Technische Universität München

TUM School of Engineering and Design

Hybrid modelling and simulation approaches for the solution of forward and inverse problems in engineering by combining finite element methods and neural networks

Rishith Ellath Meethal

Vollständiger Abdruck der von der TUM School of Engineering and Design der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz: Priv.-Doz. Dr.-Ing. habil. Stefan Kollmannsberger

Prüfer der Dissertation:

1. Prof. Dr.-Ing. Kai-Uwe Bletzinger
2. Prof. Dr.-Ing. habil. Roland Wüchner
3. Prof. Riccardo Rossi, Ph.D.

Die Dissertation wurde am 23.02.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Engineering and Design am 08.11.2023 angenommen.

Schriftenreihe des Lehrstuhls für Statik TU München

Band 60

Rishith Ellath Meethal

Hybrid modelling and simulation approaches for the solution of forward and inverse problems in engineering by combining finite element methods and neural networks

München 2023

Abstract

Executable Digital Twins (xDTs) assist engineers in assessing the current and future state of a physical component, product, or system and have become an essential part of the modern-day industry. An xDT describes the behavior and provides functionalities to optimize operation and service. It also evolves along the real system by making use of currently available knowledge about it.

An xDT utilizes different simulation models to describe a system and provide optimal operation and service functionalities. The first contribution in this dissertation is an algorithm for a hybrid simulation model which can be used in xDTs for real-time simulation. The neural network-based novel hybrid model is constructed by combining the conventional finite element method (FEM) and state-of-the-art neural network resulting in a physics-aware neural network. The residual from FEM and custom loss from the neural network are used for the hybrid model. The model is deployed along with a FEM framework to quantify the error associated with each prediction as the FEM residual. The approach is generalizable to any physics and geometric domain.

The model's suitability and scalability with respect to the different number of training samples, geometry, and neural network architecture are investigated for examples of varying complexity. The proposed hybrid model's advantages are also compared and quantified with state-of-the-art models like Physics Informed Neural Networks (PINN).

Constant updating of the Executable Digital Twin (xDT) model with currently available knowledge is essential to accurately represent and asses the physical component, product, or system. The hybrid model developed in the first contribution is extended for inverse problems in the second contribution of this dissertation. The unknown parts of the system are modelled using a neural network. The neural network surrogate prediction is then assembled in the system matrix and the residual obtained is used in the training. The unsupervised nature of the hybrid model is novel in this field. In contrast to the existing approaches, the presented hybrid model utilizes the existing frameworks without falling into the problem of matrix inversion. This novel aspect makes the hybrid model comparatively easily scalable to other models.

The inverse model is tested on the bearing parameters identification problem of rotordynamic systems. The robustness is tested on simple examples of Single Degree of Freedom (SDOF) and Multiple Degrees of Freedom (MDOF) systems with respect to different numbers of samples and a random initial guess. The scalability is tested by utilizing the model on the data from a rotordynamics *test bench for the dynamic fluid bearing parameter identification. In the tests, the hybrid model proved to be the best approach to utilizing neural networks for modeling unknown physics and capturing changes to the system parameters.*

One of the major setbacks faced by simulation or neural network community is the independent development of both areas and the need for a common framework is an absolute necessity. The Kratos-neural network application developed as part of this dissertation serves this purpose. It serves the common purposes like data creation and training of the neural network along with the interface to use a trained model in any multiphysics simulation. The suitability of the framework to perform a multiphysics simulation is demonstrated with the help of a Fluid Structure Interaction (FSI) benchmark simulation.

Acknowledgments

First on the list, I wish to express my gratitude to Prof. Dr.-Ing. Kai-Uwe Bletzinger, for the opportunity provided to me to work on this topic at the Chair of Structural Analysis. The continuous support, guidance, and encouragement helped me to finish this work without many hurdles. I sincerely thank Prof. Dr.-Ing. habil. Roland Wüchner for his supervision in organizing my research and the continuous motivation he provided. Numerous discussions I had with him have helped me to organize the research and complete it on time.

I would like to thank my advisors from Siemens Technology, Mrs. Birgit Obst and Dr. Mohamed Khalil, for their constant guidance and feedback. Without their encouragement and support, this thesis might have never reached an end. They quickly answered my queries because I never felt I was working alone on the problem for the whole time. They were also readily available to review my works, journal and conference submissions even at the late hours of the submission deadline. I appreciate the patience they had. Further, I thank Dr. Christoph Heinrich for offering me a chance to work on the topic in his group. His and the group's simulation expertise has always guided me in this research.

I am also grateful to colleagues from the Chair of structural analysis TUM for their contribution and continuous support. This includes but is not limited to Dr. Aditya Ghantasala, Philipp Bucher, and Anoop Kodakkal. Their collaboration has been vital in the journey and made my life easier by supporting me in planning, implementing and testing different ideas. Along with them I also thank Daniel Andres Arcones for contributing and collaborating in the form of his master thesis.

Last but not least, I would like to thank my family for their motivation and emotional support, which kept me going through the hard times. First among them is my wife, Sariga Premanand, whose constant reliance on a daily basis was inevitable during the whole period. She supported and encouraged me during the time when I was feeling low at work. My parents, Kunhikrishnan TP and Vijayi, were my strength the whole time. They are inspirational and supportive of my personal and professional life. And my sister Rishitha Ellath Meethal, whose motivational support and encouragement are invaluable, not only during my doctoral studies but throughout my life. Munich is my home away from home. I want to thank my friends (known by the acronym JGM) for their support in different phases of the last years. Their presence was crucial in overcoming the boredom caused by Corona lockdowns.

And this thesis is dedicated to my grandmother Janaki Kelappan, who has motivated my parents, me, and many in the family toward academics. Unfortunately, she departed us last year due to Corona. But she always encouraged me to further my education and has been doing that in my thoughts even after she left.

Rishith Ellath Meethal
Technische Universität München
February 2023

Contents

List of Abbreviations

AI	Artificial Intelligence. ix, 3, 9, 10, 13, 21, 22, 28, 30–34
ALE	Arbitrary Lagrangian Eulerian. ix, 86
ANN	Artificial Neural Networks. ix, 13, 14
BC	Boundary condition. ix
BEM	Boundary Element Method. ix, 11
CAARC	Commonwealth Advisory Aeronautical Council. ix, 132
CFD	Computational Fluid Dynamics. ix, 34, 35, 39, 40, 76, 95
CNN	Convolutional Neural Network. ix, 18, 34, 35, 47, 80, 89
CSM	Computational Solid Mechanics. ix, 76
DOF	Degrees of Freedom. ix, 110, 112, 113, 138, 140, 143, 144
DT	Digital Twin. ix, 3, 7, 8, 22

FDM	Finite Difference Method. ix, 11
FEM	Finite Element Method. ix, 11, 12, 27, 28, 59, 64, 71, 72, 99, 100, 109, 112, 113, 126, 135, 136, 138, 149
FEM-NN	Finite Element Method enhanced Neural Network. ix, 59, 61, 70, 71, 73, 99–101, 109, 112, 116, 118, 122, 126, 128, 134–138, 140, 143, 145–147, 151, 159, 160
FSI	Fluid Structure Interaction. iv, ix, 75, 76, 85, 88–90, 92–94, 131, 132, 150, 158
FVM	Finite Volume Method. ix, 11
HPC	High-performance computing. ix, 12
IC	Initial condition. ix
KPI	Key Performance Indicator. ix, 2, 4
LBFGS	Limited memory Broyden–Fletcher–Goldfarb–Shanno algorithm. ix, 17
LSTM	Long Short-Term Memory. ix, 18, 34, 35, 47, 80, 89, 90, 112
MDOF	Multiple Degrees of Freedom. iii, ix, 34, 113

ML	Machine Learning. ix, 13, 22, 34, 35
MOR	Model Order Reduction. ix, 9, 35
NLP	Natural Language Processing. ix, 13
NN	Neural Networks. ix, 27, 28, 37, 64, 100, 103, 112, 149, 150
PDE	Partial Differential Equation. ix, 7, 10, 12, 25, 26, 29, 34–36, 38, 59–61, 64, 150
PINN	Physics Informed Neural Networks. iii, ix, 6, 23, 36–38, 72–74, 101, 103, 105
RDS	Rotor dynamic system. ix
ResNet	Residual Networks. ix, 18
RNN	Recurrent Neural Network. ix, 18, 34, 47
SDOF	Single Degree of Freedom. iii, ix, 34, 113
SGD	Stochastic gradient descent. ix, 16, 17
xDT	Executable Digital Twin. iii, ix, 3–5, 7–9, 35, 149, 157

Chapter 1

Those who can imagine anything, can create the impossible.

Alan Turing

Introduction

Humankind has shown an attraction toward different natural phenomena from ancient times. The curiosity developed over such phenomena has led to continuous observations and conclusions. One notable example from history is the evolution of astronomy. Observing the night sky, stars, and moons over the years led to the beginning of astronomy. Such observations of different phenomena around us were the beginning of all sciences. In general, science is the knowledge produced by people in the form of testable explanations and predictions. In the early days, continuous observations and data recordings were used to derive such explanations, conclusions, and predictions. Over the years, data-based prediction became difficult due to the large amount of collected data. It gave rise to the concept of modeling. Modeling refers to a simplified representation of complex reality which captures the most important aspects of your system to a great extent. Often, scientific models are the solutions people come up with for explaining different phenomena. A scientific model represents objects, phenomena, and physical processes in a consistent and logical way. Rackauckas et al.

[84] state ”In the context of science, the well-known adage a picture is worth a thousand words might well be a model is worth a thousand datasets.” A model encapsulates all existing information in such a way that it can reliably predict the future state of the system. The scientific models developed and modified over the years have proven successful in providing reasonable explanations for natural phenomena around us. They were also valuable in providing reliable predictions. Later on, the scientific models have become a practice in most science and engineering problems.

When it comes to engineering, it all starts with the mathematical modeling of the physical system of interest. Research over decades or centuries has created mathematical models based on experiments and observations. Such models also evolve based on new findings. Depending on the resulting model’s complexity, it is either solved analytically (which is possible only for simplified models) or with the help of numerical methods. For years, numerical methods and simulations have been conventionally used to analyze engineering problems. Computers and computational methods have taken numerical methods to new heights, enabling the study of more significant problems in a shorter time. The use of simulations is common in the design phase. However, the primary purpose of simulations was the design and optimization in the last century. The simulation predicts different Key Performance Indicators (KPIs) of the problem at hand. A redesign may be required based on the KPIs obtained. Once design iterations are completed, the product is manufactured. In the whole process, a small amount of data is used for benchmarking and calibration of the mathematical model or simulation. Figure 1.1 illustrates the simulation as a design tool paradigm. Iterations between design and simulation can be many, based on the requirement of the problem and the simulation results.

In the last few decades, the scenario has changed drastically. The requirement of model-based systems engineering has evolved from engineering and manufacturing phases to operation and services phases. Along with the product, the product’s performance also plays an important role nowadays. In contrast to the approximate model and data available at the design phase, the system interacts with real data in real time to enable us to monitor and predict system performance. The introduction of the Digital Twin concept was the breakthrough

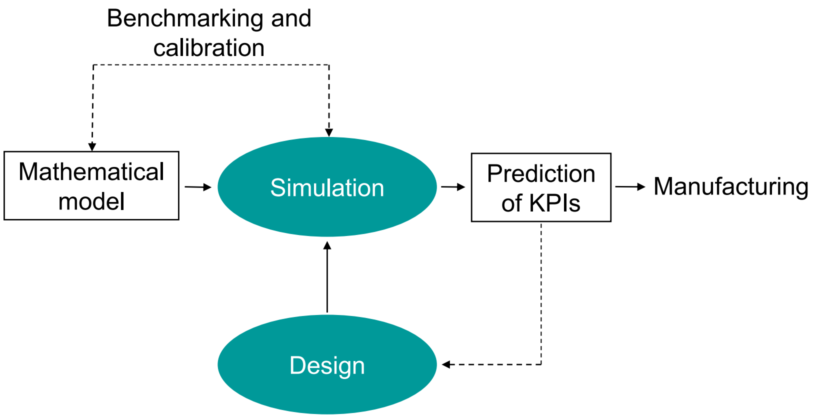


Figure 1.1: Simulation as a tool for product design

in this direction in the last few decades. As per Boschert et al. [10], Digital Twin (DT) refers to a comprehensive physical and functional description of a component, product, or system, which includes more or less all information that could be useful in all—the current and subsequent—lifecycle phases. One of the several characteristics of a DT is that it evolves along with the real system throughout the life cycle and integrates the currently available knowledge about it. The core part simulation along with the latest developments in the Internet of Things (IoT) and Artificial Intelligence (AI) are driving DTs forward. The ubiquitous presence of DTs in the industry has given rise to its self-contained realization called the Executable Digital Twin (xDT).

The xDTs enable better monitoring and decision-making for the system by making the engineering knowledge available throughout the entire life cycle of the product. The simulation acts as the core, while the changes to the physical system are captured with the help of sensors to update the xDTs. Figure 1.2 illustrates an xDT and how simulation plays an essential role. The simulation predicts the future state of the system and is then used for various decisions regarding the system. These could be decisions like stopping the system for a potential failure or changing operational parameters for improved performance. Such

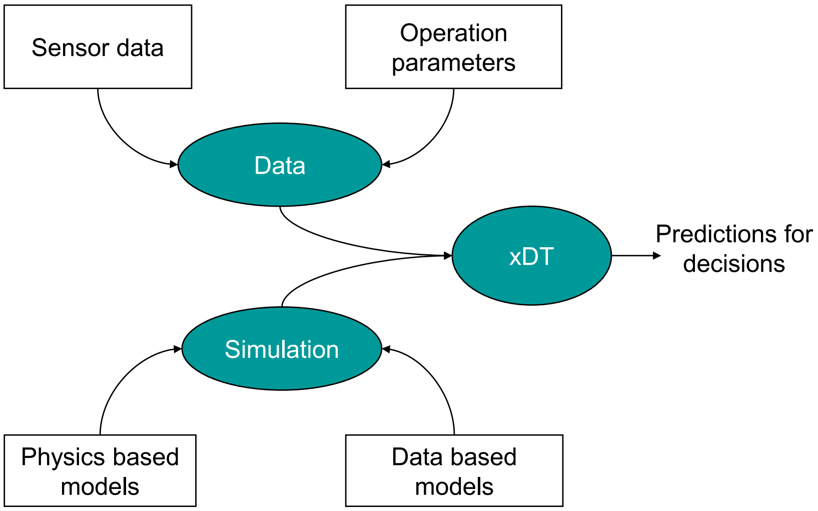


Figure 1.2: xDT and simulation

applications demand prediction in real-time. The prediction needs to be accurate along with the real-time prediction. The system undergoes many changes over time after the design phase, due to degradation or manufacturing deviations. The parameters of the system considered during the design phase may only be an approximation of the real case. Therefore, updating the system from time to time is also an indispensable characteristic of an xDT.

In the process of this dissertation project, two main characteristics of an xDT are addressed. The first one is the real-time simulation of the systems. Although real-time is relative, it means predicting useful KPIs for taking a prompt decision for performance improvement. For example, predicting a failure using the present state of the system before the failure happens can be considered as a real-time simulation. The second aspect considered in this dissertation is the model updating. A physical asset may differ from its designed state for many reasons. The changing environmental conditions, wear and tear over time are two primary causes of it. This also needs to be considered when having an executable digital twin. Actual environmental conditions and changes of the system over time are incorporated with the help of sensor data collected in an xDT. Parameter identification and updating it in the digital model is one topic that is as relevant in this direction. Most of the existing models are not real-time due to the complexity associated. Hence, a real-time and edge-device deployable models are needed.

1.1 Outline of the thesis

- **CHAPTER 2** takes the reader through the research outline. This chapter analyses both numerical simulation and AI paradigms with respect to their strengths and weaknesses for real-time engineering simulations for enabling digital twins. The chapter further discusses the directions in which simulation and AI can be combined. Then the state-of-the-art informed machine learning methods are analysed. The chapter concludes with the problem statement the dissertation addresses and the contributions.
- **CHAPTER 3** begins with the history of numerical simulations and AI. A journey through the history reveals the similarities and differences between both. This leads to the discussion on the hybrid models and methodologies combining both. The chapter also details the state-of-the-art in this direction and then discusses the research in the direction of inverse problems. Furthermore, the chapter concludes with a discussion on the recent research on the two applications considered.
- **CHAPTER 4** discusses different aspects of neural network training. This chapter begins by explaining different components of neural network and then proceeds to different aspects of its training. The chapter then details different points to consider for achieving a converged model in the training. Hyperparameter tuning, which is used to automate the architecture and parameter selection of neural networks, is discussed in the last part.
- **CHAPTER 5** introduces a novel algorithm combining numerical methods and neural networks. The first part discusses the algorithm for forward solving of the engineering problems whereas the second half discusses the version of the algorithm for the inverse

problems. The chapter also discusses different versions of the algorithm, such as those combined with Physics Informed Neural Networks (PINN), are discussed.

- **CHAPTER 6** introduces the framework created for the integration of AI and simulation. The requirement of having such a platform and the designed architecture are detailed here. The integration of the introduced novel algorithm and other state-of-the-art algorithm within the framework is explained. Additionally, the chapter also contains some benchmark results using the framework.
- **CHAPTER 7** examines the novel algorithm with the help of numerical examples. The method is compared with conventional and state-of-the-art methods and results are discussed. Examples for both forward and inverse problems are studied in detail.
- **CHAPTER 8** explores the applicability of the novel algorithm to industrial real-world applications. The simulation of the wind-load on high-rise buildings is taken as the example for forward problems. The bearing parameter identification of a fluid bearing in a rotor-dynamic system is considered for the inverse problem.
- **CHAPTER 9** concludes the dissertation. The potential of the algorithm for various kinds of problems and the limitation of the algorithm observed are discussed in this chapter.

The following footnotes are used in the thesis to reference the original publications and to mark literal transposition.[1] [2] [3]

[1] The following section is based on [72]. The main scientific research as well as the textual elaboration of the publication were performed by the authors of this work

[2] The following section is based on [6, 70]. The main scientific research as well as the textual elaboration of the publication were performed by the authors of this work

[3] The following section is based on [54]. The main scientific research as well as the textual elaboration of the publication were performed by the authors of this work

Chapter 2

Mathematics is the key and door to the sciences.

Galileo Galilei

Research Outline

As stated in Chapter 1.1, two significant requirements of an xDT are addressed in this dissertation. One is the real-time simulation of the system under consideration using forward solving of Partial Differential Equations (PDEs), and the other is the model updating of the system using inverse problems. In this chapter, we outline the essential basics, developments, and the present-day shortcomings in both the directions and conclude with the contributions of this dissertation.

2.1 Digital Twin

The term Digital Twin (DT) has revolutionized the industry in the last decade by becoming one of the most promising technologies for enabling smart manufacturing and Industry 4.0. Even though there are many definitions for DT, one of the widely accepted definition from the simulation viewpoint is from Boschert et al. [10]. It states

> “ The Digital Twin refers to a description of a component, product or system by a set of well aligned executable models with the following characteristics:
>
> - The Digital Twin is the linked collection of the relevant digital artefacts including engineering data, operation data and behaviour descriptions via several simulation models. The simulation models making-up the Digital Twin are specific for their intended use and apply the suitable fidelity for the problem to be solved.
> - The Digital Twin evolves along with the real system along the whole life cycle and integrates the currently available knowledge about it.
> - The Digital Twin is not only used to describe the behaviour but also to derive solutions relevant for the real system, i.e. it provides functionalities for assist systems to optimize operation and service. Thus, the Digital Twin extends the concept of model-based systems engineering (MBSE) from engineering and manufacturing to the operation and service phases.
>
> ”

As evident from the listed characteristics, simulation plays an integral role in DTs. Along with simulation, optimization and decision-making in the digital version, with the help of the data collected on the physical asset, are essential in creating a DT.

2.1.1 Executable digital twin (xDT)

Executable DT refers to the self contained realizations of DTs. It is a software where DT is encapsulated to run on a restricted hardware. One major intention of having an xDT is to take DTs outside the premises of the creator of DT. This enables the non-expert use of DTs. Major advantages of an xDT are,

- Non-experts can use DTs
- Protection of intellectual property
- Running DTs on restricted hardware like edge devices can enable real time decisions.

As mentioned in Section 1.1, the simulation serves as the core in xDT. Physics-driven numerical methods and data-based simulations are the two directions practiced for performing simulations. The process of determining unknown primary variables from the known input conditions is called forward simulations. Physics-based simulation methods for forward simulations are evolved over the years and provide robust solutions. Nevertheless, the time taken for simulation and hardware requirements limits its use in xDTs. The selection of the simulation methodology varies depending upon the need and model complexity. Most of the time, the simplified physical models are used to achieve the real-time nature required for the xDT. The Model Order Reduction (MOR) is also a highly appreciated way to achieve real-time simulations. Simulation of physical assets in real-time using AI is also gaining attention among researchers due to the success of AI in different areas. In contrast to numerical simulations, they are faster and easily deployable on edge devices. However, the black-box nature of many AI algorithms poses a threat to the system due to the unexplainability and incomprehensibility of the predictions. This naturally paves the way for combining explainable physics-based simulation models and unexplainable AI algorithms for forward simulations.

Along with simulation, the real data from sensors also plays a vital role in xDTs. The data can be used either as an input to simulation models or to update simulation models. Updating models is essential to track the changes in the system and environment in real-time. The modeled systems for forward simulations needed to be updated to reflect the changes. Finding out unknown or changed system parameters is called inverse problems. There are different methods in numerical methods for inverse problems. However, complexities like matrix inversion and preconditioning make the process challenging. At the same time, the data-based methods have shown partial success in inverse problems. Since both have approaches have shortcomings (which will be discussed later), the hybrid model is a good direction for inverse problems as well. In the following, we go into the details of numerical simulation and AI based surrogates to assess their strengths and weaknesses in the context of engineering problems. Only neural networks based AI surrogates are considered as they have shown great success in recent studies. The discussion covers both forward and inverse problems.

2.2 Numerical Simulations

Physical systems in engineering applications are mathematically modeled for their detailed analysis. Mathematical models help to predict the future states or analyze the past events of the system.

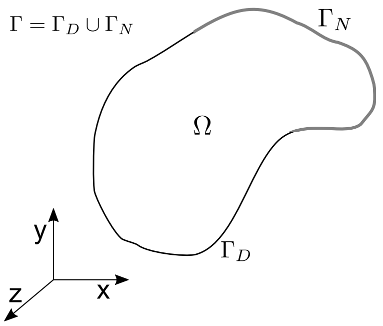


Figure 2.1: Problem definition

$$\begin{aligned} \mathcal{L}(u) &= 0 && \text{on } \Omega \\ u &= u_d && \text{on } \Gamma_D \\ \frac{\partial u}{\partial \hat{\mathbf{n}}} &= g && \text{on } \Gamma_N \end{aligned} \tag{2.1}$$

Consider a PDE governing a physical system defined on the domain Ω together with the Dirichlet (Γ_D) and Neumann (Γ_N) boundary conditions as in Equation 2.1. Here, $\mathcal{L}(u)$ is any partial differential operator and $u = u(x, t)$ is the exact solution field. However, the calculation of exact solution field u is not possible if an analytical solution does

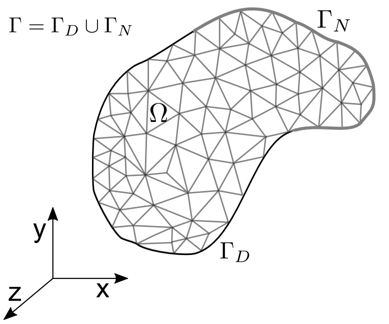


Figure 2.2: Problem definition with FEM grid

not exist for the complex mathematical model at hand. Such models are typically solved with the help of numerical methods. Numerical methods are techniques to approximate mathematical procedures. A mathematical model is converted to a discretized model using the process of discretization in numerical methods. This discretized model is solved for the unknowns in the system. Solving a discretized model results in discrete solution instead of the actual continuous physical variable under consideration. Methods such as Finite Difference Method (FDM) [31], Finite Volume Method (FVM) [115], Finite Element Method (FEM) [135] and Boundary Element Method (BEM) [18] are the commonly used numerical methods in Engineering. As an example, Figure 2.2 shows the FEM grid for the general problem given in Figure 2.1.

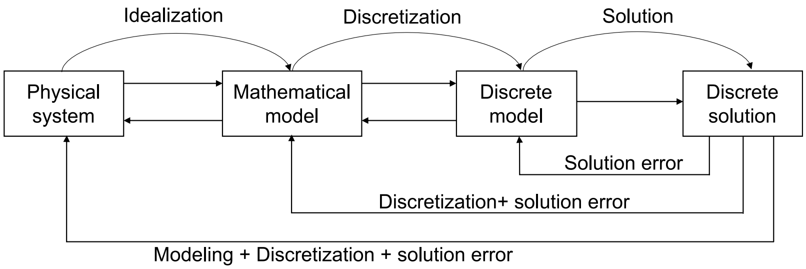


Figure 2.3: Errors in a numerical model

Although each of the numerical methods has its own advantages and disadvantages, we limit our discussion to FEM for the rest of the dissertation. The FEM decomposes the domain of interest into a finite number of subdomains called elements. The weak form of the governing equation is then integrated element-wise and results in an algebraic equation. Analytical integration of the equations is complex or infeasible in most of the PDEs. This demands the use of numerical quadrature. The algebraic equations from every element are then assembled to form the global system. The resulting global equations system is then solved using linear solvers for the unknowns. The variable $\hat{\boldsymbol{u}}$ will be used for the discretized solution from FEM in the rest of the dissertation. The accuracy of the solution increases upon increasing the number of elements. But this increases the size of the global system to be solved. It demands the use of more expensive iterative solvers instead of direct solvers.

Different steps, from modeling to solving using the numerical method, introduce different kinds of errors in the final solution. Figure 2.3 shows the steps followed in approximating a physical system to obtain a discrete solution of the variables of interest. The error associated with each of the idealization, discretization, and solution steps plays a critical role in simulation and its application. In terms of magnitude, the modeling error is considered the largest, followed by discretization and solution errors. These errors are one of the major drawbacks associated with numerical methods. Even though many of the numerical methods in use produce results meeting the demands of the present engineering regime, higher consistency between numerically simulated and actual values is always desirable.

Numerical methods play an important role in modern-day engineering. In the earlier days, they were used as a design tool. Different design alterations can be made and analyzed with the help of numerical methods. Nowadays, they are used in the entire life-cycle of a product. It is used for preventive and predictive maintenance and operation recommendations. This demands faster simulation methods. However, complex engineering problems like a simulation of a wind turbine, simulation of thermal distribution inside a motor, etc., are computationally intensive. They take hours, if not weeks, even in High-performance computing (HPC) environments.

2.3 Artificial Intelligence to Neural Networks

In general, AI is the ability of digital systems to perform tasks normally performed by intelligent beings. Automation, Machine Learning (ML), Natural Language Processing (NLP), robotics, and self-driving technology are some of the fields that encompass AI methods. Though the general term AI is used everywhere, there are many subcategories of AI which are relevant in this work. It is explained with the help of 2.4.

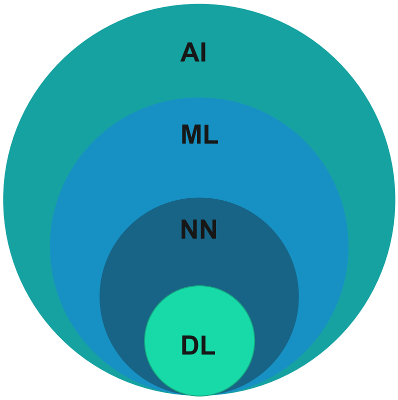


Figure 2.4: Artificial intelligence and branches

ML is the branch of AI which learns a task from data. Normally, ML results in a model based on given data to perform a given task. A simplified diagram representing a ML pipeline is given in Figure 2.5. Neural networks are a set of ML algorithms and deep learning is a subset of neural networks. In the following section we go into the details of neural networks and deep learning as we are focusing on those two subcategories of AI in this dissertation.

2.4 Artificial Neural Network and its applications

The hypothesis that models inspired by biological neural networks may help machines solve intelligent tasks gained attention among scientists in the late 40's. It resulted in Artificial Neural Networks (ANN). ANNs are computational models with interconnected smaller processing units.

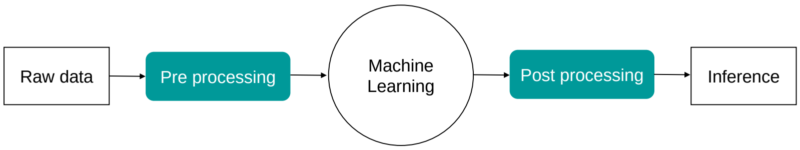


Figure 2.5: Machine learning process using data

They have become popular in the last decade after showing their success in many disciplines. ANN techniques are used in multiple domains such as computing, science, engineering, medicine, environmental, agriculture, mining, technology, climate, business, arts, and nanotechnology [1]. Neural-network models such as feed-forward and feedback propagation artificial neural networks perform better in their application to human problems. ANN showed promising results for classification, clustering, pattern recognition, speech recognition, language processing, and prediction problems in many disciplines.

Nowadays, ANNs are primarily used for universal function approximation in numerical paradigms because of their excellent quality in self-learning, adaptivity, fault tolerance, nonlinearity, and advancement in input-to-output mapping [1]. Two significant properties exhibited by neural networks make them attractive to any discipline.

- **Universal approximation theorem**: The universal approximation theorem states that a neural network with a single hidden layer having sufficient number of neurons can approximate any continuous function to a reasonable accuracy. It was George Cybenko [22] who showed that a continuous feedforward neural networks with only a single internal, hidden layer and any continuous sigmoidal nonlinearity can be used to approximate decision regions arbitrarily well. Kurt Hornik [42] applied this to all activation functions (Explained in detail in Section 4.2.2).
- **Overcomes the curse of dimensionality** Neural network has shown success in finding out relevant information from the given data. This property is normally useful as data might contain a lot of

relevant and irrelevant details. This property was one major step in the history of AI which reduced the human intervention that was essential for feature engineering.

Simplest form of a neural network is explained with the help of Figure 2.6. Here a fully connected neural network is mapping input variables $\mathbf{x} = [x_1, x_2, ...x_n]$ from domain X to output variables $\mathbf{y} = [y_1, y_2, ...y_m]$ of domain Y. The input layer is connected to the output layer by L number of layers in between, called hidden layers. Each hidden layer receives input from the previous layer and outputs $o_l = [o_l^1, o_l^2, ..., o_l^k]$. The $w_l \in \mathbb{R}^{n_l \times n_l+1}$ matrix represents the weights between layer l and $l+1$ and the vector b_l of size n_l represents the bias vector from layer l. Here, n_l represents the number of neurons in each layer of the neural network. The output from any layer is calculated using the Equation 2.2.

$$z_l = w_l o_{l-1} + b_l \tag{2.2}$$

A nonlinear activation function $\sigma(.)$ is applied to each component of the transformed vector z_l before sending it as an input to the next layer.

$$o_l = \sigma(z_l) \tag{2.3}$$

Following this sequence for all the layers, the output of the entire neural network can be written as

$$\mathbf{y}(\theta) = \sigma_L(z_L......\sigma_2(z_2.\sigma_1(z_1(x)))) \tag{2.4}$$

where $\mathbf{y}$ is the output vector for the given input vector $\mathbf{x}$, and $\theta = \{w_l, b_l\}_{l=1}^{L}$ is the set of trainable parameters of the network.

During the training process the output vector predicted by neural network $\mathbf{y}(\theta)$ is compared against the actual value $\hat{\mathbf{y}}$. It is done by formulating it as a minimization problem. The objective function for minimization can be any function of $\mathbf{y}(\theta)$ and $\hat{\mathbf{y}}$ and is known as loss function. One of the most used loss function is the mean squared error (MSE) between them.

$$\delta = \frac{1}{n}\sum_{i}^{n}(\mathbf{y_i}(\theta) - \hat{\mathbf{y}}_\mathbf{i})^2 \tag{2.5}$$

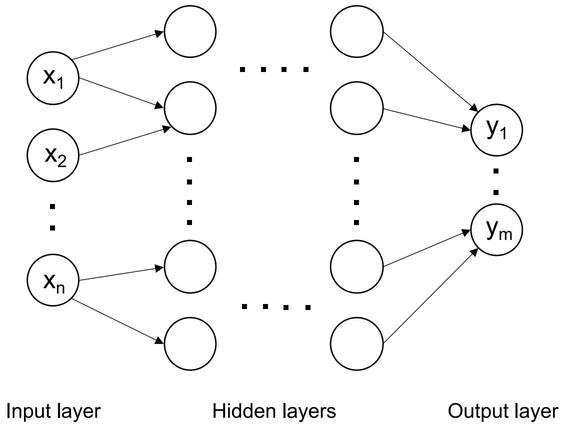


Figure 2.6: Artificial neural network

The loss $\boldsymbol{\delta}$ is reduced by updating the learnable parameters $\boldsymbol{\theta}$ of the network. The process of updating the learnable parameters is called backpropagation [40] and is performed using different optimization algorithms such as Stochastic gradient descent (SGD). In the classical SGD procedure, each learnable parameter $\boldsymbol{w_l}$ and $\boldsymbol{b_l}$ are updated as below.

$$\begin{aligned} w_l &= w_l - \eta \frac{\partial \delta}{\partial w_l} \\ b_l &= b_l - \eta \frac{\partial \delta}{\partial b_l} \end{aligned} \tag{2.6}$$

The parameter η is called learning rate and chosen by the user. For any optimizer of choice the derivatives of loss with respect to the learnable parameters $\frac{\partial \delta}{\partial w_l}$ and $\frac{\partial \delta}{\partial b_l}$ are required. They are calculated using the chain rule.

$$\begin{aligned}\frac{\partial \delta}{\partial w_l} &= \frac{\partial \delta}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial w_l} \\ \frac{\partial \delta}{\partial b_l} &= \frac{\partial \delta}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial b_l}\end{aligned} \tag{2.7}$$

Here, the first term $\frac{\partial \delta}{\partial \mathbf{y}}$ depends on the chosen loss function. In the case of a MSE loss function,

$$\frac{\partial \delta}{\partial \mathbf{y}} = 2\frac{1}{n}\sum_{i=i}^{n}(\mathbf{y_i} - \hat{\mathbf{y}}_\mathbf{i}) \tag{2.8}$$

The second term of the equation 2.7 is again calculated using chain rule. In this case the chosen architecture of the network, chosen activation function etc contributes. In any modern neural network software, the derivative of any element's output with respect to its input is readily available. In addition to SGD, different optimization algorithms are used for updating the parameters. Some of them are Adam [52], Adadelta [129] and Limited memory Broyden–Fletcher–Goldfarb–Shanno algorithm (LBFGS) [63].

In the case of simulation, primary variable is represented as $\mathbf{u}$ in this dissertation. The neural network prediction of the primary variable will be represented using $\tilde{\mathbf{u}}$. The actual value for training the neural network can either come from simulation result $\hat{\mathbf{u}}$ or measured using sensors on the physical system. The measured values from sensors are much closer to the actual physical values. Hence, they will be represented using $\mathbf{u}$.

2.4.1 Deep learning

The term "deep" in the deep learning refers to the depth of neural networks, represented by the number of layers within neural networks. Conventionally, a neural network with three or more layers is considered a deep learning algorithm. Deep learning is a subset of ML. Different processes which were done with manual interventions became automatic with the introduction of deep learning. One notable example is the feature extraction. Compared to ML, deep learning differs in two aspects. First one is the type of data and second is the method in which it learns. Conventional ML algorithms require structured, labeled data to make predictions. In another words, even if ML uses unstructured data, it goes through some pre-processing steps to organize the data into structured format. However, DL can digest and process unstructured data like images, text, speech and automates the pre-processing steps like feature extraction. DL determines by itself which features are important. Many of the modern day neural network architectures fall under the category of deep learning. Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), Residual Networks (ResNet) to name a few.

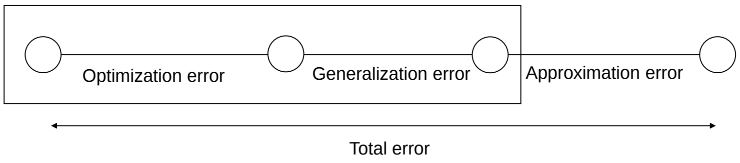


Figure 2.7: Errors in a neural network model

Similar to the numerical methods, the error analysis for neural network based models is given in Figure 2.7. Here approximation error refers to the error between real solution and the solution that a neural network of a particular architecture can result. This error can be large or small based on the function approximation capability of the chosen architecture for the function to be approximated. So, by choosing a suitable neural network this error can be reduced. Presently it is done empirically based on experience. Various parameters deciding the architecture of the neural network can be used as hyperparameters in the parameter tuning algorithms as well. However, it's practically infeasible to perfectly train a very large network, resulting in a certain level of approximation error always persisting. To make the neural network work on unseen data, bias-variance trade-off is made. Bias is the model's ability to make good prediction for all the data points. Low bias refers to the good prediction by the model. However, low bias may result in a state where model learns the given data very well and gives large error on an unseen data. This refers to the high variance of the model. This bias-variance trade off leads to generalization error. And the last part optimization error is the error caused by not achieving the minimum in the minimization process.

2.5 Neural Networks for simulation

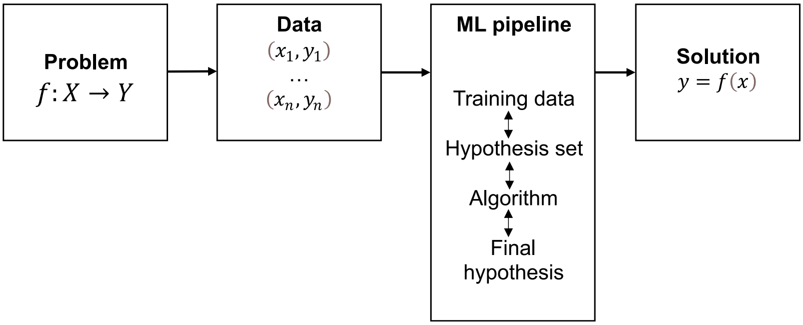


Figure 2.8: Pipeline of conventional neural network

The success of neural network in different scientific and real-life applications opened its way to simulations. Neural network can be used to perform simulation using the given data. In the initial days, even continued today, the focus were on creating surrogate models for simulation. The data required for training the network were either created by running a large number of simulations or measured from the respective real experiment. The process is given in Figure 2.8.

An example in this category is the prediction of flow field around an airfoil given the initial conditions, boundary conditions and geometry of airfoil. Here X represents the initial conditions, boundary conditions and geometry of airfoil. This can be in the form of images, as neural networks have been successful in image processing. An example of the same is given in Figure 2.9. This example is an experiment following the work of Thuerey et al. [103]. The Y represents the pressure and velocity field around the airfoil. The data pair (x_n, y_n) are the data generated by running simulation with different input variables and creating solution fields. Final solution will result in a neural network surrogate, which is also a function $f(x)$, which can predict flow around a given airfoil. The results of 750 simulation trials run using OpenFOAM is used as the training data. Training took a total of 1516 seconds for 250 epochs on a laptop using Tensorflow-GPU (NVIDIA GeForce 940MX). The prediction of pressure and velocity fields using the trained neural network is given in Figure 2.10. It can be observed in Figure 2.10 that the difference between predicted velocity and ground truth velocity is as high as 0.01 m/s in the case of x-direction and 0.006 m/s in the case of y-direction. It is observed that the accuracy of results were not consistent. Predictions for some other input scenarios had error percentage of 18.75% for x-direction velocity and 6.25% for y-direction. The major advantage of such a surrogate is the speed at which it predicts the flow fields. However, it can be observed that the error associated with it is also significant. The error is more worrying when it becomes non-physical, like non-zero velocity on the airfoil surface.

Another category in the use of neural network in simulation is the replacement of a part of the simulation pipeline with a neural network. It occurs at places where a mathematical model is difficult to obtain. For example, experimental data regarding a material can be used to

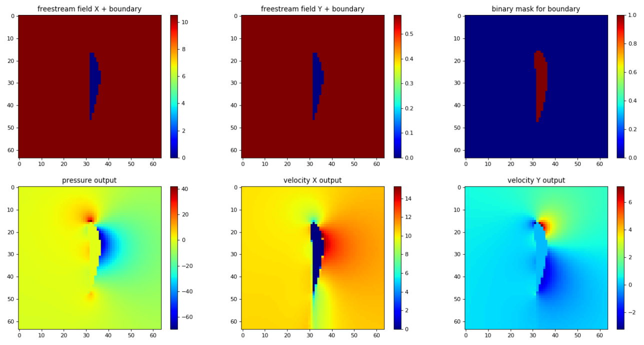


Figure 2.9: Input output data pair for predicting flow-field around an airfoil

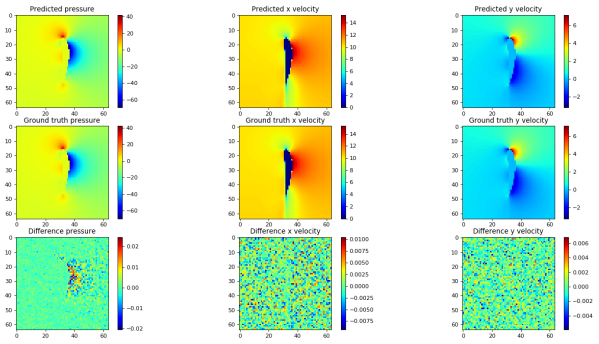


Figure 2.10: Predicted and actual flow-field around an airfoil using U-net neural network architecture

create a surrogate model. This surrogate model can be used instead of the mathematical constitutive model in the simulation.

2.5.1 How simulation and neural networks complement each other

Though developed independently over the years, the overlap between both these fields can be observed in many methods. One can consider simulation as part of the big spectrum of AI. Conventionally simulation is expert driven, whereas AI can be historic data driven or a combination of historic data and expert driven. The most important similarity between both these fields is that, they can be used for predicting the future state of the system under consideration. In the case of simulations for engineering problems, the main focus of simulation is to do experiments without hardware cost. In design phase to find the right design, in operation to optimize the operation or to do what-if scenarios for decision making and to compare expected behavior with observed behavior for failure detection. Whereas, AI methods are used to predict the future state of a system based on the model it learned from historical data. A classical example where AI is used for prediction is the stock market prediction.

Researchers worldwide are exploring how AI methods and simulations can leverage each other for better prediction of system state. Even though there are studies with promising results combining AI and simulation, there is no standard approach. One can see the combination in two different directions. The first one is for simulating a physical asset in the digital world in real-time. Real-time simulations are the need of the hour to enable the DTs for different components in industrial applications. Since each domain has its strength and weakness, there is great potential for AI and simulation to take advantage of each other's strengths. The second direction in which simulation and AI can work hand in hand to achieve digital twin is in the model updating. In this approach, one can use combined AI and Simulation to update the digital version of the physical asset according to the real changes. The real changes can be captured using sensors and other measurements.

2.5.2 How simulation and neural networks differ

As explained in previous sections, numerical methods or neural networks can be used for simulating a physical system. It is also possible to use both of them to develop hybrid models by complementing each other. A few of the directions are given in Section 2.5.1. However, there are major differences between these two approaches that are to be considered while developing hybrid models. A summary of the major differences is given in Table 2.1.

Even though ML methods have shown great success in many areas, they fail when dealing with insufficient data. Simulating physical systems with the help of machine learning methods is one such area. The generation of a large amount of quality data using numerical simulation is computationally intensive. Generating data by means of experiments also faces the same challenge. Another drawback when using machine learning in a conventional nature to physical systems is the "black-box" nature of the methods. The resulting model may not necessarily follow the laws governing the physical system. The informed machine learning methods have shown success in dealing with the above drawbacks. In this approach, the prior knowledge is integrated in the learning process of the machine learning model. The prior information can be algebraic equations, differential equations, invariances, and logic rules. This helps

Table 2.1: Comparison of numerical methods and neural networks for simulation

	Numerical methods	**Neural networks**
Physics embedding	Discretized equation	No physics embedding, (Only loss function based on data)
Solver	Linear/Non-linear solvers	Gradient based optimizer
Error	Modeling, Discretization, and solution	Optimization, generalization and approximation
Solution	Point values	weights and biases
Approximating function	Piecewise polynomials	Parametric
Mesh	Yes	No (Possible with mesh based data also)
Generalization	Extrapolates really well	May result in nonphysical results
Model	Theory based	Data based

the model to train with less amount of data as well as to follow the rules in the form of prior knowledge. The following section details informed machine learning and one of the famous approach in this direction called PINN.

2.6 Informed machine learning

The feature that purely data-driven methods fit well for the given observations is not sufficient when it comes to engineering simulations. The simulation predictions should be physically consistent. This drawback along with the lack of sufficient training data led to the ”teaching” of ML using governing physical laws. The informed machine learning paradigm coined in Von Rueden et al. [116] refers to the integration of prior knowledge into the machine learning pipeline. Figure 2.11 represents the distinction between conventional machine learning and informed machine learning.

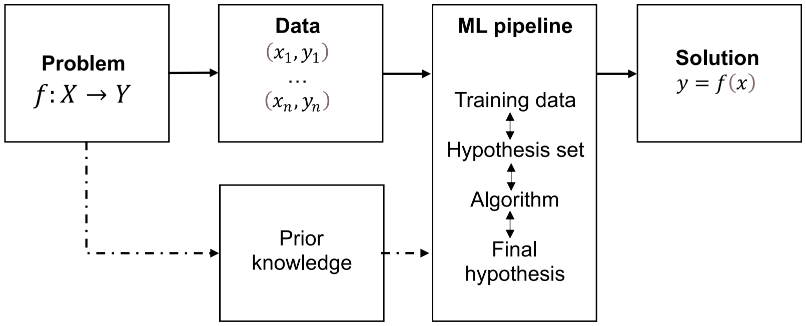


Figure 2.11: Pipeline of an Informed machine learning (adapted from Von Rueden et al. [116])

Conventional machine learning comes with a single information whereas informed machine learning can have more than one. The second information comes in the form of prior-knowledge and integrated into the machine learning pipeline via interfaces. The prior-knowledge can be logic rules, invariances, probabilistic relations, differential equations, algebraic equations or even expert knowledge. For example, if one is trying to learn relationship between mass and energy in a system's rest frame, he can use the algebraic equation 2.9 as the prior information about the data.

$$E = m \cdot c^2 \tag{2.9}$$

where E is the energy, m is the mass and c the speed of light. Similarly differential equations governing the relationship between different physical quantities can be used as the prior information.

Similar to the form of the knowledge, how they are integrated into the machine learning pipeline can be of different types. Simplest way is to use the prior knowledge to create more data and augment the existing data with this created data. Another approach is the modification of neural network's architecture according to the prior knowledge. Use of convolutional neural network for examples where location and translation invariances must be preserved falls under this category. Another prominent approach involves incorporating prior knowledge directly into the learning algorithm, as demonstrated by Von Rueden et al. [116]. This entails modifying the loss function of the learning algorithm to reflect the available prior knowledge. Such an approach has demonstrated significant success in recent years. In these instances, the loss function can be expressed as follows

$$f = argmin(\lambda_i L(f(x_i), y_i) + \lambda_j R(f) + \lambda_k L_k(f(x_i), x_i)) \tag{2.10}$$

Here, L represents the conventional labeled data-based loss function as explained in Section 2.4 and R is the regularizer. The new term, L_k represents the penalty from violating the given prior-knowledge. Such an extended loss, if it measures the inconsistencies w.r.t to physical laws, is called physics-based loss.

There is numerous research in the direction of physics-based loss functions. However, the physics informed neural network explained in Section 2.6.1 offers an easy way to integrate any type of PDE, integer-order PDEs, integro-differential PDEs, fractional PDEs, or stochastic PDEs, into the loss functions of neural networks.

2.6.1 Physics informed neural networks

Physics informed neural networks (PINNs) introduced by Raissi et al. [85] integrate different forms of PDE into the loss function of a neural network using automatic differentiation [66, 78]. Consider a parameterized partial differential equation of the form

$$\begin{aligned} & f(x, t, u, \frac{\partial u}{\partial t}, \frac{\partial u}{\partial x}, ..., \lambda) = 0, \quad x \in \Omega, \quad t \in [0, T], \\ & u(x, t_0) = g_0(x), \quad x \in \Omega, \quad t = 0, \\ & u(x, t) = g_\Gamma(x, t), \quad x \in \Gamma, \quad t \in [0, T], \end{aligned} \tag{2.11}$$

where x is the spatial coordinate and t is the time. u represents the actual solution of the equation and $\frac{\partial u}{\partial t}$ and $\frac{\partial u}{\partial x}$ are the gradient of solution with respect to spacial and temporal coordinates. Note that the gradient can be higher order terms like $\frac{\partial^2 u}{\partial x^2}$ depending the equation under consideration. The function f represents the residual of the PDE with parameters $\lambda = [\lambda_1, \lambda_2, .., \lambda_n]$. The initial and boundary conditions governing the PDE are given by $g_0(x)$ and $g_\Gamma(x,t)$, respectively. Boundary condition g_Γ on the boundary Γ can be Dirichlet, Neumann or mixed boundary conditions.

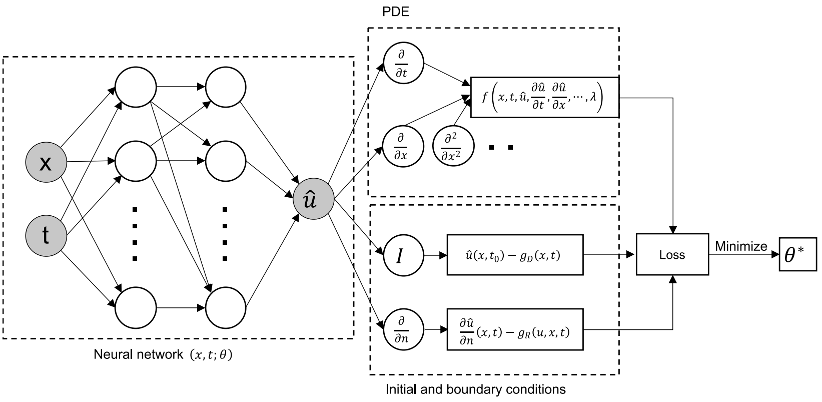

PDE
x
t
$\hat{u}$
$\frac{\partial}{\partial t}$
$\frac{\partial}{\partial x}$
$\frac{\partial^2}{\partial x^2}$
$f\left(x,t,\hat{u},\frac{\partial\hat{u}}{\partial t},\frac{\partial\hat{u}}{\partial x},\cdots,\lambda\right)$
I
$\hat{u}(x,t_0)-g_D(x,t)$
$\frac{\partial}{\partial n}$
$\frac{\partial\hat{u}}{\partial n}(x,t)-g_R(u,x,t)$
Loss
Minimize
θ^*
Neural network $(x,t;\theta)$
Initial and boundary conditions

Figure 2.12: Training of a PINN model

In PINNs, a neural network is deployed to model the solution of the PDE. The network takes space and time coordinates x, t as the input and predicts an approximate solution $\tilde{u}$ as the output. As explained in Section 2.4, the approximate solution can be written as $\tilde{u}=f_n(x,t,\theta)$ where θ represents trainable parameters such as weights and biases. The approximated solution is found out by treating the problem as an optimization task where a loss function is minimized by iteratively updating θ. The loss term is given by

$$L = L_f + L_{MSE} \tag{2.12}$$

where

$$L_f = \frac{1}{N_f}\sum_{i=1}^{N_f} \|f(x_f^i, t_f^i, u_f^i, \ldots)\|^2 \tag{2.13}$$

$$L_{MSE} = \frac{1}{N_u}\sum_{i=1}^{N_u} \|\tilde{u}(x_u^i, t_u^i) - u^i\|^2 \tag{2.14}$$

The term L_f enforces the governing equation 2.11 at a finite set of points called collocation points. The term L_{MSE} is imposed to satisfy known data points, initial and boundary conditions. The model parameters $\boldsymbol{\theta}$ are optimized using optimizers such as Adam, SGD, L-BFGS (details regarding optimizers are given in Section 2.4). As per different researchers around the world, the present generation of PINNs are not as accurate or efficient as numerical solvers [49, 85]. However, they are efficient for inverse problems. More details on the state-of-the-art is given in Section 3.4.1.

2.7 Problem statement

The earlier developments in combining neural network and simulation were focused on creating surrogate models out of simulation data. The method proved to be inefficient due to lack of enough data and loss of physics conformity of the resulting model. The idea to include more physics information in the learning process is the state-of-the-art. However, it can be observed that most of the developments in creating neural network models tend to rely on pure data-based methods while giving minimal consideration to physics and numerical methods. Many of the existing informed machine learning models are resulting in ill-posed system of equations. Despite the success that numerical methods like FEM have shown over the years, hybrid models combining numerical methods in their original form with neural networks directly have not been extensively explored by the research community. To this end, following problems are addressed,

Conceptualize and develop a hybrid model, that is generalizable to any physics, combining finite element method and neural network for predicting primary variables along with an error estimate.

The steps followed to achieve this goal are:

- Couple FEM and Neural Networks (NN) framework for the training of an informed neural network based on FEM

- Develop an optimization problem for the neural network which addresses an exact problem rather than an ill-posed one.
- Develop method to increase the accuracy, reliability and safety of neural network predictions by integrating it with existing numerical frameworks

The modeled physics may not completely represent a physical system under consideration. There is also possibility that the model parameters change over the time due to the environmental conditions or degradation. Different numerical methods struggle in this regard due to high computing time, matrix inversion, large number of possible solutions etc. More details regarding state-of-the-art and its limitations are given in 3.5. The second part of this dissertation focuses on the inverse problems. The goal is to:

> *Update numerical models used for the real-time simulation with the help of hybrid models combining FEM and NN for reducing the uncertainty in the modeling by means of sensor data.*

On walking through the demands and interests of the industry, it can be observed that there is an increased demand for real-time simulation methods and the updating of models than ever. The increased demand is partially catalysed by the success of AI in different areas of digitalization. A large number of methods (including the one in this dissertation) are proposed and being investigated now. One of the major challenge in developing and deploying new methods and models is the independent development of the numerical and data-based simulation worlds. It is imperative at this moment to have a framework which encourages the development, testing and deployment of new AI models coupled with the existing simulation frameworks. This goal of this dissertation is aimed at this shortcoming,

> *Architecture and develop a framework which enables seamless integration of numerical simulation and neural networks to facilitate and accelerate data generation, model construction and model usage in a multiphysics setting.*

The steps followed to achieve this goal are:

- Develop user friendly, intuitive and flexible methods for data generation.
- Integrate the neural network training process as part of the existing numerical framework so that an existing user can make use of it as any other feature in the framework.
- Develop interfaces to deploy neural network models independently or integrated with rest of the features in a multiphysics framework.

2.7.1 Contributions

1. **Algorithm for forward solving of parameterized PDEs**
   In this scope of the thesis, the aim is to come up with a neural network-based surrogate model for simulation. However, this model should also use the knowledge from the FEM discretized domain as in Figure 2.2. Informed machine learning has been gaining attention these days to use neural networks along with the prior knowledge we have. The process of training those networks involves including the prior knowledge into the loss function of the neural network. In this way, the neural network is trained against the equation, inequality, or conservation laws. This research proposes to use a discretized version of our PDE using a FEM mesh inside the loss function for the training. Using the FEM package along with the deployment of the neural network enables the verification of the prediction. This also helps to identify unphysical predictions from the network.

2. **Algorithm for model updating and parameter identification** It is well known that numerical methods based predictions are often challenged when they are compared with test results. This arises due to multiple reasons. Uncertainty in the concerned governing equation and uncertainty in the model parameters are the most addressed.

   Updating a numerical model by using data acquired from the physical asset is important for predicting the future states. This requirement gave rise to the field of model updating. The area model updating is concerned with the correction of numerical model/simulation models by processing records of operation data from the physical asset. This includes the reconstruction of data that are unavailable due to sensor placement.

In this scope, the aim is to extend the algorithm developed for forward problems to inverse problems. Here the measured data from real machines are used to calculate the unknown parameter of the digital model. It is common in industry that the parameters of the model change over time due to different wear and tear. So, the parameter identification is relevant to update the digital model according to the physical model.

3. **Framework for the seamless integration of neural networks in simulation**
Recent developments in the field of computational mechanics have introduced methods combining conventional numerical methods with state-of-the-art AI. These methods roughly involve two steps. First creating a surrogate for the numerical model. Second is to use the generated surrogate in place of the numerical model or along with other numerical models as a replacement to a part of multiphysics simulations. Both approaches demand integration of AI related packages into existing simulation workflows. The Neural Network Application developed in Kratos [23] is one such an application which enables the user to use neural networks in combination with Finite Element Methods (FEM). The application includes a general interface developed with the help of CoSimulation Application of the Kratos framework. This includes routines to interact with the widely-used software package for AI applications Tensorflow. Multiphysics problems present a good case to test the above developed methodology as individual numerical models of the simulation can be replaced by already generated surrogate models from AI to perform the multiphysics simulation. To this extent, a Fluid-Structure Interaction (FSI) problem is chosen. This contribution discusses different properties of the surrogate effecting the outcome of the simulation and methods to improve the stability and error in the estimates from the surrogate.

Chapter 3

Research is creating new knowledge

Neil Armstrong

State of the art

As stated in Chapter 2, this dissertation focuses on combining simulation methods and AI with the help of neural networks. This chapter surveys the main contributions of combining simulation and AI. Since the focus is on neural networks, developments in other sub-fields of AI, especially the recent ones, are not discussed in detail. After the survey on AI, the evolution of the neural network over time with a focus on informed machine learning is surveyed. Recent developments in Informed machine learning are analyzed for their strengths and weaknesses. Finally, application-specific surveys on bearing parameter identification and high-rise building simulation are performed at the end of the chapter.

3.1 Artificial intelligence

After the emergence of machines, there were attempts to mimic human actions with the help of machines. The term Artificial Intelligence (AI) originated following these attempts, refers to the intelligence demonstrated by machines as opposed to the natural intelligence human beings and animals display. The history of AI travels way back to 1940s where American Science Fiction writer Asimov [7] introduced three laws of robotics. The one stating “A robot must obey orders given it by human” received the attention of the scientific world. Later in 1950s, Alan Turing [111, 112] explained how to create intelligent machines and introduced a method to test their intelligence. The test later known as “Turing test” is still considered as benchmark to test AI systems. As per Turing test, if an evaluator fails to distinguish whether a task is performed by an human or a machine, the machine is considered to pass the test. Another contribution in the same period is from Hebb [39], where Hebbian Learning replicates the process of neurons in the human brain was introduced. Hebbian learning later led to the creation of Artificial Neural Networks (ANN). Initial works in the Game AI also occurred in 1950s. The AI based machine for playing Nim [87] was one of the first in this direction.

Initial researches such as the ones listed above made more interest towards AI among scientists. Dartmouth Summer Research Project on Artificial Intelligence hosted by Marvin Minsky and John McCarthy in 1956 united researchers from various fields to build machines that are capable of simulating human intelligence. The event is marked as the beginning of the AI spring. The conference followed a lot of research in the field of AI. One of the many notable contribution following this is the natural language processing tool ELIZA [119] developed at MIT. But, most of the works in the direction of AI and ANN stagnated due to the insufficient processing power of computers and this resulted financial setbacks for AI research. However, one contribution came during this time in 1974; backpropagation in Werbos [120] introduced a more practical method for training ANNs. The backpropagation distributed error term to different parameters in the network. Backpropagation is the major driving force behind neural network even today.

Although there has been further developments, the next major milestone happened in 1997. The world chess champion and grand master Gary Kasparov was defeated by IBM’s Deep Blue [17]. It was based on expert systems where a collection of rules assumed human intelligence can be represented as a series of ‘if-else’ statements. However, such expert systems failed when tried to learn such rules by themselves.

This led to a complete paradigm shift from expert systems to computers discovering rules by themselves. The concept of intelligent agents gained attention during this time. An intelligent agent is a system that perceives its environment and takes actions which maximize its chances of success. Later in the early years of 2000's, the success of AI was found in areas like natural language processing, human emotion detection, image recognition etc. Those developments hinted that machines could address many problems human face.

The availability of large amount of data and the introduction of highly efficient computer graphics card processors enabled AI to grow further. One of the notable examples in this direction is the ImageNet dataset [24]. The dataset motivated researchers around the world to compete for object recognition algorithms. The success of object recognition paved way for researchers to attempt more complex tasks like generating captions for scripts, image generation, etc. Artificial neural networks came back with the help of Deep learning (DL) [59] in 2015. A program developed by DeepMind was able to beat the world champion in the game of Go [95]. Within years, AI has become an essential part of our daily life. Examples like autonomous cars, natural language processing, image recognition, language processing, consumer behavior are some of the successful applications of AI that are seen today.

3.2 Artificial neural networks

The threshold logic by McCulloch et al. [68] from 1943 marks the beginning of artificial neural networks. But it took another three decades for neural networks to get traction until the use of backpropagation in neural networks in 80s. It was Paul Werbos [120] who realised the potential of backpropagation in neural networks. Backpropagation along with gradient descent form the backbone of modern day neural networks. This paper described the construction of a system that recognizes hand-printed digits, using a combination of classical techniques and neural-net methods.

Neural networks and their application has seen exponential growth from there onward. The Neural network recognizer for hand-written zip code digits [25] was first among them. It is also to be noted that the first hidden layer of this neural net was convolutional. Many of the modern day elements of neural networks originated between 1990-2000. This include but not limited to CNN, RNN, LSTM and so on.

In the last decade, neural network has been successful in many more application fields. These include computer vision [98], speech and language processing [47], drug discovery [13, 57], genomics [61], computer games [28], animation [33], robotics [51], and many more. Similarly, AI has also made a large number of contributions to computer games [27, 75, 125]. An important direction in this field is game physics, where physical effects such as smoke and fluid flows for computer graphics are simulated with neural networks. Tompson et al. [108] proposed a data-driven solution to the inviscid-Euler equations that is faster than traditional methods used in computer graphics animations. Similar to the developments in game physics, different AI methods have found their application in solving PDE for physical problems.

3.3 Simulation and neural networks

One of the first works using neural networks for simulation is dated way back to 1989. The work by Adeli et al. [3] used the perceptron (basic unit of a neural network) for the design of a beam. Later, Vanluchene et al. [114] trained neural networks on simple benchmarks such as prismatic beams, simple truss structures and plates. In the following years, more research on using neural networks for structural engineering has been witnessed. A summary of major works can be seen in Adeli [2]. Recent works also address different complexities in the structural simulation. Wu et al. [122] presented a deep CNN approach for the prediction of transient vibration response of Single Degree of Freedom (SDOF) and Multiple Degrees of Freedom (MDOF) systems using multilevel perceptron and CNN. Their ML model forecasts displacement in SDOF systems while taking velocity, acceleration and excitation as inputs. It is observed that the CNN based network could work even with the noise in the input data.

Similar to structural mechanics, the use of neural network can be seen in other simulations fields like Computational Fluid Dynamics

(CFD), thermal and electromagnetic. Surrogate modeling using neural network was more attractive to CFD due to the expensiveness of the numerical simulation. One of the beginner in this direction is the steady flow approximation using CNN by Guo et al. [35]. They showed that CNN can estimate the velocity field two orders of magnitude faster than a GPU-accelerated CFD solver and four orders of magnitude faster than a CPU-based CFD solver at a cost of a low error rate. Thuerey et al. [103] investigated the accuracy of deep learning methods for Reynolds-Averaged Navier-Stokes solutions. Pressure and velocity distributions around airfoils are predicted after training a model U-net architecture the training data. They achieved an mean relative error of less than 3% for unseen airfoil profiles. They highlight that a physical understanding of the problem helps to convert the problem to non-dimensional formulation which significantly improves the training results. In contrast to using neural networks for directly predicting the primary variables, there has been an interest in the community to model pressure projection or turbulence modeling required for the fluid simulation. Tompson et al. [108] proposed the use of CNN for the approximation of pressure projection step by using divergence free condition directly for the training of the model. This way the problem is unsupervised in nature. Jiang et al. [46] introduced an interpretable framework of data-driven turbulence modeling using deep neural networks. The framework resulted in models that exhibit good generalization across two- and three- dimensional flows. There are also other successful researches in modeling turbulence using neural networks such as Ling et al. [62], Yin et al. [126], and Zhu et al. [131]

A conventional method in accelerating the forward solving of PDEs, especially in xDTs, is the use of model order reduction techniques. Data driven methods for MOR also gained attention of the ML community. Recently the neural network based algorithms has found its place in MOR as well. Mohan et al. [73] used LSTM neural network for successfully modeling turbulent flow control. Zhuang et al. [134] introduced Runge-Kutta neural networks which learn the derivative of system state and predict the new state as a numerical integrator. Zhuang et al. [133] also introduced a method for active learning which makes the sampling of training data for MOR smarter using neural networks.

However, most of the works mentioned above follow the pure data-driven path proposed by the machine learning community. The approach faces two major challenges, lack of enough data and lack of physics conformity. This gave rise to the direction of informed machine learning.

3.4 Informed machine learning

The umbrella term informed machine learning introduced by Von Rueden et al. [116] refers to different approaches on the explicit integration of prior knowledge into machine learning pipelines. They structure different approaches according to the three above analysis questions about the knowledge source, knowledge representation and knowledge integration. In another work, Willard et al. [121] provides an overview of approaches which integrate traditional physics based modeling techniques with ML. The authors categorize these approaches into five classes; (i) Physics-guided loss function, (ii) Physics-guided initialization, (iii) Physics guided design of architecture, (iv) Residual modeling, and (v) Hybrid physics-ML models.

Among the many approaches in this direction, the Physics Informed Neural Networks gained the interest of most the researchers in the last few years. In the following section we detail the state-of-the-art in this direction.

3.4.1 Physics informed neural networks

The PINN introduced by Raissi et al. [85] is one of the widely accepted methods falling under the category of informed machine learning. PINNs embed the physics in the form of the PDE into the loss function of the neural network using automatic differentiation. More details about PINN is given in Section 2.6.1. PINNs are successfully demonstrated for different application fields in the last few years, fluid dynamics [15], thermodynamics [16], solid mechanics [36], electromagnetics [110] to name but a few examples. Eventhough PINNs present a differentiable, mesh-free approach and avoid the curse of dimensionality, the researchers found that the numerical grid-based conventional approaches outperform PINNs in forward problems. However, PINNs were found useful for parameterized PDE in low data regime or inverse problems. Some of the contributions of PINN in inverse problems are system identification in Yuan et al. [128], inverse scattering problems in photonic metamaterials [20], and seismic inversion problems Zhu et al. [132].

Recently, Zhang et al. [130] applied a multiLSTM neural network which maps the excitation force to the response of the system. They couple custom model architecture and loss functions to represent the underlying physics resulting in a model which outperforms conventional data driven LSTM models in terms of robustness and accuracy. Latterly, Wang et al. [117] devise and present a Knowledge-Enhanced Deep Learning (KEDL) algorithm which trains a NN to predict response of a system for a specific excitation. The authors used both input output data and prior knowledge in the form of equations into the NN's training loss function.

One of the main observation on PINNs failure when solving complex problems is that the soft regularization used in PINNs makes the problems ill-conditioned [55]. Krishnapriyan et al. [55] state that PINNs learn good model for relatively trivial problems and fails for complex problems. A detailed study on when and why PINNs fail is performed by Wang et al. [118] with the help of neural tangent kernal theory. One of the major observations they made is that fully-connected PINNs not only suffer from spectral bias, but also from a remarkable discrepancy of convergence rate in the different components of their loss function. Similar observations related to the multiple terms in the loss functions are made by more researchers. Bischof et al. [9] observed the need of correctly weighing the combination of multiple loss functions and introduced a self-adaptive loss balancing method. Xiang et al. [124] also observed that weighted combination of competitive multiple loss terms play an important role in the training of PINNs and introduced a self-adaptive loss balanced PINN.

As a summary, inverse problems are handled well by PINNs, whereas PINNs struggle to perform well for forward problem. As stated in the initial work on PINNs by Raissi et al. [85], PINNs should not be viewed as replacements of classical numerical methods for solving partial differential equations. More details on the scenarios where PINN may be advantageous can be found in Karniadakis et al. [49]. And latest developments hint at revisiting the loss terms or using a method to balance different loss terms in the PINN to avoid the problem becoming an ill-posed. However this also posses different difficulties as pointed out in Xiang et al. [124].

3.5 Inverse problems

Inverse problems deal with finding out the factors which caused the given observations. Historically, the discovery of planet Neptune from Uranus is the first in this direction. Systematic and formal study of inverse problems began in 20th century. Gladwell [29], Potthast [81], and Sabatier [91] discuss the directions explored and the obtained results in inverse problems in the 20th century. As per Sabatier [91], a mathematical model M enables us to predict the result of any possible measurement ϵ by giving the parameters C. Giving the model M explicitly is called direct problem. Going back from ϵ to C is called the inverse problems. The terminology inverse problem appeared first in the 1960s to designate the unknowns in the geophysics equations through experiments. Today it is widely used as the best possible reconstruction of missing information such as the loads (source identification) or the value of the undetermined parameters (model parameter identification).

Most of the works till 1970s were empirical in nature. The work of G. Backus and of F. Gilbert Backus et al. [8] introduced the use of numerical methods for the inverse problems. In 1980s Tarantola came up with the idea of using probabilistic models like Bayesian approach for the inverse problems. Tarantola authored several books on this subject [100, 101]. Later on different approaches like functional analysis [58], Regularization of Ill-Posed Problems [104], Bayesian inversion are used for inverse problems. Recent decade has seen a lot of research into the inverse problems with the help of machine learning methods. Raissi et al. [85], in the second part of the famous paper on PINN, introduced a method for inverse problems. The unknown parameters of a PDE are estimated from the measured data by backpropagating the residual of the equation.

Roehrl et al. [88] introduced a physics-informed neural ordinary differential equations (PINODE), a hybrid model that combines first principle models and data driven models. They integrated prior physics knowledge where it is available and used function approximations like neural networks—where it is not. The results on simple cart pendulum system show the advantages resulting from hybrid approach. Since we are dealing with the inverse problems in the rotordynamics systems, more state-of-the-art specific to the subject are given in section 3.6.2.

3.6 State of the art for applications

There are two engineering application examples are considered in this dissertation. The the state-of-the-art in these two applications are discussed here. In the first part, some of the major studies in the direction of wind load on high-rise buildings are discussed. In the second part, parameter identification methods developed for the fluid bearings are discussed.

3.6.1 Wind load on High-rise buildings

Conventionally the wind load on high-rise buildings are analysed with the help of wind tunnels (Building codes and standards are also used for the same. However, building codes and standards are made from wind tunnel tests). The three commonly used wind tunnel test types are high-frequency balance (HFB), high-frequency pressure integration (HFPI), and aeroelastic techniques. The use of numerical methods gained its attention later due to the high cost associated with those wind tunnel tests.

Thordal et al. [102] details the practical application of CFD for the determination of the wind load on high-rise buildings. Though the results were not matching with the wind tunnel experiments, this study highlighted the importance CFD can play in the analysis of wind load on high-rise buildings. Recently Hou et al. [43] reviewed the past studies in this direction and provided information on identification techniques. In the study of Buffa et al. [12] a Lattice-Boltzmann-based Large-Eddy Simulation approach for wind load prediction is proposed. A very good agreement has been obtained with experimental data at all validation levels using a well suited grid resolution along with a well calibrated Synthetic Eddy Method.

Different uncertain environments cause long term damages to the high-rise buildings. The terrain at a location and the wind are two major contributors of uncertainty. The uncertainties associated with them need to be studied to study the uncertainty of the load on the buildings. Different studies are done to estimate the uncertainty in the load and the displacement of the building. Tosi et al. [109] used ensemble averaging techniques for the uncertainty quantification of the CFD predictions in wind engineering problems. Kodakkal et al. [53] proposed a novel approach to risk-averse shape optimization of tall building structures using site-specific uncertainties. The study highlights the importance of fine tuned optimal design for different predominant wind directions. It also discusses the need for reducing the overall computational cost.

Recently, the use of machine learning has gained the attention of the community due to the faster prediction time using trained models. Wind-induced pressures on a building surface are predicted using neural network in Dongmei et al. [26] based on wind tunnel experimental data. As this method still demanded the use of wind tunnel tests, it is costly in nature. A multi-fidelity machine learning approach has been proposed by Lamberti et al. [56] to predict the rms pressure coefficient on a highrise building based on LES data. The proposed framework can significantly reduce the number of LES simulations needed for the design. Eventhough machine learning based methods are giving good results, there are a lot of problems we need to address. In the review paper on machine learning for wind-problems [123], Wu et al explains that the explainability and uncertainty quantification are the important research gaps that need to be addressed in ML-based wind engineering.

3.6.2 Bearing parameter identification

Rotating system are in use for centuries. They were the main driving force for factories and systems throughout the world. Hence they were one of the main system under study from ancient times. Most of the time, the corresponding industry was leading the study due to the advantage it provide in the applications. In the initial days, the effects like wirling were studied [86]. Later the studies were moved towards the stability of the system. The conditions leading to nonsynchronous precession in a rotor system are explained in Gunter Jr [34].

In 1957 the role of fluid film bearing in rotor dynamics was explained in a graphical way by Newkirk Newkirk [76]. Tiwari et al. provided a detailed review of the conventional methods of bearing parameter estimation in [60]. This include methods based on incremental static load, dynamic load, excited load, unbalance mass [60], impact hammer [83], impulse [83, 106] etc. Most of the methods either used bearings in isolation or a rigid shaft. In 2002, Tiwari et al. [107] introduced a method treating shaft as flexible for the identification of speed-dependent bearing parameters. Normally such methods do not consider the foundation flexibility, and its contribution is not estimated. The method also requires matrix inversion, which demands the consideration of the condition number of the matrix and methods to improve the condition number. Another problem associated with the method is that the data corresponding to each speed has to be considered independently to estimate the parameter corresponding to the particular speed. A model mapping speed to parameters was not possible with such methods.

A Kalman filter based approach was developed by Kang et al. [48] for the dynamic bearing coefficient identification in which displacement of the shaft is measured only at one location. Kalman filter was employed to estimate displacements of the shaft at bearings locations. The method provides a more practical approach for the estimation of bearing parameters but still uses the conventional least-square method for calculating bearing parameters from the calculated displacement at bearing locations. Least-square methods may not guarantee global optimum, especially when noise or other uncertain factors are present. Kriging surrogate model and Differential Evolution (DE) algorithm was employed in parameter identification of rotor-bearing system in Han et al. [37]. It is found that the Kriging surrogate model is more robust to the noise and costs less time but at the cost of generating a dataset prior to the initial surrogate model creation.

A neural network based approach was introduced in Pavlenko et al. [80] for learning bearing parameters against rotational speed. They used a numerically simulated dataset of possible parameters and the corresponding critical speeds. Then a neural network surrogate was created to predict bearing parameters given the critical speed. It was observed that the neural network approach resulted in more accurate results compared to conventional regression methods. They attribute this mainly to the universal approximation capacity of neural networks. However, this method demands the creation of a dataset prior and hence is costly in nature.

Chapter 4

The key to artificial intelligence has always been the representation

Jeff Hawkins

Neural network training

4.1 Introduction

An introduction to neural networks is given in Section 2.4. This chapter focuses on detailed information on the neural network with a focus on training. Even though neural networks have shown success in many fields, their training is still highly empirical in nature. Hence, fundamental knowledge of different aspects helps achieve an accurate and generalizable model. The training process of a typical neural network model has the following steps

1. Select the architecture of the neural network
2. Initialize the weights and biases of the neural networks.
3. Perform the forward pass on the input dataset x and obtain the predictions y

4. The loss function is calculated from the ground truth values y and the predictions $\hat{y}$.
5. Calculate derivatives of the loss function with respect to the parameters of the network using backpropagation.
6. Update weights and biases of the network using the calculated derivatives and selected optimization algorithm.
7. Repeat steps 2 to 6 until a convergence criterion is fulfilled or a set number of iterations (epochs) are evaluated.

The basics of each of the above steps are explained in this chapter.

4.2 Neural network architecture

A neural network is a complex structure consisting of neurons. Neurons are said to mimic the biological behavior of a brain. A typical neural network consists of an input layer, hidden layers, and output layers (Section 2.6). Every layer consists of several neurons interconnected with the neurons of other layers. Wide varieties of neural network architecture are possible depending on the connection type, number of neurons in each layer, or number of layers. The selected architecture of the network plays an essential role in approximating the unknown relation between input and output. The following part details major concepts in creating a neural network and some of the neural network architectures used in the dissertation.

4.2.1 Neurons

Neurons are fundamental units of a neural network. It is mathematically described by the function

$$N(x) = \sigma(x^T w + b) \tag{4.1}$$

where x is the input vector, w the weight vector and b the bias. The function σ is a non-linear function known as activation function.

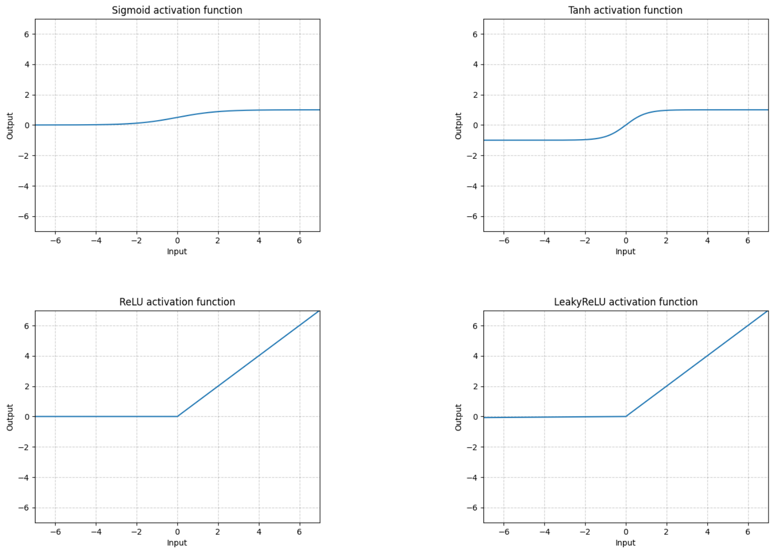


Figure 4.1: Activation functions Sigmoid, Tanh, ReLu and LeakyReLU

4.2.2 Activation functions

It is a function used in the neural network to decide if the incoming information is important or not in achieving the final goal of the network. Most activation functions map the input to a range of input $[-1,1]$ depending on the usefulness. It helps the artificial neural networks to use important information while suppressing irrelevant. A few of the widely used activation functions are detailed below and shown in Figure 4.1.

- **Perceptron** It is one of the first activation functions and an algorithm. Its range goes from 0 to 1 with a strong discontinuity at 0. Perceptron is better suited for classification problems as the results are binary. The strong discontinuity generates problems in training, as the derivatives are poorly defined at that point.

- **Sigmoid** Similar to perceptron, it has a range from 0 to 1. However, the sigmoid provides a soft slope that removes the discontinuity. Sigmoid is also generally used for classification problems, but continuity allows its use for regression.

- **Tanh** Tanh activation function evaluates the hyperbolic tangent of the input data. It gives a smooth function ranging between -1 and 1. It is the most flexible of those presented here but also the most expensive to evaluate.

- **ReLU** The Rectified Linear Unit or ReLU is defined as the maximum between 0 and the input value. It does not allow negative values in the output, and those in the input result in 0 independently of their value. This can be problematic when used in the first layers of a model. It is one of the most used activation functions and has benefits like sparseness at the activation that helps accelerate the training process. Nonetheless, the lack of differentiability at zero generates uncertainties in the training if the values are close to 0. This has promoted the appearance of similar functions with smoother behavior around 0, such as LeakyReLU and SmoothReLU.

- **LeakyReLU** It is an activation function based on ReLU, but having a small slope for negative values instead of a flat slope.

A few of the neural network architectures used in this dissertation are given below.

4.2.3 Fully connected neural network

Fully connected neural networks are the simplest form of neural network that resembles Figure 2.6. It connects every neuron in one layer to every neuron in the other layer, hence the name fully connected. They are known as “structure agnostic”, as no particular assumptions are needed to prepare the input. It is broadly applicable to any problems with the drawback of weaker performance compared to special networks.

4.2.4 Convolutional neural network

Convolutional networks [5, 77] are used with grid structure input data, or input has to be converted to a grid structure to use the CNN architecture.

The main component of a CNN architecture is the filter(kernel) of a specific dimension applied to the input layer in the grid structure to produce a new grid as output. There can be more than one filter generating those many output grids from a single input grid. In comparison to fully connected, they have three significant advantages: sparse interactions, shared parameters, and produce equivariant representations [32]. These three properties make CNN very efficient if data can be structured into a grid. They are very efficient in capturing geometrical information from the given data, hence are widely used among the simulation community as well.

4.2.5 U-Net

U-net [89] is a CNN architecture initially developed for biomedical image segmentation task. It consists of one contracting path and an expanding one. The Contracting path consists of repeated application of convolutional layers, reducing the grid structure's spatial information while increasing the features. The expanding path consists of up-convolutions and concatenations via skip connections, resulting in expanding grid structures. The contracting and expanding structure results in the name U-net. Better performance and fewer data requirements make U-net an attractive option for simulations.

4.2.6 Long Short-Term Memory (LSTM) network

LSTM[127] neural networks are mainly used for sequences of data. They have succeeded in sequence data analysis for applications like machine translation, robot control, and speech recognition. LSTMs were developed to solve the vanishing gradient problem (Section 4.9) that was encountered with the traditional RNNs [69]. The architecture of LSTM allows the gradient to flow unchanged and prevents the vanishing and exploding gradient problems.

All the networks mentioned above can take different final architectures based on the number of layers, the number of neurons in each layer, activation functions used, batch normalization, and dropout percentage. The final architecture is obtained with the help of hyperparameter tuning as explained in Section 4.10.

4.3 Parameter Initialization

After selecting the neural network architecture, the next step is to initialize the parameters of the network. The initialization step can be critical to the final performance of the model. For example, initializing with constant values results in poor performance. Constant initialization makes all the neurons in a layer have an identical contribution to the loss term and hence identical gradients. This makes those neurons evolve similarly during training and prevent the neural network from learning. Whereas initializing with very large or very small values results in divergence or slow convergence of the network. These also causes exploding and vanishing gradient problems (detailed in Section 4.9). In order to prevent from above problems, appropriate initialization is required. Making the mean of the activation zero and keeping variance similar in every layer prevents the networks from resulting in vanishing/exploding gradients [50]. This can be achieved by picking initialization values from the normal distribution. A few of the widely used initialization methods are Xavier initialization and He initialization.

4.3.1 Xavier initialization

Xavier Glorot and Yoshua Bengio proposed an initialization method called Xavier in their paper [30] in 2010. In this method, all the weights of a layer l are picked randomly from a normal distribution with a mean $\mu = 0$ and variance $\sigma^2 = \frac{1}{n^{l-1}}$. All the biases are initialized with zeros. However, it is observed that Xavier initialization performs well with tanh activation functions in comparison to other activation functions.

4.3.2 He initialization

He initialization [38] is commonly used while using ReLU activation functions. The weights are initialized by multiplying by 2 the variance of Xavier initialization in the He initialization. There are variants like He uniform and He normal initialization depending on the distribution from which values are selected.

4.4 Optimization

The process of optimizing the network starts after the selection of the neural network architecture and the initialization. This section details the basic optimization concept with a focus on neural networks.

The process of minimizing or maximizing any mathematical expression is called optimization. Optimization problems have the form,

$$\begin{array}{lll} minimize & f_0(\theta) & \\ subject\ to & f_i(\theta) \leq b_i, & i = 1, ..., m \end{array} \tag{4.2}$$

The function $f_0 : R^n \rightarrow R$ is the objective function and functions $f_i : R^n \rightarrow R$, for $i = 1, ..., m$, are the constraint functions having limits or bounds $b_1, ..., b_m$. The input to the objective function $\theta = (\theta_1, ..., \theta_n)$ is the optimization variable of the problem. A vector θ^* is called the solution of the problem 4.2, if it gives the smallest value for the objective and also satisfy the constraints.

There are two main classes of optimization problems convex optimization and non-convex optimization problems. Both the objective and constrain functions are convex in the case of convex optimization problems, which means

$$f_i(\alpha\theta + \beta\gamma) \leq \alpha f_i(\theta) + \beta f_i(\gamma) \tag{4.3}$$

When talking about optimization in the context of neural networks, we are discussing non-convex optimization. Convex optimization involves a function which there is only one optimum, the global optimum. Whereas, the non-convex optimization involves a function which has more than one optima, only one of which is the global optima. It can be very difficult to locate the global optima depending on the loss surface we define. The problems like getting stuck at local optima, too small learning rate, or loss surface morphology changes can occur while performing optimization. Optimizers are algorithms or methods used to change the parameters $\theta = (\theta_1, ..., \theta_n)$ to minimize f_0. How one should change θ is defined by the optimizers you use. Optimization algorithms are responsible for minimizing f_0 and to provide the most accurate results possible.

4.4.1 Optimizers

Over the years different optimizers are being developed to work with neural networks. A few of the widely used ones are explained below.

- **Gradient Descent**: It is an optimization algorithm used to find minimum or maximum of a given differentiable function. A function $f(\theta)$ decreases fastest in the direction of the negative gradient of f at the given point θ. The updated value for θ is given by

  $$\theta_{n+1} = \theta_n - \gamma \nabla f(\theta) \tag{4.4}$$

  where n and $n+1$ represents the iteration number. And γ represents the learning rate. Learning rate decides how big a step is taken towards the optimum.

- **Stochastic Gradient Descent (SGD)**: Gradient Descent has a disadvantage that it requires a lot of memory to load the entire dataset of n-samples at a time to compute the derivative of the loss function. In the SGD algorithm derivative is computed taking one sample at a time. In comparison to the gradient descent it is faster but the convergence rate is low.

- **Mini Batch Stochastic Gradient Descent (MB-SGD)**: It is a variation of SGD algorithm where the training data is split into small batches. The model error, gradient and model update are performed for one batch at a time. It is a balance between the robustness of SGD and the efficiency of gradient descent. It avoids the situation of having all data in the memory as well as computationally efficient due to the use of batches. It is the most commonly used implementation of gradient descent.

- **Adaptive Gradient Descent (AdaGrad)**: The learning rate is same for every dimension in the case of SGD algorithms. But it could be small in some direction and large in another. The AdaGrad [65] algorithm adaptively scale the learning rate for each of the dimension by following the update rule

$$\theta_{n+1} = \theta_n - \frac{\gamma}{\sqrt{\epsilon I + diag(G_n)}} \nabla f(\theta) \tag{4.5}$$

where G_n is given by

$$G_n = \sum_{T=1}^{n} \nabla_t \nabla_t^T \tag{4.6}$$

Hence, AdaGrad adaptively scales the learning rate at each iteration in each dimension with respect to the accumulated squared gradient.

- **RMSProp**: RMSProp is an extension to the gradient descent optimization algorithm with the concepts of AdaGrad. It uses the decaying average of partial gradients in the adaptation of the step size for each parameter. It is introduced by Hinton in his lecture titled ”rmsprop: Divide the gradient by a running average of its recent magnitude” [41].

- **SGD with momentum**: Momentum [82] helps to accelerate SGD in the relevant direction with the following update rule

$$\begin{aligned} v_n &= \eta v_{n-1} + \gamma \nabla f(\theta) \\ \theta &= \theta - v_n \end{aligned} \tag{4.7}$$

It does this by adding a fraction v of the update vector of the past time step to the current update

- **Adaptive Moment Estimation (Adam)**: The Adam [52] optimizer is described as a combination of AdaGrad and RMS prop. The adam also makes use of the average of the second moment of the gradients for updating the learning rate. The algorithm calculates an exponential moving average of the gradient and the squared gradient.

4.5 Backpropagation

Backpropagation [90] is the widely used algorithm for training neural networks. Every optimizer described in Section 4.4 uses the gradient of the loss function δ (One of the widely used loss function is the mean squared error given at 2.5) with respect to the weights and biases of the neural network $\frac{\partial \delta}{\partial w_l}$ and $\frac{\partial \delta}{\partial b_l}$.

Backpropagation computes the gradients with the help of the chain rule. The required gradient $\frac{\partial \delta}{\partial w_l}$ can be written as

$$\frac{\partial \delta}{\partial w_l} = \frac{\partial \delta}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial w_l} \tag{4.8}$$

Here the first term $\frac{\partial \delta}{\partial \hat{\mathbf{y}}}$ represents the derivative of the loss function δ with respect to the output $\hat{\mathbf{y}}$ of the network. The second term $\frac{\partial \hat{\mathbf{y}}}{\partial w_l}$ is the derivative of network output $\hat{\mathbf{y}}$ with respect to the weights w_l. This term can be further divided into smaller parts by using different layers and network activation function details.

$$\frac{\partial \hat{\mathbf{y}}}{\partial w_l} = \frac{\partial \hat{\mathbf{y}}}{\partial w_{l+1}} \frac{\partial w_{l+1}}{\partial w_l} \tag{4.9}$$

Neural network training is the process of updating the network parameters by performing optimization using the backpropagation algorithm. A few practical considerations for successful training are detailed in the following sections.

4.6 Bias-Variance tradeoff

Ideally, one needs to train a model that accurately captures training data and generalizes well to unseen data. Bias and variance are two sources of errors that prevent a model from generalizing beyond the training data. Bias is the difference between the average prediction of our model and the actual values we are trying to predict. A high-bias model makes poor training and test data predictions. Hence, a low bias is desirable. At the same time, a model with high variance predicts well on the training data but performs poorly on unseen data. So a low

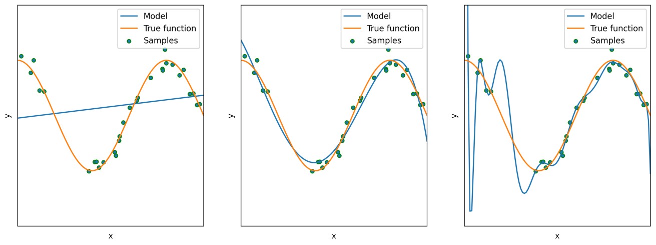


Figure 4.2: Underfit, perfect fit, and overfit models for the same dataset

variance is also preferred. However, reducing the bias increases variance. So, in any training, a bias-variance tradeoff is optimum. A model with very low bias and high variance is said to be overfitted to the training data. Similarly, a model with high bias and low variance is underfitted. Figure 4.2 shows an underfit, perfect fit and overfit models for the same dataset.

4.7 Overfitting and underfitting

Two significant concepts one needs to consider during training neural network models are overfitting and underfitting. Overfitting is a situation in data science when the statistical model exactly fits the training data and performs poorly on new data. Data augmentation, regularization, dropout, and early stopping are methods to prevent overfitting in neural network training. Underfitting is the situation where the model is unable to capture the relationship between input and output data. There will be a high error in training and testing for an underfit model.

4.7.1 Dropout

As explained in Section 4.7, overfitting is a severe problem faced by deep neural networks. Dropout [96] is a technique introduced to address this problem. The idea is to drop neurons randomly along with their connections from the neural network during training. This artificially creates different networks to be trained every epoch, which would have a different prediction for a given input. During testing, units are not dropped, and the network gives the average prediction of all the ”dropout” networks considered during training. The dropout significantly reduces overfitting and gives major improvements over other regularization methods. Srivastava et al. in [96] showed that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification, and computational biology, obtaining state-of-the-art results on many benchmark data sets.

4.7.2 Regularization

Neural networks work well for predicting outputs from inputs in the training dataset, but accurate prediction on other datasets can be challenging. The process of increasing the accuracy of predictions from inputs with a small variation from those in the training dataset is called regularization.

There are many regularization techniques that act on potential errors. One of those possible sources of error can be the overfitting of the network. An option is to apply a penalty to the loss function, called $L1$ and $L2$ regularization. This method penalizes having large values for the model weights, which typically increases the variability of the predictions. The cost function δ with regularization $L1$ and penalty factor λ results into

$$\tilde{\delta} = \delta + \lambda \|w\|_1 \tag{4.10}$$

where $\|\cdot\|_1$ is the first norm of all the weights used in the neural network. similarly, with regularization $L2$

$$\tilde{\delta} = \delta + \frac{\lambda}{2} w^T w \tag{4.11}$$

4.7.3 Early stopping

It is a regularization method used in iterative method-based training such as gradient descent. Every iterative method updates model parameters to improve its performance in each iteration. Early stopping stops

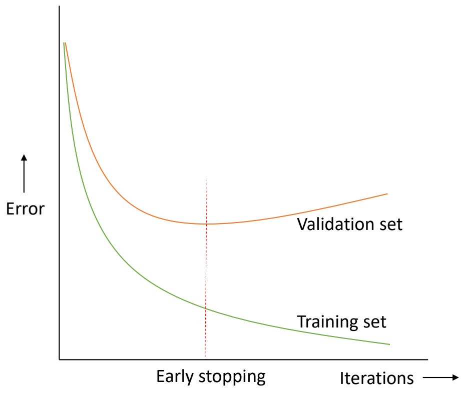


Figure 4.3: Early stopping identification

the training when parameter updates no longer improve the performance of the model on the validation set.

4.8 Batch normalization

Batch normalization [44] is an algorithm that makes the training of neural networks faster and more stable. The training process becomes complicated as each layer's input changes during training, as the previous layer change. This slows down the training and demands a lower learning rate. Batch normalization normalizes the output of hidden layers with the mean and variance of the present batch. A batch normalization layer determines the mean and the variance of the activation values across the current batch; then, it normalizes the activation vector to result in the neuron's output following a standard normal distribution across the batch. This normalization process allows the usage of much higher learning rates.

4.9 Vanishing and exploding Gradients

Vanishing and exploding gradients are significant problems while training a neural network using gradient-based learning methods and backpropagation. The gradient of weights and biases becomes vanishingly small in vanishing gradient problems. In contrast, the gradients become larger and accumulate, resulting in huge updates to weights and biases in the case of exploding gradients.

Vanishing gradients mainly occur in deep networks. When the network is more profound, the gradient gets multiplied again and again and approaches zero. It results in unchanged weights and biases and hence an unchanged model. One way to eliminate the vanishing gradient problem is to change from sigmoid activation to ReLU activation.

Exploding gradients are the problem of large gradient accumulation and large updates of the network parameters. This result in an unstable network which does not minimize the loss term. Methods such as gradient clipping [19] and weight regularization (Section 4.7.2) are used to fix the exploding gradients problem.

4.10 Hyperparameter tuning

Neural network parameters like weights and biases are adjusted to result in an optimum network that minimizes the loss. However, parameters like the number of layers and neurons in each layer are selected before feeding the data for training. Such parameters which are set for a given algorithm are called hyperparameters. Different hyperparameters result in different models for a given data set. Hyperparameter tuning finds a set of optimal hyperparameter values for an algorithm while applying this optimized algorithm to the given data set. Some of the hyperparameters in neural network training are

- **number of layers**: Making our network small with less number of layers makes it simple and generalizable. However, we may need to use deep networks with 10 or more layers for complex physics. It is recommended to start with $4-6$ layers and increase or decrease according to the accuracy.
- **number of neurons**: Even though the universal approximation theorem suggests having a larger number of neurons in a single layer, it is not possible to have too many neurons in a single layer.

Furthermore, having a large number of neurons makes the network memorize the data and results in poor prediction of new data.

- **Learning rate**: Learning rate controls how much the parameters like weights or biases change in each iteration. Keeping the learning rate low results in longer training time. At the same time, keeping its value high results in missing the minimum. Hence, we need to find the tradeoff between larger and smaller learning rates for the given problem.
- **Dropout percentage**: Dropout (Section 4.7.1) increases the generalization power of our neural network. A value between $20-50\%$ is generally used. A small value has no impact, and a high value will result in the underlearning of the network.
- **Activation function**: Activation functions also play an important role in the learning capacity of a network. Even though ReLU and Tanh are the commonly used activation functions, it is better to run them through hyperparameter search as some problems work well with only ReLu or Tanh.

It is also possible to add parameters like batch size, number of epochs, or momentum as a hyperparameter.

Different methods exist to find hyperparameters for a given problem and data set. Some of them are

- **Grid search**: A grid of possible discrete hyperparameter values is created, and the model is fitted with every possible combination. The model's performance for each set is recorded, and the one with the best performance is chosen. It is time-consuming, along with high computation requirements.
- **Random search**: Random search tries a random combination of hyperparameters and selects the one giving the best performance. It is appropriate when the number of hyperparameters has relatively large search domains. Random search typically requires less time than grid search to return a comparable result. Its drawback is that the resulting hyperparameters may not be the best possible combination.

- **Bayesian optimization**: Grid and random searches are inefficient as they do not consider the results of previous iterations. The bayesian optimization [94] considers the hyperparameter search as an optimization problem. It applies a probabilistic function to select the combination that will probably yield better results based on the previous iteration's results.

Chapter 5

Essentially, all models are wrong, but some are useful

George Box

FEM informed neural network

5.1 Introduction

As mentioned in Section 2.2, the number of elements used for discretization increases the cost of a numerical simulation. Solving the linear system of equations is the most time-consuming step in the Finite Element Methodology. It increases cubically against the number of degrees of freedom. Most linear solvers have a complexity between $O(n^2)$ and $O(n^3)$. Avoiding this step can increase the computational speed and can be used in situations where computational efficiency is critical, like real time simulations and executable digital twins. Hence, it is imperative that this step be avoided in such applications.

In the following, we discuss an algorithm that combines residual information from FEM to train a neural network for PDEs. We refer to the resulting surrogate model as the Finite Element Method enhanced Neural Network (FEM-NN).

5.2 Algorithm

The proposed algorithm results in a surrogate model for a parameterized PDE. Consider the general PDE that represents a physical problem, as given in Equation 2.1, for a general domain Ω, as illustrated in Figure 5.1.

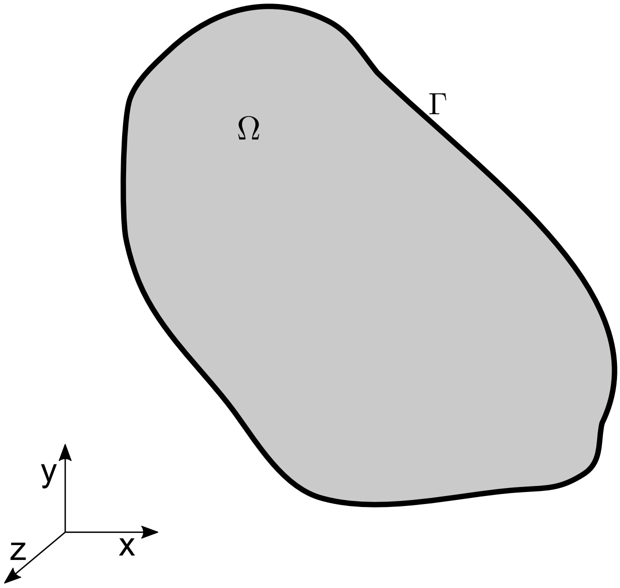


Figure 5.1: Physical problem in domain Ω with boundary Γ

The equation may include constant values for material properties, initial conditions, or boundary conditions. Considering some or all of them as parameters of the equation allows Equation 2.1 to be converted into

$$\begin{aligned} \mathcal{L}(u,\lambda) &= 0 \qquad \text{on } \Omega \\ u &= u_d \qquad \text{on } \Gamma_D \\ \frac{\partial u}{\partial \hat{\mathbf{n}}} &= g \qquad \text{on } \Gamma_N \end{aligned} \tag{5.1}$$

Here, λ may or may not include the Dirichlet and Neumann conditions. Depending on the requirements and complexity, some of the other options for λ include

- **Geometrical properties**: Examples are length of a structural beam, radius of a circular heat source or height of a high-rise building
- **Material properties**: Examples are density, specific heat, Young's modulus, or damping coefficient
- **Initial conditions**: Examples are initial temperature of the domain or initial velocity field
- **Boundary conditions**: Examples are inlet velocity, convection heat source at the boundary or fixed points of the structure

The block diagrams for training and deployment of the proposed hybrid surrogate model for the parameterized PDE are shown in Figure 5.2 and 5.3. The algorithm combines FEM and neural network to result in a surrogate model, that we refer to as FEM enhanced neural network hybrid model (FEM-NN). The training process before deploying it as a surrogate model for simulation is depicted in Figure 5.2. During the training process, the variables for simulation are taken as input parameters. This includes the parameters of the parametric PDE (λ), parameters of the neural network (θ) and constants of simulation (C). The input parameters are processed by both the neural network and the FEM library.

The neural network predicts the primary variable

$$\mathbf{\tilde{u}} = f_n(\lambda, \theta) \tag{5.2}$$

for a given sample the prediction is

$$\mathbf{\tilde{u}_i} = f_n(\lambda_i, \theta) \tag{5.3}$$

where λ_i is the i^{th} set of values for the parameters of the PDE for the given sample. The neural network outputs $\mathbf{\tilde{u}_i}$ after the forward pass

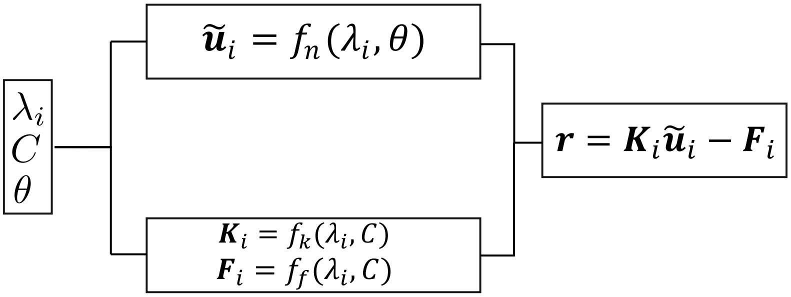

$\tilde{\boldsymbol{u}}_i = f_n(\lambda_i, \theta)$
λ_i
C
θ
$\boldsymbol{r} = \boldsymbol{K}_i\tilde{\boldsymbol{u}}_i - \boldsymbol{F}_i$
$\boldsymbol{K}_i = f_k(\lambda_i, C)$
$\boldsymbol{F}_i = f_f(\lambda_i, C)$

Figure 5.2: Training of FEM enhanced neural network

through the chosen network architecture. The output of the network is the discrete solution field vector $\mathbf{\tilde{u}_i}$ given as,

$$\mathbf{\tilde{u}_i} = \begin{pmatrix} \tilde{u}_1 \\ \tilde{u}_2 \\ \vdots \\ \tilde{u}_n \end{pmatrix}_i \tag{5.4}$$

The parameters λ_i and other constants required for a simulation are inputted into the simulation unit to derive and extract resulting stiffness matrix and force vector.

$$\mathbf{K_i} = f_k(\lambda_i, C) \tag{5.5}$$

$$\mathbf{F_i} = f_f(\lambda_i, C) \tag{5.6}$$

Here, $\mathbf{K_i}$ and $\mathbf{F_i}$ are the stiffness matrix and force vector for the parameters λ_i of the given sample. The C represents other constant values required for the simulation framework such as mesh size, number of nodes and time-step.

The residual $\mathbf{r}$ is calculated using the prediction $\mathbf{\tilde{u}_i}$ from the neural network and $\mathbf{K_i}$ and $\mathbf{F_i}$ from FEM. Loss for the neural network prediction is defined as Euclidean norm of the residual vector $\mathbf{r_i}$.

$$\delta = ||\mathbf{r_i}||_2 \tag{5.7}$$

where $\mathbf{r_i}$ is given by

$$\begin{aligned}
\mathbf{r_i} &= \mathbf{K_i}\mathbf{\tilde{u}_i} - \mathbf{F_i} \\
&= \begin{pmatrix} k_{1,1} & k_{1,2} & \cdots & k_{1,n} \\ k_{2,1} & k_{2,2} & \cdots & k_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ k_{n,1} & k_{n,2} & \cdots & k_{n,n} \end{pmatrix}_i \begin{pmatrix} \tilde{u}_1 \\ \tilde{u}_2 \\ \vdots \\ \tilde{u}_n \end{pmatrix}_i - \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}_i \\
&= \begin{pmatrix} k_{1,1}\tilde{u}_1 + k_{1,2}\tilde{u}_2 + .. + k_{1,n}\tilde{u}_n - f_1 \\ k_{2,1}\tilde{u}_1 + k_{2,2}\tilde{u}_2 + .. + k_{2,n}\tilde{u}_n - f_2 \\ \vdots \\ k_{n,1}\tilde{u}_1 + k_{n,2}\tilde{u}_2 + .. + k_{n,n}\tilde{u}_n - f_n \end{pmatrix}_i
\end{aligned} \tag{5.8}$$

This gives loss δ as,

$$\begin{aligned}
\delta &= ||\mathbf{r_i}||_2 \\
&= \sqrt{(k_{1,1}\tilde{u}_1 + .. + k_{1,n}\tilde{u}_n - f_1)^2 + ... + (k_{n,1}\tilde{u}_1 + .. + k_{n,n}\tilde{u}_n - f_n)^2} \\
&= \sqrt{\sum_{j=1}^{n}\sum_{i=1}^{n}(K_{j,i}\tilde{u}_i - f_j)^2}
\end{aligned} \tag{5.9}$$

As explained in Section 2.7, the learnable parameters θ are updated using the backpropagation algorithm to minimize the loss. Backpropagation calculates the gradients of the loss with respect to the learnable parameters θ. Since we use a custom loss function specific to FEM, we need to calculate the second part of the Equation 2.7, $\frac{\partial \delta}{\partial \mathbf{y}}$, for the custom loss used here. In the case of a hybrid model, the output $\mathbf{y}$ of the neural network is the predicted discrete solution field $\mathbf{\tilde{u}_i}$.

$$\mathbf{y} = \begin{pmatrix} \tilde{u}_1 \\ \tilde{u}_2 \\ \vdots \\ \tilde{u}_n \end{pmatrix}_i$$

The gradient has to be calculated with respect to each member of the output. $\frac{\partial \delta}{\partial \mathbf{y}}$ becomes,

$$\begin{pmatrix} \frac{\partial \delta}{\partial \tilde{u}_1} \\ \frac{\partial \delta}{\partial \tilde{u}_2} \\ \vdots \\ \frac{\partial \delta}{\partial \tilde{u}_n} \end{pmatrix}_i = \frac{1}{2\delta} \begin{pmatrix} k_{1,1}\tilde{u}_1 + k_{1,2}\tilde{u}_2 + .. + k_{1,n}\tilde{u}_n - f_1 \\ \vdots \\ k_{n,1}\tilde{u}_1 + k_{n,2}\tilde{u}_2 + .. + k_{n,n}\tilde{u}_n - f_n \end{pmatrix}^T$$

$$\begin{pmatrix} k_{1,1} & k_{1,2} & \cdots & k_{1,n} \\ k_{2,1} & k_{2,2} & \cdots & k_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ k_{n,1} & k_{n,2} & \cdots & k_{n,n} \end{pmatrix}$$

which gives

$$\frac{\partial \delta}{\partial \mathbf{y}} = \frac{\mathbf{r_i^T}\,\mathbf{K_i}}{\delta} \tag{5.10}$$

where $\mathbf{r_i^T}$ is the transpose of the residual vector $\mathbf{r_i}$ and $\mathbf{K_i}$ is the stiffness matrix of the i^{th} sample. We can train the network against the residual of the differential equation with the implementation of the above in the machine learning frameworks. A sample code for the implementation in the PyTorch framework is given in code 5.1. The second part of the Equation 2.7 is readily available in all the neural network frameworks like PyTorch or Tensorflow.

Once trained, the neural network is deployed using a similar hybrid approach for a new set of input parameters. The deployment process is depicted in Figure 5.3. The input variables are the parameters λ_i of the parametric PDE for the given sample, the learned parameters $\boldsymbol{\theta}^*$ of the neural network and the constants for simulation C. The trained neural network predicts the output $\mathbf{\tilde{u}_i}$ and the FEM gives $\mathbf{K_i}$ and $\mathbf{F_i}$. For a given sample the prediction is

$$\mathbf{\tilde{u}_i} = f_n(\lambda_i, \theta^*) \tag{5.11}$$

where θ^* is the optimized NN parameters resulted after training the neural network

In a conventional way of deployment of networks, the prediction accuracy is not measurable. However, here the output is used along with the stiffness matrix $\mathbf{K_i}$ and force vector $\mathbf{F_i}$ from FEM to calculate the residual $\mathbf{r_i}$ in Equation 5.8. The residual $\mathbf{r_i}$ is a measure of how much the output deviates from the actual solution of the governing equation. This way, prediction accuracy of FEM-neural network is quantifiable.

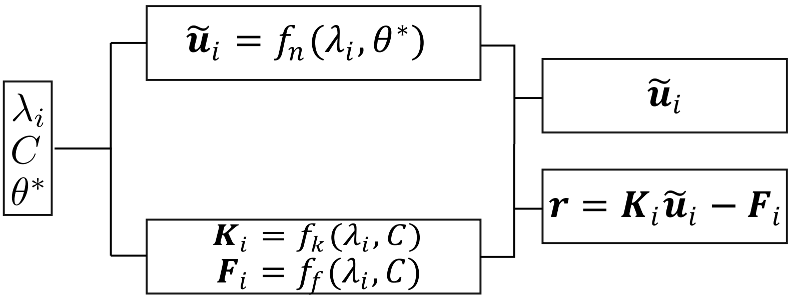

λ_i
C
θ^*
$\tilde{\boldsymbol{u}}_i = f_n(\lambda_i, \theta^*)$
$\boldsymbol{K}_i = f_k(\lambda_i, C)$
$\boldsymbol{F}_i = f_f(\lambda_i, C)$
$\tilde{\boldsymbol{u}}_i$
$\boldsymbol{r} = \boldsymbol{K}_i \tilde{\boldsymbol{u}}_i - \boldsymbol{F}_i$

Figure 5.3: Deployment of FEM enhanced neural network

```python
import torch
from torch.autograd import Function
import numpy as np

class Linear_residual_loss(Function):
    @staticmethod
    def forward(ctx, u:torch.tensor, K:torch.tensor, F:torch.
    tensor):
    """
    Forward pass of calculating residual
    Parameters:
        u: Output of your neural network
        K: Stiffness matrix
        F: Force vector
    """
        n_samples = u.shape[0]
        R = K.matmul(u) - F
        loss = torch.norm(R) / n_samples
        ctx.save_for_backward(K, R, loss)
        return final_loss

    @staticmethod
    def backward(ctx, grad_output:torch.tensor):
    """
    Backward pass of residual which return the gradient
    Parameters:
        grad_output: gradient till this function in the
     computational graph
    """
        K, R, loss = ctx.saved_tensors
        g_input = grad_output = None
        R_t = R.permute(0,2,1)
        g_output_c = R_t.matmul(K)/ loss
        g_out = g_output_c.permute(0,2,1)
        n_samples, shape_2 = g_out.shape[0], g_out.shape[1]
        g_out2 = g_out.reshape(n_samples, shape_2) / n_samples
        return g_input, g_out2, None, None
```

Listing 5.1: Implementation of forward and backward for residual based loss in PyTorch

The procedure for training and prediction are detailed in Algorithms 1 and 2.

5.3 Inverse problems

The algorithm introduced in Section 5.2 has been extended for inverse problems as well. Forward problems estimate the results for a defined cause, whereas inverse problems typically estimate the cause for the observed results. In such cases, the inverse problem is formulated as a parameter identification problem, where the unknown parameters of the forward problem are determined by minimizing an appropriate cost function. The estimation of unknown parameters results in correcting or updating the mathematical model used.

In the case of inverse problems for Equation 5.8, the primary variable $\tilde{\mathbf{u}}$ is known, whereas forces $\mathbf{F}$ or stiffness matrix $\mathbf{K}$ can have unknown parts. We consider the category where a stiffness matrix has unknown parts for the rest of the discussion. It is also possible to have unknown forces and its calculation also falls under the category of inverse problems. A problem having unknown parts in stiffness matrix can be described using the following equation

Algorithm 1: FEM-NN training for forward problems

```
    Read simulation parameters $\lambda$, $C$
    Select neural network architecture
    Initialize weights and biases
    Let $L$ be the number of layers in the neural network
    Initialize FEM Package with $C$
    $\mathbf{K_i} \leftarrow f_k(\lambda_i, C)$ and $\mathbf{F_i} \leftarrow f_f(\lambda_i, C)$ for all samples
    create $\mathbf{K}$ and $\mathbf{F}$ by assembling all $\mathbf{K_i}$ and $\mathbf{F_i}$
    while not Stop Criterion do
      $\tilde{\mathbf{u}}_\mathbf{i} \leftarrow f_n(\lambda_i, \theta)$
      Compute the residual $\mathbf{r} \leftarrow \mathbf{K}\tilde{\mathbf{u}} - \mathbf{F}$
1     Compute the loss $\delta = ||\mathbf{r}||_2$
      Compute the derivative $\frac{\partial \delta}{\partial \mathbf{u}} = \frac{\mathbf{r}^T \times \mathbf{K}}{\delta}$
      for all $l \in \{1, \dots, L\}$ do
        $\frac{\partial \delta}{\partial w_l} \leftarrow \frac{\partial \delta}{\partial \tilde{\mathbf{u}}} \frac{\partial \tilde{\mathbf{u}}}{\partial w_l}$
        $\frac{\partial \delta}{\partial b_l} \leftarrow \frac{\partial \delta}{\partial \tilde{\mathbf{u}}} \frac{\partial \tilde{\mathbf{u}}}{\partial b_l}$
        Update weights and biases
        $w_l = w_l - \eta \frac{\partial \delta}{\partial w_l}$
        $b_l = b_l - \eta \frac{\partial \delta}{\partial b_l}$
      end for
    end while
```

Algorithm 2: FEM-NN deployment for forward problems

```
    Initialize FEM Package
    Compute the system matrices $\mathbf{K}$ and $\mathbf{F}$
1   Predict $\tilde{\mathbf{u}} \leftarrow f_n(\lambda_i, \theta^*)$ using the trained neural network
    Compute the Residual $\mathbf{r} = \mathbf{K}\tilde{\mathbf{u}} - \mathbf{F}$
    return $\tilde{\mathbf{u}}$ and $\mathbf{r}$
```

$$\begin{pmatrix} \mathbf{K_k} & \mathbf{K_{ku}} \\ \mathbf{K_{uk}} & \mathbf{K_u} \end{pmatrix} \begin{pmatrix} \mathbf{u_k} \\ \mathbf{u_u} \end{pmatrix} = \begin{pmatrix} \mathbf{F_k} \\ \mathbf{F_u} \end{pmatrix} \tag{5.12}$$

where $\mathbf{K_k}$ is the known part and $\mathbf{K_u}$ the unknown part of the system matrix. $\mathbf{K_{ku}}$ and $\mathbf{K_{uk}}$ are the contribution of unknown part to the remaining DOFs of the system matrix. They are zero unless there is any physical connection between them. In cases having physical connections, those parts of the system also fall under the unknown category. Similarly, $\mathbf{u_u}$ are the responses corresponding to the unknown part of the system matrix and $\mathbf{u_k}$ are the responses at the rest of the system. It is to be noted that the responses are known in advance in contrast to the neural network prediction in the case of forward algorithm. Similar to the forward algorithms, the loss for the neural network prediction is defined as the Euclidean norm of the residual vector $\mathbf{r}$.

$$\delta = ||\mathbf{r}||_2 \tag{5.13}$$

where $\mathbf{r}$ is given by

$$\begin{aligned} \mathbf{r} &= \mathbf{K\tilde{u}} - \mathbf{F} \\ &= \begin{pmatrix} \mathbf{K_k} & \mathbf{K_{ku}} \\ \mathbf{K_{uk}} & \mathbf{K_u} \end{pmatrix} \begin{pmatrix} \mathbf{\tilde{u}_k} \\ \mathbf{\tilde{u}_u} \end{pmatrix} - \begin{pmatrix} \mathbf{F_k} \\ \mathbf{F_u} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{K_k\tilde{u}_k} + \mathbf{K_{ku}\tilde{u}_u} - \mathbf{F_k} \\ \mathbf{K_{uk}\tilde{u}_k} + \mathbf{K_u\tilde{u}_u} - \mathbf{F_u} \end{pmatrix} \end{aligned} \tag{5.14}$$

This gives loss δ as,

$$\begin{aligned} \delta &= \big||\mathbf{r}|\big|_2 \\ &= \sqrt{(\mathbf{K_k\tilde{u}_k} + \mathbf{K_{ku}\tilde{u}_u} - \mathbf{F_k})^2 + (\mathbf{K_{uk}\tilde{u}_k} + \mathbf{K_u\tilde{u}_u} - \mathbf{F_u})^2} \end{aligned} \tag{5.15}$$

In the case of inverse problems, the unknown part of the matrix is predicted using neural network. Hence,

$$\mathbf{K_u} = f_n(\lambda, \theta) \tag{5.16}$$

Algorithm 3: FEM-NN training for inverse problems

```
1   Read known simulation parameters $\lambda$ and $C$
    Select neural network architecture
    Initialize weights and biases
    Initialize FEM Package
    Let $L$ be the number of layers in the neural network
    while not Stop Criterion do
        Compute the matrices and vectors $\mathbf{K_k}, \mathbf{K_{ku}}, \mathbf{K_{uk}}, \mathbf{F_k}, \mathbf{F_u}, \mathbf{U_k}$ and $\mathbf{U_u}$
        $\mathbf{K_u} \leftarrow f_n(\lambda, \theta)$
        assemble known and unknown matrices to result $\mathbf{K}$
        Compute the residual $\mathbf{r} = \mathbf{K}\mathbf{U} - \mathbf{F}$
        Compute the loss $\delta = ||\mathbf{r}||_2$
        Compute the derivative $\frac{\partial \delta}{\partial \mathbf{K_u}} = \frac{1}{\delta}(\mathbf{K_{uk}}\mathbf{U_k} + \mathbf{K_u}\mathbf{U_u} - \mathbf{F_u})\mathbf{U_u}$
        for all $l \in \{1, \ldots, L\}$ do
            Compute the derivative using chain rule $\frac{\partial \delta}{\partial w_l} = \frac{\partial \delta}{\partial \mathbf{K_u}} \frac{\partial \mathbf{K_u}}{\partial w_l}$
            Update trainable parameters (weights and biases)
            $w_l = w_l - \eta \frac{\partial \delta}{\partial w_l}$
            $b_l = b_l - \eta \frac{\partial \delta}{\partial b_l}$
        end for
    end while
```

Algorithm 4: FEN-NN deployment for inverse problems

```
1   1: Initialize FEM Package
    2: Predict $\mathbf{K_u}$
    3: Assemble to result in system matrix $\mathbf{K}$
    4: Use $\mathbf{K}$ for forward simulation or other analysis
```

Similar to the calculation performed for forward problems, we need to calculate the derivative of the residual with respect to neural network prediction to perform the backpropagation. In the case of Equation 5.15 it is $\frac{\partial \delta}{\partial \mathbf{K_u}}$.

$$\frac{\partial \delta}{\partial \mathbf{K_u}} = \frac{1}{\delta}(\mathbf{K_{uk}}\mathbf{\tilde{u}_k} + \mathbf{K_u}\mathbf{\tilde{u}_u} - \mathbf{F_u})\mathbf{\tilde{u}_u} \tag{5.17}$$

Equation 5.17 is used along with the second part of the Equation 2.7 to update the neural network parameters during training the neural network to identify the unknown part of the matrix. The procedure for training and prediction for inverse problems is detailed in Algorithms 3 and 4.

5.4 Variants of the FEM-NN algorithm

5.4.1 Transient solver

The simulation of transient problems are conventionally started from the initial condition. Then the next time values for primary variables are calculated using any time stepping scheme like Runge-Kutta or Euler. The FEM-NN algorithm introduced in section 5.2 is extended to transient problems as well. For example an explicit time-integration scheme will result an equation of the form

$$\mathbf{A}\mathbf{\tilde{u}_{t+1}} = \mathbf{\tilde{u}_t}$$

here the residual can be written as

$$\mathbf{r} = \mathbf{A}\mathbf{\tilde{u}_{t+1}} - \mathbf{\tilde{u}_t} \tag{5.18}$$

The loss is given by

$$\delta = ||\mathbf{A}\mathbf{\tilde{u}_{t+1}} - \mathbf{\tilde{u}_t}||_\mathbf{2} \tag{5.19}$$

Neural network surrogates can be trained for transient problems using the above loss function. However in practice the training is much harder to converge as the values at times $t+1$ and t are neural network predictions. Examples for transient problems using the Equation 5.19 is given in Section 7.1.3.

5.4.2 Physics guided timeseries predictor

The second algorithm explores the possibility of conducting the transient simulation using a neural network after completing n timesteps via numerical methods.

The algorithm for training the system is illustrated in Figure 5.4. Initially, the simulation runs for the first n timesteps using FEM, during which the corresponding stiffness matrix $\mathbf{K}$ and force vector $\mathbf{F}$ for each timestep are recorded. The input Z to the system comprises both the system parameters and numerical method specific parameters of the problem. A subset of these parameters, which varies for each timestep, along with the responses of the system for the last three timesteps, serve as the input to the neural network. The matrices from FEM and prediction $\mathbf{x}$ from the neural network is utilized in the FEM-NN loss function to train the model. The algorithm employs an equation akin to Equation 5.19, with the difference being that data for timestep t is known and only the data for timestep $t+1$ is predicted. Once trained, the model is deployed to predict the simulation from timestep $n+1$ onwards (Figure 5.5). Further details of the algorithm can be found at Meethal et al. [72]. Examples utilizing this algorithm are provided in Section 7.1.3.

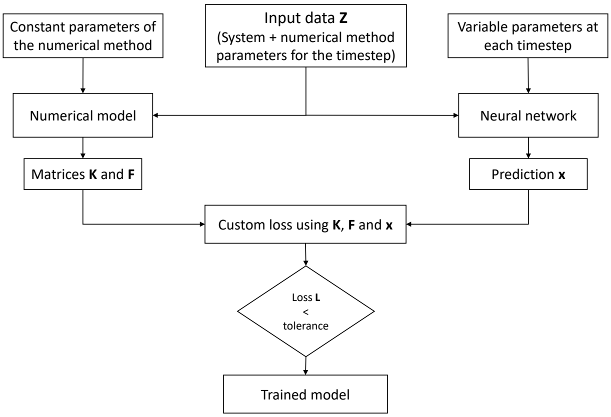

Constant parameters of the numerical method
Input data **Z** (System + numerical method parameters for the timestep)
Variable parameters at each timestep
Numerical model
Neural network
Matrices **K** and **F**
Prediction **x**
Custom loss using **K**, **F** and **x**
Loss **L** < tolerance
Trained model

Figure 5.4: Training of physics guided timeseries predictor

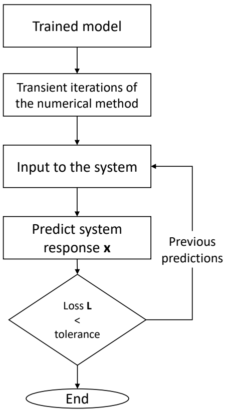


Figure 5.5: Deployment of physics guided timeseries predictor

5.4.3 PINN and FEM-NN hybrid learning

Consider the physical problem governed by a partial differential equation in domain Ω as in Figure 5.1. The PINN introduced by Raissi et al in [85] embed the physics in the form of the PDE into the loss function of the neural network using automatic differentiation. In PINNs, the solution to the equation 5.1 is approximated by a feed-forward neural network $\bar{\mathbf{U}}(x, t, \lambda; \theta)$ using the loss function in Equation 2.12. The loss term L_f enforces the equation given by 5.1 on a set of random points called collocation points $\{t_f^i, x_f^i\}_{i=1}^{N_f}$ and loss term L_{MSE} enforces the initial and boundary training data $\{t_b^i, x_b^i, u_b^i\}_{i=1}^{N_b}$. Figure 5.6 shows the collocations and boundary points for the general physical problem under PINN.

The collocation points and boundary points can be connected to form a mesh as in Figure 5.7, that can be used for deriving FEM formulation. In another words, the nodes from an FEM can be treated as the collocation and boundary points for the PINN training.

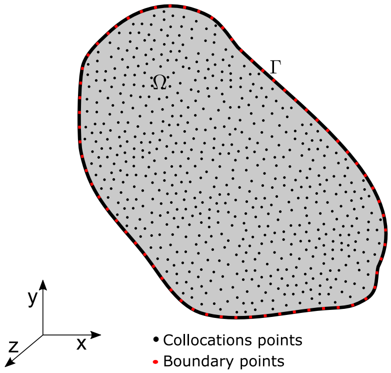


Figure 5.6: Physical problem in domain Ω with collocations and boundary points for PINN training

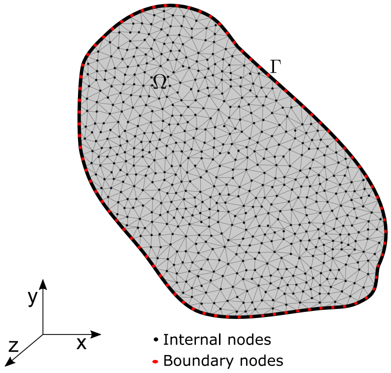


Figure 5.7: Physical problem in domain Ω with internal and boundary nodes for FEM formulation

In this PINN and FEM-NN hybrid learning approach, we combine the FEM-NN loss along with other two loss terms for a neural network training. We call the FEM-NN loss term as L_r

$$L_r = ||\mathbf{K}\tilde{\mathbf{u}} - \mathbf{F}||_2 \tag{5.20}$$

In order to achieve this, one needs to predict the solution at nodal points of the discretization of the numerical method used. We can achieve this by selecting nodal points as the collocation points of PINN training. Total loss for training the neural network is

$$\begin{aligned}
L &= L_p + L_r \\
L &= L_f + L_\Gamma + L_r \\
&= \frac{1}{N_f}\sum_{i=1}^{N_f} |\mathcal{L}(\tilde{u}^i)| + \frac{1}{N_b}\sum_{i=1}^{N_b} |\tilde{u}(t_b^i, x_b^i) - u_b^i| \\
&+ ||\mathbf{K}\tilde{\mathbf{u}} - \mathbf{F}||_2
\end{aligned} \tag{5.21}$$

Adding the third loss term which is based on the discretized physics helps the PINNs to learn quickly and accurately. The ill-posedness of PINNs is addressed with this third term. The examples and advantages of this combination are given in Chapter 7.

Chapter 6

The purpose of software engineering is to control complexity, not to create it

Pamela Zave

Kratos neural network application

Developments combining simulation and neural networks raise the opportunity to introduce neural network models in coupled multiphysics simulations. Coupled multiphysics simulations appear in many engineering designs and are one of the challenging areas for engineers. Some notable examples where coupled multiphysics systems analysis is essential are wind turbines, flutter of airplane wings, bridge excitation and collapses due to wind, and risers in offshore applications. Some known problems associated with simulations of such coupled multiphysics problems, especially Fluid Structure Interaction (FSI), are the accuracy, stability, computational cost, and more considerable computing time. Introducing neural network models to replace one or more coupled solvers can significantly reduce the computing time and, hence, become cheaper. Neural network surrogates can also be used to model part of the simulation that lacks a mathematical model but has enough data.

In this contribution of the dissertation, a software application as part of Kratos multiphysics [23] solver to use neural network models for coupled multiphysics simulations is developed. To avoid conflicts with other parts of the simulation, an interface for the coupling of neural network-based surrogate models with other solvers is implemented. The framework also supports the generation of training data from simulation as well as training neural network models using some of the famous libraries. The implemented methodologies for data generation and training are tested on Computational Solid Mechanics (CSM) and CFD problems and the interface is tested on FSI benchmark problems.

A framework to enabling all the above mentioned points need the software to follow the design principles of software development. We made sure that the framework avoid rigidity, fragility and immobility. For example, one major rigidity aspect that occur is the dependency on any neural network package. But the software is designed in such a way that any new neural network package can be easily integrated to the current framework. Presently both Keras [21] and PyTorch [79] frameworks are integrated. The developed application has three parts data generation, neural network training and cosimulation interface (Figure 6.1). They all are implemented under the neural network application in Kratos.

Kratos neural network application

Kratos is an open-source finite element based code developed in collaboration of Chair of Structural Analysis, TU Munich and International Center for Numerical Methods in Engineering (CIMNE) of the Polytechnic University of Catalonia, BarcelonaTech. It is “a framework for building parallel, multi-disciplinary simulation software, aiming at modularity, extensibility, and high performance” [23]. The term multi-disciplinary refers to the different applications for different physical problems. An example application is the FSI application in the Kratos. Implementation of new finite element applications and interaction between them on a common platform is possible with Kratos with little effort due to the modularity structure it offers.

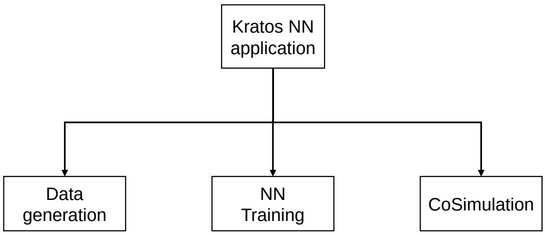


Figure 6.1: Parts of the neural network application in Kratos

6.1 Neural network application

6.1.1 Data generation

One of the evident applications of combining simulation and Neural networks are the data generation for neural networks using simulation. So, a data generation process is created as part of this application to generate training data. The generated data can be used by Kratos or any external application.

The data generation process starts by creating a standard Kratos simulation of interest and adding an extra process to the simulation. This additional process is called the *data generation process* and it enables the user to define different aspects of data generation. Users can select the part of the model for which input or output data is to be generated. Users can also provide the distribution from which input data is generated. Presently distributions like normal and binomial are supported. New algorithms for sampling can be easily added to the framework. Both input and output data can be either the nodal values or any process data that Kratos supports. An example of nodal value is the force applied on a structure at a given location. An example for a process data is the inlet velocity of a fluid simulation which is defined as "inlet velocity process" in Kratos. Similar to the input data, output data can also be specified as nodal or process. Nodal output value can be the displacement or pressure at a given node. The process output value can be the drag coefficient resulting from the "calculate drag coefficient process". The algorithm followed by the data generation process is explained in the algorithm 5.

Algorithm 5: Data generation process in Kratos NN application

```
1
  1: Procedure - DATA GENERATION:
  2: Initialize Analysis
  3: Initialize Perturbation
  4: Select input model part/s from all available model parts
  5: Select input data source(solution step, nodal value, or process)
  6: Select output model part/s from all model parts
  7: Select output data source(solution step, nodal value, or process)
  8: $d, d_v$ ← input variable distribution and its parameters
  9: While not End of simulations do
 10:     New input and save input variables
 11:     Initialize FEM analysis
 12:     $u$ ← calculate outputs
 13:     $f$ ← calculate derived outputs
 14:     save output variables
 15: end While
```

The Figure 6.2 shows the simulation of a cantilever beam exerted by force on top. A surrogate model giving the displacement of the tip of the beam for the given force on top is of interest. The data required for that are generated using the "data generation process" by giving force as input and tip displacement of point A as output. The distributions of the input data sampled from normal and uniform distributions are given in Figure 6.3.

6.1.2 Neural network training

Neural network training is the second part of the neural network application in Kratos. The core part of the neural network training is the *neural network analysis* implemented in Neural network application. Neural network analysis contains the preprocessing, setting up model,

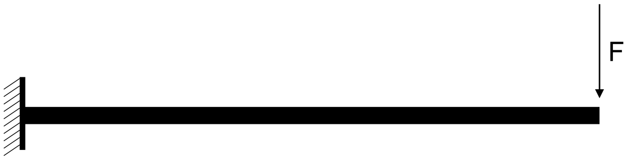
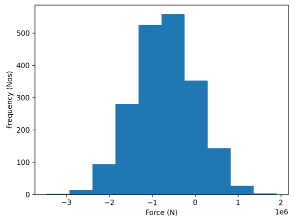
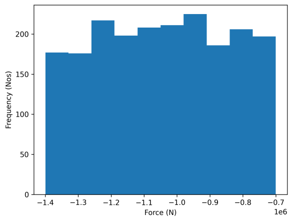


Figure 6.2: Cantilever beam with fixed support and point load



(a) Normal



(b) Uniform

Figure 6.3: Input data generated using normal and uniform distribution

training, testing and post-processing parts. An overview of the neural network analysis and its different parts and functionalities are given in Figure 6.4.

The data generated from data generation process or other source are to be pre-processed first. The data pre-processing part in the neural network application can perform three functionalities normalisation, masking and look back setup. Normalisation takes care of normalising the input data to the prescribed range of $[0, 1]$ or $[-1, 1]$. It is also possible to define custom ranges like $[0.2, 0.8]$ to handle outliers easily. This is since there can be situations in multiphysics simulation where the data during prediction is not falling in the trained region. Normalisation helps the optimization algorithms to converge faster.

There can be situations where the output vector has a few identical elements for any given input. For example, the Dirichlet boundary condition we assign is known in prior and can be same in every output. Such elements can be masked so that neural network need not predict

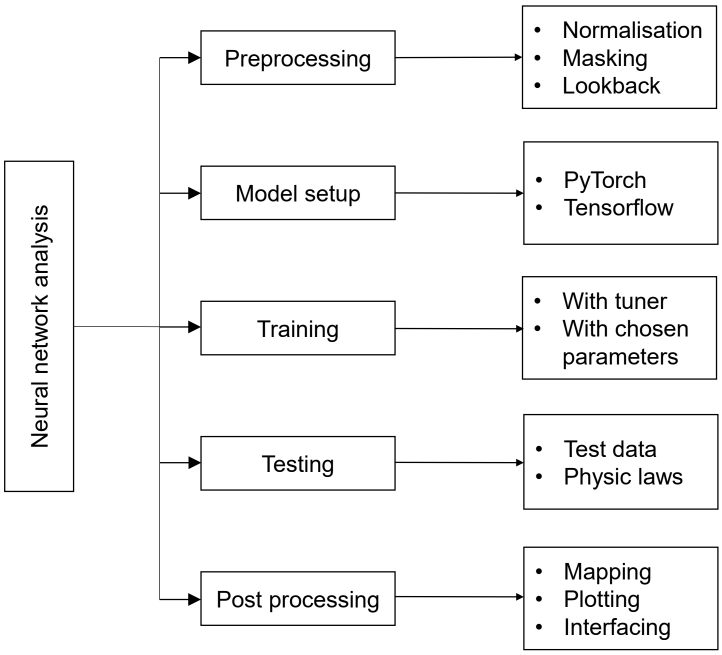


Figure 6.4: Components of neural network analysis inside Kratos neural network application

them. This is done with the help of the masking process in the pre-processing. Such masking process helps the neural network to learn the rest of the output better and faster. The purpose of *lookback setup* is to deal with time-series data. Normally we use last n-timestep values for timeseries analysis and model training. The lookback setup is used for such purposes by specifying number of timesteps to look back.

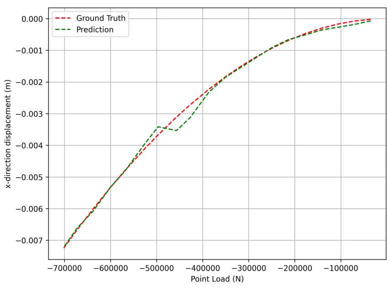
After preprocessing the data, neural network is setup with the help of standard libraries. Presently both Keras and PyTorch libraries are supported. But the application is designed such a way that the addition of a new library can easily be done. User can also select what kind of network to use and select the parameters accordingly. For examples, one may use LSTM for time-series prediction or CNN for predicting the velocity distribution in a domain. Once the model setup is done we can train the model using training process. In the training process once can either train using a known set of parameters for number of neurons in each layer, filter size, depth of the network, learning rate and dropout. If the parameters needed to be tuned, the tuner process can be employed. The tuner process uses the Keras tuner or Optuna tuner based on the user input.

After the training, testing and post-processing can also be done with the help of *neural network analysis*. In the testing, conventional neural network testing methods can be done using the test data prepared. Neural network application plots the prediction and corresponding error on the test data to inspect the training accuracy. Figure 6.5 shows the prediction and the corresponding error for the cantilever beam example. One also have the opportunity to cross-validation with the help of the framework. The integrated nature with the Kratos help to verify the physical conformity of the results if the network is designed to predict the complete solution on every nodes. Then corresponding stiffness matrices are constructed and solution accuracy is tested by calculating the residual associated.

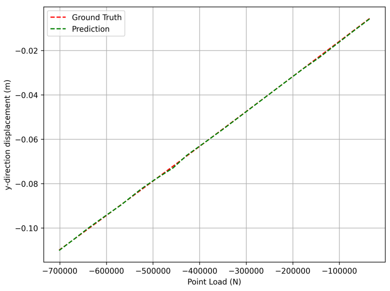
In the post-processing part, different metrics are plotted to analyse the training process. User can choose to visualise different metrics to verify the stability of training process. There are also options like plotting the Fast Fourier Transform (FFT) of the prediction to better analyse the prediction for cases like fluid simulations. It is also possible to map the prediction results onto the geometry with the help of Kratos modules. This enables the visualisation and other post processing with the help of softwares such as paraview and GiD accessible. Interfacing the results to use along with other Kratos applications is also possible. This part is detailed in the deployment section below.

6.1.3 Deployment

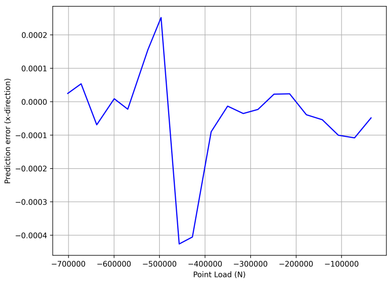
The generated neural network from the training can be directly used to predict output for the corresponding inputs. As an example one can train a surrogate against inlet velocity and coefficient of drag and use the trained model to predict coefficient of drag for an unseen inlet velocity.



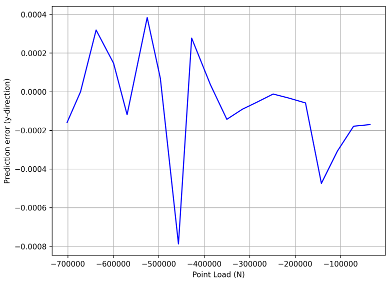
(a) x-direction displacement prediction



(b) y-direction displacement prediction



(c) x-direction displacement prediction error



(d) y-direction displacement prediction error

Figure 6.5: Prediction and error from the trained neural network for cantilever beam

But, when it comes to simulation, especially in multiphysics simulation, the purpose is to use it along with other elements of the simulation. One use case of such generated network will be to simulate a single physics simulation against time. In that case we need to predict the output for a particular position for the entire duration of the simulation. Another use case will be the use in a multiphysics simulation. In a partitioned multiphysics simulation, neural network receives input from the other simulation solvers and need to output data to them. To enable the smooth and flexible use of neural network surrogate, we use it as a black box solver for the particular physics. This way, we can use it along any existing physics just by calling neural network solver. A workflow explaining the use of the surrogate model as a solver is shown in Figure 6.6.

In Section 6.2 we discuss some of the examples using the Kratos neural network application. Only few examples are considered here, detailed study can be found in [6].

6.2 Examples using Kratos NN application

6.2.1 Static non-linear diamond shape

In this example we consider a static non-linear diamond shape as given in Figure 6.7. The features of the example also helps in understanding how we can use neural network models for symmetric problems. The analytic solution for the benchmark is provided in Mattiasson [67] and the implementation in Kratos multiphysics, which is used for generating the data, is validated in Sautter [92]. The data generation follows the procedure detailed in section 6.1.1. The input variable is the load F and the output variables are displacements u and v. The forces are acted on the top and bottom corners of the structure with a magnitude of $2F$. The beam parameters are $E = 210 \times 109 N/m^2$, $\rho = 7850 kg/m^3$, $A = 0.01 m^2$ and $I_z = I_y = 0.00001 m^4$. A total of 500 data points are gathered with values of F varying from 0 kN to -7 kN. Only half of the system is used the structural simulation utilizing the symmetry condition. For testing, the data is again generated using a random normal distribution with mean 0 and standard deviation 0.1. The neural network architecture is a fully connected network with $128, 64, 16, 4, 2$ number of neurons from second to output layers. In the data generation process, the load is recorded in both the nodes where displacements u and v are observed. This results in duplicated data with opposite signs. The duplication could be avoided with a redefinition in the geometry file of the part we use for training. Instead of the redefinition, the masking process implemented in the data preprocessing is used in this example.

The results obtained in this benchmark are close to the ground truth values (see Figures 6.8). Both the u and v predictions follow the benchmark curve for the whole test region. The error is very small compared to the actual displacement. It can also be observed that the use of symmetry for the problem has no effect on the neural network

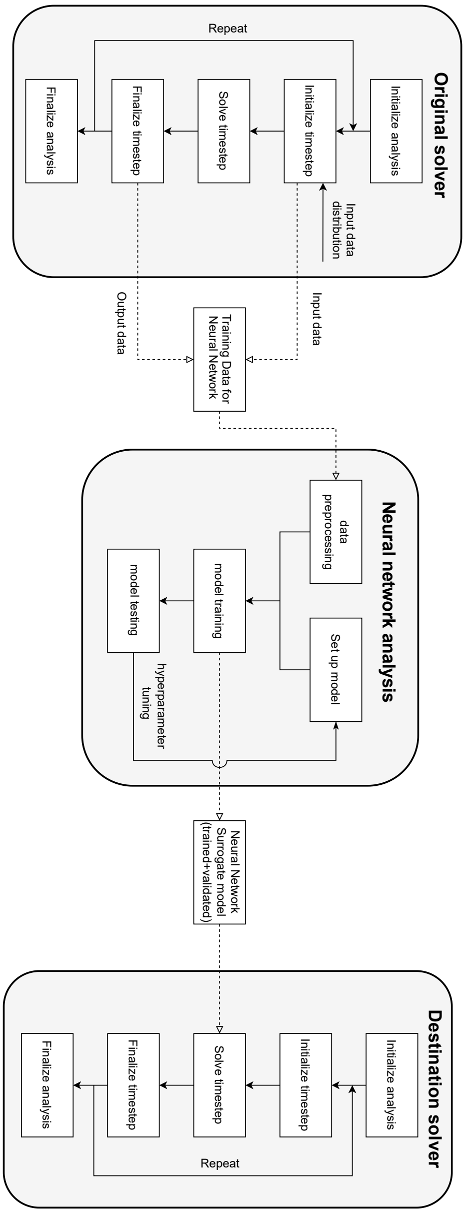


Figure 6.6: Workflow of neural network application for a coupled simulation

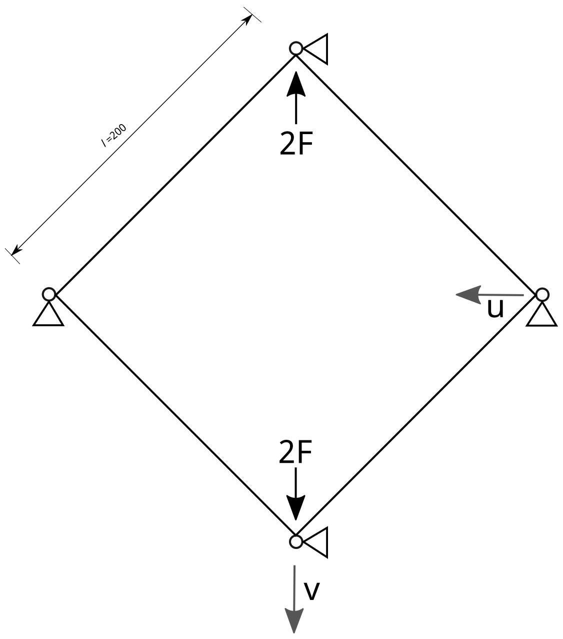


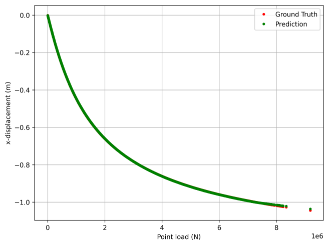
Figure 6.7: Diamond shaped non-linear structure

results. It also reiterates that the neural network only consider the data it is trained with. Which is a strength and weakness of the pure data driven approach. Depending on the use case, this lack of physics understanding can be alarming.

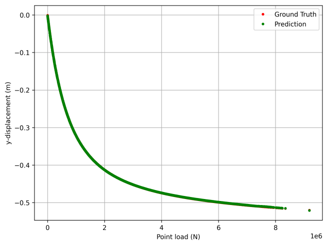
6.2.2 Fluid-Structure Interaction problems

In this example[1] we consider the deployment of the neural network application for a multiphysics coupled problem. We consider an FSI problem with either of the fluid or structural solver replaced with the

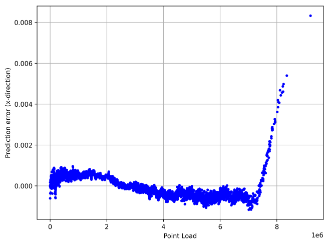
[1] The following section is based on [6, 70]. The main scientific research as well as the textual elaboration of the publication were performed by the authors of this work



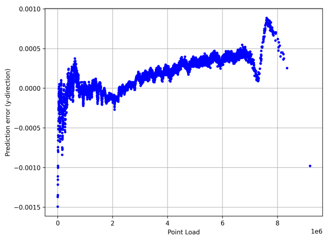
(a) x-direction displacement



(b) y-direction displacement



(c) x-direction prediction error



(d) y-direction prediction error

Figure 6.8: Prediction and error for the non-linear diamond shape structure

neural network model. We consider the standard Mok's [74] benchmark problem to demonstrate the ability of Kratos neural network application for simulating multiphysics problems.

Mok's benchmark consists of a flexible wall in a channel flow. The flexible wall is fixed at its base and is displaced by the force of the fluid flow. The fluid is modeled using an Arbitrary Lagrangian Eulerian (ALE) formulation, where the wall displacement is reflected as a modification of the fluid domain boundary conditions. The velocity and pressure of the fluid are affected by the fluid near the interfaces. Significant interactions exist between the convergent flow and the flexible wall as a result of having the same order of magnitude for the densities. It results in a strongly coupled FSI problem. The first reference solution to this problem is provided by Mok (2001) [74]. More refined studies are conducted by Valdés [113] in 2007. This particular benchmark problem is already implemented in Kratos and the results obtained are closer to

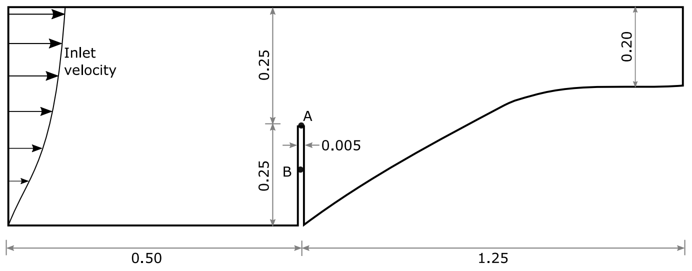


Figure 6.9: Mok's benchmark of flexible wall structure in a convergent fluid channel

the results from Valdés. This implementation in Kratos is used as the basis to validate our tests using neural network application.

In the benchmark, the inlet velocity has a parabolic profile with

$$v(y,t) = 4\bar{v}y(1-y) \tag{6.1}$$

Where $\bar{v} = \frac{0.06067}{2}\left(1-\cos\frac{\pi t}{10}\right)$ m/s in the first 10 seconds (the ramp-up phase) and $\bar{v} = 0.06067$ m/s otherwise. The fluid domain is considered with a Newtonian law having a density of $\rho = 956$ kg/m^3 and a kinematic viscosity of $\nu = 0.145$ m^2/s. The structure follows the linear elastic plane stress constitutive law with density $\rho = 1500$ kg/m^3, Young's modulus $E = 2.6\times10^6$ Pa and Poisson's ratio $\nu = 0.45$. A timestep of 0.1 seconds is chosen for the 20.0 seconds simulation.

The results are recorded for the two points at the fluid-structure interface. Point A is at the top of the structure, and Point B is in the center (shown in figure 6.9). The Gauss-Seidel scheme is used to test both the weak or explicit coupling and the strong or implicit coupling cases. The structure's surface is defined as the interface for information exchange. The reaction from the fluid is synchronized as the load on the structure. The structure displacement is correspondingly coupled with the mesh displacement of the fluid. Both the fluid and structural solvers use identical timescales. The fluid and structure domains have coinciding meshed at the interface. Each fluid node has a coincident node from the structure at the same position.

Structural surrogate

In this case, we replaced the structural solver with a neural network-based surrogate model. Similar to conventional FSI, the forces on the interface from the fluid model are mapped to the structural model. Instead of solving the whole structural problem with the corresponding solver, here the neural network solver reads the loads and predicts the displacements at the nodes. The loads at the nodes are taken as input after preprocessing them. The predicted displacement from the neural network solver is also stored at the nodes. Afterwards, the new coupling step maps the node displacements of the structure model part as the fluid mesh displacements. The data for training (point loads and displacements) are generated from structural model following the method explained in 6.1.1.

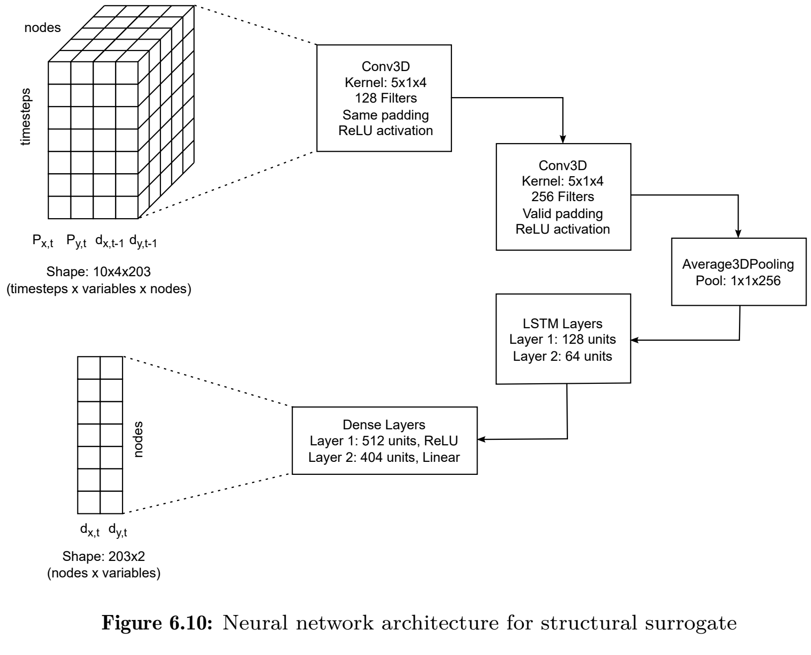
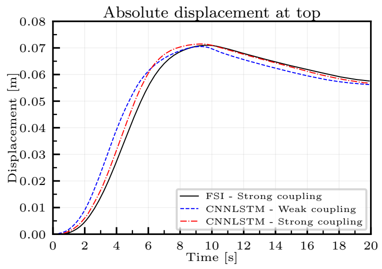


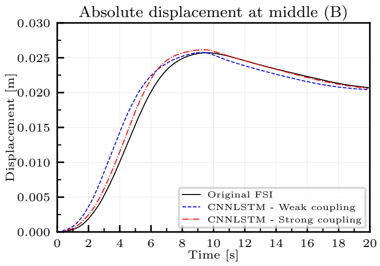
Figure 6.10: Neural network architecture for structural surrogate

The final neural network architecture after hyperparameter tuning is shown in Figure 6.10. The loads and previous timestep displacements at each node in $x-$ and $y-$ directions are taken as input. The variables $[P_{X,t}, P_{Y,t}, d_{X,t-1}, d_{Y,t-1}]$ represents the load at the given timestep P_t and the displacement of previous timestep d_{t-1}. A total of 200 data points with 10 lookback timestpes makes the final input shape $(200, 10, 4, 203)$. Hence, a 3D-CNN LSTM was used as the neural network surrogate. The output tensor is composed of the displacements d_t in both directions. The neural network is trained as explained in Section 6.1.2.

The Figures 6.11a and 6.11b shows the displacement of points A and B on the flexible wall. The displacement prediction from the benchmark is compared against both strong and weak coupling using the neural network surrogate. It can be observed that the displacement predictions are close to the benchmark in both cases. The error in the strong coupling is smaller than the error in the weak coupling. Both the points A and B follow the same behaviour.



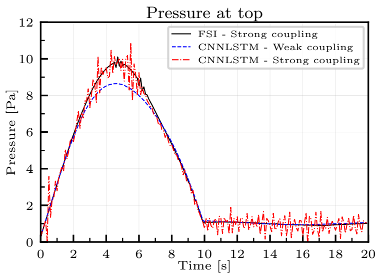
(a) Point A



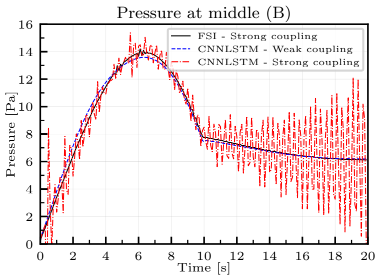
(b) Point B

Figure 6.11: Comparison of displacement of points A and B on the wall for FSI Mok benchmark using structural surrogate

In contrast to the trend in predicting displacement, the pressure prediction showed more error when comparing with the benchmark. The structural surrogate model when used with strong coupling results in significant instabilities in the pressure. The pressure can be seen fluctuating around the benchmark results. This can be attributed to the unphysical displacements predicted by the neural network for the wall. It is also observed that the instability increased after the ramp-up phase and diverges later. In general, LSTM layers are suitable for transient



(a) Point A



(b) Point B

Figure 6.12: Comparison of pressure at points A and B on the wall for FSI Mok benchmark using structural surrogate

and oscillatory signals, but they introduce errors when the input and the output vary relatively little. These errors are accumulated with time, leading to an eventual loss of convergence. However, this instability is not present in weak coupling. But there is a pressure drop of upto 14% at the point A when compared with benchmark.

Fluid surrogate

In this second case, we replace the fluid with a surrogate model. Since fluid simulations are computational intensive, this approach is more promising than replacing the structural model. Similar to the structural model, we start with generating the input-output data required for training the neural network. The velocity at the inlet and the displacement at the fluid-structure interface are the relevant boundary conditions that vary at every timestep. This is taken as input to the model to predict the reaction forces at the interface. The predicted reaction forces are mapped afterwards to the structural model.

The neural network architecture after the hyperparameter tuning is similar to the structural surrogate. As shown in Figure 6.13, it is a combination of 2D-convolutional layer with LSTM layers. Since the input data are coming from two model parts (inlet and interface), it is not possible to group them using nodes. They are simply combined and

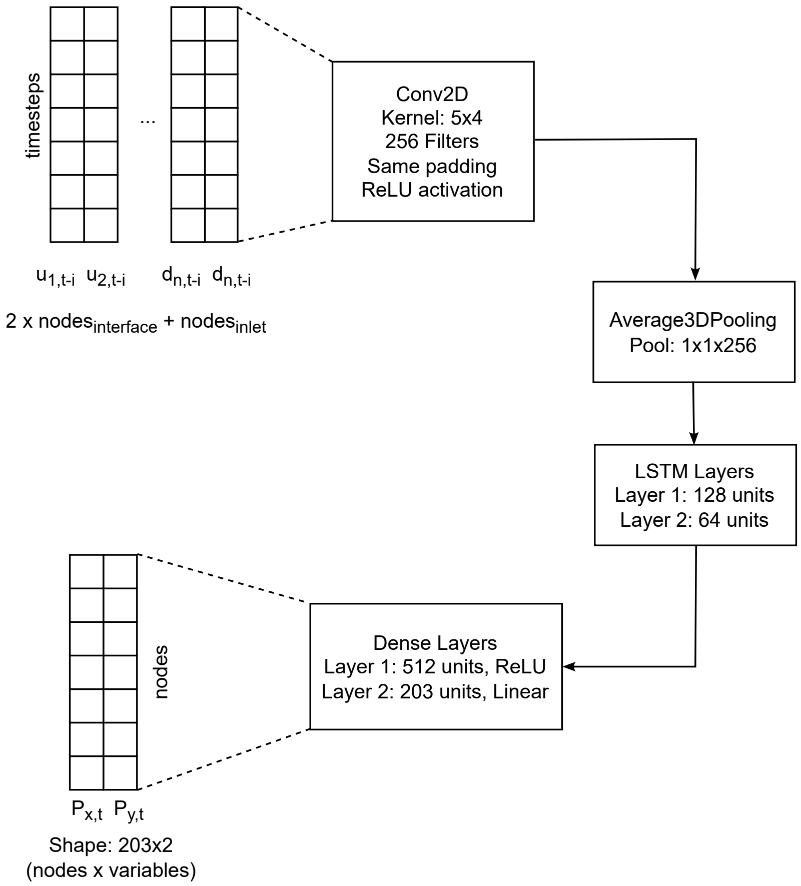
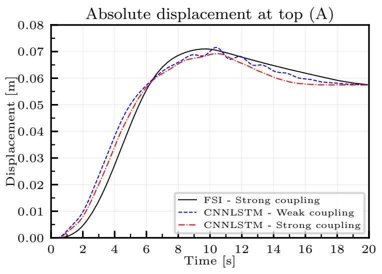


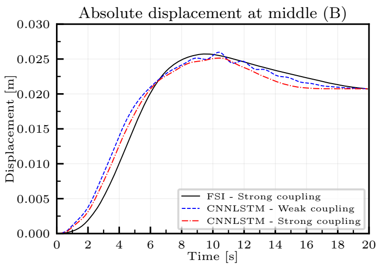
Figure 6.13: Neural network architecture for fluid surrogate model

used as input to the network. This results in an input tensor of shape $(10, 854)$, with the total training dataset shape being $(200, 10, 854)$.

The displacement of points A and B can be observed in Figure 6.14a and 6.14b. Both the Weak and strong coupling schemes using a fluid surrogate is analysed. The results are close to the benchmark in both the cases and for both points. However, there is an oscillatory



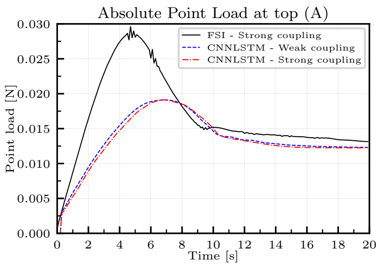
(a) Point A



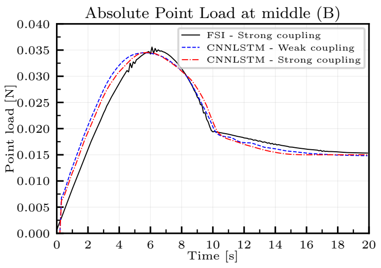
(b) Point B

Figure 6.14: Comparison of displacement at points A and B on the wall for FSI Mok benchmark using fluid surrogate

behavior when using weak coupling in the stabilization phase. The solution loop is run only once per timestep with weak coupling, which amplifies any inaccuracy produced by the neural network. In such a case, the variation in the loads causes oscillations in the system, giving results to the pattern observed. When it comes to the prediction of point load, the difference is very high for point A. This result might be due to the inaccuracies in the displacement as the point loads are affected by the rotation of the structure. This conclusion is supported



(a) Point A

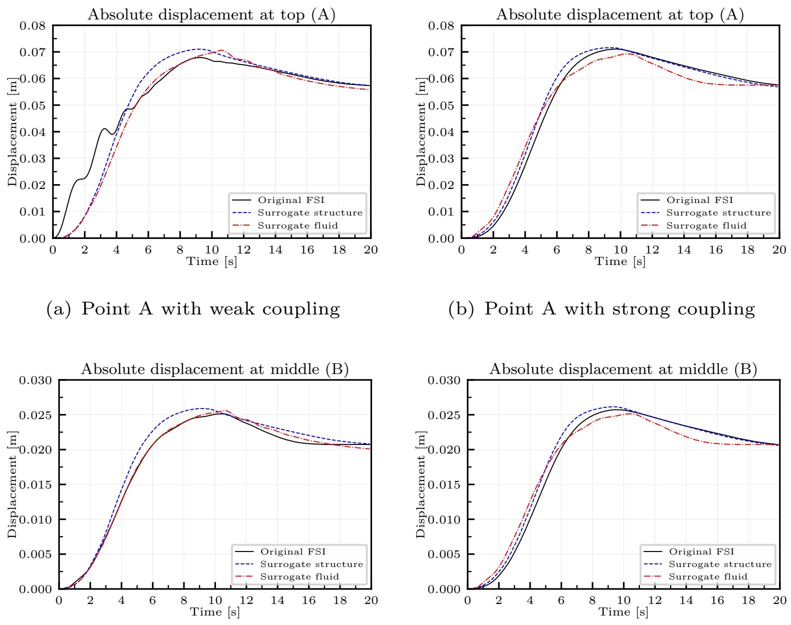


(b) Point B

Figure 6.15: Comparison of point load at points A and B on the wall for FSI Mok benchmark using fluid surrogate

by the point load prediction plot for point B. Point B predictions are more accurate in both strong and weak coupling. It is to be noted that, there is no significant difference between the point loads in weak and strong coupling. This might be due to the reason that they are a direct result of the predictions of the neural network which is trained using strong coupling data.

Comparison between structural and fluid surrogates



(a) Point A with weak coupling (b) Point A with strong coupling

(c) Point B with weak coupling (d) Point B with strong coupling

Figure 6.16: Comparison of displacement of points A and B on the wall for FSI Mok benchmark

This section compares displacements of the interface and the total time taken for simulation for the surrogate model-based simulations against the benchmark. A comparison between the structural surrogate and the fluid surrogate is made in this section.

Figure 6.16 shows the displacements of points A and B from the surrogates and the benchmark. In general, the surrogate models predict values close to the benchmark. Structural surrogate models are closer to the benchmark, whereas fluid surrogates result in undesirable oscillations. It is to be noted that the weak coupling results from the surrogate for point A are more accurate than the benchmark result due to the training data we used.

Table 6.1 shows the simulation time taken for different FSI simulations. The simulation time taken with the structural and the fluid surrogate is compared against the benchmark for both the weak and strong coupling scenarios. It is observed that replacing the structure increases the simulation time instead of decreasing it. In conventional FSI, the structural solver takes much less time than the fluid solver. The neural network predictions may not be accurate, and more iterations may be needed for convergence. However, the weak coupling simulation is faster than the original conventional strong coupling one with a negligible loss in accuracy. Nevertheless, it has to be remembered that the neural network surrogate was created using the strongly coupled simulation data. So, this approach can be used to create faster simulation models without losing accuracy (One time training time of the surrogate model is not considered).

Table 6.1: Simulation times for different configurations on Mok's benchmark.

Model	Time weak coupling simulation [s]	Time strong coupling simulation [s]
Full FSI	90.34	181.42
Surrogate structure 3DCNNLSTM	104.43	280.03
Surrogate fluid 2DCNNLSTM	40.45	44.65

The simulation time improves significantly when replacing the fluid model with a surrogate model. The time taken reduces to less than half for weak coupling and less than a quarter in the case of strong coupling. The percentage improvements are 55.2% and 75.4%, respectively. Hence, replacing the fluid solver with the surrogate model is recommended for a nearly real-time simulation. However, this transition entails the necessity of training a neural network with a sizable dataset, incurring costs from two primary facets: the time required to execute numerous CFD simulations and the duration of hyperparameter tuning and training. This can vary from weeks to months depending on the complexity of the problem and the data required for a successful training.

Chapter 7

Everything around you is mathematics. Everything around you is numbers

Shakuntala Devi

Numerical study

In this chapter, we analyse the suitability of the introduced hybrid model and approach to different numerical problems. The algorithm is tested on one-dimensional and two-dimensional thermal and structural problems. Both steady state and transient problems are analysed.

7.1 Forward solving of PDEs

7.1.1 Convection diffusion simulations

Convection diffusion equations describe the physics of the distribution of particles, energy, or other particles inside a domain due to convection and diffusion. Convection diffusion in general form is given by

$$\frac{\partial c}{\partial t} = \nabla \cdot (D \nabla c) - \nabla \cdot (\mathbf{v} c) + R \tag{7.1}$$

where, c is the variable of interest. It can be mass, density, temperature etc. D represents diffusivity and v the velocity. R represents the source or sink term. In the following examples we consider temperature T as the variable. The Equation 7.1 for one-dimensional convection diffusion of temperature is given by

$$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} = k\frac{\partial^2 T}{\partial x^2} + S \tag{7.2}$$

where T is the temperature, u the convection velocity, k the thermal diffusion coefficient and $S(x)$ the heat source.

Case 1

In the first case[1], we consider a steady one-dimensional convection diffusion problem. The equation becomes

$$\begin{aligned} &u\frac{\partial T}{\partial x} = k\frac{\partial^2 T}{\partial x^2} + S \\ &T(0) = T_1 \\ &T(1) = T_2 \end{aligned} \tag{7.3}$$

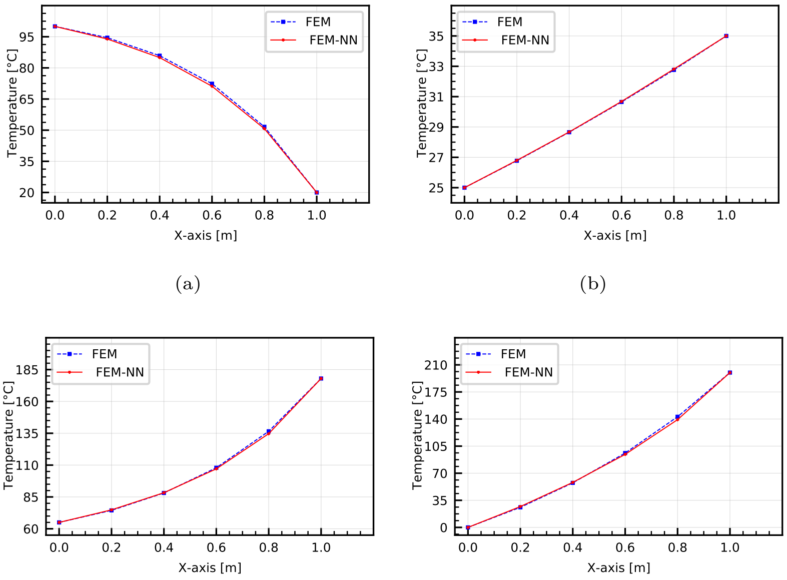
We consider the boundary conditions T_1 and T_2, source S, thermal diffusivity k and convection velocity u as the input parameters to obtain the nodal temperature values as output.

The parameters for the neural network $T(x)$ are learned by minimizing the FEM-based loss. A standard Adam optimizer [52] is applied to minimize the loss function based on the residual of the discretized equation. In this particular example, the residual is

$$\delta = \frac{1}{N}\sum_{i=i}^{N} ||\mathbf{K_i}\,\mathbf{T_i} - \mathbf{F_i}||_2^2 \tag{7.4}$$

where $\mathbf{K}$ is the stiffness matrix for the given input, $\mathbf{F}$ is the force vector, $\mathbf{T}$ is the output from the neural network and N is the total

[1] The following section is based on [71]. The main scientific research as well as the textual elaboration of the publication were performed by the authors of this work



(a) (b) (c) (d)

Figure 7.1: Distribution of temperature along x-axis for the steady state convection diffusion example for cases (a) $T_1 = 100, T_2 = 20, k = 10, u = 20, S = 100$ (b) $T_1 = 25, T_2 = 35, k = 10, u = 3, S = 1$ (c) $T_1 = 65, T_2 = 178, k = 6, u = 11, S = [5,2,3,4,5,1]$ and (d) $T_1 = 0, T_2 = 200, k = 10, u = 30, S = [5,2,3,4,5,1]$

number of samples used for training. Equation 7.4 helps the neural network to learn the physics rather than mere data. The trained surrogate model is used to predict the temperature distribution for different input combinations. Figure 7.1 compares the prediction from FEM-NN with the exact solution calculated using conventional FEM.

It can be observed from the predictions that the neural network learns the physics well. It is able to predict the temperature distribution accurately in different scenarios. The four sub-figures of Figure 7.1 shows the prediction for different combinations of k, u and S. The average absolute error between FEM results and FEM-NN results are 0.616,

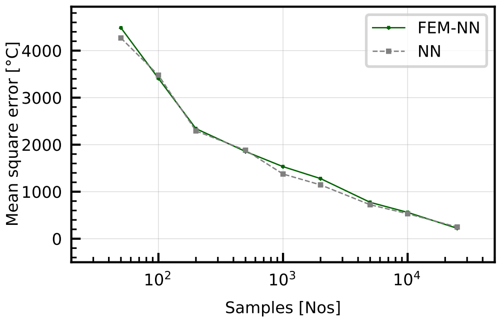


Figure 7.2: Error comparison of conventional neural networks and FEM-NN

0.022, 0.352, and 0.642 degree Celsius for Figures 7.1a, 7.1b, 7.1c and 7.1d respectively.

In the first two cases, Figures 7.1a and 7.1b, a constant heat source was applied on every node. Whereas, Figures 7.1c and 7.1d have different and random values for the heat source on each node. All four examples use random values for other input parameters. The parameters used are, $T_1 = 65$, $T_2 = 178$, $S = [5, 2, 3, 4, 5, 1]$, $u = 11$, $k = 6$ for Figure 7.1c. Similarly, the parameters for the Figure 7.1d are $T_1 = 0$, $T_2 = 200$, $S = [5, 2, 3, 4, 5, 1]$, $k = 1$, and $u = 1$. It can be observed that the model generalizes quite well in predicting the distribution of temperature even for such complicated cases.

One of the main advantage of this hybrid algorithm is that it does not use any target value during the training. Since it uses the system matrix and vector from FEM, it saves the computational time in the linear solvers. Hence, it does not fall under supervised learning and can be regarded as a semi-supervised or unsupervised learning approach.

The accuracy of FEM-NN is compared against standard NN in Figure 7.2. The loss is the L_2 of the difference between the prediction and the actual solution of $50,000$ samples. The actual solution of $50,000$ samples is created using a standard FEM method. It can be observed that the error is similar to that of the conventional neural network, whereas the time taken for conventional neural network training, including data creation, is more than the introduced FEM-NN.

Case 2

In this case, we consider a unsteady one dimensional convection diffusion problem. This case is used to demonstrate the strength of the introduced algorithm in comparison to state-of-the-art methods like PINN. The hybrid model introduced in Section 5.4.3 is used for training the model. The hybrid model is compared with the conventional PINN to compare the strengths and weaknesses. Similar to a conventional PINN, the spatial and temporal coordinates are the input and primary variable is the output.

We consider an unsteady 1D convection equation of the form,

$$\frac{\partial \phi}{\partial t} = -U_0 \frac{\partial \phi}{\partial x} \tag{7.5}$$

in the domain $x \in [0, 2\pi]$ with a periodic boundary condition. The initial condition for this problem is given by

$$\phi(x, t = 0) = sin(x) \tag{7.6}$$

In transient problems, the numerical residual loss takes different form as that of Equation 5.9. For example, applying central difference scheme for spatial discretization and an implicit Euler scheme for time integration results in an equation of the form

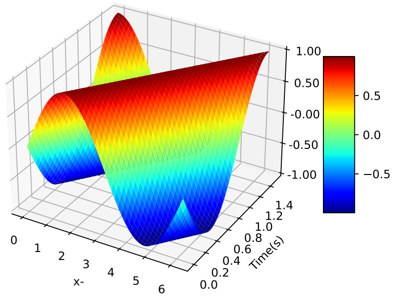
$$\mathbf{A\Phi^{t+1}} = \mathbf{\Phi^t} \tag{7.7}$$

where $\mathbf{\Phi}$ is the vector containing all the nodal values of ϕ. Here the residual can be written as

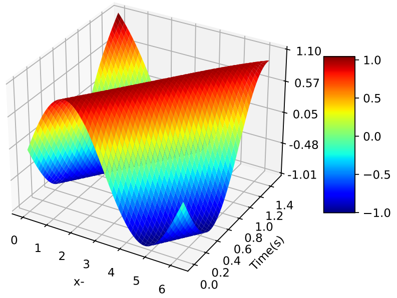
$$\mathbf{r} = \mathbf{A\Phi^{t+1}} - \mathbf{\Phi^t} \tag{7.8}$$

The Euclidean norm of this residual vector r can be used as the numerical residual loss.

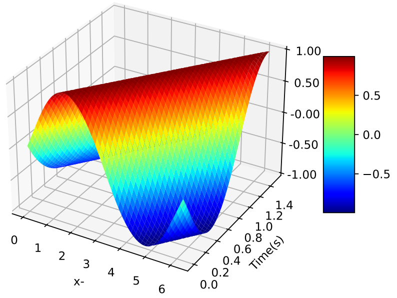
$$L_r = ||\mathbf{A\Phi^{t+1}} - \mathbf{\Phi^t}||_\mathbf{2} \tag{7.9}$$



(a) Exact



(b) Hybrid



(c) PINN

Figure 7.3: Unsteady 1D convection equation solution using different methods

Both the vectors $\mathbf{\Phi_{t+1}}$ and $\mathbf{\Phi_t}$ are predicted using a NN and substituted in Equation 7.9 to calculate the residual. It is backpropagated along with other two loss terms while learning the neural network.

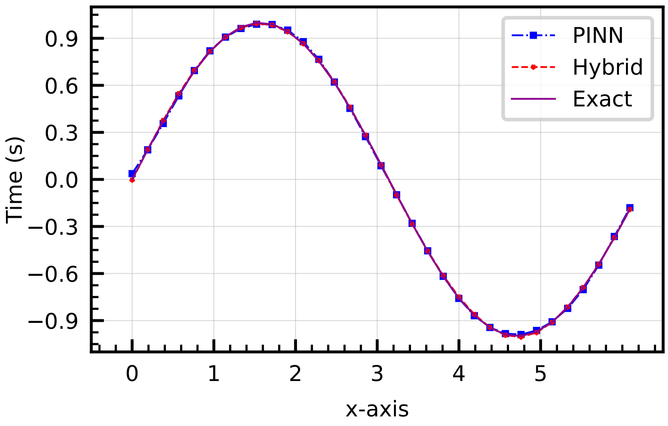
Figure 7.3 shows the results of simulation and prediction from different models after 5000 epochs. A comparison between exact solution, hybrid model and conventional PINN is made. In this example, exact solution is calculated by using numerical method having 50 grid points along the $x-$axis and using an Implicit Euler time integration method. A timestep of 10^{-3}s is used for the time integration. The hybrid model is able to predict the results matching with that of exact solution. However, the conventional PINN predictions are not matching with exact solution except at the initial timesteps. Solution along the $x-$axis for different number of timesteps from start is plotted in Figure 7.4. The results show that PINN is only able to predict the results in the initial timesteps, whereas the hybrid model is able to predict for longer time duration.

Case 3

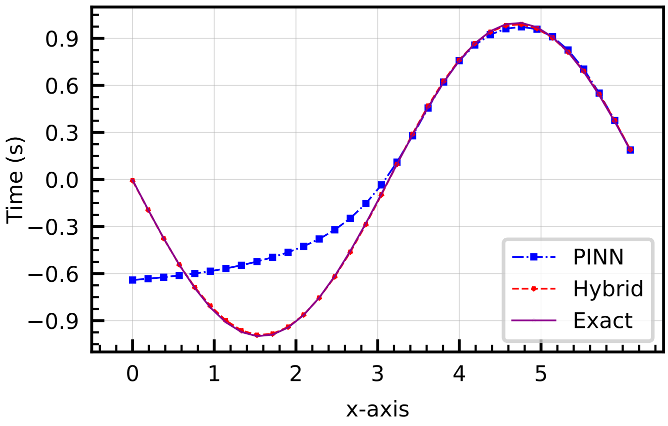
Here we consider a two-dimensional Poisson problem on a rectangular domain as in Figure 7.5. The numerical grid and the corresponding collocation points used for PINN training are shown. Dirichlet boundary conditions governing the problem are $u(x,y)=g(x,y)$ for $x \in \delta\Omega$.

$$\begin{aligned}
&\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0\\
&\qquad u=0.5 \quad at \quad x=-1\\
&\qquad u=1.0 \quad at \quad x=1\\
&\qquad u=-2.0 \quad at \quad y=1\\
&\qquad u=-1.0 \quad at \quad y=-1
\end{aligned} \tag{7.10}$$

In a first case, the domain is divided into 50 nodes in both x and y directions. The hybrid model converges to actual solution after training for 3000 epochs. Similar to a conventional PINN, the spatial and temporal coordinates are the input and primary variable is the output. On the other hand, the conventional PINN failed to reach convergence within 3000 epochs. Figure 7.6 shows the prediction from

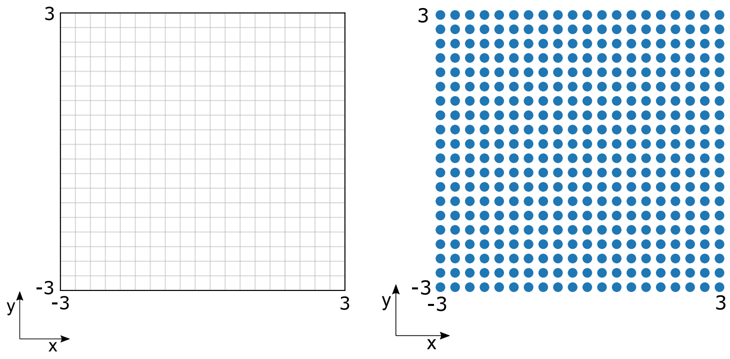


(a) Initial condition



(b) 1000 timesteps

Figure 7.4: Unsteady 1D convection equation solution using different methods at different timesteps



(a) Numerical grid (b) Collocation points

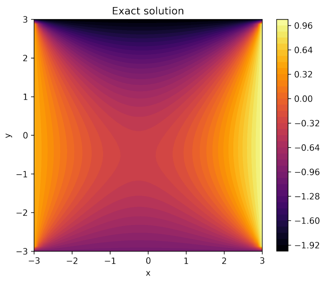
Figure 7.5: Domain for two dimensional Poisson problem

hybrid methodology along with the actual solution. The exact solution is calculated by taking 500 nodal points in each direction. The figure shows that the exact and predicted solutions are almost the same. Mean squared error between exact and predicted is 0.844.

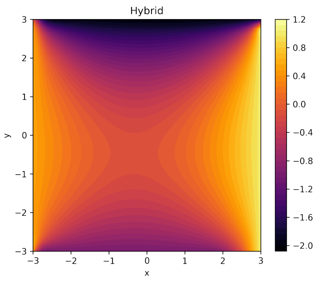
Figure 7.7 shows the prediction using conventional PINN after 3000 epochs. A comparison with actual solution shows that the solution has not converged. The error with exact solution is high throughout the domain. Mean squared error between exact and predicted is 1.7285.

The optimum model was found out using hyperparameter tuning. Similar to other example, Optuna [4] tuner was used to obtain the best parameters for number of layers, units per layer and learning rate. The final model used 2 layers with 155 neurons in each layer. A learning rate of 0.001199 was used with the Adam optimizer.

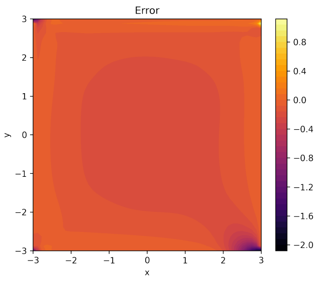
In Table 7.1, a comparison between exact solution and prediction from different models are compared. Nodes of numbers 20, 30 and 50 in each direction are considered. It can be seen that hybrid model performs better upon increasing the number of nodes. However, the increase is minimal compared to the computational time increase it results.



(a) Exact



(b) Hybrid

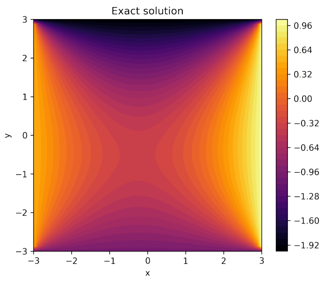


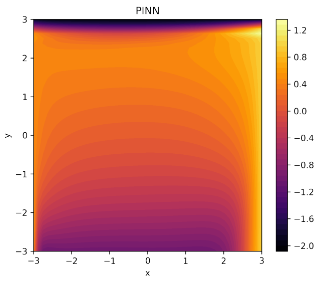
(c) Error

Figure 7.6: Two-dimensional Poisson Equation solution and absolute error using hybrid methodology

Table 7.1: Comparison of the accuracy for different numerical grids

Number of nodes	**Accuracy**
50 x 50	0.702
30 x 30	0.840
20 x 20	0.844


(a) Exact


(b) PINN

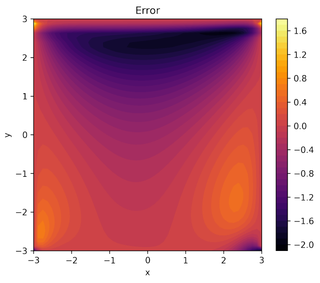

(c) Error

Figure 7.7: Two-dimensional Poisson Equation solution and absolute error using PINN

7.1.2 Truss

For a linear elastic prismatic bar under axial force (T), the equilibrium of forces is given by,

$$AE\frac{du(x)}{dx} = T \tag{7.11}$$

where A, E are the cross-sectional area and modulus of elasticity of the material. Taking the derivative of Equation 7.11 with respect to local coordinate x

$$\frac{d}{dx}\left[AE\frac{du(x)}{dx}\right]=0 \tag{7.12}$$

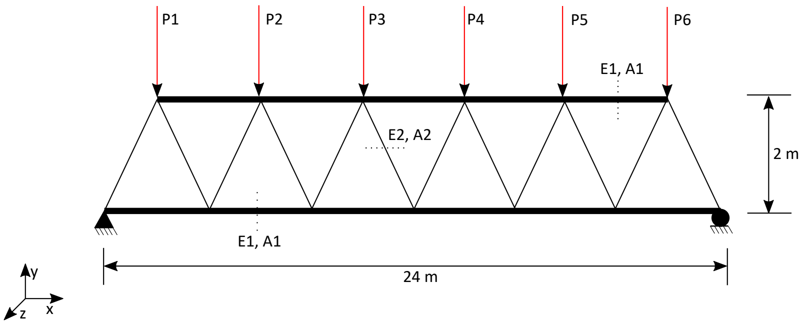


Figure 7.8: Sudret truss

In this example, we consider a 23-member, simply-supported truss structure example taken from [97]. The geometrical dimensions of the truss structure are given in Figure 7.8. Young's modulus of the horizontal bars is given by E_1 and for inclined bars E_2. Similarly, the horizontal bars have a cross-sectional area of A_1 and inclined bars have A_2. The truss is loaded by vertical forces $P1-P6$. All of these 10 variables are taken as an input for the neural network to predict the 39 nodal displacements for the 13 nodes of the truss.

Table 7.2: Input parameters for the 23-bar truss problem

Input variable	**Distribution**	**Low**	**High**
Horizontal cross-section area $A_h(m^2)$	Uniform	1.0×10^{-3}	1.0×10^{-4}
Inclined cross-section area $A_v(m^2)$	Uniform	2.0×10^{-3}	2.0×10^{-4}
Horizontal Young's modulus $E_h(Pa)$	Uniform	2.1×10^{11}	2.1×10^{10}
Inclined Young's modulus $E_v(Pa)$	Uniform	2.1×10^{11}	2.1×10^{10}
Inclined forces $P1-P6(N)$	Uniform	-5.0×10^{5}	5.0×10^{4}

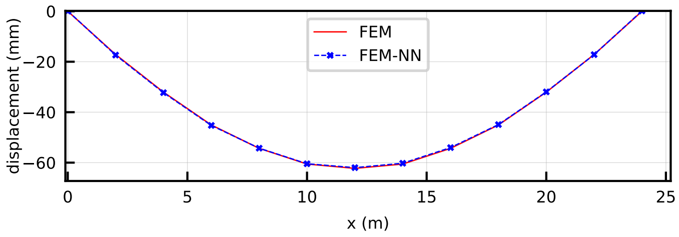


Figure 7.9: Vertical displacement of the nodes of the structure

The parameters used for training the model are given in Table 7.2. The system is modelled using Kratos Multiphysics and is then coupled with FEM-NN algorithm for training the model. The input parameters are given to both the Kratos and neural network to generate matrices and neural network prediction. We train the model as explained in the Algorithm 1.

Figure 7.9 shows the predicted and actual results for the input parameters $E_1 = 2.1\times10^{11}$, $E_2 = 2.32\times10^{11}$, $A_1 = 9.2\times10^{-4}$, $A_2 = 1.89\times10^{-3}$, $P_1 = -5.2\times10^4$, $P_2 = -5.2\times10^4$, $P_3 = -5.4\times10^4$, $P_4 = -3.6\times10^4$, $P_5 = -6.5\times10^4$ and $P_6 = -4.4\times10^4$. It can be observed that the prediction matches closely with the actual FEM based simulation results. The mean error associated with the prediction is 4.1×10^{-4}. As expected the prediction of $z-$direction displacements from the neural network are all close to zero. They all fall in the range of 10^{-6}m, which can be considered as zero considering the machine precision that limits the convergence of the neural network training.

7.1.3 Transient structural simulation

In this example, we consider a transient structural simulation using the variant of the FEM-NN model explained in Section 5.4.2. In structural dynamics, the equation of motion of a multi degree of freedom system (MDOF) is given by

$$\mathbf{M}\frac{d^2\mathbf{X}}{dt^2} + \mathbf{C}\frac{d\mathbf{X}}{dt} + \mathbf{K}\mathbf{X} = \mathbf{F}(t) \tag{7.13}$$

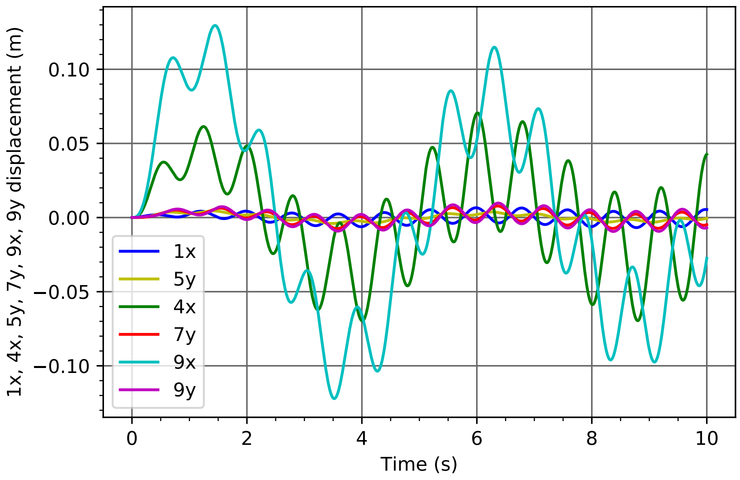
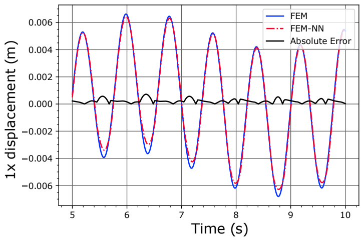


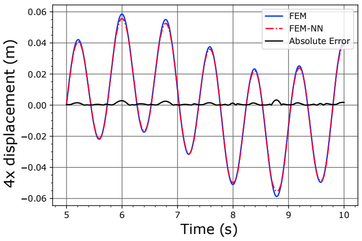
Figure 7.10: Response of MDOF system under external excitation

where, **M**, **C** and **K** are the global mass, damping and stiffness matrices and **F** is the external force on the system. Here **X** represents the collection all degrees of freedoms of the system. We use a 20 DOF system to demonstrate the model. Each DOF in this system is either the $x-$direction or the $y-$direction displacement of one of the 10 masses in the system. The external force applied is $\mathbf{F}(t) = sin(1.25\times 2\pi t)N$ on all masses. The displacement of the masses of the system for the first 10 seconds is given in Figure 7.10. It is calculated using FEM with generalized alpha time integration scheme. We used a timestep of $\Delta t = 10^{-3}$s. Only selected DOFs are plotted in the figure.

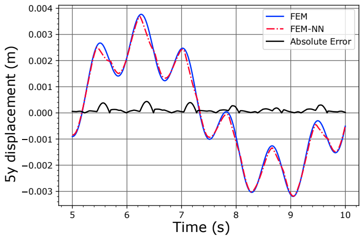
The Figure 7.10 shows the displacements of different Degrees of Freedoms (DOFs) in the system. The displacement of first mass in $x-$ direction is represented with the label $1x$ and similarly the $y-$direction displacement is represented using $1y$. It can be observed that each DOF behaves differently according to the properties of the system and applied force. This makes the training process of the neural network harder.



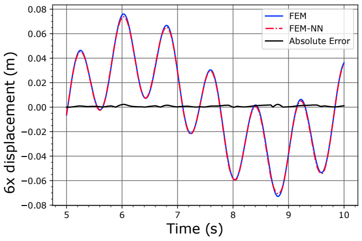
(a) 1x



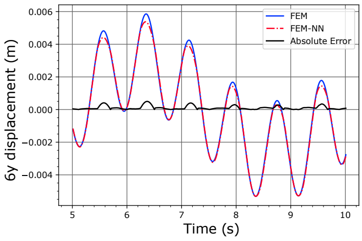
(b) 4x



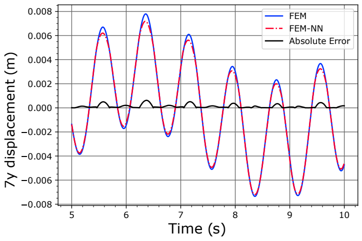
(c) 5y



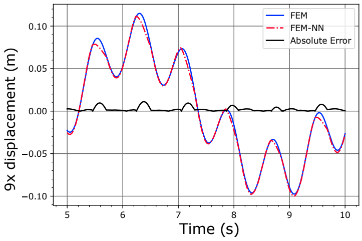
(d) 6x



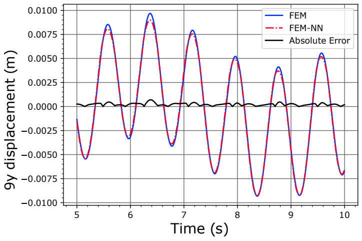
(e) 6y



(f) 7y



(g) 9x



(h) 9y

Figure 7.11: Plots of predicted displacement for the selected DOFs

This example used a three-layer LSTM network with 200 hidden units as the NN hybrid model. The training used the first 5000 time steps from the simulation. After the successful training, the trained model was used to predict the displacements of the remaining part of the simulation. An Adamax optimizer with a learning rate of $1e^{-4}$ and dropout value of 0.3 was used for optimization. The beta parameters for the optimizer were $\beta_1 = 0.9$ and $\beta_2 = 0.99$.

In Figure 7.11, the accuracy of prediction from the FEM-NN hybrid model is compared against the FEM based solution. The FEM solution is calculated using a Generalized Alpha time integration scheme. Displacements from both FEM-NN and FEM are plotted for the time interval between $5s$ and $10s$. The displacement of only a few selected DOFs are plotted in the Figure. The prediction from FEM-NN is closely matching with the exact solution with minimal error. The error associated with each prediction is also shown for a better comparison. Even though the displacements of different DOFs differ in scale and pattern across time, the model maintained the prediction accuracy reasonably. Even though the prediction is error is close to zero for all DOFs, there

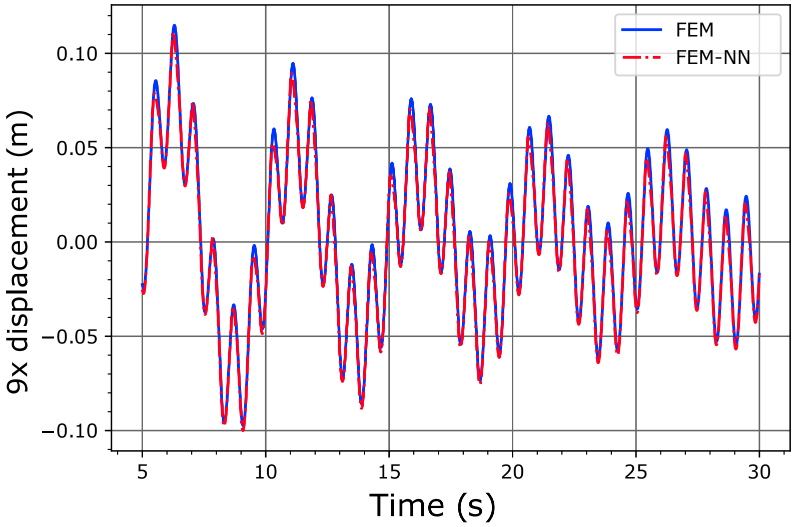


Figure 7.12: 9x displacement for longer time

is high error close to the crest and trough of the displacement curves. The high gradient of the displacement might be the reason for this. However, solving this by training on more epochs results in overfitted model which fails to generalize for future timesteps. Since the force **F** is changing over time sinusoidally, using it also in the input might solve this problem.

MDOF prediction for longer duration The trained network is tested to predict the solution of the simulation for much longer time duration as well. The displacement from $5s$ to $30s$ were predicted using the model and compared against the standard FEM based model. Interestingly, the model was able to predict the displacement accurately. The displacement for the DOF $9x$ is shown in Figure 7.12. The model maintains same level of accuracy at $30s$ as that of $5s$. This shows the potential of the model to generalize well.

7.2 Inverse problems

Numerical study is performed on different examples with varying complexity in the following section for the inverse problems. We consider a rotordynamic system as the industrial example in Chapter 8. Hence, we start with a simplified model of a rotordynamic system and increase the complexity in the following examples. We start with a SDOF system with a spring and a mass. Then, we add a damper also to the problem. After the analysis on SDOF system, we consider MDOF. Finally we analyse a complete rotordynamic system modeled using the open-source software ROSS rotordynamics [105].

7.2.1 SDOF with spring and mass

Many of the systems of practical importance can be reduced to a SDOF system for dynamic analysis. A few of the examples in the rotor dynamics are shown Figure 7.13.

Figure 7.13a shows a rotor with mass m supported by two flexible springs having stiffnesses k_1 and k_2. The effective stiffness can be assumed to be $k_{eff} = k_1 + k_2$. Figure 7.13b considers a rotor supported by a flexible shaft but with pin supports. In this case the effective stiffness is $k_{eff} = \frac{48EI}{l^3}$. Here E is Young's modulus of the shaft and I is the moment of inertia. The moment of inertia is calculated using the diameter of the shaft d using the expression $I = \frac{\pi d^4}{64}$. Similarly for a flexible hanging shaft in Figure 7.13c the effective stiffness is $k_{eff} = \frac{3EI}{l^3}$. Here also E and I represents the Young's modulus and moment of inertia respectively. Figure 7.13d represents the SDOF approximation of a rigid shaft under rotation. The effective rotation stiffness is given by $k_{eff} = \frac{GJ}{l}$. G is the modulus of rigidity (shear modulus) of the material and J is the torsion constant. $J = \frac{\pi d^4}{32}$ for a shaft with uniform circular cross section with diameter d.

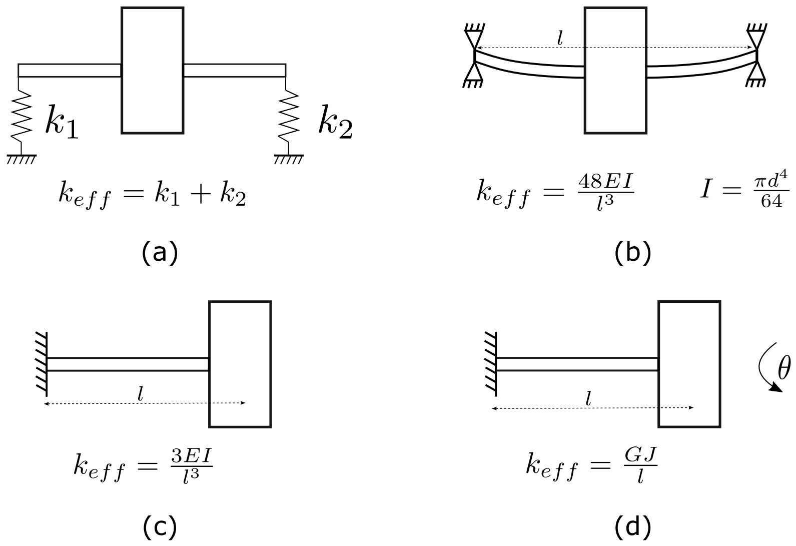


Figure 7.13: SDOF approximation for different systems

After the approximation, an SDOF with mass and spring is represented by the equation

$$m\ddot{u} + ku = f \tag{7.14}$$

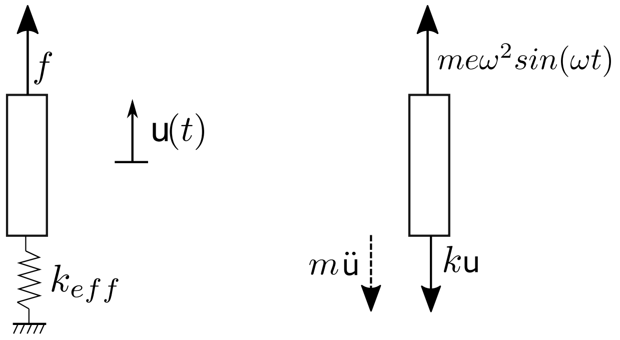


Figure 7.14: SDOF problem and the corresponding free body diagram

where m is the effective lumped mass, k the effective stiffness and f the force acting on the system. The primary variable u represents the displacement of the mass. For a rotordynamic system under study the force f is modeled as an unbalance in the system. Let m_b be the unbalance mass, e the distance of unbalance from centre of gravity, and ω the frequency of rotation. Then the unbalance force f is,

$$f = m_b e w^2 sin(w t) \tag{7.15}$$

which results in

$$m\ddot{u} + ku = m_b e w^2 sin(w t) \tag{7.16}$$

One approach to solve systems in rotordynamics is to convert the equation to frequency domain by the use of Fourier transform. Let us assume that the solution of this equation is given by, $u = U \sin(\omega t)$, where U is the amplitude of vibration. Then $\ddot{u} = -\omega^2 U \sin(\omega t)$. Hence, transforming equation 7.16 to frequency domain gives,

$$(-m\omega^2 + k)U = m_b e w^2 \tag{7.17}$$

Here $(-m\omega^2 + k)$ can be considered as the effective system stiffness K, and $m_b e w^2$ as the force F. Hence the residual of the equation will take the form

$$\begin{aligned} R &= KU - F \\ &= (-m\omega^2 + k)U - m_b e w^2 \end{aligned} \tag{7.18}$$

The stiffness k can be constant or a function of some operating(or environmental) variable depending on the system model we are using. At first we consider k as constant, which resemble the conditions of using a ball bearing. When considering constant k, we model it as a learnable parameter instead of treating it as a function of any variable.

Constant spring stiffness

In the first case we consider a constant value for the spring stiffness. Different parameters used for the experiment is given in table 7.3. Both vibration and force data are generated assuming the stiffness value of $13000\ N/m$. Then the generated data is used to calculate the assumed stiffness using the FEM-NN algorithm for inverse problems explained in 5.3.

Table 7.3: Experimental parameters for SDOF system with spring

Parameters	**Value**
Mass m (kg)	10
Eccentricity e(m)	0.02
Spring coefficient k (N/m)	13000

We start with a random initial guess for the unknown stiffness in the inverse problem. The residual calculated using the assumed stiffness value is then optimized using the FEM-NN inverse algorithm. Here the assumed stiffness is declared as a variable for the optimizer to optimize. Figure 7.16 shows the prediction of spring stiffness against number of iterations in the training loop. Five different initial values for the stiffness are assumed and training was performed. It is observed that the training converges to the assumed stiffness value for any of the initial guess we make.

7.2.2 SDOF with spring, mass and damper

An increased complexity from the basic spring mass system is the addition of a damper. Adding a damper to the system results in the equation of the form

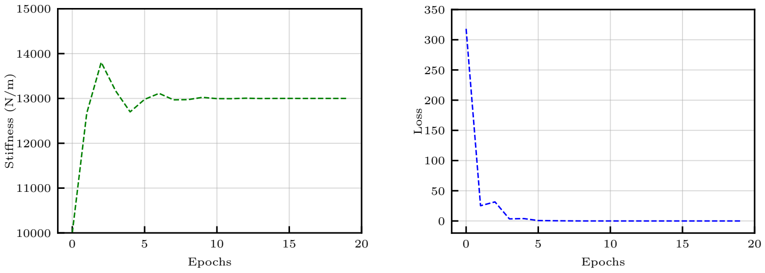
$$m\ddot{u} + c\dot{u} + ku = mew^2 sin(wt) \tag{7.19}$$

where c is the damping coefficient. Converting to frequency domain following the solution assumption of $u = Usin(\omega t)$ gives,

$$\begin{aligned} (-m\omega^2 U + jc\omega U + kU)\sin(\omega t) = me\omega^2 \sin(\omega t) \\ (-m\omega^2 + jc\omega + k)U = me\omega^2 \end{aligned} \tag{7.20}$$

The actual response with respect to time $u(t)$ is given by:

$$\begin{aligned} u(t) &= U\sin(\omega t) \\ u(t) &= \left| \frac{me\omega^2}{k + jcw - m\omega^2} \right| \sin(\omega t) \end{aligned} \tag{7.21}$$



(a) Stiffness prediction (b) Loss curve

Figure 7.15: Stiffness prediction and corresponding loss curve during training for spring, mass system

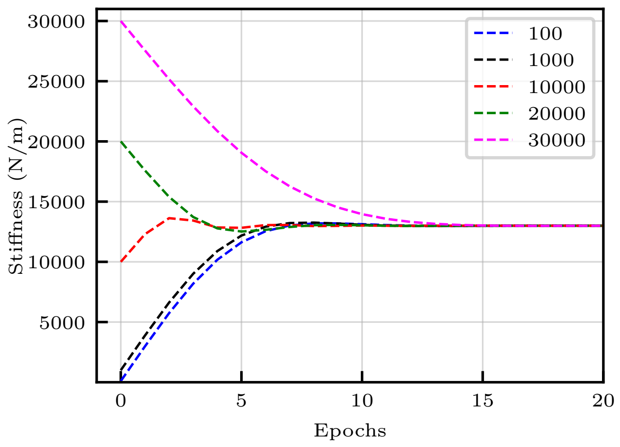


Figure 7.16: Robustness of the method against different initial values for the unknown stiffness

The assumptions result in effective stiffness K and force F as follows

$$\begin{aligned} K &= k + jcw - m\omega^2 \\ F &= me\omega^2 \end{aligned} \tag{7.22}$$

Hence the residual which is to be optimized is

$$R = (k + jcw - m\omega^2)U - me\omega^2 \tag{7.23}$$

The values for different parameters for this experiment are given in Table 7.4. Similar to the previous experiments, vibration and force data are created using a stiffness value of 13000 N/m. An initial guess is made for the stiffness and is optimized using the FEM-NN algorithm. Major difference this example has compared to the previous one is the presence of damper and the resulting complex equations. The gradient calculation and optimization algorithms must consider this into account. Details on this can be found in Appendix A.

Table 7.4: Experimental parameters for SDOF system with spring and damper

Parameters	**Value**
Mass m (kg)	10
Eccentricity e (m)	0.02
Spring coefficient in k (N/m)	13000
Damping coefficient (Ns/m)	5

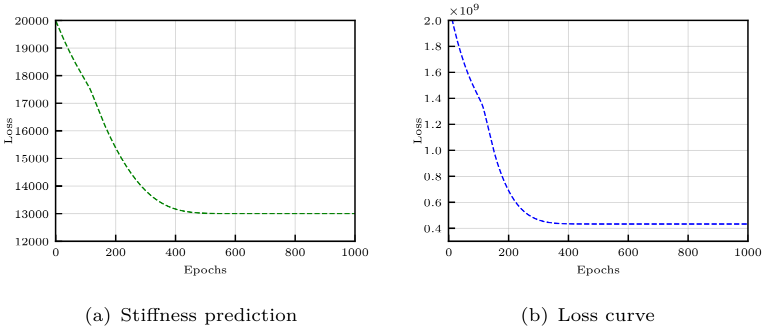


Figure 7.17: Stiffness prediction and corresponding loss curve during training for spring, mass and damper system

The Figure 7.17 shows the stiffness prediction and the corresponding loss curve. It can be seen that model is able to find exact value without any difficulty.

7.2.3 MDOF Jeffcott rotor

Jeffcott rotor is the simplified lumped parameter model used to solve a complete rotordynamic system. The Jeffcott rotor is a single disk symmetrically mounted on a uniform elastic shaft. The shaft is considered massless and disc having a mass m. It is also possible to include the mass of the shaft by including half of the total mass of the shaft (m_s) along with the mass of the disc (m_d).

$$m = m_d + \frac{m_s}{2} \tag{7.24}$$

We start with the equations of motion,

$$\begin{aligned} m_x \ddot{u}_x + c\,\dot{u}_x + k_x u_x &= m_x e \omega^2 \cos(\omega t) \\ m_y \ddot{u}_y + c\,\dot{u}_y + k_y u_y &= m_y e \omega^2 \sin(\omega t) \end{aligned} \tag{7.25}$$

Table 7.5: Experimental parameters for Jeffcott rotor

Parameters	**Value**
Mass m (kg)	10
Eccentricity e (m)	0.02
Spring coefficient in x-direction $k_x(N/m)$	13000
Spring coefficient in y-direction $k_y(N/m)$	10000
Damping coefficient (Ns/m)	5

The amplitudes of vibration in $x-$ and $y-$directions are given by,

$$\begin{aligned} U_x &= \left(\frac{m_x \omega^2 e}{\sqrt{(k_x - m\omega^2)^2 + (c\,\omega)^2}} \right) \\ U_y &= \left(\frac{m_y \omega^2 e}{\sqrt{(k_y - m\omega^2)^2 + (c\,\omega)^2}} \right) \end{aligned} \tag{7.26}$$

The actual response of the system in $x-$ and $y-$directions with respect to time are given by:

$$\begin{aligned} u_x(t) &= \left(\frac{m_x \omega^2 e}{\sqrt{(k_x - m\omega^2)^2 + (c\omega)^2}} \right) \cos(\omega t - \phi) \\ u_y(t) &= \left(\frac{m_y \omega^2 e}{\sqrt{(k_y - m\omega^2)^2 + (c\omega)^2}} \right) \sin(\omega t - \phi) \end{aligned} \tag{7.27}$$

when converting into frequency domain and rewriting

$$\begin{aligned} K_1 &= k_x + jcw - m_x \omega^2 \\ K_2 &= k_y + jcw - m_y \omega^2 \end{aligned} \tag{7.28}$$

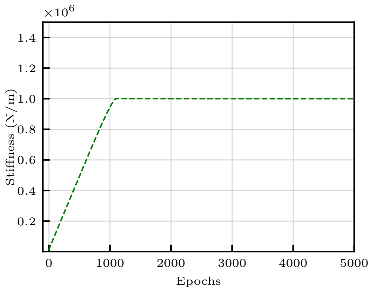
$$\begin{aligned} F_1 &= m_x e \omega^2 \\ F_2 &= -jw m_y e \omega^2 \end{aligned} \tag{7.29}$$

The same equation system can be written in matrix form as

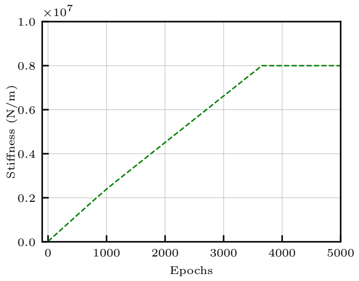
$$\begin{aligned} &\begin{bmatrix} m_x & 0 \\ 0 & m_y \end{bmatrix} \begin{pmatrix} \ddot{u}_x \\ \ddot{u}_y \end{pmatrix} + \begin{bmatrix} c_x & 0 \\ 0 & c_y \end{bmatrix} \begin{pmatrix} \dot{u}_x \\ \dot{u}_y \end{pmatrix} + \\ &\qquad \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix} \begin{pmatrix} u_x \\ u_y \end{pmatrix} = \begin{pmatrix} m_x e \omega^2 \cos(\omega t) \\ m_y e \omega^2 \sin(\omega t) \end{pmatrix} \end{aligned} \tag{7.30}$$

Converting the equation to frequency domain results in

$$\begin{aligned} &\begin{bmatrix} -m_x \omega^2 + jc_x w + k_x & 0 \\ 0 & -m_y \omega^2 + jc_y w + k_y \end{bmatrix} \begin{pmatrix} U_x \\ U_y \end{pmatrix} \\ &\qquad = \begin{pmatrix} m_x e \omega^2 \\ -jw m_y e \omega^2 \end{pmatrix} \end{aligned} \tag{7.31}$$



(a) Stiffness in x-direction



(b) Stiffness in y-direction

Figure 7.18: Stiffness prediction of Jeffcot rotor along the training

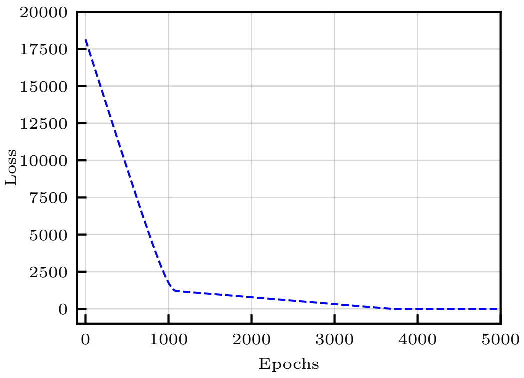


Figure 7.19: Loss curve of Jeffcott rotor stiffness prediction using FEM-NN algorithm

The equation 7.31 is used to calculate the residual and use for optimization process. The values for different parameters for this experiment are given in table 7.5. Similar to the previous experiments, vibration and force data are created using assumed stiffness values of $k_x = 10^6\ N/m$ and $k_y = 10^8\ N/m$. Then initial guesses are made for the stiffnesses and is then optimized using the FEM-NN algorithm.

7.3 Complete rotordynamic system

So far we have considered simplified examples of rotordynamic systems. Here we consider a real-scale model of a rotordynamic system modelled using the open source software called ROSS [105]. Before going in the details of the complete model, we first consider the basics of fluid bearings and how they are modeled in rotating systems as we are considering a fluid bearing example in Chapter 8.

Industrial bearings on which load is supported by a thin layer of fluid are called fluid bearing. They are of mainly two types i) Hydrostatic, and ii) Hydrodynamic. Hydrostatic fluids have pressurised fluid with the help of a pump. whereas hydrodynamic bearings have the pressure generated with the speed of the shaft. We discuss about hydrodynamic bearings in the rest of the dissertation. Consider the mathematical model of a hydrodynamic bearing given in Figure 7.20.

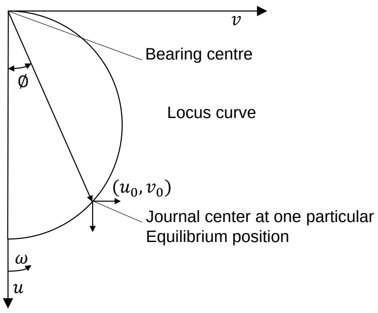


Figure 7.20: Mathematical model of a fluid bearing

The total reaction forces at the centre of bearing are given by R_x and R_y

$$R_x = R_{u_0} + K_{xx}\,u_x + K_{xy}\,u_y + C_{xx}\,\dot{u}_x + C_{xy}\,\dot{u}_y + m_{xx}\,\ddot{u}_x + m_{xy}\,\ddot{u}_y \quad (7.32)$$

$$R_y = R_{v_0} + K_{yx}\,u_x + K_{yy}\,u_y + C_{yx}\,\dot{u}_x + C_{yy}\,\dot{u}_y + m_{yx}\,\ddot{u}_x + m_{yy}\,\ddot{u}_y \quad (7.33)$$

here

$$R_x = f_x - m\ddot{x} \tag{7.34}$$

$$R_y = f_y - m\ddot{y} \tag{7.35}$$

The stiffness and damping coefficients are defined as

$$K_{xy} = \left(\frac{\partial R_x}{\partial y}\right)_{(u_0,v_0)} \tag{7.36}$$

$$C_{xy} = \left(\frac{\partial R_x}{\partial \dot{y}}\right)_{(u_0,v_0)} \tag{7.37}$$

$$m_{xy} = \left(\frac{\partial R_x}{\partial \ddot{y}}\right)_{(u_0,v_0)} \tag{7.38}$$

In a nutshell we can arrive at the following observations from modeling the hydrodynamic bearings

- Equation is a complete form of linearized fluid-film dynamic equation and it contains twelve stiffnesses, damping and added-mass coefficients.
- The coefficients depend on the equilibrium position (u_0, v_0) which depends on the dimensionless Sommerfeld number S $S = \frac{\mu \Omega R L}{W}\left(\frac{R}{c_r}\right)^2\left(\frac{L}{D}\right)^2$,
- The coefficients can also depend on external excitation frequency ω
- So, bearing coefficients are a function of the form $f(\Omega, \omega)$. The stiffness and damping coefficients can be numerically obtained by a finite difference solution of the perturbed Reynolds equation [64].

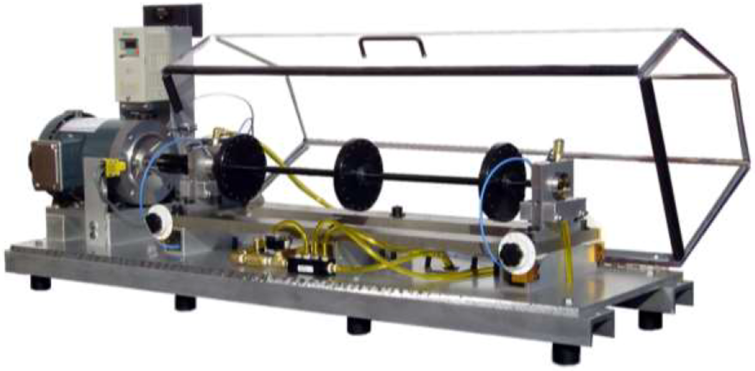


Figure 7.21: Experimental setup for rotordynamic bearing study

- Damping coefficients are symmetric but stiffnesses are not.
- Although the load–displacement characteristics of a journal bearing is non-linear, the concept of linear dynamic coefficients is still used for modern rotor dynamic calculations for unbalance response, damped natural frequencies and stability since experience has demonstrated the usefulness of the coefficients (Applicable only if the amplitude of vibration is small)

In Chapter 8, we consider a rotordynamic test-bench as given in Figure 7.21. In this system, a motor is driving a shaft on which two discs are mounted. The shaft is supported by two fluid bearings at the driving end (DE) and non-driving end (NDE). In order to create forces on the system, known unbalance masses are attached to the discs at known distance from the disc centre. The discs have uniformly distributed mass and do not result in unbalance unless an external mass is attached.

Figure 7.22 represents the model of the system given in Figure 7.21. The system consists of l number of nodes from driving to non-driving end. The nodes n_{b1} and n_{b2} represents the nodes where bearing is attached. Nodes n_{d1} and n_{d2} represents the locations where discs are

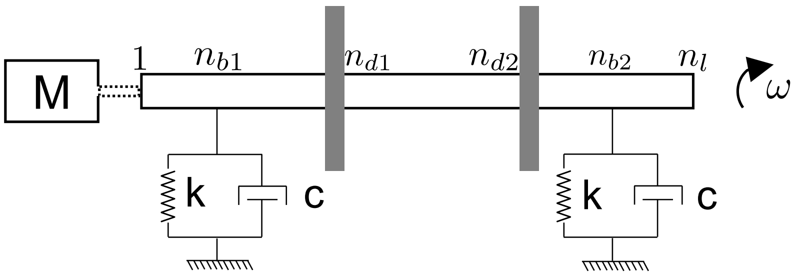


Figure 7.22: A general rotordynamic system with two bearings and two discs

attached. n_l is the node at the end of the shaft and also the total number of nodes in the system. The M in Figure 7.22 represents the motor which drives the system at a rotational frequency of ω.

The equation of motion of the system in frequency domain is given below

$$\begin{aligned} \left[-\omega^2\mathbf{M} + j\omega(\mathbf{G}\omega + \mathbf{C}) + \mathbf{K}\right]\mathbf{U} &= \mathbf{F} \\ \mathbf{D}\,\mathbf{U} &= \mathbf{F} \end{aligned} \tag{7.39}$$

where $\mathbf{M}, \mathbf{G}, \mathbf{C}$ and $\mathbf{K}$ are the mass, gyroscopic, damping and stiffness matrices respectively. $\mathbf{U}$ is the displacement vector and F is the force vector in the frequency domain. $\mathbf{D} = \left[-\omega^2\mathbf{M} + j\omega(\mathbf{G}\omega + \mathbf{C}) + \mathbf{K}\right]$ is called dynamic stiffness matrix.

The parameters used for modeling the shaft and discs of the system in ROSS [105] are given in Table 7.6. The bearing coefficients of the modelled fluid bearing follows as in Figure 7.24. All 8 coefficients (stiffness and damping) are functions of the rotational speed.

First part of this experiment generates the data required for the algorithm to train on. A FEM based model of the system is created using the open-source software ROSS rotordynamics [105]. Different unbalance configurations are created and corresponding forces on the system are recorded. The displacements at each node of the system are also recorded. This is done numerically modeling the fluid bearing. After creating the data, we also create the matrices of the FEM system without having bearings in the system. The FEM-NN algorithm is

Table 7.6: Parameters of the rotordynamic system modelled using ROSS

Part	**Dimension**	**Value in SI unit**
Shaft	Length	0.25m
	Diameter	0.05m
	Density	7800kg/m^3
Disc	Inside diameter	0.05m
	Outside diameter	0.28m
	Width	0.07m
	Density	7800kg/m^3
Bearing	Inlet pressure	10^{5}Pa
	Outlet pressure	0Pa
	Density of the fluid	860kg/m^3
	Viscosity of the fluid	860kg/m^3
	Outer diameter	0.05m
	Inner diameter	0.049m
	Width	0.03m
	Static load	525N

tested on the created data to calculate the bearing coefficients that we modeled. The training for bearing coefficients follow the pipeline given in Figure 7.23.

Initially both the displacement and force signals generated are converted to the frequency domain and their amplitudes are calculated. Mass, gyroscopic, damping and stiffness matrices of the rotordynamic system without fluid bearings are created. Bearing coefficients of the fluid bearings are predicted using a neural network which takes the rotational speed as the input. The predicted coefficients are assembled at

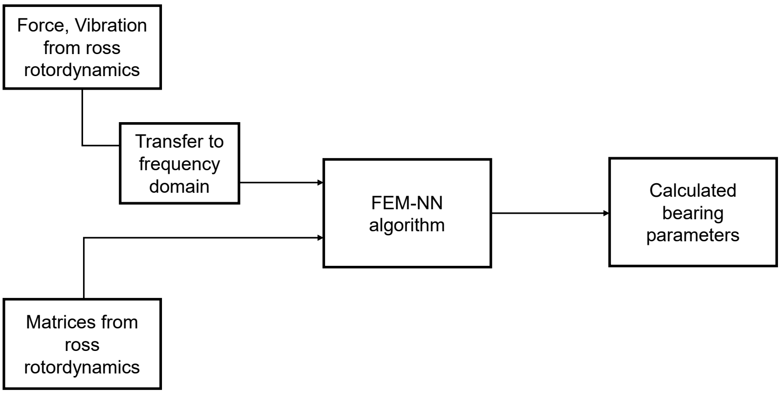

Force, Vibration from ross rotordynamics
Transfer to frequency domain
FEM-NN algorithm
Calculated bearing parameters
Matrices from ross rotordynamics

Figure 7.23: Bearing identification process using simulated data

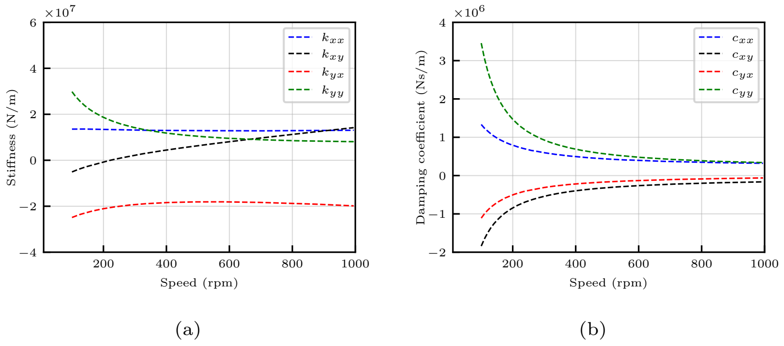
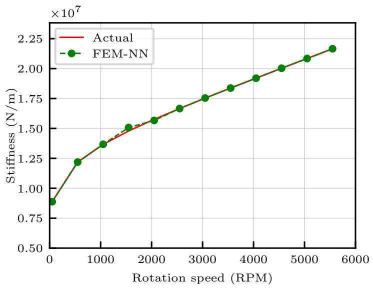


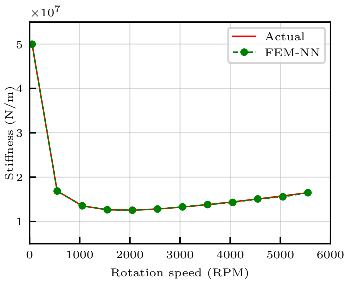
Figure 7.24: Speed dependent bearing coefficients of the hydrostatic bearing modeled using ROSS rotordynamics

the respective positions of the global system matrix and then optimized using the FEM-NN algorithm.

Figures 7.25 and 7.26 show the prediction from the neural network after the training process. It can be observed that the prediction is able to capture the relation between speed and coefficients pretty well.

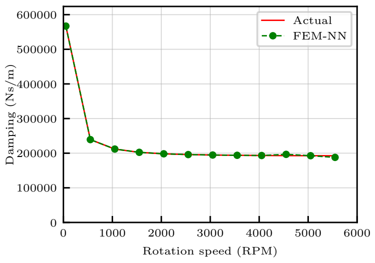


(a)

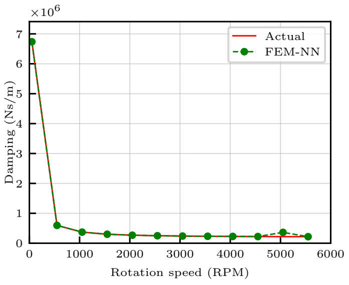


(b)

Figure 7.25: Speed dependent bearing stiffness of the hydrostatic bearing actual vs learned



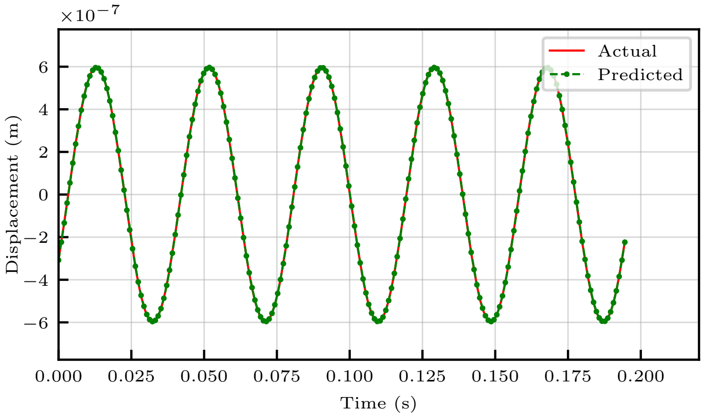
(a)



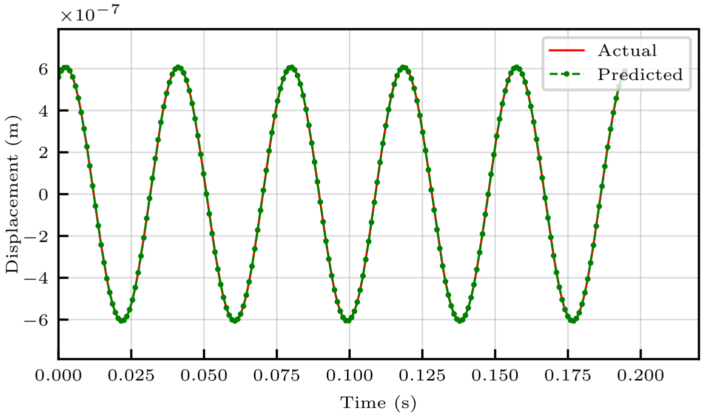
(b)

Figure 7.26: Speed dependent bearing damping coefficients of the hydrostatic bearing actual vs learned

Once a model is trained for the unknown coefficients, we can use the trained model in the forward simulations. Such an experiment is also conducted using the trained neural network to predict the vibrations at different points on the shaft. Figure 7.27 shows the comparison between the predicted vibration using the learned coefficients and the actual vibration based on the fluid bearing numerically modeled in the ROSS rotordynamics.



(a) x-direction displacement of the disc node



(b) y-direction displacement of the disc node

Figure 7.27: Predicted vs actual displacement of the disc node of the rotordynamic system

Chapter 8

Scientists study the world as it is; engineers create the world that has never been

Theodore von Karman

Industrial examples

8.1 Wind load on high-rise building

Wind load on different structures, such as wind turbines, bridges and buildings, has been a major concern for engineers during its design and operational phase. It is said that $70-80\%$ of economic losses due to natural disasters in the world are caused by wind related hazards [99]. FSI simulations are performed in the design phase for the accurate understanding and prediction of the impact of fluid force on such systems. The wind load on high-rise buildings is extensively studied during the design of buildings as the wind can cause vibration and structural damage to the buildings. Additionally, the effect of wind also plays an important role in the serviceability and habitability of buildings. Hence, the effect of wind is an important and challenging issue in building and structural engineering field. Altogether, the field of structural wind engineering is concentrated on studying the effects of wind load on structures.

The FSI approach is generally employed to predict the flow-induced vibration and estimate the forces exerted by the wind flow. However, FSI simulations are not always easy and feasible due to their considerably complicated algorithm and communication scheme, which sometimes require high computational resources. In this example, we perform the multi-physics study of wind-structure interaction in a simpler manner. We implement the wind flow in a relatively simple yet comprehensive fashion, followed by the assessment of design criteria of highrise structures from the structural engineering and design perspective

The Commonwealth Advisory Aeronautical Council (CAARC) [11] building geometry is used for the study as it is widely used in literature for benchmarking. The CAARC is a parallelopiped building with the dimensions listed in Table 8.1. The parameters of the building height(H), width(B) and length(D) are given in Table 8.1. In this study, we do not attempt a full model analysis as it becomes numerically expensive. Alternatively, we model the building as an Euler-Bernoulli beam described by the following equations.

$$\begin{gathered}\frac{\partial^2}{\partial y^2}(EI\frac{\partial^2 u_x(y)}{\partial y^2})=f(x)\\ \frac{\partial^2}{\partial y^2}(EI\frac{\partial^2 u_z(y)}{\partial y^2})=f(z)\end{gathered} \tag{8.1}$$

The variables $u_x(y)$ and $u_z(y)$ denotes the deflection of the beam in the x and z directions at some position y. E is the elastic modulus and I is the second moment of area of the cross section. The forces in x and z directions are generated by the wind load on the building. The wind-load on the building is calculated using the mean wind velocity u_{ref} and roughness length y_0. The static wind load f at each height of the building is calculated as

$$f=\frac{\rho_{air}V(y)^2AC_d}{2} \tag{8.2}$$

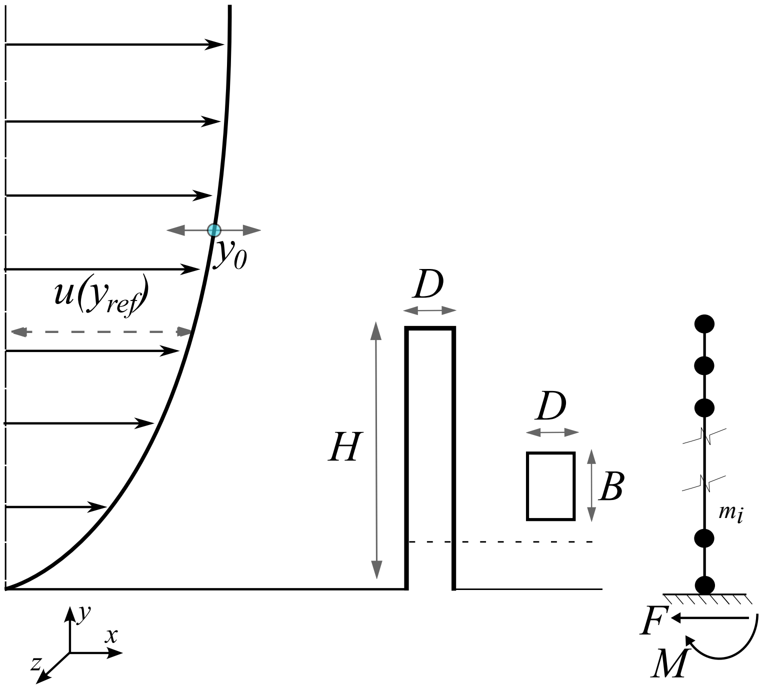


Figure 8.1: Wind load on a building

where ρ_{air} is the air density, A is the reference area and C_d the drag coefficient for the cross section of the building. $V(y)$ represents the wind velocity at height y calculated using an assumed profile for the wind.

The calculation of wind profile starts with the calculation of the friction velocity using the mean velocity u_m.

$$u_* = \frac{u_m \kappa}{log((y_{ref} + y_0)/y_0)} \tag{8.3}$$

Table 8.1: Details of the building - geometry and structural

Parameters	**Values**
Height (H)	180 m
Width (W)	45 m
Length (D)	30 m
Frequency (f)	0.2 Hz
Density (ρ)	160 kg/m^3
Damping ratio (ζ)	0.01

The friction velocity is then used to calculate the velocity at each height y using

$$u_y = \frac{u_*}{\kappa} ln(\frac{y}{y_0} + 1) \tag{8.4}$$

where, κ is the Von-karmann constant with a value 0.41. Using the velocity u_y, the force at the height y is calculates as

$$f_y = 0.5 \times 1.2 \times u_y^2 \times 45 \tag{8.5}$$

This force is then used along with the Equation 8.1 to calculate the displacement of the structure. Among all the parameters, the uncertainty in wind and the terrain affect the long term performance of the structure. Hence, the uncertainty in these parameters and their effects has to be studied. The effect of uncertainty in the wind load on the horizontal displacement at the top of the building is studied in this example. Figure 8.1 shows the schematic diagram of the approximation of the high-rise building and the wind load on it.

In this experiment, a surrogate model for the structural simulation of the high-rise building is developed using the FEM-NN algorithm. The surrogate model takes the mean wind velocity $u(y_{ref})$ and roughness length y_0 as the input to predict the displacement of the building at different nodal points. The surrogate model is trained by generating training data by choosing different values for the mean velocity and

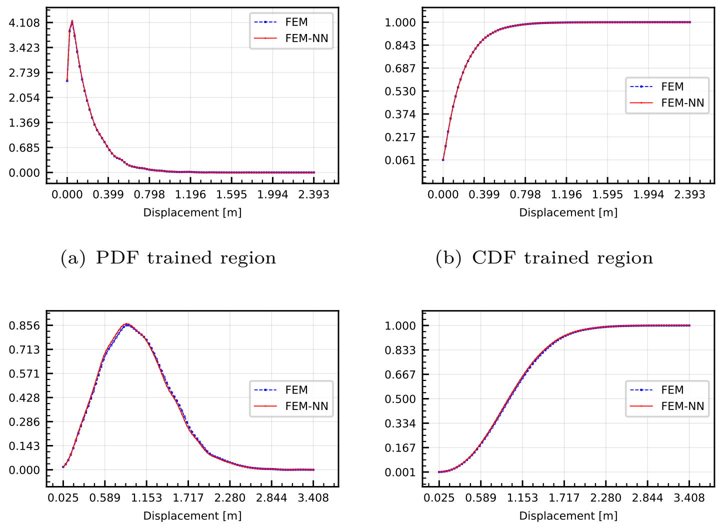
Table 8.2: Details of uncertain wind parameters

Uncertain parameters	**Distribution**	**Values**
Mean wind velocity u(y_{ref})	Weibull	Mean= 40 m/s Shape parameter = 2
Roughness length, y_0	Uniform	[0.1, 0.7]

roughness length. The mean velocity is sampled from a Weibull distribution and the roughness length from a uniform distribution. The parameters used for the distribution are given in Table 8.2.

The trained model is used to run a large of number of simulations. Such simulation results are then used for the Monte Carlo uncertainty analysis. The Monte Carlo-based analysis is a widely used uncertainty analyses method that we can now see in almost all engineering fields [45]. This analysis expresses the quantity of interest as a complementary cumulative distribution function (CDF). The results from simulations are used for the probability distributions of the targeted variable. In conventional Monte Carlo method, a large number of computationally expensive simulations are required for generating the probability distributions. By using a surrogate model made of limited number of simulations, we reduce the total number of expensive simulations required for the analysis.

In our experiment, the displacement of the topmost point is the quantity of interest. Figure 8.2 illustrates the probability distribution using the FEM-NN surrogate model in comparison to a conventional FEM bases analysis. A total of 2000 samples were utilized for the training of the neural network. A fully connected network with 200 hidden units and 4 layers were used. A Monte Carlo method with 5000 samples were used for the FEM based solution. Figure 8.2 (a) shows the probability distribution of the top displacement of the building and Figure 8.2 (b) shows the cumulative distribution function. It can be



(a) PDF trained region

(b) CDF trained region

(c) PDF untrained region

(d) CDF untrained region

Figure 8.2: PDF and CDF in trained region in (a) and (b) and untrained region in (c) and (d)

observed that the FEM-NN is able to reproduce the results obtained using FEM with reasonable accuracy. In this case, the input parameters were taken from the same distribution as that used for training.

Figures 8.2 (c) and (d) are the results of an analysis with different distribution of input parameters than that used for training. The statistical quantities of the analysis for both cases are given in Table 8.3. It can be observed that the FEM-NN was able to produce accurate results in both the cases. Hence, it can be concluded that the FEM-NN can be used to run Monte Carlo uncertainty analysis with lesser number of samples and faster. Running a large number of simulations using a neural network only takes seconds in contrast to to the conventional FEM. However, the training time and hyperparameter tuning efforts of neural network are not considered.

Table 8.3: Statistical quantities of Monte-Carlo analysis

	Trained region				Untrained region			
	mean	standard deviation	skewness	kurtosis	mean	standard deviation	skewness	kurtosis
FEM	0.183	0.199	2.679	12.278	1.016	0.470	0.667	0.717
FEM-NN	0.181	0.197	2.706	12.555	1.004	0.468	0.693	0.790

8.2 Fluid bearing parameter identification

In this example, we test the FEM-NN inverse algorithm on the data collected from the experimental setup in Figure 7.21. The typical workflow for the bearing parameter identification on an industrial rotordynamic system is given in Figure 8.3. The same workflow is applicable to any industrial settings with slight modification in the data acquisition or preprocessing.

The workflow begins with the vibration measurements at different feasible points on the system. The measured signals are preprocessed

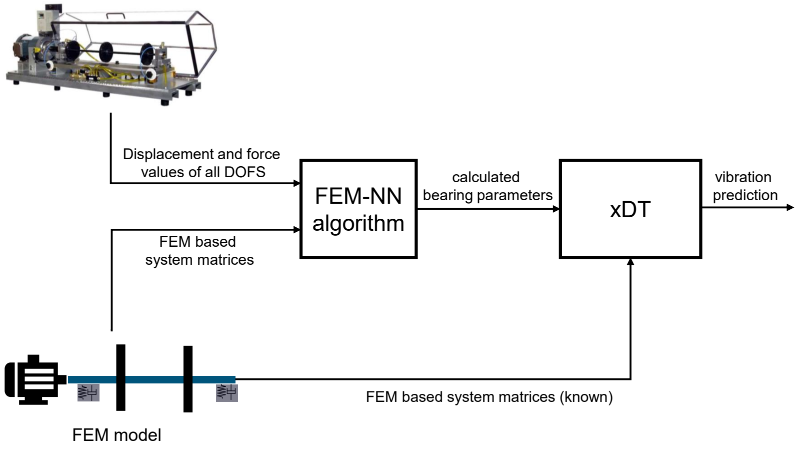

Displacement and force values of all DOFS
FEM-NN algorithm
calculated bearing parameters
xDT
vibration prediction
FEM based system matrices
FEM based system matrices (known)
FEM model

Figure 8.3: Bearing parameter identification workflow for the rotordynamic system experimental setup

Table 8.4: Different unbalance configurations used for the experiment

Unbalance	**Disc 1**		**Disc 2**	
configuration	**Radial**	**Angular**	**Radial**	**Angular**
Unbalance 1				
Unbalance 2	Out	1		
Unbalance 3			Out	1
Unbalance 4	Out	1	Out	1
Unbalance 5	Out	14	Out	1
Unbalance 6	Out	9	Out	1
Unbalance 7	Out	5	Out	1

to remove noise and other vibrations contributed from nearby machines. Then the vibration values for DOFs which are not measured are reconstructed. The force values for each configuration is also recorded using the known unbalance configuration. At the same time, an FEM model of the system is constructed using any software which permits the access to the system matrices and vectors. Then the FEM-NN algorithm for inverse problem is executed to calculate the bearing coefficients. Known coefficients are then used for the prediction of the future state of the system.

In the following each of the above steps are detailed

Data acquisition

Data acquisition is performed by attaching known unbalance mass on the disc at different radial distances from the centre and at different angles from the reference position. The diagram of the disk with the holes positions for attaching the unbalance is given in Figure 8.4. Two of such discs are attached to the shaft. Different configurations used for the data acquisition are given in Table 8.4.

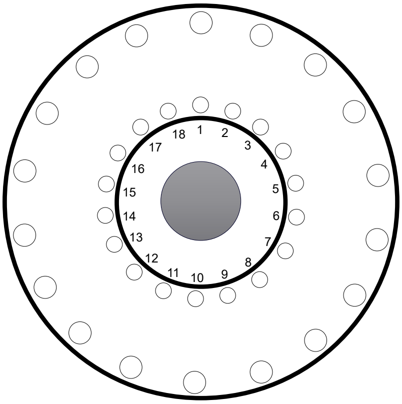


Figure 8.4: Discs on the rotating shaft with the provisions for attaching unbalance mass

In each of the configurations, five sensors are used at the locations motor drive end (Motor), drive end z-direction (S_DE_Z), non-drive end z-direction (S_NDE_Z), drive end y-direction (S_DE_Y) and non-drive end y-direction (S_NDE_Y). The experiments are performed with rotating speed ranging from 10 to 60 Hz with an interval of 10 Hz. The measured signals from two of the configurations are given in Figures 8.6 and 8.7.

Data cleaning

Vibration sensors mounted on the system collect vibrations over time at 5 different locations on the system. However, due to the sensor precision and environmental effects, the collected raw data contains noisy values. Appropriate data cleaning is to be performed to eliminate noisy data from the measurement. Since the vibration follows the rotating speed of the system, we used different filtering techniques for the data cleaning. Figure 8.5 shows the workflow of the data cleaning process. A Butterworth filter is used for the filtering of noises from the measured signal. The Butterworth filter is a type of signal processing filter designed to have a frequency response, that is as flat as possible in the range of frequencies. We used a range of 2 Hz around the

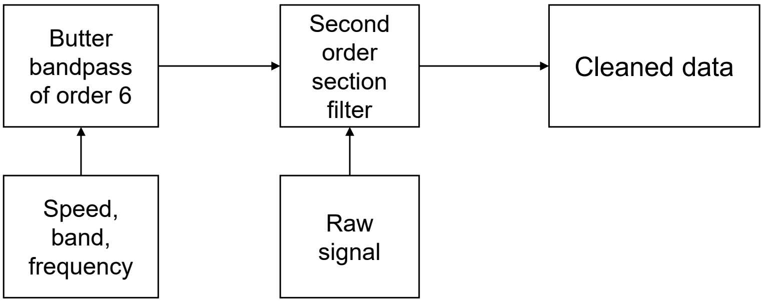


Figure 8.5: Workflow for cleaning the measured data

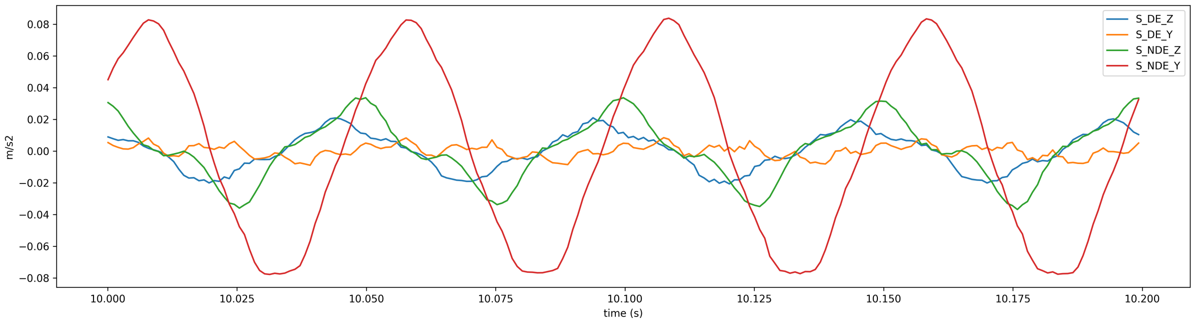
rotating frequency as the band. Rotating speed, band and frequency of measurement are used to create a butter bandpass. The created bandpass is used as a second order section filter on the raw signal to get the cleaned data. Second order section filters the data along one dimension using cascaded second-order sections. More details on butterworth filter is available in [14, 93]. Figures 8.6 and 8.7 show the raw data and cleaned data using the data cleaning process in Figure 8.5.

Data synchronisation

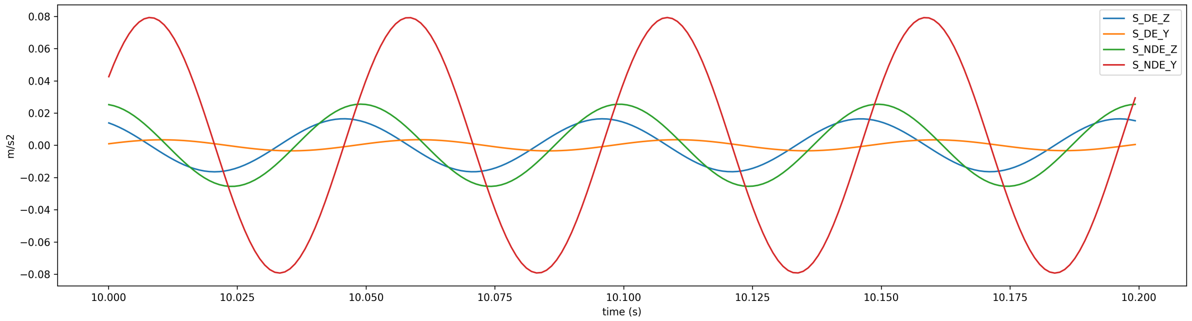
The measurements of vibrations were conducted on different days by different people, making it difficult to maintain the same starting position of the shaft for all measurements. Hence, a position on the shaft was marked and a sensor was used to detect the mark on each revolution. The signal before the first occurrence of that mark needs to be removed to keep the initial position of the shaft the same for all measurements. We took the hundredth instead of the first occurrence to allow the initial dynamics to dampen.

Data reconstruction

The proposed FEM-NN algorithm uses the primary variables and force vector corresponding to all DOF for running the inverse problems. Due to the practical limitations for sensor placement, we can only measure

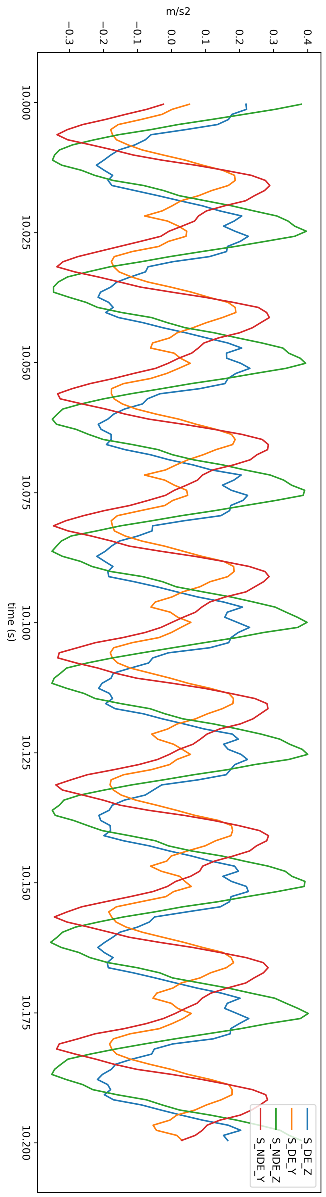


(a) Uncleaned

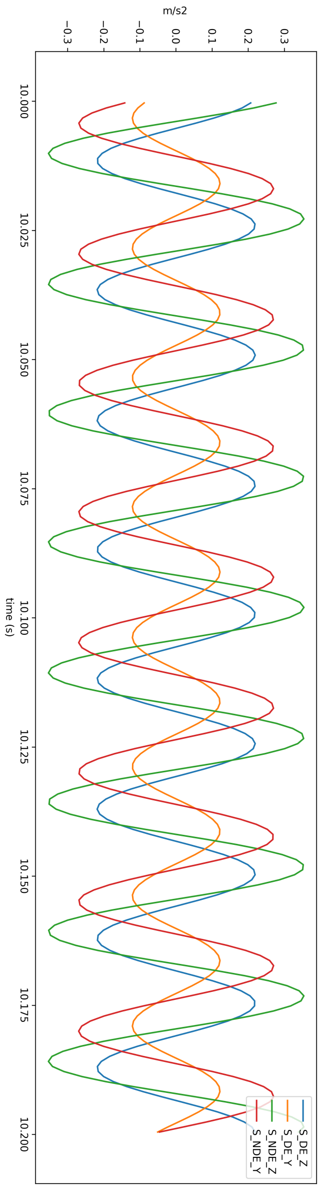


(b) cleaned

Figure 8.6: Data from unbalance configuration 5 with running frequency 20Hz



(a) Uncleaned



(b) Cleaned

Figure 8.7: Data from unbalance configuration 5 with running frequency 40Hz

vibration at few DOFs. Due to nature of the experimental setup, it is possible to reconstruct the rest of the DOFs using the reformulation of the equation of motion for the rotordynamic system.

Consider the model given in Figure 7.22. The system of equations representing the dynamics can be written as

$$\begin{bmatrix} \mathbf{D}_{\mathbf{R,ii}} & \mathbf{D}_{\mathbf{R,ib}} \\ \mathbf{D}_{\mathbf{R,bi}} & \mathbf{D}_{\mathbf{R,ii}} + \mathbf{D}_{\mathbf{B,bb}} \end{bmatrix} \begin{pmatrix} \mathbf{U}_{\mathbf{R,i}} \\ \mathbf{U}_{\mathbf{R,b}} \end{pmatrix} = \begin{bmatrix} \mathbf{F}_{\mathbf{R}} \\ 0 \end{bmatrix} \tag{8.6}$$

Here the elements of the dynamics system matrix $\mathbf{D}$ are grouped into different submatrices. The $\mathbf{D}_{\mathbf{R,ii}}$ represents the part of the matrix that belongs to the rotor but not containing bearings. The subsystem $\mathbf{D}_{\mathbf{R,ii}} + \mathbf{D}_{\mathbf{B,bb}}$ represents the part contributed to by bearing nodes. This part can be divided into two subparts; the contribution on bearing nodes from the rotor $\mathbf{D}_{\mathbf{R,ii}}$, and the contribution from bearing $\mathbf{D}_{\mathbf{B,bb}}$. The subsystems $\mathbf{D}_{\mathbf{R,ib}}$ and $\mathbf{D}_{\mathbf{R,bi}}$ represent the cross-coupling terms between rotor and bearings. Since the bearing does not contribute to those terms, they are completely known from rotor design. Correspondingly, the displacements $\mathbf{U}_{\mathbf{R,i}}$ and $\mathbf{U}_{\mathbf{R,b}}$ represent the displacement of rotor at non-bearing and bearing locations. Since we only have sensors at bearing locations, the $\mathbf{U}_{\mathbf{R,i}}$ is unknown on the right-hand-side, the forces are only acting on the discs. Hence, the forces on the bearing locations are $\mathbf{0}$ and on the rest of the system are calculated/known from the known unbalance configuration.

Modifying results from Equation 8.6 as

$$\mathbf{U}_{\mathbf{R,i}} = \mathbf{D}_{\mathbf{R,ii}}^{-1}[\mathbf{F}_{\mathbf{R}} - \mathbf{D}_{\mathbf{R,ib}}\mathbf{U}_{\mathbf{R,b}}] \tag{8.7}$$

More details on this reconstruction can be found in [107].

8.2.1 Bearing parameter identification

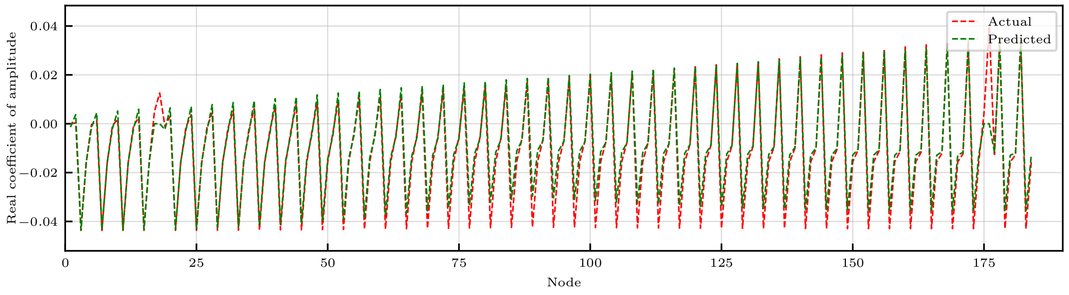
Once the displacement vector and force vector are available, the FEM-NN inverse algorithm can be used for identifying the unknown coefficients $\mathbf{D}_{\mathbf{R,bb}}$ of the dynamic stiffness matrix. Since the bearing we use is a hydrostatic fluid bearing, we assume the parameters as functions of speed. For example, different stiffness coefficients and damping coefficients can be written as

$$\begin{pmatrix} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{pmatrix} = f(\omega, \theta) \tag{8.8}$$

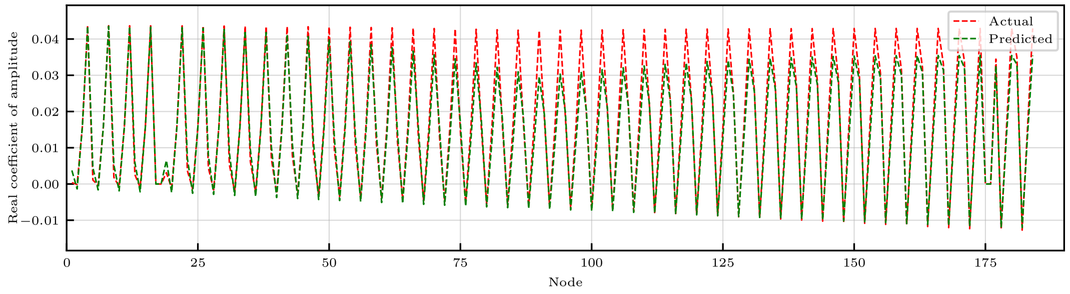
$$\begin{pmatrix} c_{xx} & c_{xy} \\ c_{yx} & c_{yy} \end{pmatrix} = f(\omega, \theta) \tag{8.9}$$

where, ω is the speed and θ is the neural network parameters. We use a neural network which predicts the bearing parameters (stiffness and damping coefficients) by taking rotating speed as input. The predicted parameters are assembled into the dynamic system matrix as explained in Algorithm 4. In contrast to the example discussed in Section 7.3, we do not know the actual values for the bearing coefficients in this case. Hence, we predict the vibration for a few of the experiments using the predicted bearing coefficients and compare it with the actual vibration data collected. Figure 8.8 shows the predicted and measured amplitudes of vibration for the unbalance setting 2 given in Table 8.4 with a rotational speed of 20Hz. Figures 8.8(a) and 8.8(b) show the real and imaginary parts of the complex amplitude for each of the nodes. It can be observed that the model is able to capture the amplitude pretty well for most of the nodes. However, at the bearing nodes the amplitudes are not perfectly captured and have high error in comparison to rest of the nodes. This can be attributed to the error accumulated from the different processes involved in the pipeline, starting from data collection to the model training.

Figures 8.9 and 8.10 show the vibration of each node in the time domain. Figure 8.9 shows the transient response of the disc nodes. The predictions are having amplitudes more than that of the measured. The difference between the prediction and error is on an average of 20% for the disc nodes. Figure 8.9 shows the transient response of some other DOFs in the system. It can be observed that the error is as low as 3% in these cases.

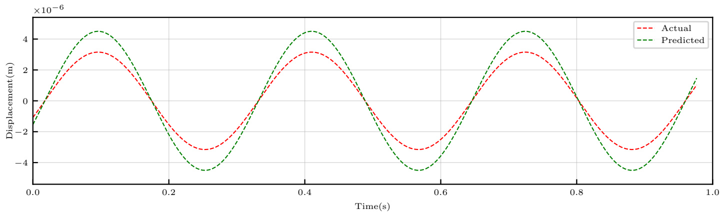


(a) Real part

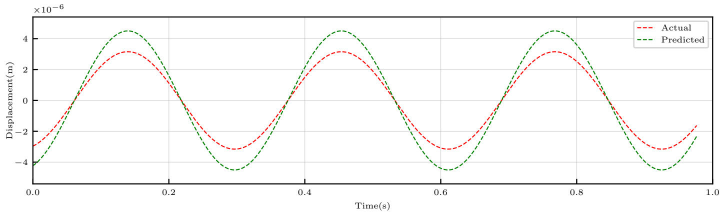


(b) Imaginary part

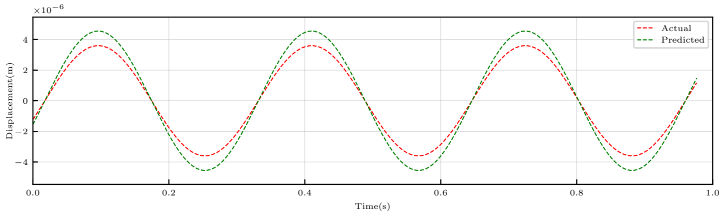
Figure 8.8: Measured vs predicted amplitudes of the rotordynamic system using the FEM-NN calculated bearing parameters



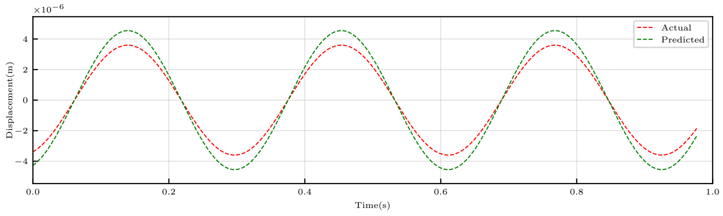
(a) First disc x-direction



(b) First disc y-direction

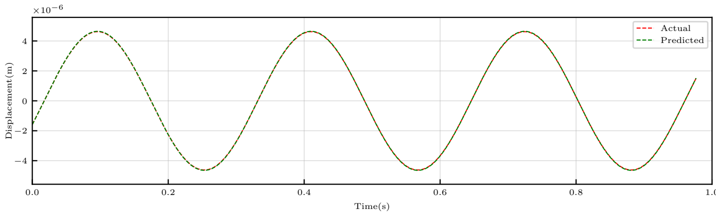


(c) Second disc x-direction

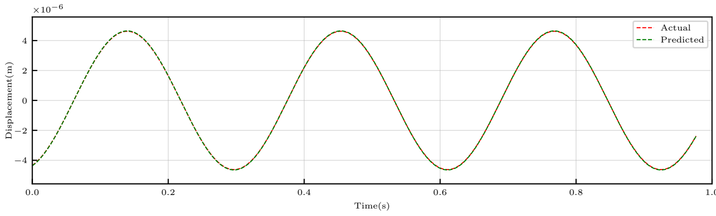


(d) Second disc y-direction

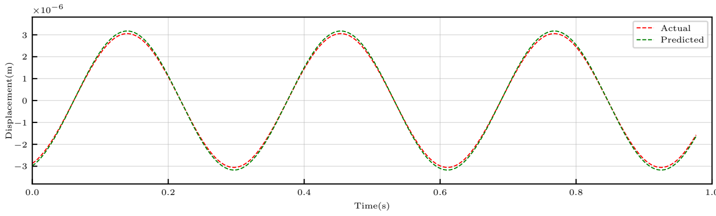
Figure 8.9: Predicted and actual transient response of disc nodes after calculating bearing parameters using FEM-NN



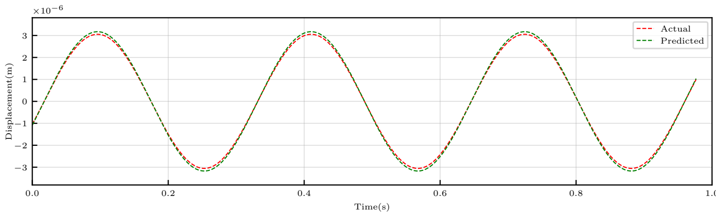
(a) DOF 10



(b) DOF 11



(c) DOF 150



(d) DOF 151

Figure 8.10: Predicted and actual transient response of random nodes after calculating bearing parameters using FEM-NN

Chapter 9

Conclusions and outlook

Conclusions

The xDT provides functionalities for operation and service optimization in industries with the help of real-time simulations. The xDTs constantly evolve along with a system to represent the current state of the system. A hybrid model combining FEM and NN is presented in this dissertation to support xDTs. This hybrid approach is a highly effective and reliable way of creating and using a surrogate model. The developed hybrid approach offers a general workflow for creating a neural network surrogate for any given physics and geometry. It also provides an efficient way to update the system parameters.

The fundamentals behind simulations using numerical methods and neural networks are examined to identify their strengths and weaknesses. Furthermore, the limitations of existing approaches combining simulation and neural networks are studied in detail. The state-of-the-art neural networks and neural networks for forward and inverse problems are thoroughly examined to analyze the research gap. As a result, a novel algorithm for training a physics-aware neural network and the concept of the hybrid model are introduced to realize faster simulation surrogate models. The discretized form of the governing PDE is used as the custom loss in the neural network for training. During training, the residual associated with the discretized form is backpropagated through NN. Deploying the resulting neural network with the numerical framework allowed us to verify the correctness of the prediction and discard if the residual is high. The same algorithm is extended to deal with inverse problems as well. Here the unknown part of a system is modeled as a neural network and assembled into the system matrix. This novel way of training is unsupervised as we use the governing PDE instead of the target data. The introduced algorithm and hybrid model are analyzed using examples of increasing complexity from different physics and geometry. The hybrid model is tested on thermal and structural problems in one-dimensional and two-dimensional problems of varying complexity. Both steady-state and transient problems were analyzed. The proposed method outperformed state-of-the-art neural network-based methods in terms of accuracy, speed, and reliability. Compared to many state-of-the-art methods, the introduced method results in a well-posed optimization problem representing the exact problem at hand and results in an accurate solution, provided the training has converged.

Along with the hybrid models, an application called Kratos neural network application is developed as part of the dissertation. As there is increasing interest in combining simulation and neural networks, a platform that enables easier integration of both worlds was necessary. The developed application can be used for simulation data generation, neural network training, and for using surrogate models along with other solvers in a multiphysics simulation. The feasibility of the application for running a multiphysics simulation was tested by running an FSI benchmark problem by replacing either fluid or structural with a surrogate model. The approach resulted in much less simulation time, provided we omitted the computational time and effort to train the surrogate.

Outlook

The introduced FEM-NN offers further research opportunities for the simulation and the neural network communities to improve aspects like training time. Some of them are beyond the scope of this thesis and are indicated in this section.

As pointed out in Chapters 6.2 and 8, one major challenge in training a neural network is hyperparameter tuning. Finding the right architecture, which represents the relation between input variables and primary output variable distribution in the domain, is time-consuming. Currently, the design of effective architecture is done empirically by researchers. Some parts of the hyperparameter tuning are now automatized using different hyperparameter tuners. However, such tuners also have various parameters to choose manually, starting from the sampling algorithm. This is one research area that the community must focus on to create an immediate impact. Even after finding the suitable architecture, the training part is expensive, and options like transfer learning must be explored with the given algorithm to speed up the training. A similar observation has also been made for inverse problems in Section 7.2. The amount of data required can be significantly reduced with the help of transfer learning approaches. Different transfer learning approaches are already quite successful in image processing. When it comes to physical problems, new algorithms that describe the underlying physics must be developed.

Another major issue when using neural network-based surrogates for physical problems is the need for a metric to measure their correctness. This dissertation proposed using the numerical grid to map the neural network surrogate solution and calculate the associated residual as a measure of correctness. However, this also demands the calculation of system matrices during the deployment. This contrasts with the advantage of using neural network surrogates for predicting in real-time. Hence, other methods are required to verify the correctness of neural network predictions.

Appendix A

Gradient calculation for complex numbers

There can be situations where complex numbers are used as input/-variables in a neural network training. The rotor dynamic example we considered in Section 8.2 belong to this. However, the final loss is always a real number.

Consider the gradient for a real cost function $\delta(z)$ defined on complex plane with argument $z = x + i\,y$

$$\begin{aligned}\nabla =& \frac{\partial \delta}{\partial x} + i\frac{\partial \delta}{\partial y}\\ =& \frac{\partial \delta}{\partial z}\frac{\partial z}{\partial x} + \frac{\partial \delta}{\partial z^*}\frac{\partial z^*}{\partial x} + i\left[\frac{\partial \delta}{\partial z}\frac{\partial z}{\partial y} + \frac{\partial \delta}{\partial z^*}\frac{\partial z^*}{\partial y}\right]\\ =& 2\frac{\partial \delta}{\partial z^*}\\ =& 2\left(\frac{\partial \delta}{\partial z}\right)^*\end{aligned} \tag{A.1}$$

When it comes to the parameters of a neural network, the derivative after backpropagation can be written as (similar to Equation 2.7)

$$\frac{\partial \delta}{\partial w_l} = \sum \frac{\partial \delta}{\partial w_{l+1}}\frac{\partial w_{l+1}}{\partial w_l} + \frac{\partial \delta}{\partial w^*_{l+1}}\frac{\partial w^*_{l+1}}{\partial w_l} \tag{A.2}$$

$$\frac{\partial \delta}{\partial b_l} = \sum \frac{\partial \delta}{\partial b_{l+1}}\frac{\partial b_{l+1}}{\partial b_l} + \frac{\partial \delta}{\partial b^*_{l+1}}\frac{\partial b^*_{l+1}}{\partial b_l} \tag{A.3}$$

When the function is differentiable, the second term in both Equations A.2 and A.3 vanishes and results an expression similar to that of real numbers. However, we need to take the conjugate in contrast to that of real numbers. This particular implementation is missing in most of the currently used neural network packages.

An example with the default PyTorch version and the effect of implementing above mentioned Equation A.1 is demonstrated below. We consider learning the complex number $1j$ beginning with the real number 1. In every iteration, a loss function calculates the error between the assumed complex number and the target. The mean squared error is backpropagated and the assumed complex number is adjusted using Adam optimizer implemented in PyTorch. In a second case, the Adam optimizer is modified according to Equation A.1 using the following change in the code.

```
exp_avg_sq.mul_(beta2).addcmul_(grad,grad.conj(),value=1-beta2)
```

Listing A.1: Change in Adam optimizer

A detailed discussion on the above topic is available at
https://github.com/pytorch/pytorch/issues/59998

```python
import torch as t
from matplotlib import pyplot as plt

complex_param = t.tensor([1],dtype=t.complex64, requires_grad=
    True)

# We will optimize on the mean squared error from a 1j
target = 1j

def calc_loss(x):
    return t.abs(x - target)**2

optimizer = t.optim.Adam([complex_param], lr=0.001)

n = 10000
values = t.zeros(n, dtype=t.complex64)
for i in range(n):
    optimizer.zero_grad()
    loss = calc_loss(complex_param)
    loss.backward()
    optimizer.step()
    values[i] = complex_param.detach()

# Plot the results
plt.plot(values.real, label='Real Part')
plt.plot(values.imag, label='Imaginary Part')
plt.legend()
plt.xlabel('Iteration')
plt.ylabel('Complex Parameter')
plt.title('Optimization Progress with as-implemented Adam')
plt.show()
```

Listing A.2: Code snippet to test complex gradient implementation in PyTorch

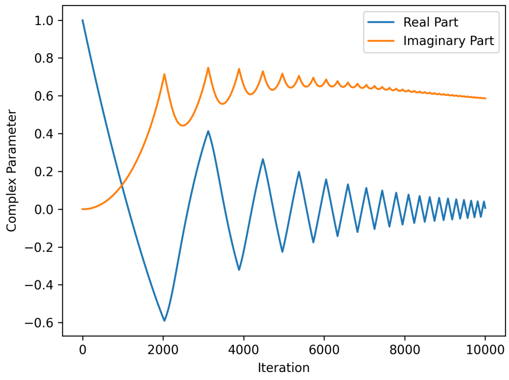


Figure A.1: Optimization progress with unresolved PyTorch error

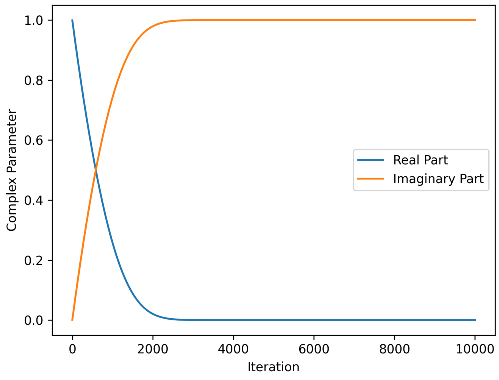


Figure A.2: Optimization progress with modification in PyTorch

List of Figures

List of Tables

Bibliography

[1] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad. "State-of-the-art in artificial neural network applications: A survey". In: *Heliyon* 4.11 (2018), e00938.

[2] H. Adeli. "Neural networks in civil engineering: 1989–2000". In: *Computer-Aided Civil and Infrastructure Engineering* 16.2 (2001), pp. 126–142.

[3] H. Adeli and C Yeh. "Perceptron learning in engineering design". In: *Computer-Aided Civil and Infrastructure Engineering* 4.4 (1989), pp. 247–256.

[4] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* 2019.

[5] S. Albawi, T. A. Mohammed, and S. Al-Zawi. "Understanding of a convolutional neural network". In: *2017 international conference on engineering and technology (ICET).* Ieee. 2017, pp. 1–6.

[6] D. Arcones, R. Meethal, B. Obst, and R. Wüchner. "Neural Network-Based Surrogate Models Applied to Fluid-Structure Interaction Problems". In: *Collection of papers presented at the WCCM-APCOM 2022, Yokohama, July 31 to August 5, 2022.*

[7] I. Asimov. "Runaround". In: *Astounding science fiction* 29.1 (1942), pp. 94–103.

[8] G. E. Backus and J. F. Gilbert. “Numerical applications of a formalism for geophysical inverse problems”. In: *Geophysical Journal International* 13.1-3 (1967), pp. 247–276.

[9] R. Bischof and M. Kraus. “Multi-objective loss balancing for physics-informed deep learning”. In: *arXiv preprint arXiv:2110.09813* (2021).

[10] S. Boschert and R. Rosen. “Digital twin—the simulation aspect”. In: *Mechatronic futures*. Springer, 2016, pp. 59–74.

[11] A. L. Braun and A. M. Awruch. “Aerodynamic and aeroelastic analyses on the CAARC standard tall building model using numerical simulation”. In: *Computers & Structures* 87.9-10 (2009), pp. 564–581.

[12] E. Buffa, J. Jacob, and P. Sagaut. “Lattice-Boltzmann-based large-eddy simulation of high-rise building aerodynamics with inlet turbulence reconstruction”. In: *Journal of Wind Engineering and Industrial Aerodynamics* 212 (2021), p. 104560.

[13] R. Burbidge, M. Trotter, B Buxton, and S. Holden. “Drug design by machine learning: support vector machines for pharmaceutical data analysis”. In: *Computers & chemistry* 26.1 (2001), pp. 5–14.

[14] S. Butterworth et al. “On the theory of filter amplifiers”. In: *Wireless Engineer* 7.6 (1930), pp. 536–541.

[15] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis. “Physics-informed neural networks (PINNs) for fluid mechanics: A review”. In: *Acta Mechanica Sinica* (2022), pp. 1–12.

[16] S. Cai, Z. Wang, S. Wang, P. Perdikaris, and G. E. Karniadakis. “Physics-informed neural networks for heat transfer problems”. In: *Journal of Heat Transfer* 143.6 (2021).

[17] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu. “Deep blue”. In: *Artificial intelligence* 134.1-2 (2002), pp. 57–83.

[18] G. Chen and J. Zhou. *Boundary element methods*. Vol. 92. Academic press London, 1992.

[19] X. Chen, S. Z. Wu, and M. Hong. “Understanding gradient clipping in private SGD: A geometric perspective”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 13773–13782.

[20] Y. Chen, L. Lu, G. E. Karniadakis, and L. Dal Negro. “Physics-informed neural networks for inverse problems in nano-optics and metamaterials”. In: *Optics express* 28.8 (2020), pp. 11618–11633.

[21] F. Chollet et al. “Keras: Deep learning library for theano and tensorflow”. In: *URL: https://keras. io/k* 7.8 (2015), T1.

[22] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[23] P. Dadvand, R. Rossi, and E. Oñate. “An object-oriented environment for developing finite element codes for multi-disciplinary applications”. In: *Archives of computational methods in engineering* 17.3 (2010), pp. 253–297.

[24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition.* Ieee. 2009, pp. 248–255.

[25] J. Denker, W Gardner, H. Graf, D. Henderson, R Howard, W Hubbard, L. D. Jackel, H. Baird, and I. Guyon. “Neural network recognizer for hand-written zip code digits”. In: *Advances in neural information processing systems* 1 (1988).

[26] H. Dongmei, H. Shiqing, H. Xuhui, and Z. Xue. “Prediction of wind loads on high-rise building using a BP neural network combined with POD”. In: *Journal of Wind Engineering and Industrial Aerodynamics* 170 (2017), pp. 1–17.

[27] C. Fairclough, M. Fagan, B. Mac Namee, and P. Cunningham. *Research directions for AI in computer games.* Tech. rep. Trinity College Dublin, Department of Computer Science, 2001.

[28] J. D. Funge. *Artificial intelligence for computer games: an introduction.* CRC Press, 2004.

[29] G. M. Gladwell. “Inverse problems in vibration”. In: (1986).

[30] X. Glorot and Y. Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[31] S. Godunov and I Bohachevsky. “Finite difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics”. In: *Matematičeskij sbornik* 47.3 (1959), pp. 271–306.

[32] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[33] R. Grzeszczuk, D. Terzopoulos, and G. Hinton. “Neuroanimator: Fast neural network emulation and control of physics-based models”. In: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. 1998, pp. 9–20.

[34] E. J. Gunter Jr. *Dynamic stability of rotor-bearing systems*. Tech. rep. 1966.

[35] X. Guo, W. Li, and F. Iorio. “Convolutional neural networks for steady flow approximation”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 481–490.

[36] E. Haghighat, M. Raissi, A. Moure, H. Gomez, and R. Juanes. “A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics”. In: *Computer Methods in Applied Mechanics and Engineering* 379 (2021), p. 113741.

[37] F. Han, X. Guo, and H. Gao. “Bearing parameter identification of rotor-bearing system based on Kriging surrogate model and evolutionary algorithm”. In: *Journal of Sound and Vibration* 332.11 (2013), pp. 2659–2671. doi: `10.1016/j.jsv.2012.12.025`.

[38] K. He, X. Zhang, S. Ren, and J. Sun. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[39] D. O. Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[40] R. Hecht-Nielsen. “Theory of the backpropagation neural network”. In: *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[41] G. Hinton, N. Srivastava, and K. Swersky. “Neural networks for machine learning lecture 6a overview of mini-batch gradient descent”. In: *Cited on* 14.8 (2012), p. 2.

[42] K. Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural networks* 4.2 (1991), pp. 251–257.

[43] F. Hou and M. Jafari. “Investigation approaches to quantify wind-induced load and response of tall buildings: A review”. In: *Sustainable Cities and Society* 62 (2020), p. 102376.

[44] S. Ioffe and C. Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[45] H. Janssen. “Monte-Carlo based uncertainty analysis: Sampling efficiency and sampling convergence”. In: *Reliability Engineering & System Safety* 109 (2013), pp. 123–132.

[46] C. Jiang, R. Vinuesa, R. Chen, J. Mi, S. Laima, and H. Li. “An interpretable framework of data-driven turbulence modeling using deep neural networks”. In: *Physics of Fluids* 33.5 (2021), p. 055133.

[47] D. Jurafsky. *Speech & language processing*. Pearson Education India, 2000.

[48] Y. Kang, Z. Shi, H. Zhang, D. Zhen, and F. Gu. “A novel method for the dynamic coefficients identification of journal bearings using Kalman filter”. In: *Sensors* 20.2 (2020), p. 565.

[49] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. “Physics-informed machine learning”. In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440.

[50] K. Katanforoosh and D. Kunin. “Initializing neural networks”. In: *DeepLearning. ai* (2019).

[51] M. Kawato, Y. Uno, M. Isobe, and R. Suzuki. “Hierarchical neural network model for voluntary movement with application to robotics”. In: *IEEE Control Systems Magazine* 8.2 (1988), pp. 8–15.

[52] D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[53] A. Kodakkal, B. Keith, U. Khristenko, A. Apostolatos, K.-U. Bletzinger, B. Wohlmuth, and R. Wuechner. “Risk-averse design of tall buildings for uncertain wind conditions”. In: *arXiv preprint arXiv:2203.12060* (2022).

[54] A. Kodakkal, R. E. Meethal, B. Obst, and R. WUCHNER. “A Finite Element Method-Informed Neural Network For Uncertainty Quantification”. In: *14th WCCM-ECCOMAS Congress 2020*. Vol. 800. 2021.

[55] A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, and M. W. Mahoney. “Characterizing possible failure modes in physics-informed neural networks”. In: *Advances in Neural Information Processing Systems* 34 (2021).

[56] G. Lamberti and C. Gorlé. “A multi-fidelity machine learning framework to predict wind loads on buildings”. In: *Journal of Wind Engineering and Industrial Aerodynamics* 214 (2021), p. 104647.

[57] A. Lavecchia. “Machine-learning approaches in drug discovery: methods and applications”. In: *Drug discovery today* 20.3 (2015), pp. 318–331.

[58] L. P. Lebedev, I. I. Vorovich, and G. M. L. Gladwell. *Functional analysis: applications in mechanics and inverse problems*. Vol. 41. Springer Science & Business Media, 2012.

[59] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444.

[60] A. Lees. “Identification of dynamic bearing parameters: a review”. In: *The Shock and Vibration Digest* 36.2 (2004), pp. 99–124.

[61] M. W. Libbrecht and W. S. Noble. “Machine learning applications in genetics and genomics”. In: *Nature Reviews Genetics* 16.6 (2015), pp. 321–332.

[62] J. Ling, A. Kurzawski, and J. Templeton. “Reynolds averaged turbulence modelling using deep neural networks with embedded invariance”. In: *Journal of Fluid Mechanics* 807 (2016), pp. 155–166.

[63] D. C. Liu and J. Nocedal. “On the limited memory BFGS method for large scale optimization”. In: *Mathematical programming* 45.1 (1989), pp. 503–528.

[64] J. Lund and K. Thomsen. “A calculation method and data for the dynamic coefficients of oil-lubricated journal bearings”. In: *Topics in fluid film bearing and rotor bearing system design and optimization* 1000118 (1978).

[65] A. Lydia and S. Francis. “Adagrad—an optimizer for stochastic gradient descent”. In: *Int. J. Inf. Comput. Sci* 6.5 (2019), pp. 566–568.

[66] C. C. Margossian. “A review of automatic differentiation and its efficient implementation”. In: *Wiley interdisciplinary reviews: data mining and knowledge discovery* 9.4 (2019), e1305.

[67] K. Mattiasson. “Numerical results from large deflection beam and frame problems analysed by means of elliptic integrals”. In: *International journal for numerical methods in engineering* 17.1 (1981), pp. 145–153.

[68] W. S. McCulloch and W. Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[69] L. R. Medsker and L. Jain. “Recurrent neural networks”. In: *Design and Applications* 5 (2001), pp. 64–67.

[70] R. E. Meethal, A. Ghantasala, P. Bucher, R. Wüchner, C. Heinrich, and K.-U. Bletzinger. “Co-Simulation of Multiphysics Problems with Data Driven SurrogateModels”. en. In: *WCCM-ECCOMAS*. Virtual, 2021.

[71] R. E. Meethal, A. Kodakkal, M. Khalil, A. Ghantasala, B. Obst, K.-U. Bletzinger, and R. Wüchner. “Finite element method-enhanced neural network for forward and inverse problems”. In: *Advanced Modeling and Simulation in Engineering Sciences* 10.1 (2023), p. 6.

[72] R. E. Meethal and L. S. P. R. Kondamadugula. “Generalized physics-informed machine learning for numerically solved transient physical systems”. In: *AAAI MLPS 2021, https://ceur-ws.org/Vol-2964/* (2021).

[73] A. T. Mohan and D. V. Gaitonde. “A deep learning based approach to reduced order modeling for turbulent flow control using LSTM neural networks”. In: *arXiv preprint arXiv:1804.09269* (2018).

[74] D. P. Mok. *Partitionierte Lösungsansätze in der Strukturdynamik und der Fluid-Struktur-Interaktion.* 2001.

[75] A. Nareyek. “AI in computer games”. In: *Queue* 1.10 (2004), pp. 58–65.

[76] B. Newkirk. “Varieties of shaft disturbances due to fluid films in journal bearings”. In: *Transactions of the American Society of Mechanical Engineers* 78.5 (1956), pp. 985–987.

[77] K. O’Shea and R. Nash. “An introduction to convolutional neural networks”. In: *arXiv preprint arXiv:1511.08458* (2015).

[78] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. “Automatic differentiation in pytorch”. In: (2017).

[79] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32 (2019), pp. 8026–8037.

[80] I. Pavlenko, V. Simonovskiy, V. Ivanov, J. Zajac, and J. Pitel. “Application of artificial neural network for identification of bearing stiffness characteristics in rotor dynamics analysis”. In: *Design, Simulation, Manufacturing: The Innovation Exchange.* Springer. 2018, pp. 325–335.

[81] R. Potthast. “A survey on sampling and probe methods for inverse problems”. In: *Inverse Problems* 22.2 (2006), R1.

[82] N. Qian. “On the momentum term in gradient descent learning algorithms”. In: *Neural networks* 12.1 (1999), pp. 145–151.

[83] Z. Qiu and A. Tieu. “Identification of sixteen force coefficients of two journal bearings from impulse responses”. In: *Wear* 212.2 (1997), pp. 206–212.

[84] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman. “Universal differential equations for scientific machine learning”. In: *arXiv preprint arXiv:2001.04385* (2020).

[85] M. Raissi, P. Perdikaris, and G. E. Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational physics* 378 (2019), pp. 686–707.

[86] W. M. Rankine. “On the centrifugal force of rotating shafts”. In: *Van Nostrand’s Eclectic Engineering Magazine (1869-1879)* 1.7 (1869), p. 598.

[87] R. Redheffer. “A machine for playing the game nim”. In: *The American Mathematical Monthly* 55.6 (1948), pp. 343–349.

[88] M. A. Roehrl, T. A. Runkler, V. Brandtstetter, M. Tokic, and S. Obermayer. “Modeling system dynamics with physics-informed neural networks based on lagrangian mechanics”. In: *IFAC-PapersOnLine* 53.2 (2020), pp. 9195–9200.

[89] O. Ronneberger, P. Fischer, and T. Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention.* Springer. 2015, pp. 234–241.

[90] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[91] P. C. Sabatier. “Past and future of inverse problems”. In: *Journal of Mathematical Physics* 41.6 (2000), pp. 4082–4124.

[92] K. B. Sautter. “Dynamic simulation of rock-fall protection nets: implementation and validation of large deformation finite elements in an open-source code”. In: (2017).

[93] I. W. Selesnick and C. S. Burrus. “Generalized digital Butterworth filter design”. In: *IEEE Transactions on signal processing* 46.6 (1998), pp. 1688–1694.

[94] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. “Taking the human out of the loop: A review of Bayesian optimization”. In: *Proceedings of the IEEE* 104.1 (2015), pp. 148–175.

[95] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587 (2016), pp. 484–489.

[96] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[97] B. Sudret. “Uncertainty propagation and sensitivity analysis in mechanical models–Contributions to structural reliability and stochastic spectral methods”. In: *Habilitationa diriger des recherches, Université Blaise Pascal, Clermont-Ferrand, France* 147 (2007), p. 53.

[98] R. Szeliski. *Computer vision: algorithms and applications.* Springer Science & Business Media, 2010.

[99] Y. Tamura and R. Yoshie. *Advanced environmental wind engineering.* Springer, 2016.

[100] A. Tarantola. *Inverse problem theory and methods for model parameter estimation.* SIAM, 2005.

[101] A. Tarantola, B. Valette, et al. “Inverse problems= quest for information”. In: *Journal of geophysics* 50.1 (1982), pp. 159–170.

[102] M. S. Thordal, J. C. Bennetsen, and H. H. H. Koss. “Review for practical application of CFD for the determination of wind load on high-rise buildings”. In: *Journal of Wind Engineering and Industrial Aerodynamics* 186 (2019), pp. 155–168.

[103] N. Thuerey, K. Weißenow, L. Prantl, and X. Hu. “Deep learning methods for Reynolds-averaged Navier–Stokes simulations of airfoil flows”. In: *AIAA Journal* 58.1 (2020), pp. 25–36.

[104] A. N. Tikhonov, V. J. Arsenin, V. I. Arsenin, V. Y. Arsenin, et al. *Solutions of ill-posed problems*. Vh Winston, 1977.

[105] R. Timbó, R. Martins, G. Bachmann, F. Rangel, J. Mota, J. Valério, and T. G. Ritto. “ROSS - Rotordynamic Open Source Software”. In: *Journal of Open Source Software* 5.48 (2020), p. 2120. doi: `10.21105/joss.02120`.

[106] R Tiwari and V Chakravarthy. “Simultaneous identification of residual unbalances and bearing dynamic parameters from impulse responses of rotor–bearing systems”. In: *Mechanical systems and signal processing* 20.7 (2006), pp. 1590–1614.

[107] R Tiwari, A. Lees, and M. Friswell. “Identification of speed-dependent bearing parameters”. In: *Journal of sound and vibration* 254.5 (2002), pp. 967–986.

[108] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin. “Accelerating eulerian fluid simulation with convolutional networks”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3424–3433.

[109] R. Tosi, M. Núñez, J. Pons-Prats, J. Principe, and R. Rossi. “On the use of ensemble averaging techniques to accelerate the Uncertainty Quantification of CFD predictions in wind engineering”. In: *Journal of Wind Engineering and Industrial Aerodynamics* 228 (2022), p. 105105.

[110] M. von Tresckow, S. Kurz, H. De Gersem, and D. Loukrezis. “A Neural Solver for Variational Problems on CAD Geometries with Application to Electric Machine Simulation”. In: *Journal of Machine Learning for Modeling and Computing* 3.1 (2022).

[111] A. M. Turing. “Computing machinery and intelligence”. In: *Parsing the turing test*. Springer, 2009, pp. 23–65.

[112] A. M. Turing and J Haugeland. “Computing machinery and intelligence”. In: *The Turing Test: Verbal Behavior as the Hallmark of Intelligence* (1950), pp. 29–56.

[113] J. G. Valdés Vázquez. “Nonlinear Analysis of Orthotropic Membrane and Shell Structures Including Fluid-Structure Interaction.” In: (2007).

[114] R. Vanluchene and R. Sun. “Neural networks in structural engineering”. In: *Computer-Aided Civil and Infrastructure Engineering* 5.3 (1990), pp. 207–215.

[115] H. K. Versteeg and W. Malalasekera. *An introduction to computational fluid dynamics: the finite volume method*. Pearson education, 2007.

[116] L. Von Rueden, S. Mayer, K. Beckh, B. Georgiev, S. Giesselbach, R. Heese, B. Kirsch, J. Pfrommer, A. Pick, R. Ramamurthy, et al. “Informed Machine Learning–A Taxonomy and Survey of Integrating Knowledge into Learning Systems”. In: *arXiv preprint arXiv:1903.12394* (2019).

[117] H. Wang and T. Wu. “Knowledge-enhanced deep learning for wind-induced nonlinear structural dynamic analysis”. In: *J. Struct. Eng* 146.11 (2020), p. 04020235.

[118] S. Wang, X. Yu, and P. Perdikaris. “When and why PINNs fail to train: A neural tangent kernel perspective”. In: *Journal of Computational Physics* 449 (2022), p. 110768.

[119] J. Weizenbaum. “ELIZA—a computer program for the study of natural language communication between man and machine”. In: *Communications of the ACM* 9.1 (1966), pp. 36–45.

[120] P. Werbos. “Beyond regression:” new tools for prediction and analysis in the behavioral sciences”. In: *Ph. D. dissertation, Harvard University* (1974).

[121] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar. “Integrating physics-based modeling with machine learning: A survey”. In: *arXiv preprint arXiv:2003.04919* 1.1 (2020), pp. 1–34.

[122] R.-T. Wu and M. R. Jahanshahi. “Deep convolutional neural network for structural dynamic response estimation and system identification”. In: *Journal of Engineering Mechanics* 145.1 (2019), p. 04018125.

[123] T. Wu and R. Snaiki. “Applications of machine learning to wind engineering”. In: *Frontiers in Built Environment* 8 (2022).

[124] Z. Xiang, W. Peng, X. Zheng, X. Zhao, and W. Yao. “Self-adaptive loss balanced physics-informed neural networks for the incompressible navier-stokes equations”. In: *arXiv preprint arXiv:2104.06217* (2021).

[125] G. N. Yannakakis. “AI in computer games : generating interesting interactive opponents by the use of evolutionary computation”. PhD thesis. University of Edinburgh, UK, 2005.

[126] Y. Yin, P. Yang, Y. Zhang, H. Chen, and S. Fu. “Feature selection and processing of turbulence modeling based on an artificial neural network”. In: *Physics of Fluids* 32.10 (2020), p. 105117.

[127] Yu, Y. Jianxun, X. Si, C. Hu, and Zhang. “A review of recurrent neural networks: LSTM cells and network architectures”. In: *Neural computation* 31.7 (2019), pp. 1235–1270.

[128] F.-G. Yuan, S. A. Zargar, Q. Chen, and S. Wang. “Machine learning for structural health monitoring: challenges and opportunities”. In: *Sensors and smart structures technologies for civil, mechanical, and aerospace systems 2020* 11379 (2020), p. 1137903.

[129] M. D. Zeiler. “Adadelta: an adaptive learning rate method”. In: *arXiv preprint arXiv:1212.5701* (2012).

[130] R. Zhang, Y. Liu, and H. Sun. “Physics-informed multi-LSTM networks for metamodeling of nonlinear structures”. In: *Computer Methods in Applied Mechanics and Engineering* 369 (2020), p. 113226.

[131] L. Zhu, W. Zhang, J. Kou, and Y. Liu. “Machine learning methods for turbulence modeling in subsonic flows around airfoils”. In: *Physics of Fluids* 31.1 (2019), p. 015105.

[132] W. Zhu, K. Xu, E. Darve, and G. C. Beroza. “A general approach to seismic inversion with automatic differentiation”. In: *Computers & Geosciences* 151 (2021), p. 104751.

[133] Q. Zhuang, D. Hartmann, H. J. Bungartz, and J. M. Lorenzi. “Active-learning-based non-intrusive Model Order Reduction”. In: *arXiv preprint arXiv:2204.08523* (2022).

[134] Q. Zhuang, J. M. Lorenzi, H.-J. Bungartz, and D. Hartmann. “Model order reduction based on Runge–Kutta neural networks”. In: *Data-Centric Engineering* 2 (2021).

[135] O. Zienkiewicz and R. Taylor. *The Finite Element Method, The Basis*. The Finite Element Method. Wiley, 2000. isbn: 9780470395042.

Bisherige Titel der Schriftenreihe

Band	**Titel**
23	Rupert Fisch, *Code Verification of Partitioned FSI Environments for Lightweight Structures*, 2014.
24	Stefan Sicklinger, *Stabilized Co-Simulation of Coupled Problems Including Fields and Signals*, 2014.
25	Madjid Hojjat, *Node-based parametrization for shape optimal design*, 2015.
26	Ute Israel, *Optimierung in der Fluid-Struktur-Interaktion - Sensitivitätsanalyse für die Formoptimierung auf Grundlage des partitionierten Verfahrens*, 2015.
27	Electra Stavropoulou, *Sensitivity analysis and regularization for shape optimization of coupled problems*, 2015.
28	Daniel Markus, *Numerical and Experimental Modeling for Shape Optimization of Offshore Structures*, 2015.
29	Pablo Suárez, *Design Process for the Shape Optimization of Pressurized Bulkheads as Components of Aircraft Structures*, 2015.
30	Armin Widhammer, *Variation of Reference Strategy - Generation of Optimized Cutting Patterns for Textile Fabrics*, 2015.
31	Helmut Masching, *Parameter Free Optimization of Shape Adaptive Shell Structures*, 2016.
32	Hao Zhang, *A General Approach for Solving Inverse Problems in Geophysical Systems by Applying Finite Element Method and Metamodel Techniques*, 2016.
33	Tianyang Wang, *Development of Co-Simulation Environment and Mapping Algorithms*, 2016.
34	Michael Breitenberger, *CAD-integrated Design and Analysis of Shell Structures*, 2016.

Band	**Titel**
45	Franz-Josef Ertl, *Vertex Morphing for Constrained Shape Optimization of Three-dimensional Solid Structures*, 2020.
46	Daniel Baumgärtner, *On the Grid-based Shape Optimization of Structures with Internal Flow and the Feedback of Shape Changes into a CAD Model*, 2020.
47	Mohamed Khalil, *Combining Physics-based models and machine learning for an Enhanced Structural Health Monitoring*,2021.
48	Long Chen, *Gradient Descent Akin Method*, 2021.
49	Aditya Ghantasala, *Coupling Procedures for Fluid-Fluid and Fluid-Structure Interaction Problems Based on Domain Decomposition Methods*, 2021.
50	Ann-Kathrin Goldbach, *The Cad-Integrated Design Cycle for Structural Membranes*, 2022.
51	Iñigo Pablo López Canalejo,, *A Finite-Element Transonic Potential Flow Solver with an Embedded Wake Approach for Aircraft Conceptual Design*, 2022.
52	Mayu Sakuma, *An Application of Multi-Fidelity Uncertainty Quantification for Computational Wind Engineering*, 2022.
53	Suneth Warnakulasuriya, *Development of methods for Finite Element-based sensitivity analysis and goal-directed mesh refinement using the adjoint approach for steady and transient flows*, 2022.
54	Klaus Bernd Sautter, *Modeling and Simulation of Flexible Protective Structures by Coupling Particle and Finite Element Methods*, 2022.
55	Efthymios Papoutsis, *On the incorporation of industrial constraints in node-based optimization for car body design*, 2023.