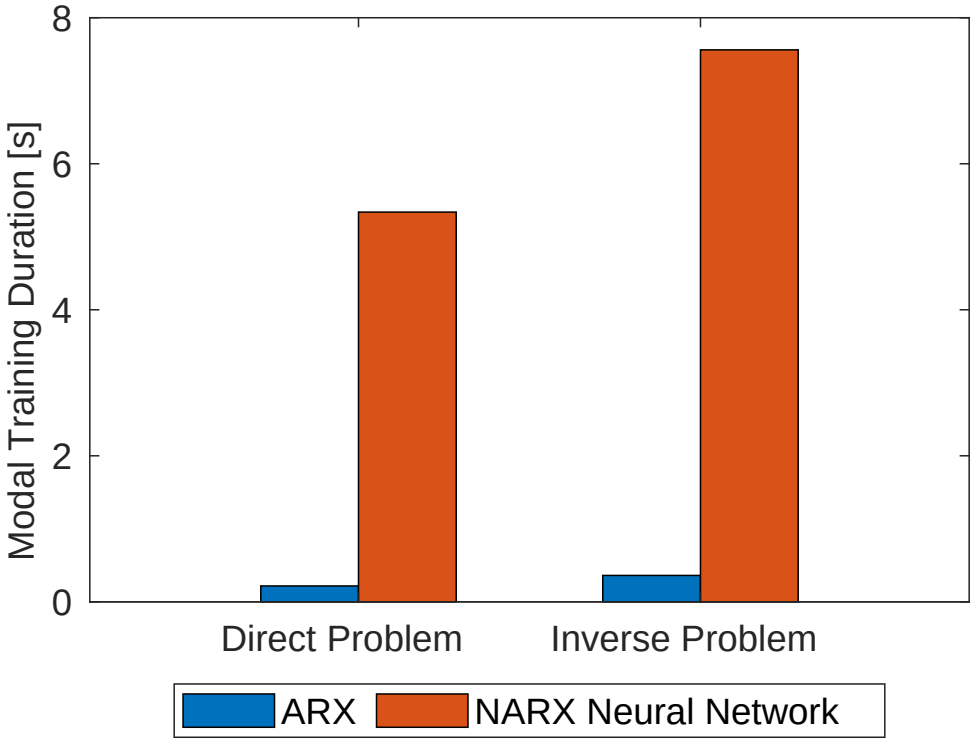


Figure 3.19: Model training duration comparison

4 Numerical Study of Wave Tank

4.1 Data generation and pre-processing

In this study, first the surface waves were generated in the numerical wave tank (NWT) subjected to the multi-frequency harmonic excitations at the source region of the NWT as shown in Fig 2.11. The simulation was carried out in OpenFOAM solver. Each simulation lasted for almost 4 hours on an average to finish, and ran in parallel processing over 4 CPU i7-7700k cores. All Input signals were multi-sinusoidal in nature, equally spaced frequency with a random phase based on Pierson Moskowitz Spectrum. Python was used

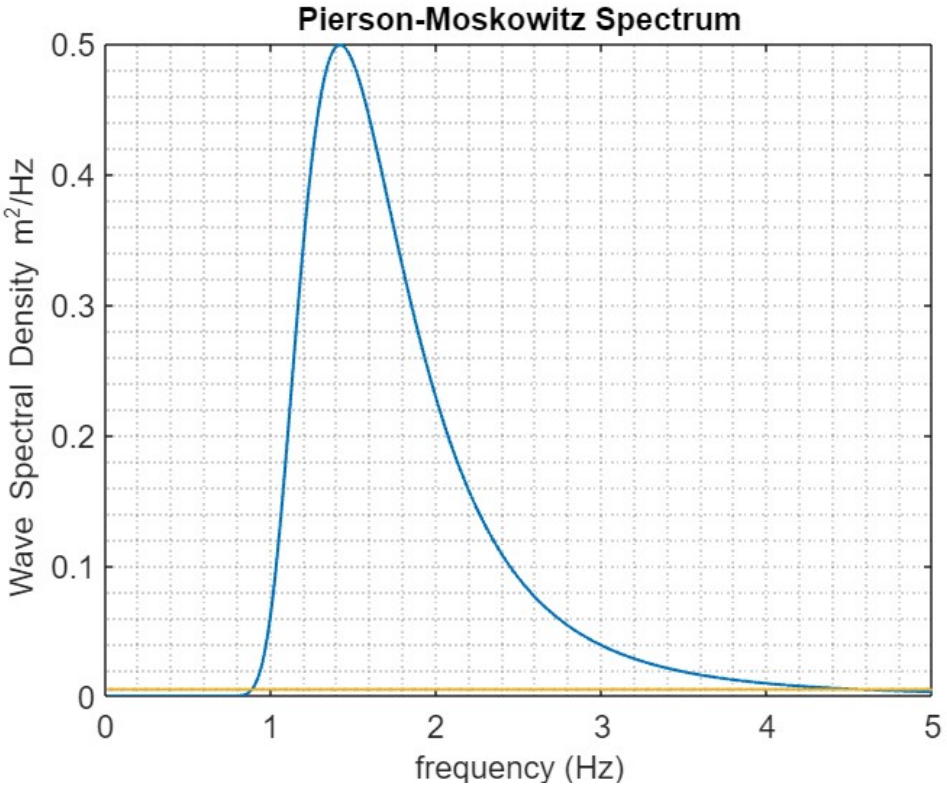


Figure 4.1: Frequency Spectrum of Input Excitation

to extract data from the OpenFOAM which was then processed in MATLAB and the results for different input signals are shown in the following figures. The actual datasets included 20001 data points. After looking carefully into the fast fourier transformation of the training dataset, it is convenient and worth to downsize the sampling frequency by 1/10 of the actual size using an inbuilt MATLAB function, *interp1*. Otherwise the system identification process becomes time consuming to carry out on a normal computer. Results obtained from the OpenFOAM simulations for the corresponding inputs are shown in the following figures. A single dataset is used for training of all models and is named as 'Training dataset' and rest of the datasets are used for validation of models. All datasets consist of different magnitudes of amplitudes and frequency ranges.

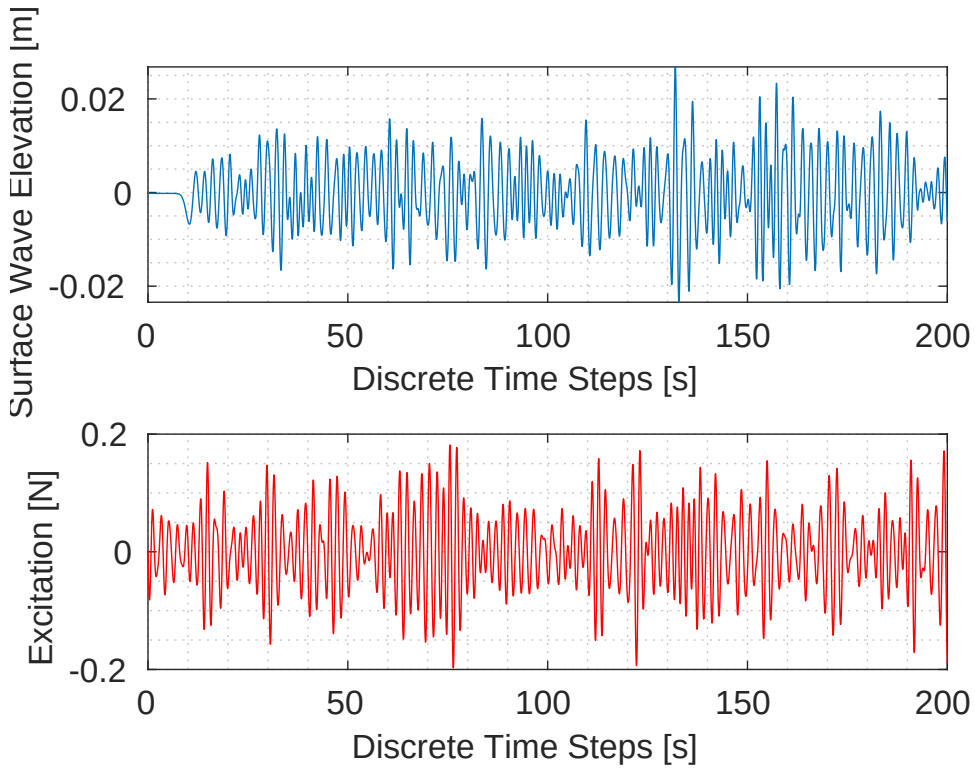


Figure 4.2: Surface waves generated in OpenFOAM. (Training Dataset)

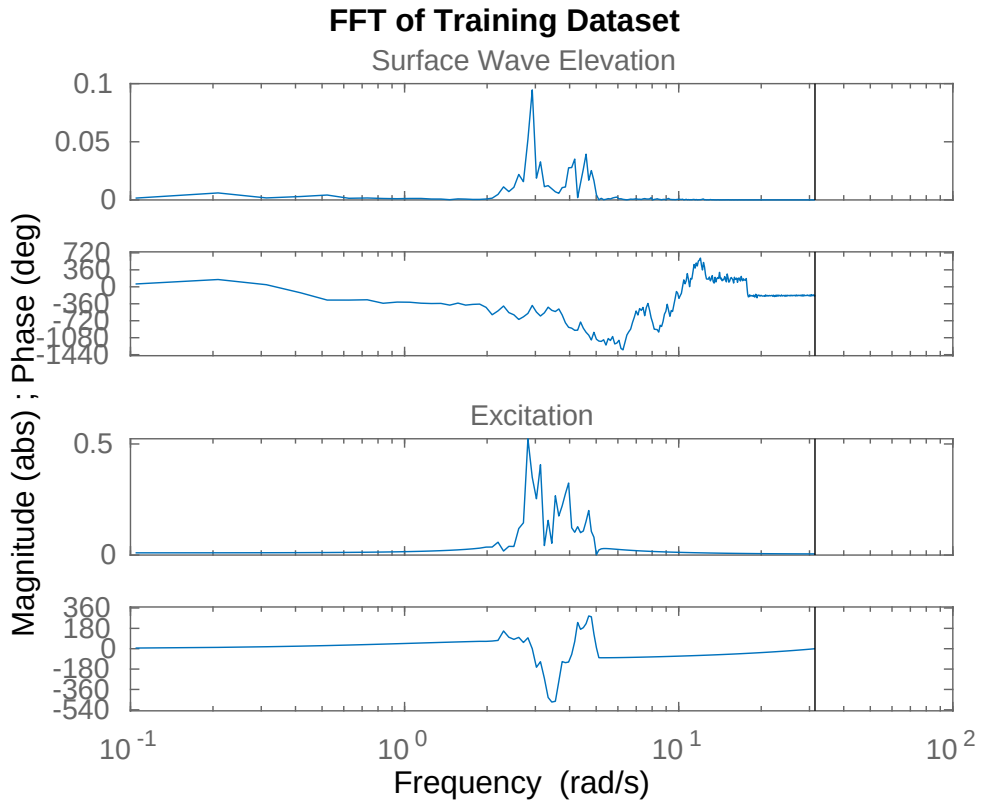


Figure 4.3: FFT of Training Dataset

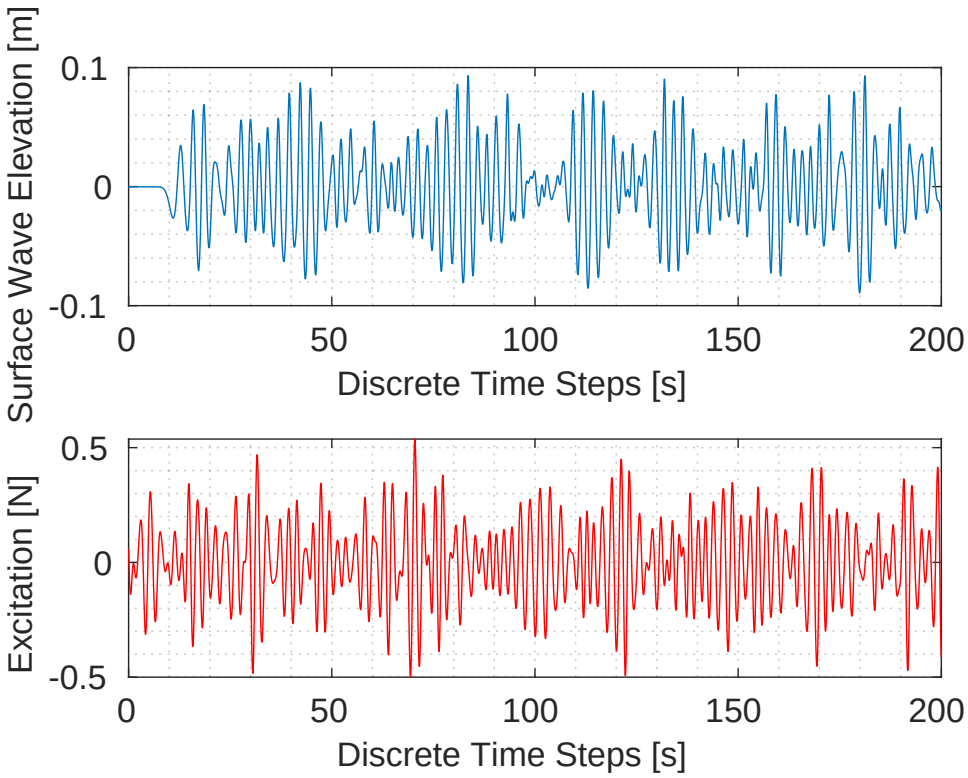


Figure 4.4: Surface waves generated in CFD solver for an Excitation with the highest amplitude (Testing Dataset I)

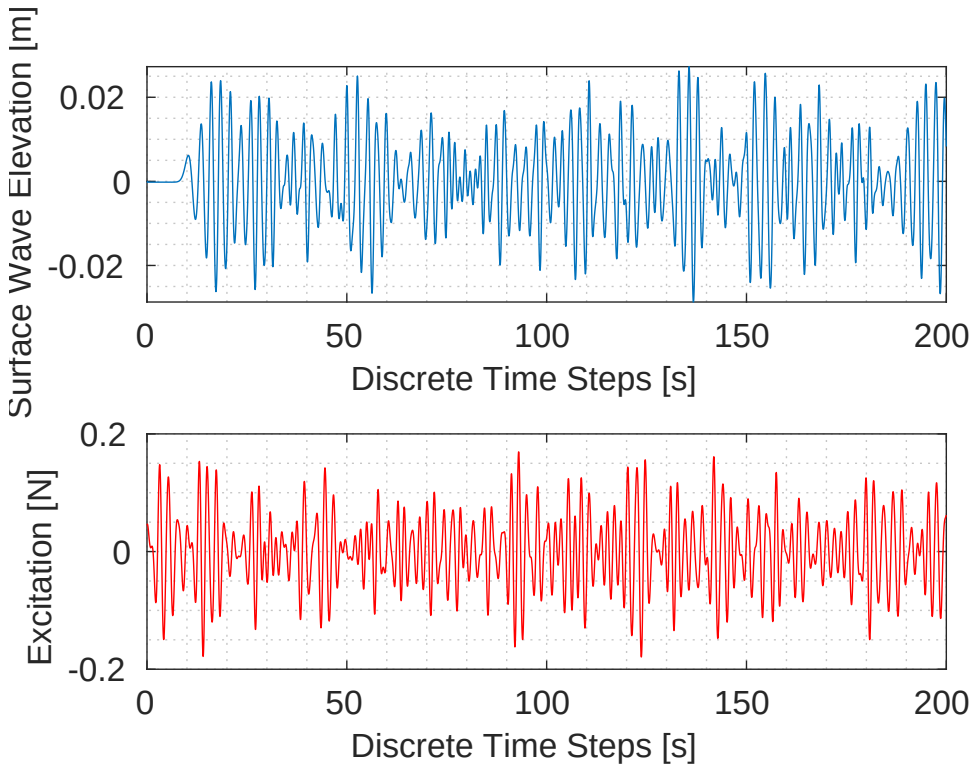


Figure 4.5: Surface waves generated in CFD solver for a medium sized Excitation (Testing Dataset II)

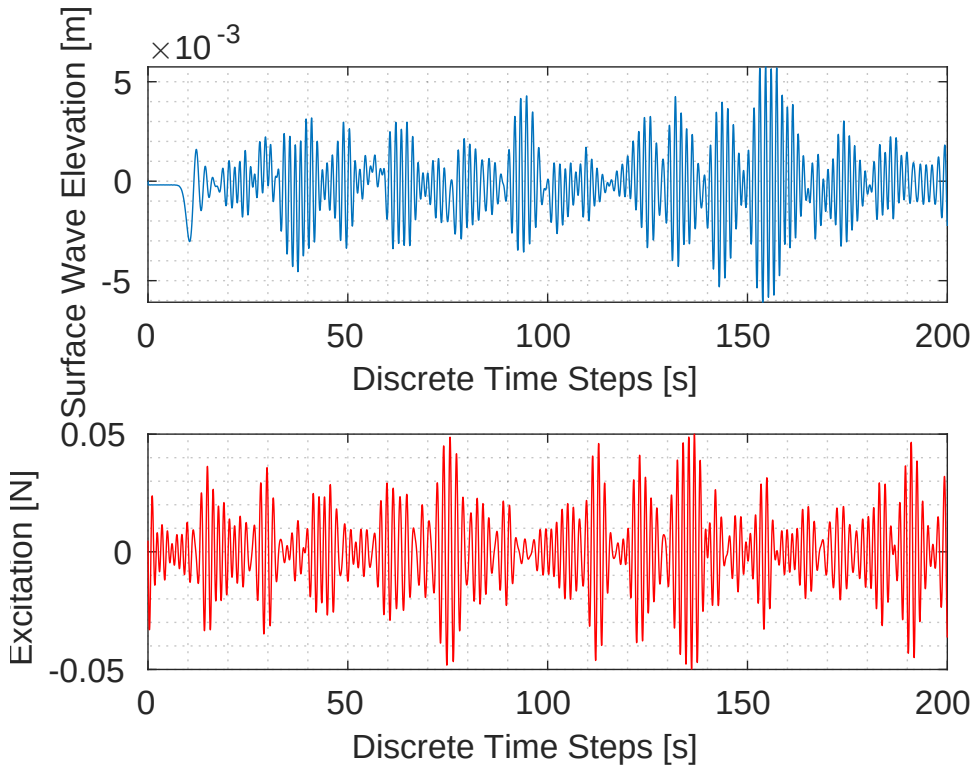


Figure 4.6: Surface waves generated in CFD solver for an Excitation with the smallest amplitude (Testing Dataset III)

4.2 Direct Numerical Wavetank (NWT) problem

After acquisition of data, the second most important task is to find out a model which can map a non-linear relationship between the desired surface wave elevation generated at some probe distance in NWT to the required Input (excitation). Additionally, there is a delay at the beginning of each output signal which accounts to the fact that the probe is located at some distance away from the point source in NWT. It is also desirable that the identified model should also capture this time varying delay accurately. Since these generated waves are multi-frequency in nature, therefore, require different amount of time to reach to the probe with different velocities.

In this section, black-box models such as ARX, NARX, HW and OE models are trained over a single training example, capturing the most of the dynamics of the system then tested over different datasets with amplitude variations of almost 10 times the amplitude of the training dataset. Moreover, a comparison of models based on individual complexity, computational cost and accuracy is also presented at the end.

4.2.1 Autoregressive with External Inputs (ARX)

For ARX model, MATLAB's inbuilt functions such as *arxstruc* and *selstruc* with *AIC* criteria can provide some intuitive ideas about the initial model order selections. In this case, surface plots consisting of variation of loss function with respect to n_y *and* n_u for

different values of delay n_d are shown in the Fig. 4.7. *It can be observed that higher*

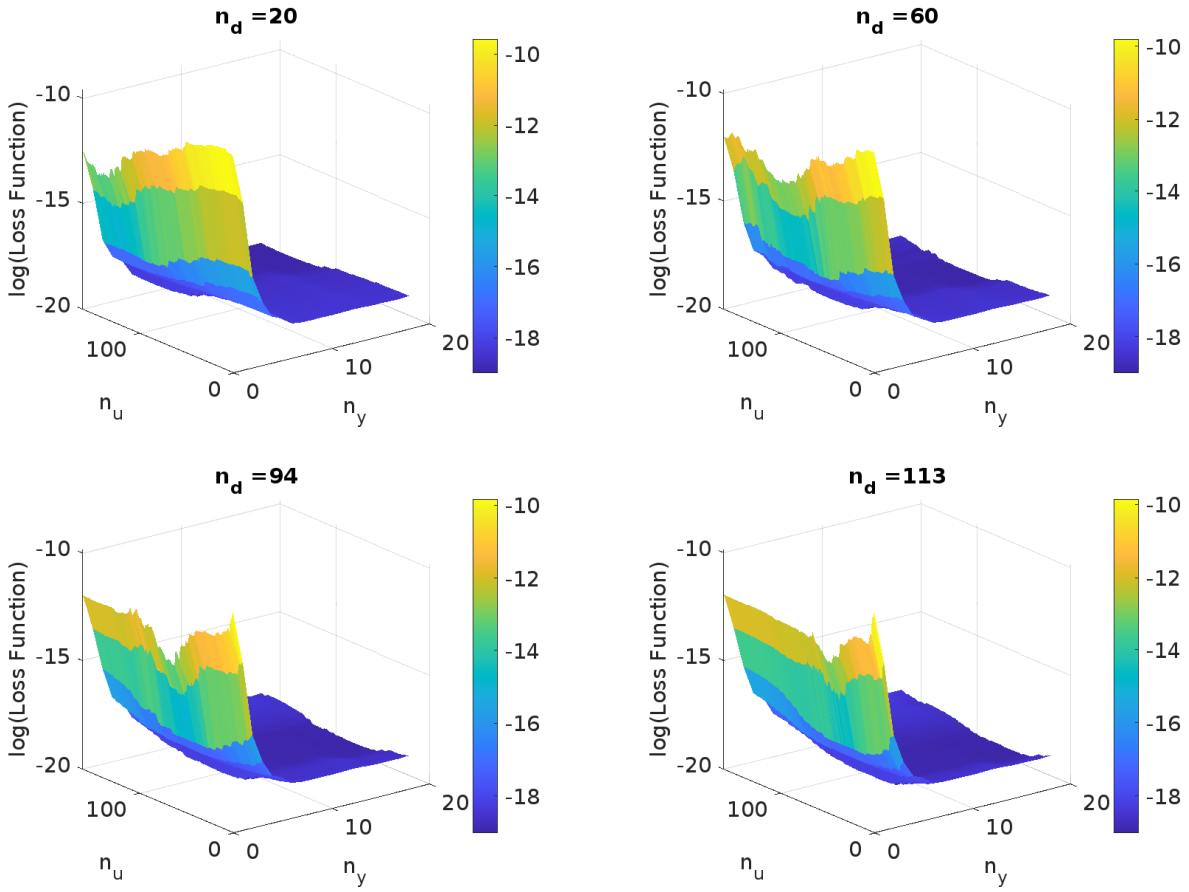


Figure 4.7: Variation of loss function (see 2.1) w.r.t. n_y, n_u, for different values of n_d

the value of delay n_d is chosen, smaller the value of n_u is required to find the optimum model order at which loss function value is minimum. The value of the delay n_d if set to 94 was found to be the best performing model during the training and validation while maintaining a trade off between lower model complexity and higher accuracy.

For $n_y > 2$ loss function has no significant reduction, while keeping $n_u = 103$. Similarly, for $n_y = 2$, $n_u = 103$ provides a model order which performs better across all datasets and has lower number of parameters,thus lesser complex. With this criteria, ARX with model order [2 103 94] was chosen to be the best performing ARX model while looking into different model order configurations from the following Table 4.2. To find the performance of the ARX with the other model orders, refer to Table A.2 in Appendix. The performance of the arx during training and validation with only the first 800 data points is shown in Fig 4.10.

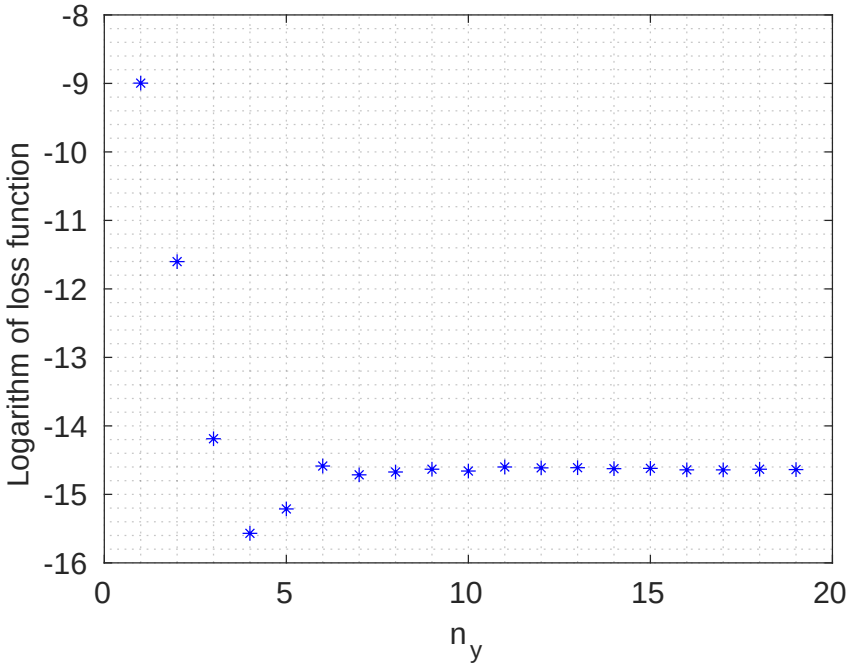


Figure 4.8: Variation of logarithm of loss function with n_y for $n_u = 103, n_d = 94$

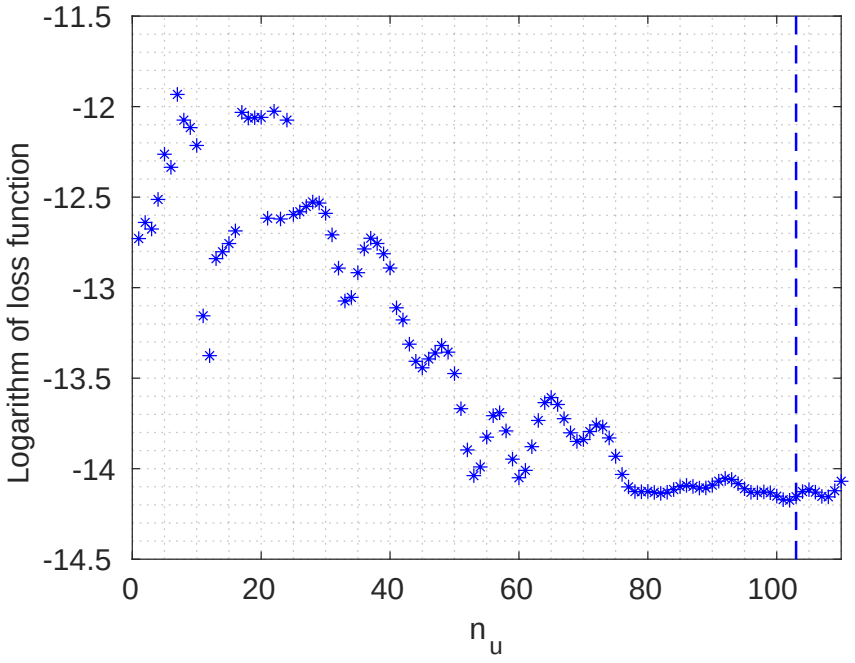


Figure 4.9: Variation of logarithm of loss function n_u for $n_y = 2$, $n_d = 94$

ARX	Performance				
	Training Dataset		Testing Datasets		
$[n_y\ n_u\ n_d]$	Loss Function	Fit (%)	Dataset I Fit (%)	Dataset II Fit (%)	Dataset III Fit (%)
[2 103 94]	$\mathbf{1.057 \times 10^{-7}}$	**86.83**	**83.50**	**91.27**	**70.05**
[2 108 94]	1.121×10^{-6}	86.09	81.92	91.28	68.51
[3 103 94]	2.451×10^{-7}	87.67	81.20	90.69	68.45
[4 103 94]	1.377×10^{-7}	87.56	80.61	89.52	67.65
[5 103 94]	6.501×10^{-8}	87.96	80.91	90.81	69.41
[6 103 94]	2.558×10^{-8}	87.39	80.37	89.49	68.03
[7 103 94]	1.004×10^{-8}	87.70	79.04	89.36	69.50
[8 103 94]	5.331×10^{-9}	87.33	80.34	89.37	68.13

Table 4.1: Goodness of fit of ARX with different model orders for Direct NWT problem

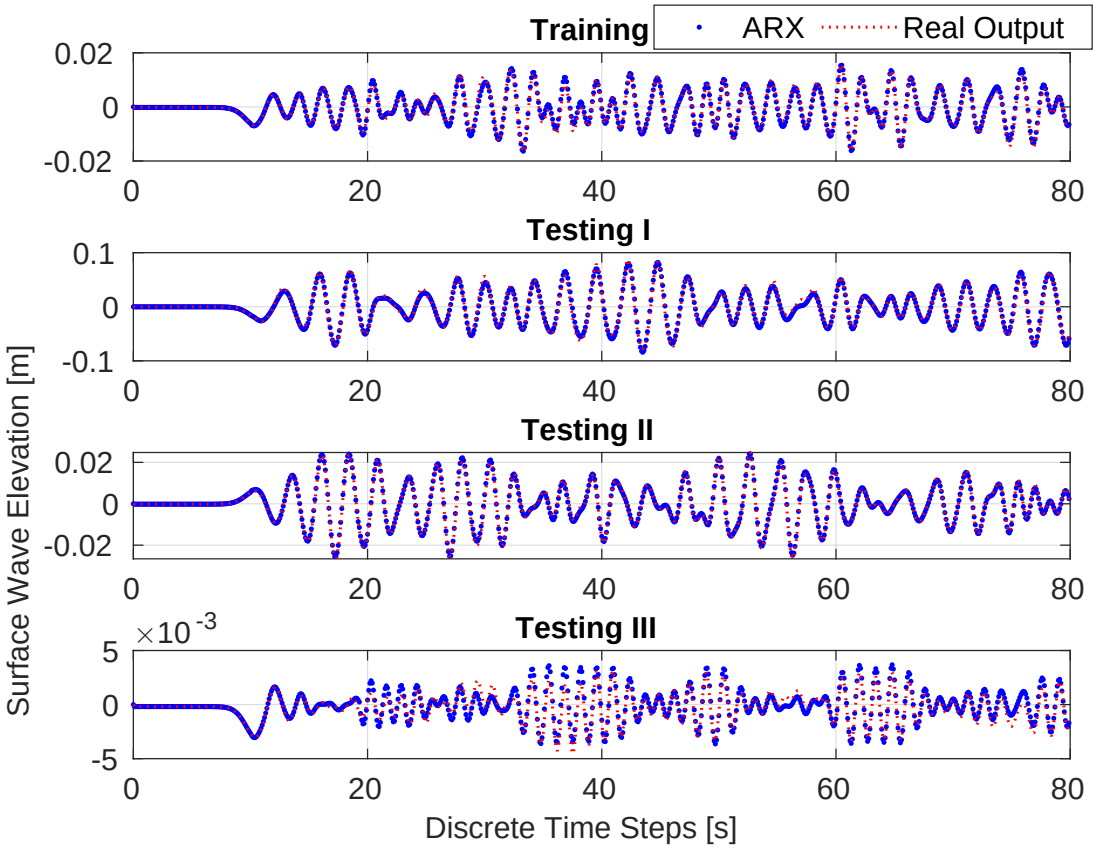


Figure 4.10: Performance of ARX model during training and testing for Direct NWT

4.2.2 Output Error (OE) Model

The Output Error (OE) method deals with minimizing a nonlinear least square (NLS) loss function, using a nonlinear programming algorithm e.g. Levenberg-Marquardt (LM).

In this case the best model order at which training as well as validation results obtained were satisfactory was found to be [99 3 88] with NLS as the loss function and LM optimization algorithm over a maximum of 30 iterations. Here the first term in the model order represents the number of the past inputs required, followed by the model outputs and finally the input delay parameter, for further details, see section 2.4.5. The results obtained are shown in Table 4.2.

4.2.3 Hammerstein-Wiener (HW) Model

The dynamics of the required surface wave elevation to the given excitation was represented by a linear transfer function and the associated non-linearities were added using static nonlinear functions placed at the input and the output of the linear system. For an estimation of the linear transfer function for the Hammerstein-Wiener (HW) model, various pole-zero configurations were explored. The effect on the loss function and the fitness percentage for the various pole-zero and delay configurations, $[n_b\ n_f\ n_k]$ to the linear dynamic subsystem of the HW model can be seen in appendix Table A.4, where n_b is the number of zeros plus 1, n_f is the number of poles of polynomials B and F respectively used in linear transfer function block, and n_d is the input delay (see section 2.4.6, for further details). The optimal pole-zero and deadtime configuration was chosen after examining the maximum fitness percentage values obtained during both training and validation for various model order configurations and was found to be [103 3 96] with nonlinear least square loss function, and Levenberg-Marquardt as optimization algorithm over a maximum of 50 iterations. Moreover, a sigmoidal layer of 14 units and piecewise linear functions of 10 units, were used as static functions at the input and output of the linear system block respectively to obtain this best peforming HW model.

4.2.4 Nonlinear Autoregressive with exogeneous variables

Nonlinearities can be added to the ARX model found by including nonlinear mapping functions, such as Sigmoid network or Wavelet network or Partition tree network in the model. A Sigmoid network for instance, is a sum of 3 components - a linear function, an offset term, and a nonlinear function (see section 2.4.4 to get further details). Here, a partition tree network of 2 units together with the best performing ARX model ([2 103 94]) enhanced the ARX model performance slightly and can be seen in Table A.3 of the Appendix.

Additionally, use of NARX-Neural Network was also investigated in this case. After performing an hyperparameter optimization search, a two hidden layered cascaded neural network with 2 and 4 hyperbolic tangent activation functions in first and second layer respectively and a linear function at the output layer with model order $n_u = 3$, $n_y = 103$ and $n_d = 94$ as shown in the Fig. 4.11 produced satisfactory results during training as well as testing.

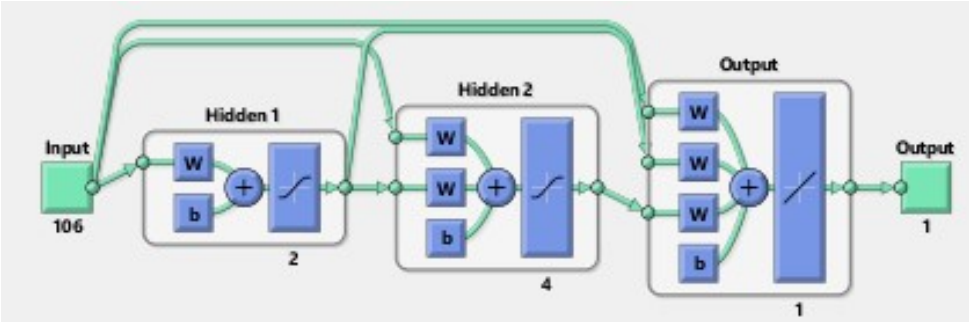


Figure 4.11: Cascade Neural Network Structure for Direct NWT problem

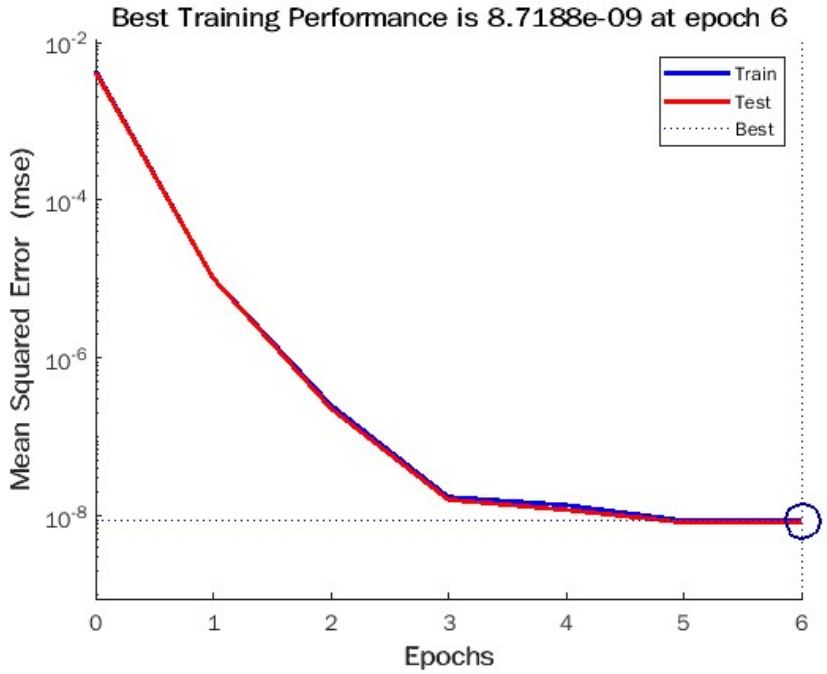


Figure 4.12: Loss Function vs Epochs during NARX Neural Network

Models	Model order	Training perf.(%)	Testing I Perf.(%)	Testing II Perf.(%)	Testing III Perf.(%)
ARX	[2 103 94]	86.83	81.81	91.27	70.05
NARX NN	[3 103 94]	86.26	80.62	91.01	70.93
OE	[99 3 88]	87.52	80.55	90.56	68.21
HW	[103 3 96]	87.10	81.41	92.73	71.79

Table 4.2: Model performance comparison during training and validation for NWT direct problem

Only the first 800 data points are shown in the following figures to get a more detailed view of model performances. All of the models used, performed comparably well during training and testing. But it was rather the use of simple yet powerful machine learning

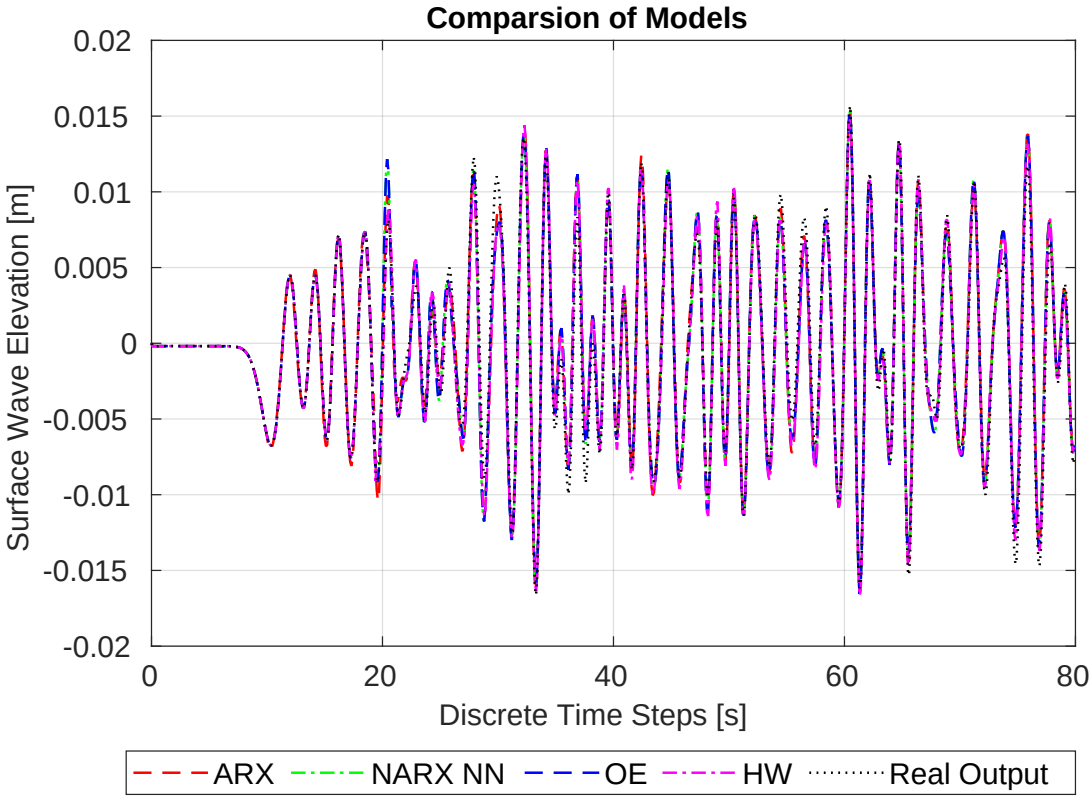


Figure 4.13: Model performance during Training.

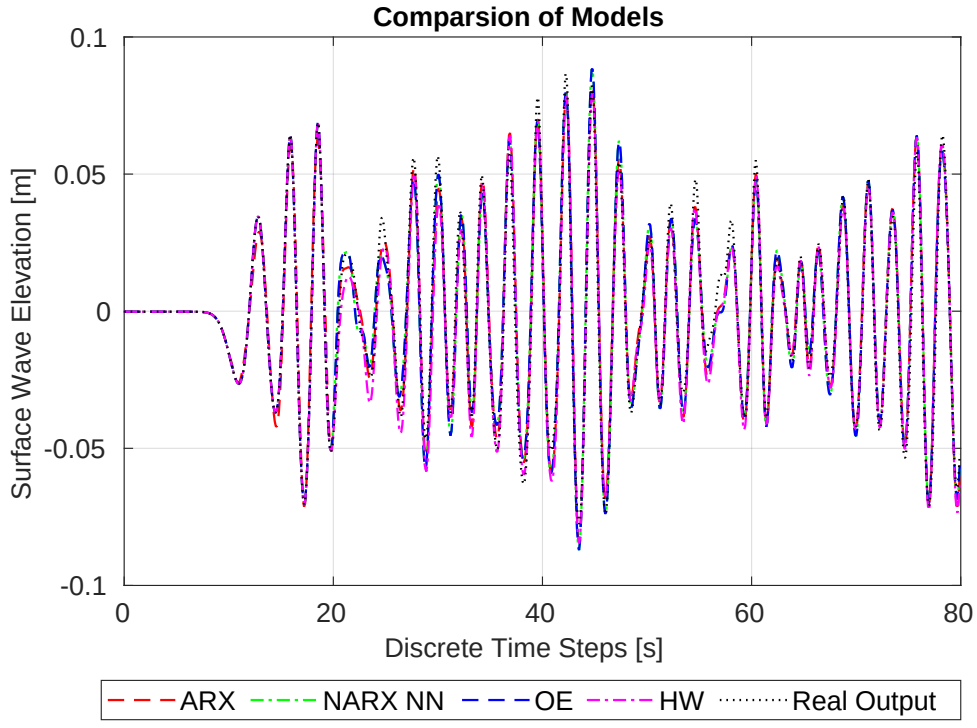


Figure 4.14: Model performance during Testing I.

model such as ARX, which surprisingly captured the nonlinear behaviour of the NWT. Based on the model training duration required for each model, from Fig 4.17, it can be deduced that model training of ARX was almost 13 times shorter than NARX neural

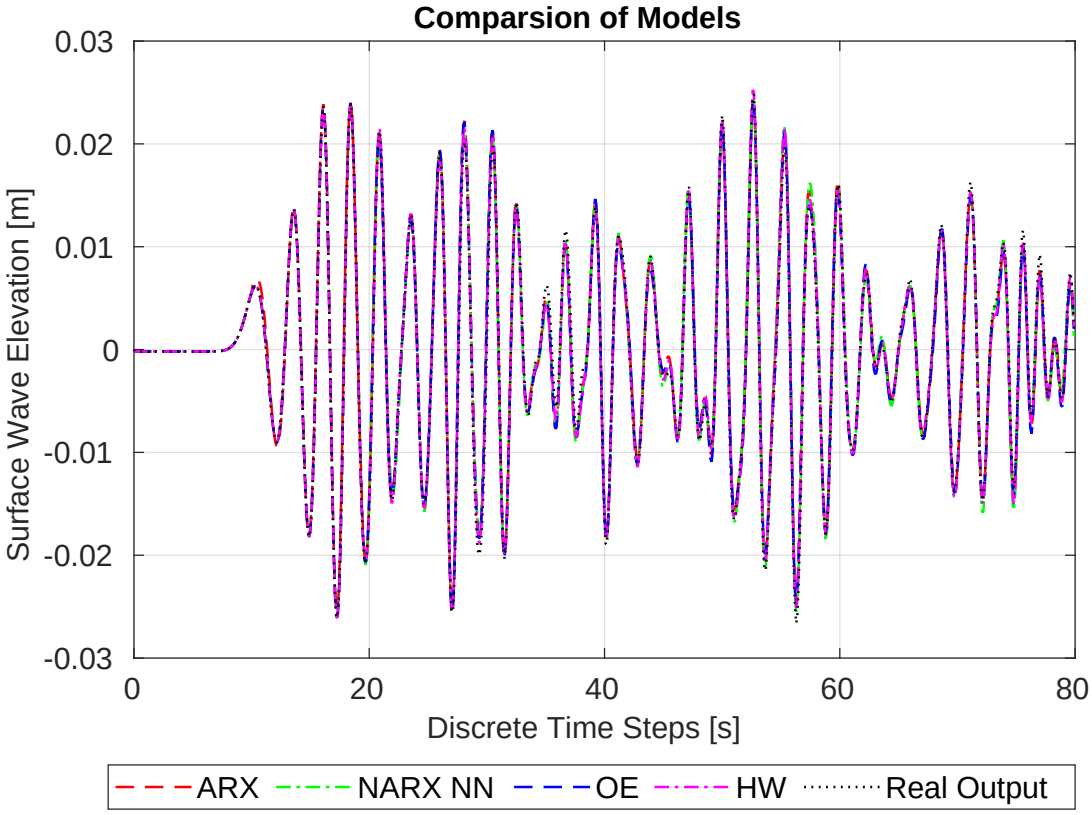


Figure 4.15: Model performance during Testing II.

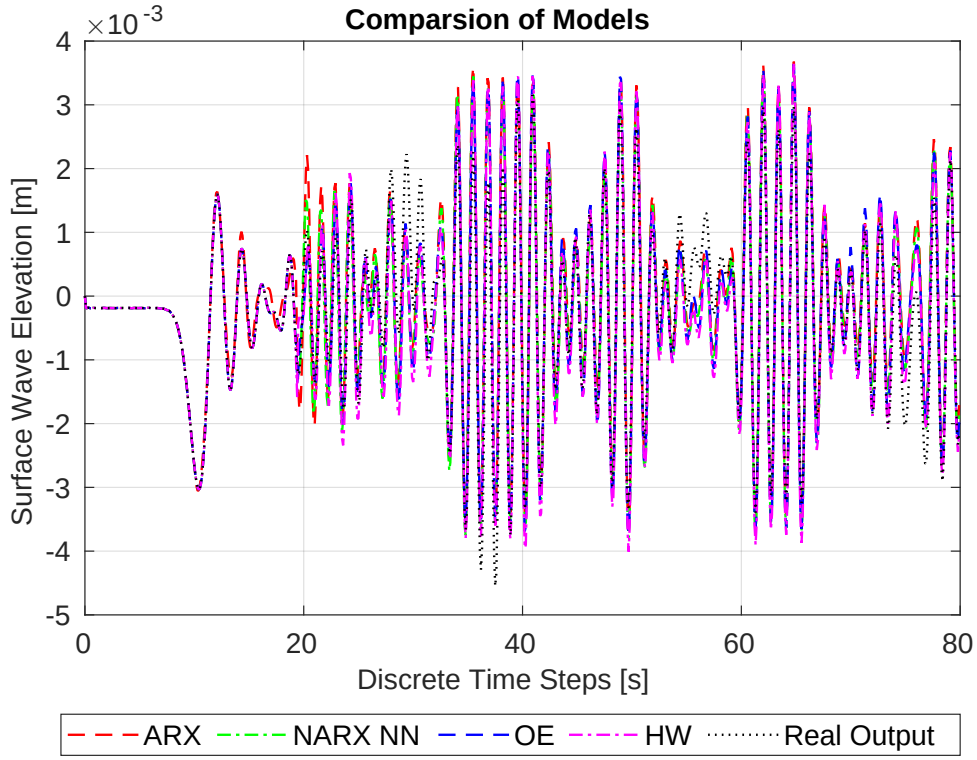


Figure 4.16: Model performance during Testing III.

Network, 11 times shorter than nonlinear HW model and 4 times shorter than nonlinear OE model. In terms of model complexity, NARX-NN and HW models consist of the largest

number of model parameters, 106 followed by ARX, 105 and finally nonlinear Output Error model, 102. But all models other than ARX are nonlinear in nature and involve nonlinear function in their structure which therefore increases the model complexity.

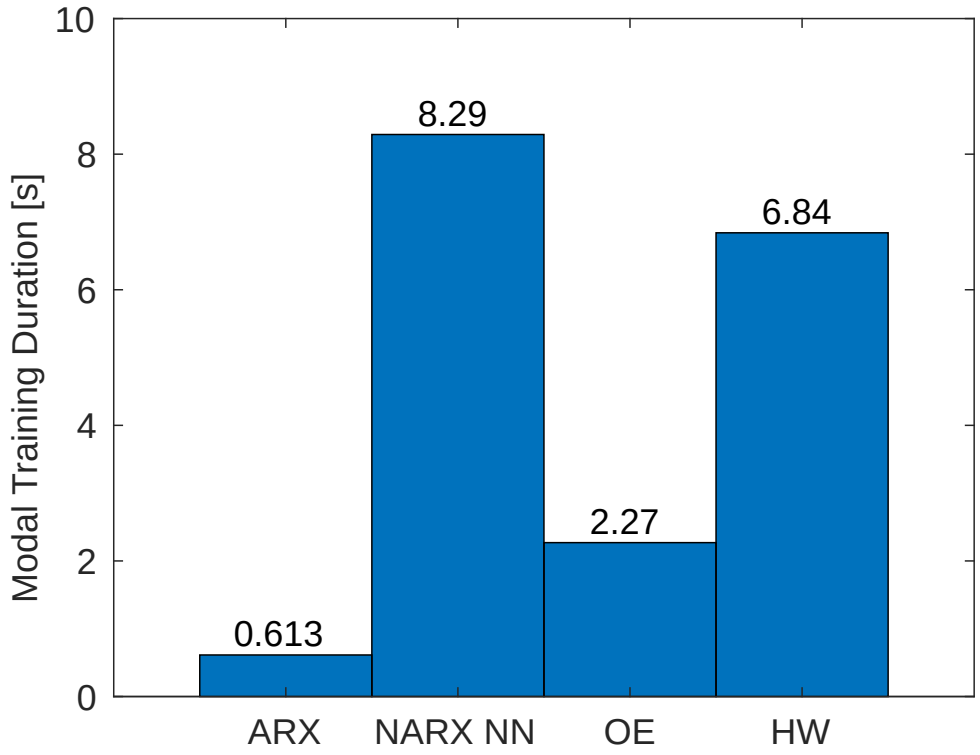


Figure 4.17: Model training duration comparison for NWT direct problem

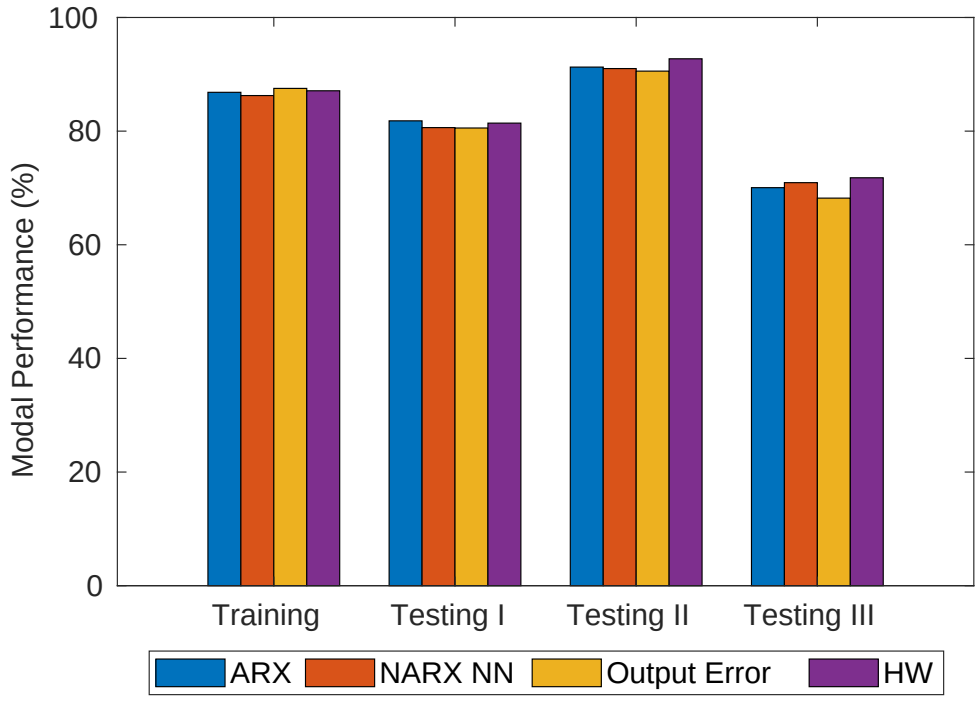


Figure 4.18: Model performance comparison over training and testing datasets

4.3 Nonlinear Inverse Numerical Wave Tank Problem

Finally, a highly significant problem yet difficult to solve with OpenFOAM, at least in its current setup, is the inverse problem of NWT, where it is required to determine the 'Excitation' or 'Impulse' applied at the source location to produce a desired Surface Wave Elevation at some probe length in the numerical Wavetank. In this problem, we have chosen the same set of data obtained after performing CFD simulations (see section 4.1). One of the datasets named as 'Dataset A' was split into training and testing on a 55:45 ratio as shown in Fig 4.19. Other datasets were also checked against their suitability to train the models but 'Dataset A', which consists of manily medium sized amplitude waves, emerged as the best training set upon which the models performed well during the validation. Here the data labelled as "Surface Wave Elevation", obtained from the CFD solver, was fed into the identified models as the Input and the "Impulse" is the required output, the models should predict/simulate. For further model validations, additional datasets named as 'B', 'C' and 'D' were used. To simplify the problem, all the signals were shifted by 200 time steps to compensate for the delay in "Surface Wave Elevation" obtained from CFD simulations, owing to the fact that the probe in the NWT is located at some distance away from the source region.

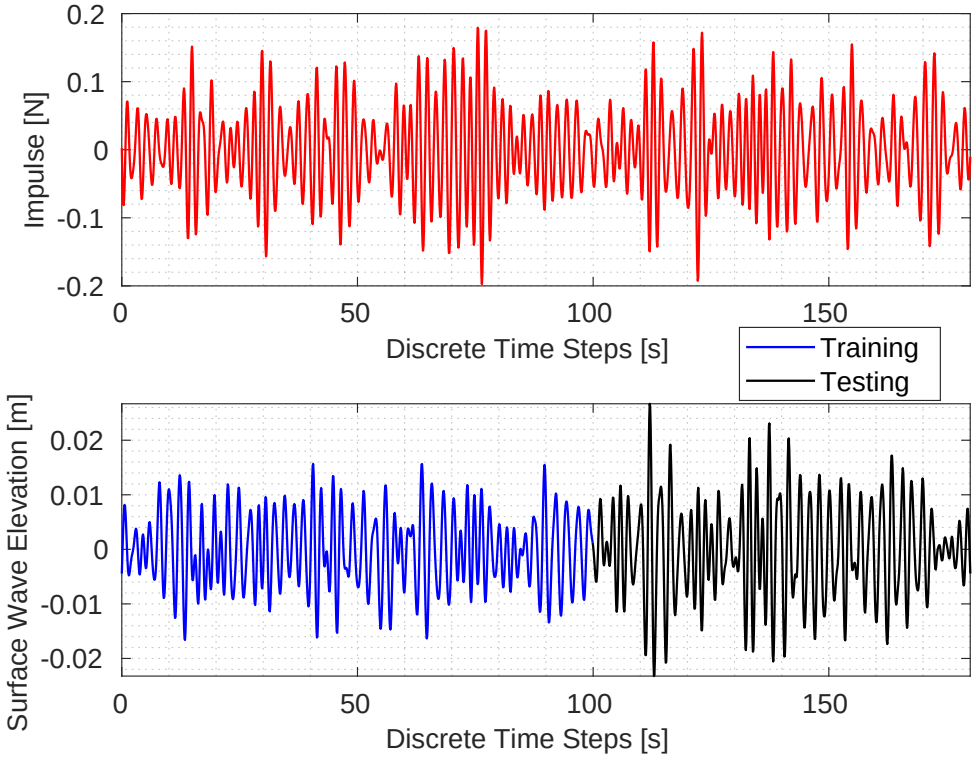


Figure 4.19: Dataset A with medium sized amplitude for inverse NWT problem

Black-Box models such as ARX, Hammerstein-Wiener and Output Error models were investigated and the accuracy of the models was tested on each available datasets and finally a comparison of models based on individual complexity, computational cost and accuracy is presented at the end.

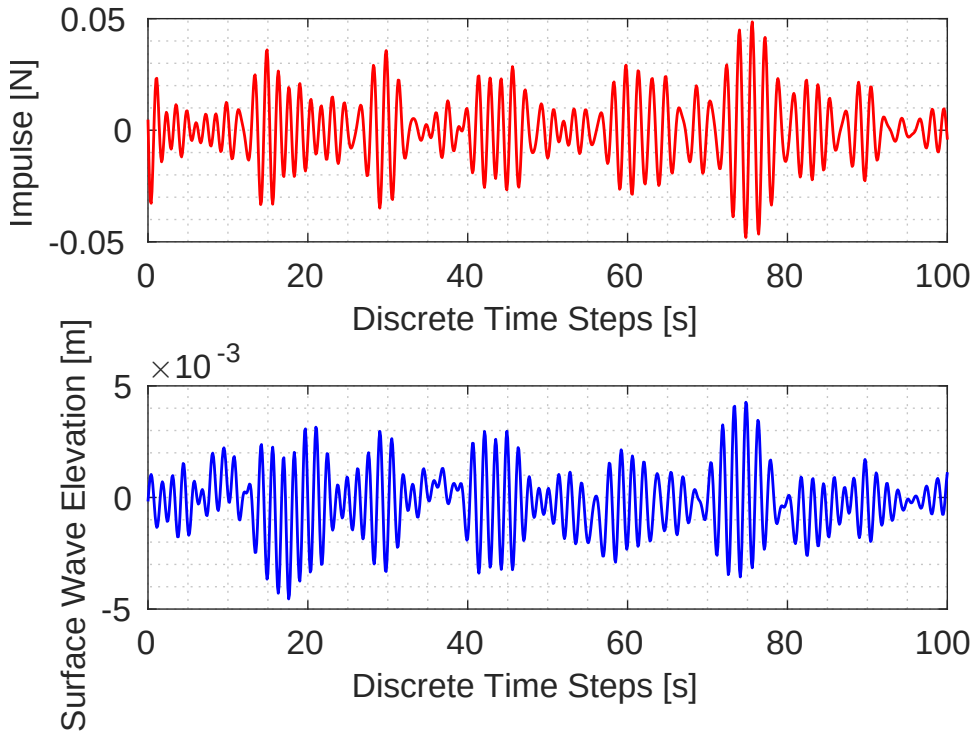


Figure 4.20: Dataset B with the smallest amplitude used for model validation.

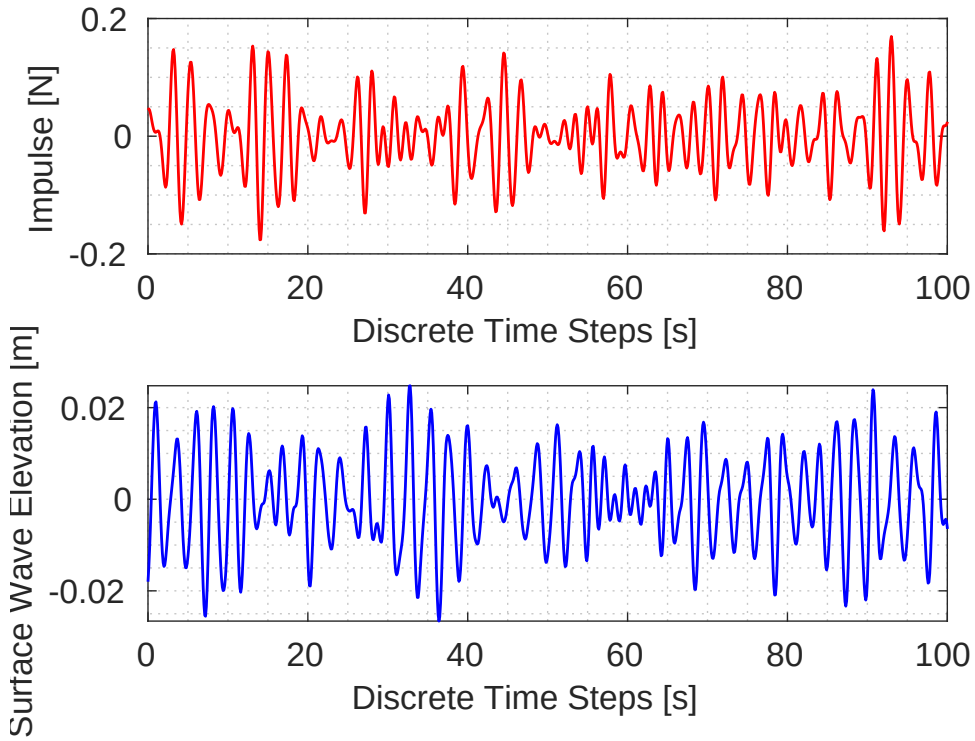


Figure 4.21: Dataset C with comparable amplitude used for model validation.

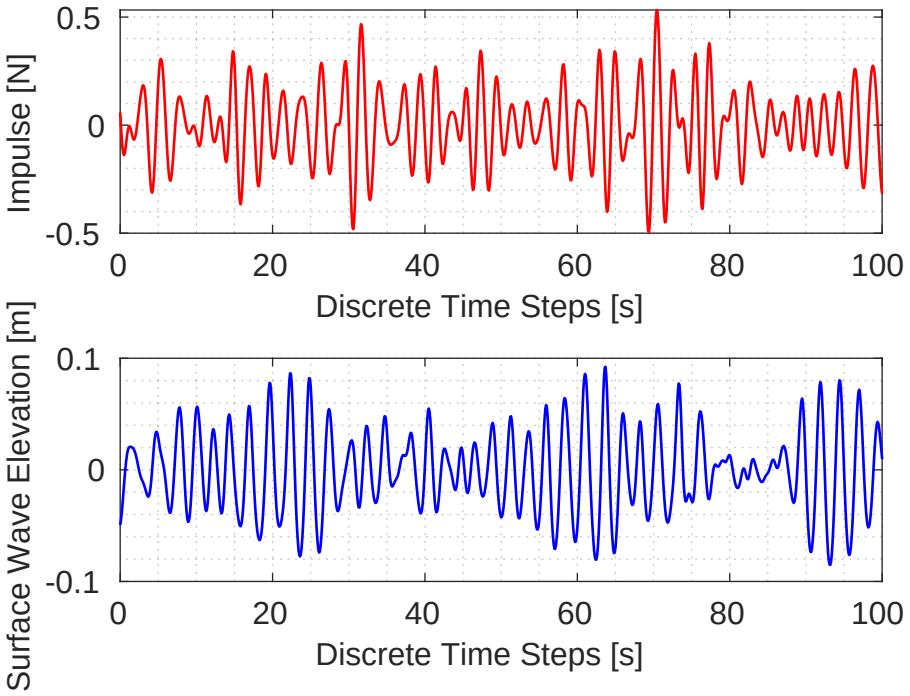


Figure 4.22: Dataset D with the highest amplitude for additional validation

4.3.1 Autoregressive with External Inputs (ARX)

For ARX model, MATLAB's inbuilt functions such as *arxstruc* and *selstruc* with *AIC* criteria can provide some intuitive ideas about the initial model order selections. In this case, there is no need to keep the delay as it is already compensated during preprocessing of data. Surface plot consisting of variation of loss function with respect to n_y *and* n_u for value of delay $n_d = 0$ is shown in the Fig. 4.23. For $n_y > 7$ loss function has no significant

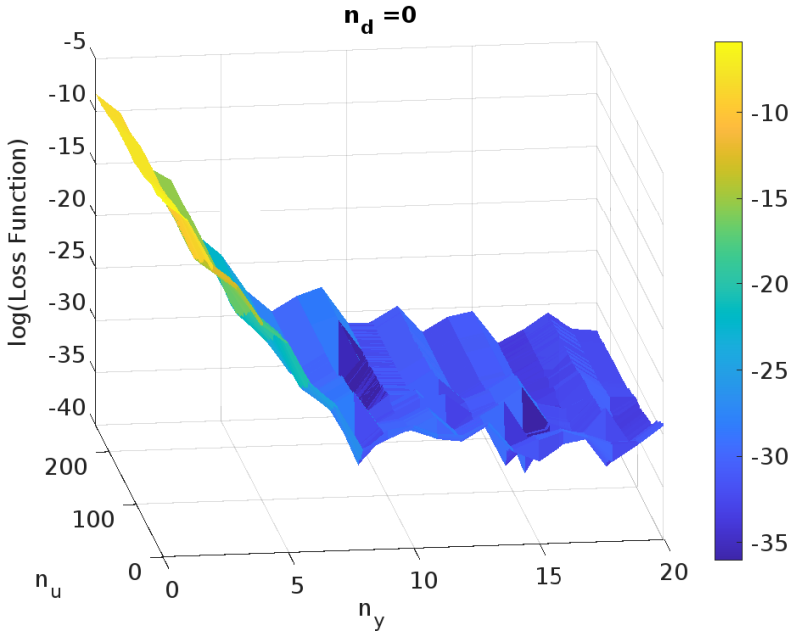


Figure 4.23: Variation of loss function w.r.t. n_y *and* n_u for $n_d = 0$

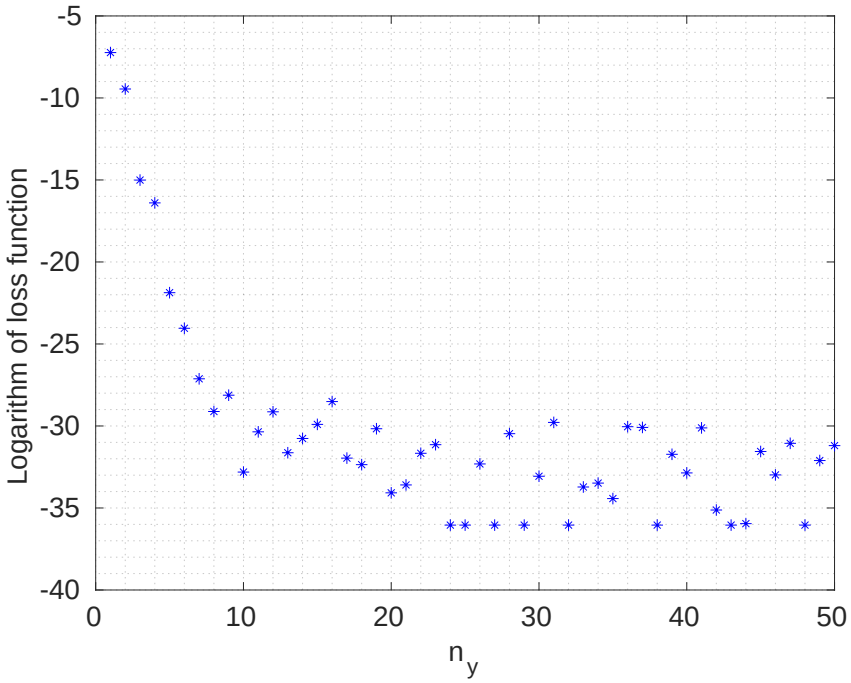


Figure 4.24: Variation of logarithm of loss function with n_y for $n_u = 93, n_d = 0$

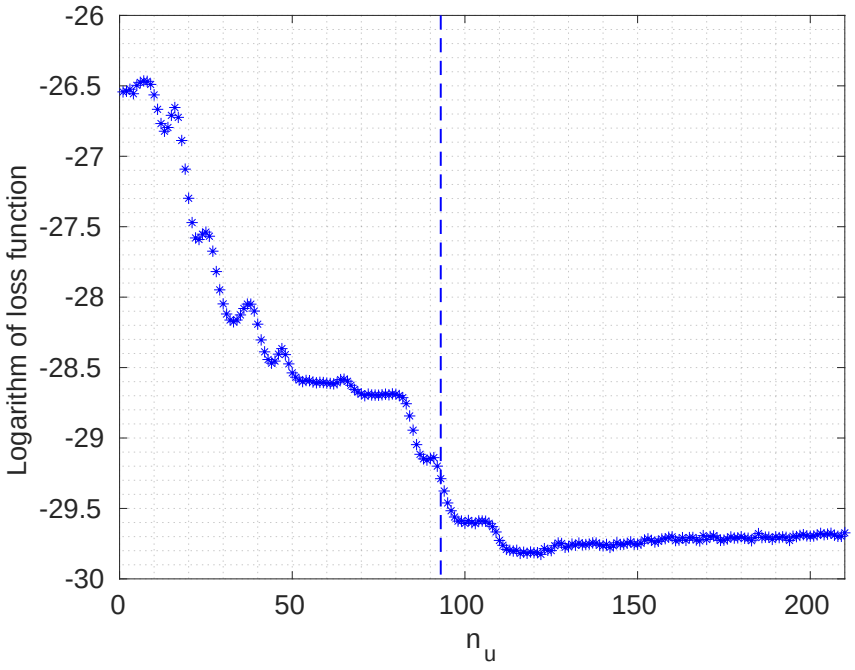


Figure 4.25: Variation of logarithm of loss function with n_u for $n_y = 7, n_d = 0$

reduction, while keeping $n_u = 93$. Similarly, for $n_y = 7$, $n_u = 93$ provides a model order which performs better across all datasets and has lower number of parameters, thus lesser complex. With this criteria, ARX with model order of [7 93 0] was chosen to be the best performing ARX model while looking into different model order configurations in Table 4.3. To find the performance of the ARX model with different model order configurations, refer to Table A.5 in Appendix. Fig. 4.26 shows the comparison of ARX model response

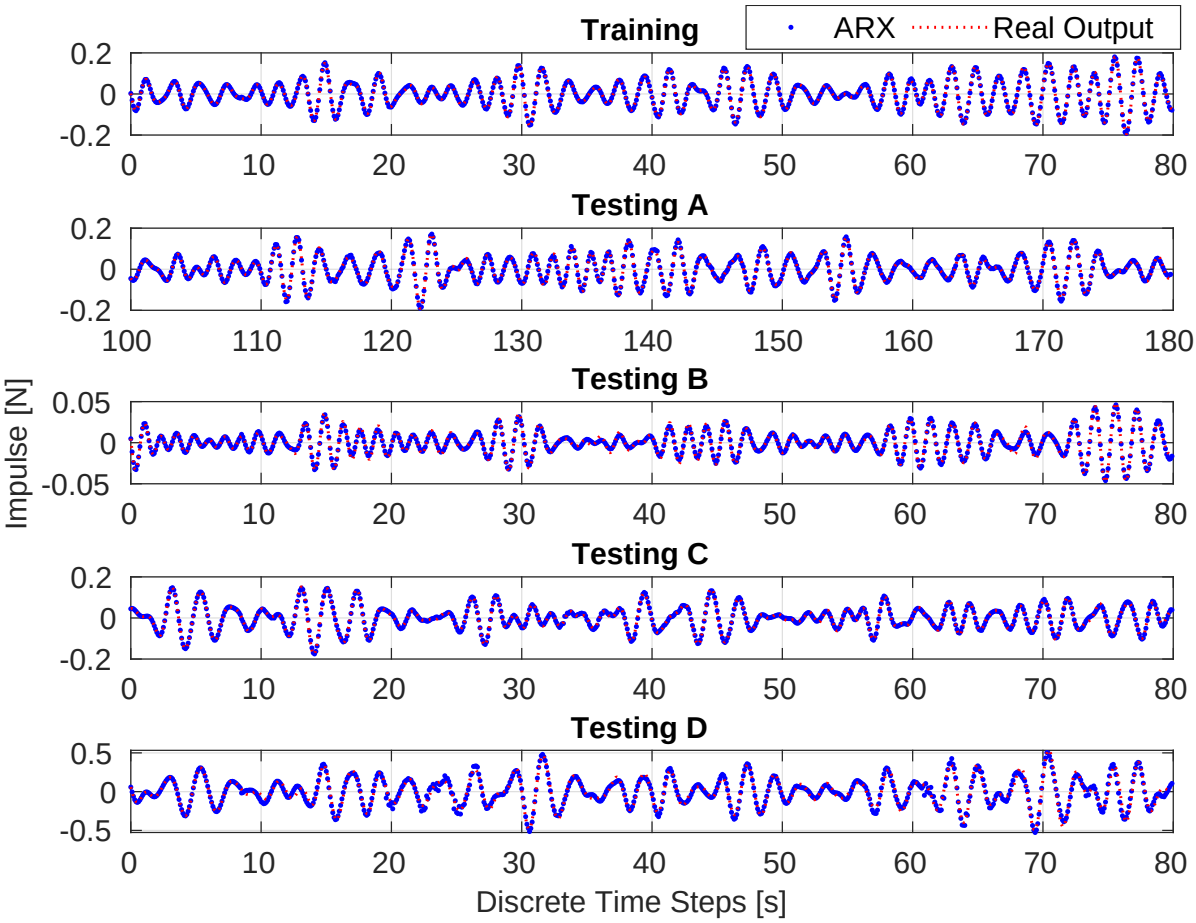


Figure 4.26: Performance of ARX model during training and testing for Inverse NWT

and the actual output for training as well as testing datasets. Here only first 800 data points are shown.

ARX	Performance					
	Training Dataset		Testing Datasets			
$[n_y\ n_u\ n_d]$	Loss Function	Fit (%)	Testing A Fit(%)	Testing B Fit(%)	Testing C Fit(%)	Testing D Fit(%)
[7 92 0]	2.069×10^{-5}	91.40	92.32	83.19	89.58	72.31
[7 93 0]	$\mathbf{2.159} \times 10^{-5}$	**92.85**	**92.25**	**84.18**	**90.04**	**72.72**
[7 94 0]	2.323×10^{-5}	92.63	90.18	82.38	89.83	75.82
[7 95 0]	2.518×10^{-5}	92.48	90.10	82.27	89.83	75.95
[7 100 0]	3.585×10^{-5}	91.46	79.34	78.15	85.63	71.72
[7 120 0]	5.522×10^{-5}	79.27	88.39	76.30	85.46	64.86
[10 93 0]	2.259×10^{-5}	87.33	77.77	74.17	81.19	67.84

Table 4.3: Comparsion of goodness of fit values for various ARX model orders for Inverse NWT problem

4.3.2 Output Error (OE) Model

The OE method deals with minimizing a nonlinear least square loss function, that is based on the output error ϵ_{OE}, using a nonlinear programming algorithm e.g. Levenberg-Marquardt (LM). This error, ϵ_{OE} is between the measured output y of the system and the output $\hat{y}_M$ of the transfer function model (see section 2.4.5, for further details). In this case the best model order which performed well during training as well as testing was found to be [96 3 0] with the nonlinear least square and LM algorithms used over a maximum of 30 iterations. Here the first term in the model order represents the number of the past inputs required, followed by the model outputs and finally the input delay parameter. The results obtained are shown in Table 4.4. To check the performance of OE model for other model order configurations, see Table A.6 in Appendix.

4.3.3 Hammerstein-Wiener (HW) Model

The dynamics of the required impulse to the given surface wave elevation was represented by a linear transfer function and the associated nonlinearities were added using static nonlinear functions placed at the input and the output of the linear system. For an estimation of the linear transfer function, various pole-zero configurations were explored. The effect on the loss function and the fitness percentage for the various pole-zero and delay configurations, $[n_b\ n_f\ n_k]$ to the linear dynamic subsystem of the HW model can be seen in appendix Table A.7, where n_b is the number of zeros plus 1, n_f is the number of poles of polynomials B and F respectively used in the linear transfer function block, and n_d is the input delay (see section 2.4.6). The optimal model order configuration was chosen after examining the maximum fitness values obtained during both training and validation for different model orders and was found to be [141 6 0] with a nonlinear least square loss function, Levenberg-Marquardt as optimization algorithm for a maximum of 50 iterations. Moreover, 8 and 11 units of piecewise linear functions were used as the static input and output functions to the linear system block respectively to obtain this best performing HW model.

Models	Model Order	Training perform(%)	Testing A Perform(%)	Testing B Perform(%)	Testing C Perform(%)	Testing D Perform(%)
ARX	[7 93 0]	92.85	92.25	84.18	90.14	72.72
OE	[96 3 0]	89.97	89.81	80.03	86.46	65.85
HW	[141 6 0]	86.73	90.32	80.01	87.03	72.84

Table 4.4: Model performance comparison on available data in case of Inverse NWT problem

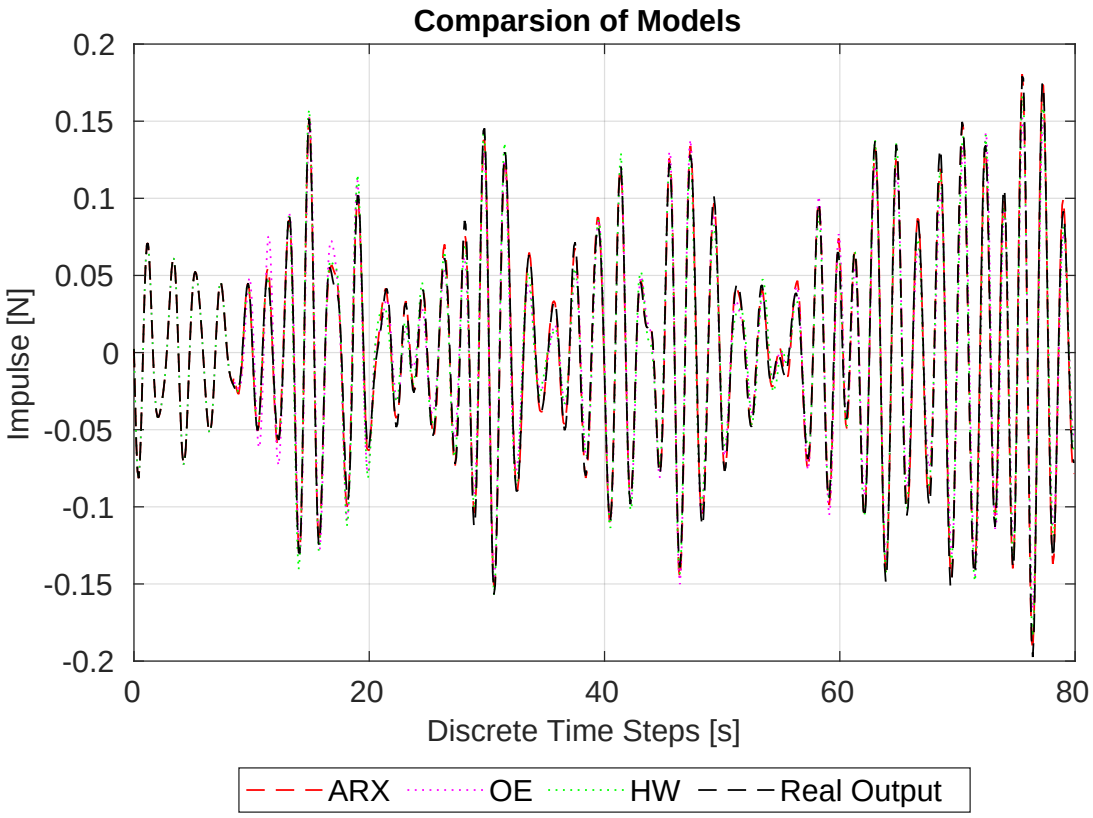


Figure 4.27: Model performance comparison during training for inverse NWT .

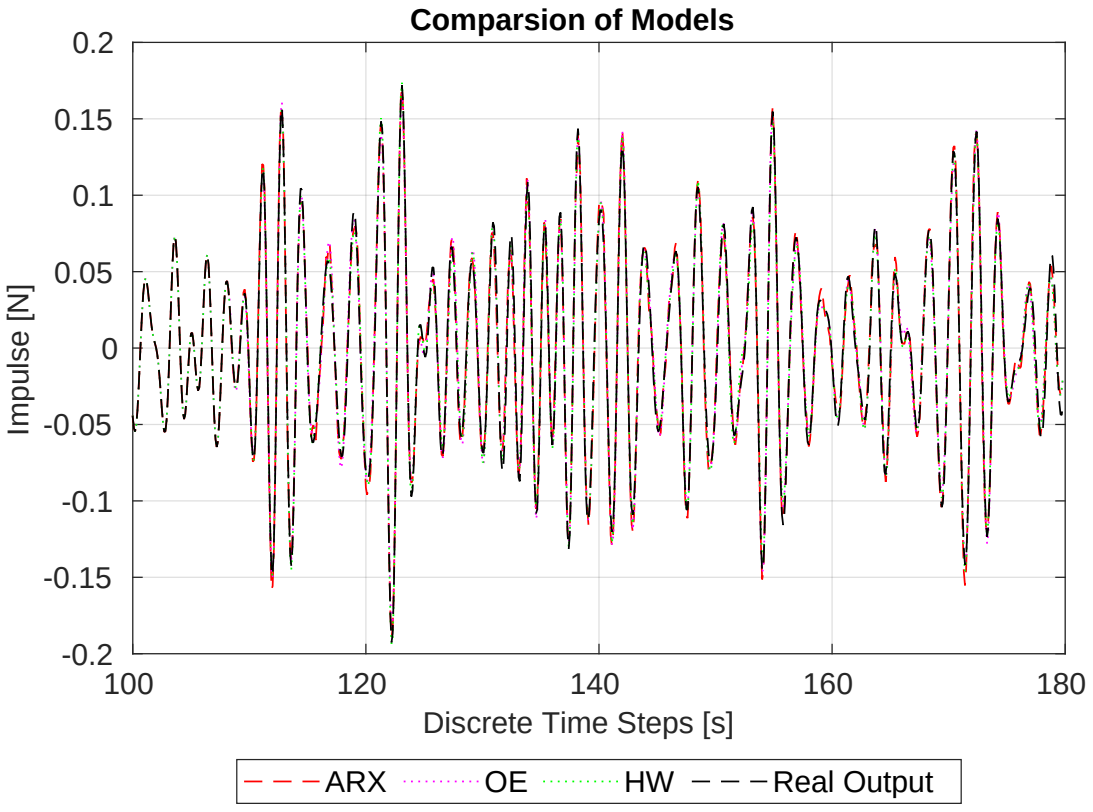


Figure 4.28: Model performance comparison during testing Dataset A for inverse NWT.

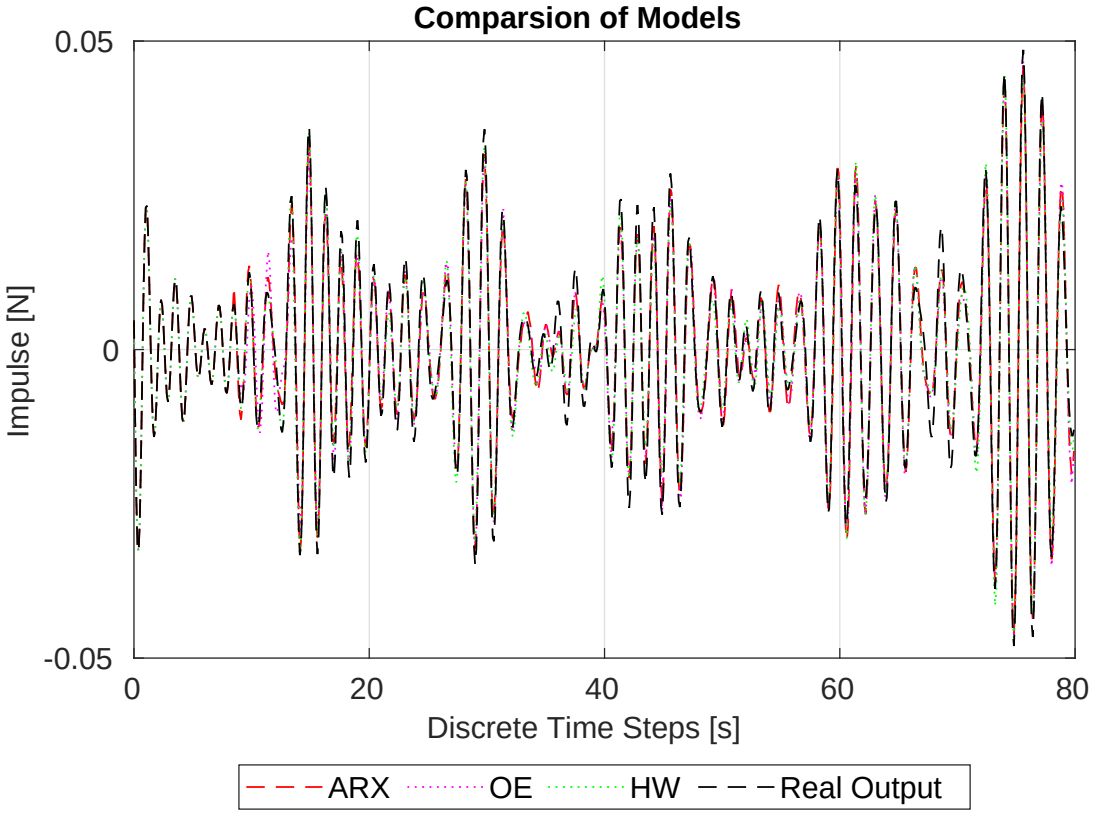


Figure 4.29: Model performance comparison during testing Dataset B for inverse NWT.

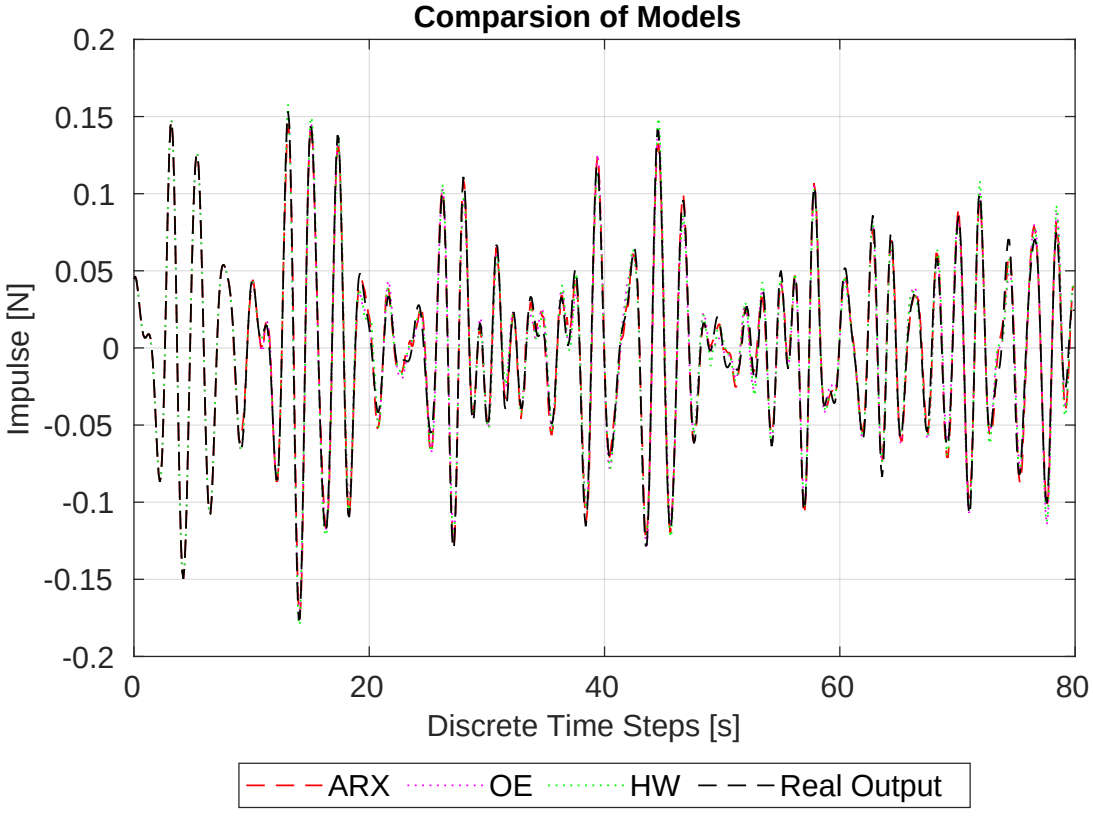


Figure 4.30: Model performance comparison during testing Dataset C for inverse NWT.

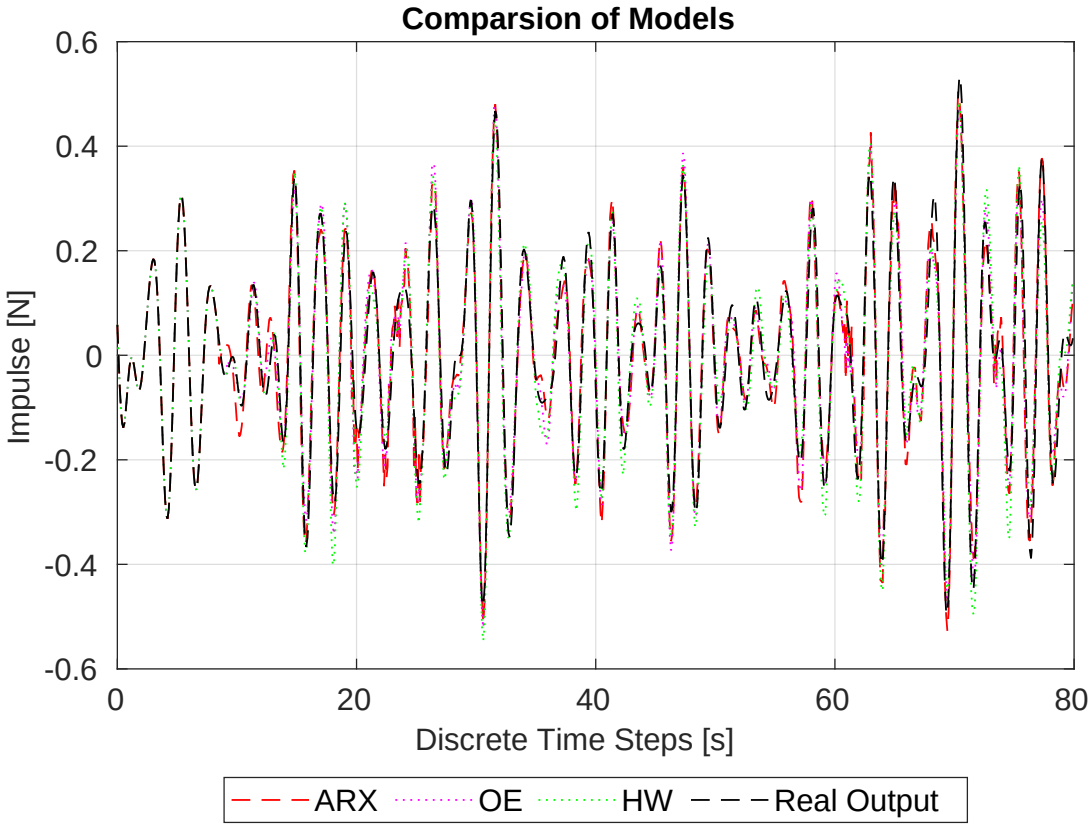


Figure 4.31: Model performance comparison during testing Dataset D for inverse NWT.

Only the first 800 data points are shown in the above figures in order to get a more detailed view of model performances. Attempt to use NARX Neural Network in this case was also explored. Performance of NARX Neural Network was found descent during training but deteriorated during validation (see Table A.8 in Appendix). Therefore, additional efforts are required in fine tuning of its hyper-parameters in future. All of the models used, performed comparably well during training and testing. But it was again the ARX, which captured most of the dynamics of the system with the highest accuracy. Based on the time duration required to train each model, from Fig 4.17, training duration of ARX was almost 4 times shorter than Output Error model and 10 times shorter than Hammerstein Wiener model. In terms of model complexity, Output Error model consists of lesser number of model parameters, 99, followed by ARX, 100 and finally Hammerstein Wiener model, 147. All the models except ARX employed are nonlinear in nature, therefore more complex than ARX.

Moreover, in both cases, direct and inverse NWT problems, all the models were trained over for just a single training example of medium sized amplitude and were tested against all other datasets with different amplitudes. It was exciting to see the validation performance of all the models especially ARX was satisfactory when subjected to inputs with amplitudes varying by a manifold of 10 times the amplitude of the training dataset.

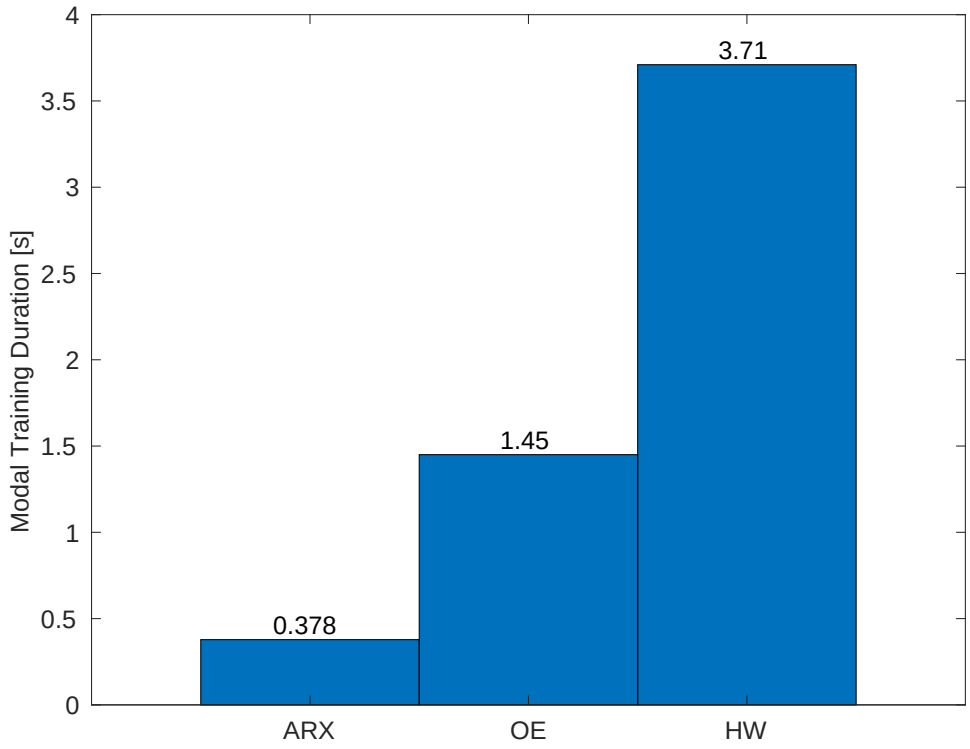


Figure 4.32: Model training duration comparison for Inverse NWT problem

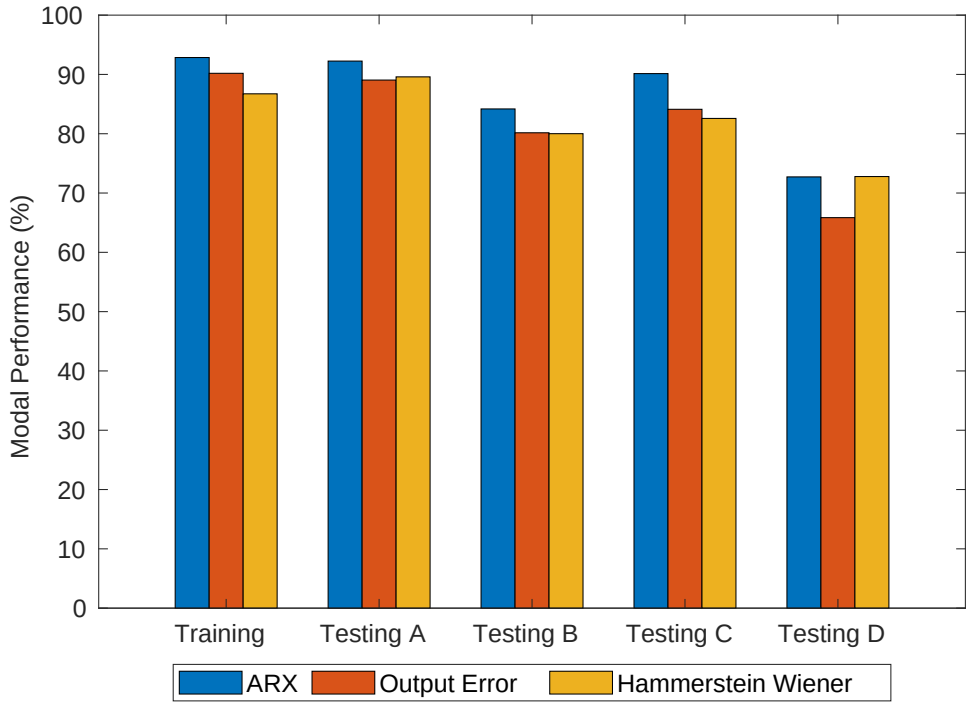


Figure 4.33: Model performance comparison over training and testing datasets for Inverse NWT problem

5 General Conclusions and Perspective

The main objective of this thesis is to improve the generation of waves in numerical wave tanks (NWTs) by using data-driven system identification techniques including machine learning and deep learning and create models mapping the relationship between the surface wave elevation in NWT to the required impulse source function. Another objective is to implement a NWT in a CFD solver, OpenFOAM with a impulse source wave maker and carry out multi-phase simulations and generate free surface waves which are then utilized for training and validating identified models. The first chapter states this objective and the motivation behind this work undertaken.

The second chapter introduces the system identification techniques, optimization algorithms and use of various classical and modern state-of-the-art black-box modelling approaches such as ARX, Hammerstein-Wiener (HW), NARX and Output-Error Models. Hybrid models involving use Deep learning for system identification such as NARX Neural Network is also discussed. In the last part of this chapter, a brief overview of OpenFOAM, Numerical wave tank and the theory behind its implementation and other necessary tools and set-up information required to carry out multiphase simulations in opensource CFD solver is presented to obtain gravity waves in NWT.

In the third chapter a practical application of data-driven system identification to a mass-spring-damper system is introduced. The ability of both ARX and Narx Neural Network to model the dynamics of such a system is demonstrated. Moreover, the inverse of this forward problem, and the relative accuracy, along the time duration to train such models is clearly presented. It is found that ARX is much faster (almost 25 times) than NARX Neural Network. Both the model structures bears the same level of accuracy over the training and validation data.

In the fourth chapter, an open source CFD solver, OpenFOAM, has been used to implement a numerical wave tank. The study is based on the interFoamsrc solver, which is a modified interFoam solver for incompressible multiphase flow problems. The solver uses the finite volume method for the spatial discretization of equations and applies the VOF approach for the free surface modeling. In the next step, after acquisition of data from the CFD solver, system identification techniques are used and comparative study of four different black box models namely. ARX, NARX Neural Network, Hammerstein-Wiener and output are shown, each model is attempting to map the nonlinear relationship between the surface wave elevation at some probe length in NWT to the applied impulse source excitation. Results obtained during training and validation from all these models are compared against each other based on the model complexity, computational cost and accuracy. All models performed relatively well in terms of accuracy but ARX stand out as the most favourable model owing to its highest accuracy, minimum model training duration and relatively lower complexity.

At the end of this chapter, an inverse problem of this forward NWT problem is pre-

sented. Such a problem is not solvable in OpenFOAM, which is still lacking the ability to directly correlate the required impulse source wavemaker amplitude to the given surface wave elevation. The current method of generating waves in NWT is an inefficient approach based on hit and trial method or iterative calibrations which is computationally intensive, highly time consuming and yet uncertain to produce the desired free surface waves in NWT. To counter this problem, system identification methods such as ARX, Hammerstein-Wiener and OE models are put forth. The results obtained are satisfactory and are validated over three different datasets. Based on the criteria of maximum accuracy with minimum model complexity, it is shown that ARX model is efficient to establish a relationship between the required excitation to the desired surface wave elevation in NWT. It was demonstrated that ARX, if trained for an input of medium sized amplitude can validate satisfactorily over the data having amplitude variation of almost 10 times the amplitude of the training dataset both in forward and inverse NWT problems.

In general, all identified models were able to simulate the response of the dynamic systems well when subjected to the same input conditions during the forward problems as well as performed satisfactory for inverse problems too but such models may not be able to extrapolate well when subjected to input condition outside of the frequency and amplitude ranges they are trained of achieving global minimum which is a limitation of black box modelling. Additionally, the advantages of using these models over the classical Deep learning is such that these models do not require large number of training examples and model gets trained within a very short duration of time. Unlike traditional Neural networks e.g. Artificial Neural Network (ANN), which can act as universal functional approximators, but the performance of such models is highly dependent on the amount of the available data to train these models. Moreover, these black box models are very complex in nature and can incur much higher computational costs.

The following future works and perspectives can be mentioned

- Use of classical neural network approach e.g. ANN is beyond the scope of this study due to the limited amount of available data with restricted computational resources and time limitation. It will be noteworthy to check the simulation performance of such models against other discrete model structures discussed earlier.
- Recent advance in deep learning called Physics-Informed Neural Networks (PINNs), for learning unknown dynamics and constitutive relations can be another area to explore [50]. Instead of replacing known physics with purely data-driven deep neural networks in a wholesale fashion, the idea is to blend physical constraints into the workflow in an attempt to combine the best of both worlds. One of the advantages of this method is its ability to train models over limited datasets which should be investigated in future studies.

Reference

[1] MATLAB. *What are Nonlinear ARX models*. Mathsworks.Inc, 2021. [Online; accessed 21-Dec-2021].

[2] W. Hongwei and S. Xiaodong . Nonlinear system identification based on narx network. In *2015 10th Asian Control Conference (ASCC)*, volume 1, pages 1–6. IEEE, 2015.

[3] J. Davidson, M. Cathelain, L. Guillmet, T. L. Huec, and J. Ringwood. Implementation of an openfoam numerical wave tank for wave energy experiments. In *11th European Wave and Tidal Energy Conference (EWTEC)*, 2015.

[4] J. Choi and S. B. Yoon. *Numerical simulations using momentum source wave-maker applied to RANS equation model*, volume 56. Coastal Engineering, 2009.

[5] P. Schmitt, C. Windt, J. Davidson, J. V. Ringwood, and T. Whittaker. *The Efficient Application of an Impulse Source Wavemaker to CFD Simulations*. Journal of Marine Science and Engineering, 2019.

[6] L. Ljung. *System Identification: Theory for the User*. 2nd ed, Theory for the User Englewood Cliffs, NJ, USA, 1999.

[7] O. Nelles. *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models*. New York, NY, USA: Springer, 2001.

[8] G. Golub C. Van Loan. *Matrix Computations*. 4th ed. Baltimore, MD, USA, The Johns Hopkins Univ. Press,, 2012.

[9] K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. *Learning phrase representations using rnn encoder-decoder for statistical machine translation*. arXiv preprint, 2014.

[10] D. P. Kingma J. L. Ba. *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION*. arXiv preprint, 2017.

[11] K. Madsen H.B Nielsen O. Tingleff. *Methods for Non-Linear Least Squares Problems*. Technical University of Denmark, 2004.

[12] J. Richalet, S.Abu. el Ata-Doss, and A. Coïc. *Global Identification and Optimal Input Design*, volume 24r. IFAC Proceedings Volumes, 1991.

[13] Himmelblau and M. David. *Applied nonlinear programming*. McGraw-Hill Book Company, New York, 1972.

[14] J. E. Dennis and B. Schnabel. Robert. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Society for Industrial and Applied Mathematics; Illustrated edition, 1987.

[15] S. Boyd and L. Vandenberghe. *Convex Optimization Equations*. Cambridge University Press, 2004.

[16] Y. Chetouani. *Using ARX and NARX approaches for modeling and prediction of the process behavior: application to a reactor-exchanger*, volume 3. Asia-Pacific Journal of Chemical Engineering, 2008.

[17] D.E. Rivera and S.V. Gaikwad. *Systematic techniques for determining modelling requirements for SISO and MIMO feedback control*, volume 5. Journal of Process Control, 1995.

[18] X. Liu and Y. Zhu. *ARX Model Estimation of Multivariable Errors-in-Variables Systems*, volume 51. IFAC-PapersOnLine, 2018.

[19] M.L. Fravolini, A. Ficola, and M. La Cava. *Optimal operation of the leavening process for a bread-making industrial plant*, volume 60. Journal of Food Engineering, 2003.

[20] H.U. Frausto, J. G. Pieters, and J. M. Deltour. *Modelling Greenhouse Temperature by means of Auto Regressive Models*, volume 84. Biosystems Engineering, 2003.

[21] J. V. Ringwood, P. C. Austin, and W. Monteith. *Development of ARX model based off-line FDD technique for energy efficient buildings*, volume 22. Renewable Energy, 2001.

[22] H. Yoshida and S. Kumar. *Forecasting weekly electricity consumption*, volume 15. Energy Economics, 1993.

[23] L. Ljung and B. Wahlberg. *Asymptotic properties of the least-squares method for estimating transfer functions and disturbance spectra*, volume 24. Advances in Applied Probability, 1992.

[24] M. Galrinho, N. Everitt, and H.Hjalmarsson. *ARX modeling of unstable linear systems*, volume 75. Automatica, 2017.

[25] S. A. Billings. *NARMAX Methods in the Time, Frequency,and Spatio-Temporal Domains*. John Wiley Sons, Ltd., West Sussex, United Kingdom,, 2013.

[26] A. Leva and L.Piroddi. *NARX-based technique for the modelling of magneto-rheological damping devices*, volume 11. Smart Materials and Structures, 2002.

[27] A. Rahrooh and S. Shepard. *Identification of nonlinear systems using NARMAX model*, volume 79. Nonlinear Analysis: Theory, Methods Applications, 2009.

[28] SA. Billings and H. L. Wei. *The wavelet-NARMAX representation: A hybrid model structure combining polynomial models with multiresolution wavelet decompositions*, volume 36. International Journal of Systems Science, 2005.

[29] W. Zhang, J. Zhu, and D. Gu. *Identification of robotic systems with hysteresis using Nonlinear AutoRegressive eXogenous input models*, volume 14. International Journal of Advanced Robotic Systems, 2017.

[30] J. V. Ringwood, J. Davidson, and S. Giorgi. *Identifying Models Using Recorded Data.* Numerical Modelling of Wave Energy Converters, 2016.

[31] MATLAB. *What are Hammerstein-Wiener models?* Mathsworks.Inc, 2021.

[32] J.L.Figueroa, S.I.Biagiola, and O.E.Agamennoni. *An approach for identification of uncertain Wiener systems*, volume 48. Mathematical and Computer Modelling, 2008.

[33] J. W. Wingerden, and M. Verhaegen. Closed-loop subspace identification of hammerstein-wiener models. In *48th IEEE conference on decision and control and 28th Chinese control conference - Shanghai, China*, pages 3637–3642, 2009.

[34] L. Ljung, C. Andersson, K. Tiels, and T. B. Schön. *Deep Learning and System Identification*, volume 53. IFAC-PapersOnLine, 2020.

[35] Q. Liu, W. Chen, H. Hu, and Q. Zhu. *An Optimal NARX Neural Network Identification Model for a Magnetorheological Damper With Force-Distortion Behavior*, volume 7. Frontiers in Materials, 2020.

[36] R. W.K. Chan, J. Yuen, E. Lee, and M. Arashpour. *Application of Nonlinear-Autoregressive-Exogenous model to predict the hysteretic behaviour of passive control systems*, volume 85. Engineering Structures, 2015.

[37] C. Andersson, A. H. Ribeiro, K. Tiels, N. Wahlstrom, and T. B.Schon. *Deep Convolutional Networks in System Identification*, volume 2019-December. Proceedings of the IEEE Conference on Decision and Control, 2019.

[38] J. M. Nóbrega and H. Jasak. *OpenFOAM: Selected Papers of the 11th Workshop*, volume 11. Springer International Publishing, 2019.

[39] J. Ferziger and M. Peric. *Computational methods for fluid dynamics.* 3rd edn. Springer, Berlin, 2002.

[40] C. W. Hirt and B. D. Nichols. *Volume of fluid (VOF) method for the dynamics of free boundaries*, volume 39. Journal of Computational Physics, 1981.

[41] A. M. Miquel, A. Kamath, M. A.Chella, R. Archetti, and H. Bihs. *Analysis of Different Methods for Wave Generation and Absorption in a CFD-Based Numerical Wave Tank*, volume 6. Journal of Marine Science and Engineering, 2018.

[42] B. Bouali and S. Larbi. *Contribution to the Geometry Optimization of an Oscillating Water Column Wave Energy Converter*, volume 36. Energy Procedia, 2013.

[43] Du. Qingjie, Y. C. Dennis, and Leung. '2d numerical simulation of ocean waves. In *World Renewable Energy Congress, Sweden*, 2011.

[44] G. Chawla, J.T. Kirby, and A. Sinha. *Generation of waves in Boussinesq models using a source function method*, volume 36. Coastal Engineering, 1999.

[45] A. Chawla and J.T. Kirby. *A source function method for generation of waves on currents in Boussinesq models*, volume 22. Applied Ocean Research, 2000.

[46] J. Orszaghova, A.G.L. Borthwick, and P.H. Taylor. *From the paddle to the beach - A Boussinesq shallow water numerical wave tank based on Madsen and Sørensen's equations*, volume 2. Journal of Computational Physics, 2012.

[47] C.M. Dong and C. J. Huang. *Generation and propagation of water waves in a two-dimensional numerical viscous wave flume*, volume 3. Journal of Waterway, Port, Coastal and Ocean Engineering, 2004.

[48] P. Schmitt, C. Windt, J. Davidson, J. V. Ringwood, and T. Whittaker. *Beyond VoF: alternative OpenFOAM solvers for numerical wave tanks*, volume 6. Journal of Ocean Engineering and Marine Energy volume, 2020.

[49] P. Schmitt and B. Elsaesser. A review of wave makers for 3d numerical simulations. In *Proceedings of the 6th International Conference on Computational Methods in Marine Engineering, Rome, Italy*, volume 56, page 437–446, 2015.

[50] R. Tipireddy, P. Perdikaris, P. Stinis, and A. Tartakovsky. *A comparative study of physics-informed neural network models for learning unknown dynamics and constitutive relations.* arXiv:1904.04058, 2019.

A Additional Model Performance Data Tables

A.1 ARX performance vs model orders for Inverse MSD problem

ARX	Performance			
	Training Dataset		Testing Datasets	
$[n_y\ n_u\ n_d]$	Loss Function	Fit (%)	Testing Fit I(%)	Testing Fit II (%)
[1 7 2]	5.544×10^{-1}	52.12	−75.07	69.87
[0 12 3]	2.514×10^{-1}	66.72	−154.35	53.38
[0 7 0]	8.089×10^{-4}	98.11	89.52	99.35
[3 11 0]	2.643×10^{-4}	98.80	95.65	99.21
[0 8 0]	$\mathbf{3.013} \times 10^{-4}$	**98.85**	**95.27**	**99.14**
[0 15 0]	9.047×10^{-5}	99.36	87.22	98.47
[2 1 1]	19.05×10^{-1}	2.058	−0.09	−0.28
[3 4 1]	4.182×10^{-1}	27.58	−43.19	22.18
[3 4 0]	2.248×10^{-1}	95.01	85.84	93.53
[1 3 0]	4.691×10^{-1}	59.84	26.19	40.11
[3 9 0]	1.400×10^{-3}	98.19	90.99	99.05
[1 9 0]	2.966×10^{-4}	98.88	95.00	99.10
[0 9 0]	2.617×10^{-4}	98.93	93.17	93.17
[0 70 0]	5.395×10^{-9}	99.99	14.99	99.98
[1 70 0]	5.517×10^{-9}	99.99	14.04	99.98
[1 11 0]	1.920×10^{-4}	99.08	93.65	98.86
[0 11 0]	1.908×10^{-4}	99.08	93.47	98.82

Table A.1: Goodness of fit values for various ARX Model orders for Inverse MSD problem

A.2 ARX performance vs model orders for Direct NWT problem

ARX	Performance				
	Training Dataset		Testing Datasets		
$[n_y\ n_u\ n_d]$	Loss Function	Fit (%)	Dataset I Fit (%)	Dataset II Fit (%)	Dataset III Fit (%)
[1 7 2]	2.518×10^{-5}	28.58	−0.70	21.23	1.5
[1 3 0]	5.228×10^{-5}	1.48	−1.35	1.37	0.64
[2 103 94]	$\mathbf{1.057} \times 10^{-7}$	**86.83**	**83.50**	**91.27**	**70.05**
[2 150 94]	4.798×10^{-6}	87.65	22.03	74.04	69.10
[3 5 4]	4.497×10^{-5}	0.25	−0.28	0.15	−0.76
[3 12 17]	4.433×10^{-5}	0.14	−1.36	−1.19	−0.90
[3 103 94]	2.545×10^{-7}	87.66	81.20	90.69	68.44
[3 103 96]	2.531×10^{-7}	87.71	81.10	90.42	68.96
[3 96 96]	3.364×10^{-7}	87.42	79.33	89.30	68.22
[3 97 100]	3.679×10^{-7}	87.36	80.46	89.56	68.07
[3 129 96]	1.377×10^{-7}	87.56	80.61	89.52	69.72
[4 103 94]	1.377×10^{-7}	87.56	80.61	89.52	67.64
[5 23 39]	5.456×10^{-7}	21.00	−5.87	7.88	−22.49
[6 95 80]	3.796×10^{-8}	85.19	76.09	87.71	64.97
[9 107 100]	3.810×10^{-9}	87.71	76.38	88.76	68.93
[11 17 29]	6.825×10^{-5}	25.66	17.49	23.75	−7.90
[15 103 96]	3.176×10^{-9}	87.68	81.20	90.62	68.86

Table A.2: Performance of ARX with different model order configurations for NWT direct problem

A.3 NARX performance comparison for Direct NWT problem

NARX	Performance				
	Training Dataset		Testing Datasets		
idTreePartition	Loss Function	Fit (%)	Dataset I Fit (%)	Dataset II Fit (%)	Dataset III Fit (%)
1 *unit*	3.318×10^{-8}	87.02	83.93	91.21	54.23
2 *units*	3.396×10^{-8}	87.02	83.93	91.62	54.23
3 *units*	2.377×10^{-8}	78.98	75.20	81.57	54.22
6 *units*	2.367×10^{-8}	78.99	75.21	81.57	54.22

Table A.3: Effect of changing number of units of nonlinear *idTreePartition* function on the performance of the best performing ARX model order **[2 103 94]** for direct NWT problem

A.4 HW model performance for Direct NWT problem

HW	Performance				
	Training Dataset		Testing Datasets		
$[n_b\ n_f\ n_k]$	Loss Function	Fit(%)	Dataset I Fit (%)	Dataset II Fit (%	Dataset III Fit(%))
[5 23 39]	1.166×10^{-5}	53.49	5.95	61.73	0.17
[9 5 0]	4.875×10^{-5}	4.92	-274.31	-34.01	13.44
[11 21 67]	4.170×10^{-6}	72.19	46.60	60.49	15.88
[11 17 29]	1.479×10^{-5}	47.62	34.82	55.04	7.29
[14 43 54]	1.509×10^{-5}	47.08	36.80	53.76	16.74
[54 32 22]	1.006×10^{-5}	56.80	9.5401	57.23	15.55
[76 13 98]	9.995×10^{-7}	86.38	75.17	87.56	70.74
[102 2 96]	8.477×10^{-7}	87.46	18.14	86.68	71.08
[103 3 94]	1.059×10^{-6}	87.17	77.86	89.32	70.90
[103 3 96]	$\mathbf{8.869} \times 10^{-7}$	**92.73**	**81.41**	**90.69**	**71.08**

Table A.4: Comparison of goodness of fit values for various Hammerstein Wiener model orders for Direct NWT problem

A.5 ARX performance vs model orders for Inverse NWT problem

ARX	Performance					
	Training Dataset		Testing Datasets			
$[n_y\ n_u\ n_d]$	Loss Function	Fit (%)	Testing A Fit(%)	Testing B Fit(%)	Testing C Fit(%)	Testing D Fit(%)
[2 23 0]	1.186×10^{-3}	38.24	30.37	42.55	8.62	−44.17
[6 103 0]	3.091×10^{-5}	74.27	81.02	77.16	82.68	68.36
[7 90 0]	3.091×10^{-5}	90.67	92.32	79.34	89.30	68.36
[7 91 0]	2.663×10^{-5}	91.74	92.36	83.18	89.67	70.86
[7 92 0]	3.117×10^{-5}	91.40	92.32	83.19	89.58	72.31
[7 93 0]	$\mathbf{2.159} \times 10^{-5}$	**92.85**	**92.25**	**84.18**	**90.04**	**72.72**
[7 94 0]	2.323×10^{-5}	92.63	90.18	82.38	89.83	75.82
[7 95 0]	2.518×10^{-5}	92.48	90.10	82.27	89.83	75.95
[7 100 0]	7.391×10^{-6}	91.46	79.34	78.15	85.63	71.72
[7 107 0]	5.604×10^{-5}	91.33	79.34	77.67	84.96	49.58
[7 114 0]	6.871×10^{-6}	90.55	79.34	72.24	83.04	65.74
[7 120 0]	5.522×10^{-5}	79.27	88.39	76.30	85.46	64.86
[7 128 0]	6.144×10^{-5}	91.33	79.34	77.67	84.96	49.58
[7 130 0]	6.218×10^{-5}	79.19	85.55	76.05	80.15	58.04
[7 200 0]	3.124×10^{-5}	92.48	79.34	77.92	80.76	54.78
[8 93 0]	2.948×10^{-5}	72.69	81.94	70.46	78.80	68.81
[9 93 0]	3.862×10^{-5}	65.89	73.82	61.15	71.72	67.53
[10 93 0]	2.259×10^{-5}	87.33	77.77	74.17	81.19	67.84
[11 93 0]	2.051×10^{-5}	82.44	78.07	75.90	73.77	63.81
[15 93 0]	1.0×10^{4}	−0.72	0.002	−1.46	−0.56	−1.02
[20 93 0]	1.042×10^{-4}	81.39	83.14	78.54	75.91	56.91
[25 93 0]	1.544×10^{-3}	81.79	74.94	79.27	64.31	46.97

Table A.5: Comparison of goodness of fit values for various ARX model orders for Inverse NWT problem

A.6 OE and HW model performances for Inverse NWT problem

OE	Performance					
	Training Dataset		Testing Datasets			
$[n_b\ n_f\ n_d]$	Loss Function	Fit (%)	Testing A Fit(%)	Testing B Fit(%)	Testing C Fit(%)	Testing D Fit(%)
[67 3 0]	9.054×10^{-5}	87.69	87.14	76.66	75.82	37.41
[74 3 0]	8.362×10^{-5}	88.38	83.75	76.84	78.31	30.81
[75 3 0]	1.113×10^{-4}	87.18	80.42	75.61	77.55	16.93
[91 3 0]	7.214×10^{-5}	89.52	89.74	79.59	86.53	60.44
[96 3 0]	$\mathbf{4.696} \times 10^{-5}$	**89.97**	**89.81**	**80.03**	**86.46**	**65.85**
[98 3 0]	7.841×10^{-5}	88.32	84.88	77.39	82.62	33.48
[99 3 0]	4.985×10^{-5}	90.19	89.05	80.16	84.12	55.31
[102 3 0]	1.527×10^{-4}	86.54	76.91	72.08	73.79	−6.47

Table A.6: Comparison of goodness of fit values for various Output Error model orders for Inverse NWT problem

HW	Performance					
	Training Dataset		Testing Datasets			
$[n_b\ n_f\ n_d]$	Loss Function	Fit (%)	Testing A Fit(%)	Testing B Fit(%)	Testing C Fit(%)	Testing D Fit(%)
[62 6 0]	2.421×10^{-4}	80.88	87.90	77.62	81.19	57.67
[75 6 0]	2.389×10^{-4}	87.96	88.08	**7**6.67	82.66	56.65
[76 6 0]	2.309×10^{-4}	88.17	88.46	78.20	82.30	59.04
[82 6 0]	2.389×10^{-4}	88.42	87.95	76.52	81.60	57.71
[89 6 0]	2.381×10^{-4}	88.46	88.02	76.07	81.17	52.47
[92 6 0]	2.448×10^{-4}	88.36	88.63	78.15	82.54	59.24
[95 6 0]	2.443×10^{-4}	87.93	88.75	78.20	81.32	49.57
[99 6 0]	2.266×10^{-4}	84.07	88.03	75.00	80.68	42.25
[106 6 0]	2.197×10^{-4}	88.54	88.14	74.65	82.90	62.50
[107 7 0]	2.468×10^{-4}	88.91	88.56	79.80	82.62	60.59
[108 7 0]	2.261×10^{-4}	88.55	89.11	79.24	83.85	69.84
[141 6 0]	$\mathbf{2.215} \times 10^{-4}$	**86.73**	**90.32**	**80.01**	**87.03**	**72.84**
[147 7 0]	2.193×10^{-4}	84.67	90.89	80.08	86.39	71.69

Table A.7: Comparison of goodness of fit values for various Hammerstein Wiener model orders for Inverse NWT problem

A.7 NARX Neural Network performance for Inverse NWT problem

NARX NN	Performance					
	Feedforwardnet		Testing Datasets			
$[n_y\ n_u\ n_d]$	Neurons per layer	Activation func. used	Training Fit(%)	Testing B Fit(%)	Testing C Fit(%)	Testing D Fit(%)
[2 98 0]	[12 8]	[*linear linear*]	73.09	40.24	66.59	64.61
[2 103 0]	[12 8]	[*linear linear*]	71.02	48.05	69.48	65.6
[2 103 0]	[2 2]	[*linear linear*]	71.33	37.89	66.78	50.32
[2 103 0]	[2 4]	[*linear linear*]	65.77	36.77	64.73	62.08
[2 103 0]	[3 2]	[*linear linear*]	**69.18**	**51.42**	**64.52**	**75.79**
[2 103 0]	[3 4]	[*linear linear*]	72.97	23.85	67.32	22.60
[2 103 0]	[12 2]	[*linear linear*]	69.41	45.19	66.03	65.57
[2 98 0]	[12 3]	[*linear linear*]	73.19	44.51	62.67	69.08
[2 98 0]	[12 4]	[*linear linear*]	74.48	16.62	37.44	36.36
[2 98 0]	[12 8]	[*linear linear*]	73.09	40.24	64.60	66.59
[3 96 0]	[12 4]	[*linear linear*]	66.40	41.93	42.88	51.58
[2 93 0]	[12 2]	[*tansig tansig*]	79.88	31.04	43.81	35.47
[3 98 0]	[2 2]	[*tansig tansig*]	52.33	31.21	2.67	73.84
[3 98 0]	[3 2]	[*tansig tansig*]	53.55	46.24	16.06	61.53
[3 98 0]	[3 3]	[*tansig tansig*]	50.01	51.96	30.27	18.86
[3 96 0]	[12 2]	[*tansig tansig*]	57.29	31.38	32.41	39.88

Table A.8: Comparison of goodness of fit values for various NARX Neural Network for Inverse NWT problem

B MATLAB And Python Codes

B.1 NARX Neural Network Code

```
##########################################################################
########## NARX-Neural Network for Spring Mass Damper System ##########
##########################################################################
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import RMSprop
from sklearn.metrics import, mean_absolute_error,r2_score
from matplotlib import pyplot as plt
plt.rcParams.update({'font.size': 18})
##########################################################################
# Function to determine goodness of fit
##########################################################################
def fit(targets, predictions):
    from numpy import linalg as LA
    targets=targets.reshape(targets.shape[0],1)
    predictions = predictions.reshape(predictions.shape[0],1)
    return (1-LA.norm(targets - predictions)/LA.norm
                        ....(targets-targets.mean()))*100
##########################################################################
# Function to simulate and plot the results
##########################################################################
def simulate(X_test,Y_test):
    predict_time=Y_test.shape[0]
    Y_predicted_offline=np.zeros(shape=(predict_time,1))
    X_predict_offline = X_test[0].reshape(1,pastoutput+pastinput);
    Y_predicted_offline[0] = Y_test[0]
    y_predict_tmp = Y_predicted_offline[0]
    Y_past =X_test[0,pastinput:pastinput+pastoutput].reshape(1,pastoutput)
    for i in range(0,predict_time-1):
        Y_past[:,0:pastoutput-1]=Y_past[:,1:] #Rolling window by 1
        Y_past[:,-1]=y_predict_tmp
        X_predict_offline[:,0:pastinput]=X_test[i+1,0:pastinput]
        X_predict_offline[:,pastinput:pastinput+pastoutput]=Y_past
        y_predict_tmp= model.predict(X_predict_offline)
        Y_predicted_offline[i+1]=y_predict_tmp
```

```
    Y_test = Y_test.reshape(Y_test.shape[0],1)
    import matplotlib.pyplot as plt
    import math
    Fit = fit(Y_test[:predict_time],Y_predicted_offline[:predict_time])
    if Fit == -math.inf:
        return print("Bad Fit -Inf")
    else:
        plt.plot(Y_test[:predict_time],label='Real output')
        plt.plot(Y_predicted_offline[:predict_time], label='NARX_NN')
        plt.xlabel('Discrete time steps')
        plt.ylabel('Force')
        plt.title("Fit :" +str(f'{Fit:.2f}')+"%" )
        plt.legend()
        plt.grid()
    return plt.show()
##############################################################################
# Function to Plot the given data
##############################################################################
def plots(t,x,y):
    plt.figure(figsize=(14,6))
    plt.subplot(2,1,1)
    plt.plot(t,x,label=r'$Input Force$')
    plt.legend();plt.grid()
    plt.ylabel('Excitation')
    plt.subplot(2,1,2)
    plt.plot(t,y,label=r'$Displacement$')
    plt.legend()
    plt.ylabel('Displacement')
    plt.xlabel('Time')
    plt.grid()
    return plt.show()
##############################################################################
#            Load the Data
##############################################################################
data = pd.read_csv('https://raw.githubusercontent.com/Mirirfan11/
                                System-identification/main/MSD.csv')
Time = data['Time'].values
################################## Training ##################################
Input1 = data['Input1'].values  # Excitation
output1 = data['output1'].values    # Displacement
plots(Time,Input1,output1)
################################ Validation ##################################
```

```
Input4 = data['Input4'].values  # Excitation
output4 = data['output4'].values   # Displacement
plots(Time,Input4,output4)
################################ Testing ###################################
Input3 = data['Input3'].values  # Excitation
output3 = data['output3'].values   # Displacement
plots(Time,Input3,output3)

"""System Identification"""

#############################################################################
#        This function formats the input and output data
#############################################################################
def form_data(input_seq, output_seq,pastoutput,pastinput):
    data_len=np.max(input_seq.shape)
    X=np.zeros(shape=(data_len-pastinput,pastinput+pastoutput))
    Y=np.zeros(shape=(data_len-pastinput,))
    for i in range(0,data_len-(pastinput)):
        X[i,0:pastinput]=input_seq[i:i+pastinput,0]
        X[i,pastinput:]=output_seq[i:i+pastoutput,0]
        Y[i]=output_seq[i+pastoutput,0]
    return X,Y
###################### Direct Problem ###################################
pastinput=4;pastoutput=3;
#############################################################################
#                    Create the training data
#############################################################################
input_seq_train = Input1.reshape(Input1.shape[0],1)
output_seq_train = output1.reshape(output1.shape[0],1)
X_train,Y_train = form_data(input_seq_train, output_seq_train,
                                          pastoutput,pastinput)
#############################################################################
#                    Create the validation data
#############################################################################
input_seq_validate = Input4.reshape(Input4.shape[0],1)
output_seq_validate = output4.reshape(output4.shape[0],1)
X_validate,Y_validate =  form_data(input_seq_validate,
                   output_seq_validate, pastoutput,pastinput)
#############################################################################
#                    Create the test data
#############################################################################
input_seq_test = Input1.reshape(Input1.shape[0],1)
```

```
output_seq_test = output1.reshape(output1.shape[0],1)
X_test,Y_test = form_data(input_seq_test, output_seq_test,
                                        pastoutput,pastinput)
##############################################################################
#                 Create the MLP network and train it
##############################################################################
model = Sequential()
model.add(Dense(12,activation='linear',use_bias=False,
                              input_dim=(pastinput+pastoutput)))
model.add(Dense(8,activation='linear'))
model.add(Dense(1))
opt = tf.keras.optimizers.Adam(lr=0.001, decay=1e-6)
model.compile(optimizer=opt, loss='mse')
history=model.fit(X_train, Y_train, epochs=500, batch_size=20,
          validation_data=(X_validate,Y_validate), verbose=2)
##############################################################################
#                  Plot training and validation curves
##############################################################################
loss=history.history['loss']
val_loss=history.history['val_loss']
epochs=range(1,len(loss)+1)
plt.figure()
plt.plot(epochs, loss,'b', label='Training loss')
plt.plot(epochs, val_loss,'r', label='Validation loss')
plt.title('Training and validation losses')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.xscale('log')
plt.grid();plt.legend();
plt.show()
##############################################################################
#   use the test data to investigate the prediction performance
##############################################################################
network_prediction = model.predict(X_test)
Y_test = np.reshape(Y_test, (Y_test.shape[0],1))
plt.figure(figsize=(14,5))
plt.plot(Y_test, 'r', label='System')
plt.plot(network_prediction,'b', label='1 step ahead Prediction')
plt.xlabel('Discrete time steps')
plt.ylabel('Output');plt.legend();plt.grid();plt.show()
###################### Plot the Simulation ###############################
simulate(X_test,Y_test)
```

B.2 ARX Code for Mass-Spring-Damper System

```
t_final=100;
dt=0.1;
time=0:dt:t_final; %time vector
%-------------------------Input-----------------------------------------%
type =1; % Choose 1 for single frequency and 2 for multiple frequency
trainingAmp=5;
trainingFreq=9;
validationAmp=2;
validationFreq=11;
if type ==1
    Input = trainingAmp*sin(trainingFreq*time);
else
   freqs=1:10; % Mulitple frequencies
    Input = time*0; % Initial input signal to 0
    %Add multiple frequency components to input signal
    for i=1:length(freqs)
        % Give random amplitude and phase to each frequency component
        Input=Input+rand()*sin(freqs(i)*time+rand()*2*pi);
    end
end

%------------------ Model for mass-spring-damper system----------------%
m=1;
c=1;
k=1;
%----------------------- Simulate the system-----------------------------%
x0=[0 0]; % initial conditions
option=odeset('RelTol', 1e-8, 'AbsTol', 1e-8);
[t,x]=ode45(@(t,x) MSD_system(m,c,k,Input,time,t,x),[0 t_final],x0,option);

%----------------------   differential equations ----------------------%

%-----Interpolate the output to the same time sampling as the input---%
Output=interp1(t,x,time);
position=Output(:,1);

%---------------------------------ARX model-----------------------------%
na = 2; % Number of past outputs
nb = 7; % Number of inputs (current+past)
```

```
nd = 0; % Number of future inputs
np = na+nb+nd; % Total number of parameters in ARX model

%---------------------- Parameter identification ------------------------%
transient_sec=10; % Number of seconds to disregard from the start
%of the data
transient=ceil(transient_sec/dt); % Number of data points to disregard
%from the start of the data
dataLength=length(position)-transient;
z = position(transient:end-1,1); % observation vector / position

%----------------------output from the simulation ----------------------%
phi = zeros(dataLength,np); % Data matrix
%Load past output values into the Data matrix
for i=1:na
    % Load na columns into the matrix, each one time shifted one step
    phi(:,i)=position(transient-i:end-1-i);
end
%Load past input values into the Data matrix
for i=1:nb
    %Load nb columns into the matrix,
     phi(:,na+i)=Input(transient-i+1:end-i);
end
%---------------  Least squares parameter identification ----------------%
theta = inv(phi'*phi)*phi'*z
%----------------- Simulate system using ARX model-----------------------%
z_hat=position(1:max(na,nb)+2); %Initialise first few points from data

%-------------Implement ARX model predicting one step -----------------%
%------------ ahead and then iterating to nexttime step ----------------%
for i = max([na,nb])+1:length(Input)
    z_hat(i)=flip(z_hat(i-na:i-1))'*theta(1:na)+

    flip(Input(i-nb+1:i))*theta(na+1:end);
end
%--------------- Calculate error metric for the fit ---------------------%
error = position-z_hat;
errorSquared=0;
for i=1:length(error)
    errorSquared=errorSquared+error(i)^2;
end
RMS = sqrt(errorSquared/length(error)); %Root mean sqaured value
```

```
trainingFit = RMS/max(abs(position))  %Normalised RMS, against the
%  amplitude of the signal

%------------------------ Plot results ------------------------------------%
figure, plot(time,position,'r',time,z_hat,'b'), xlabel('Time (s)'),
ylabel('Position (m)'), legend('System','ARX model'),
title(strcat('Training data : Fit =',num2str(trainingFit)))

%-------------------- Validate on new data --------------------------------%
if type == 1
    Input = validationAmp*sin(validationFreq*time);
else
   freqs = 1:10; %Mulitple frequencies
    Input = time*0; %Initial input signal to 0
    %Add multiple frequency components to input signal
    for i=1:length(freqs)
     %%% Give random amplitude and phase to each frequency component
        Input=Input+rand()*sin(freqs(i)*time+rand()*2*pi);
    end
end
%-------------------- Simulate the system ---------------------------------%
x0=[0 0]; % initial conditions
option = odeset('RelTol', 1e-8, 'AbsTol', 1e-8);
[t,x]=ode45(@(t,x) MSD_system(m,c,k,Input,time,t,x),[0 t_final],x0,option);
Output = interp1(t,x,time);
position = Output(:,1);
%------------  Simulate system using ARX model-------------------------%
z_hat = position(1:max(na,nb)+2); % Initialise first few points from data

for i = max([na,nb])+1:length(Input)
    z_hat(i)=flip(z_hat(i-na:i-1))'*theta(1:na)+

    flip(Input(i-nb+1:i))*theta(na+1:end);
end
%------------ Calculate error metric for the fit-----------------------%
error = position-z_hat;
errorSquared=0;
for i = 1:length(error)
    errorSquared=errorSquared+error(i)^2;
end
RMS = sqrt(errorSquared/length(error));
validationFit = RMS/max(abs(position))
```

```
%------------------Plot results-----------------------------------------%
figure, plot(time,position,'r',time,z_hat,'b'), xlabel('Time (s)'),
ylabel('Position (m)'), legend('System','ARX model'),
title(strcat('Validation data : Fit =',num2str(validationFit)))

%---------- Function to calculate the mass spring damper system ------%
function [x_dot]=MSD_system(m,c,k,Input,time,t,x)
    input=interp1(time,Input,t);
    x_dot(1,1)=x(2);
    x_dot(2,1)= (input - c*x(2) - k*x(1))/m;
end
```