Modelling of Reed Valves with Neural Networks based on FSI Data

Bhagyashree Bhagat
Modelling of Reed Valves with Neural Networks based on FSI Data

Bhagyashree Bhagat
Masterarbeit

Aufgabensteller: Prof. Dr. -Ing Harald Köstler
Betreuer: Prof. Dr. -Ing Harald Köstler
Bearbeitungszeitraum: 1.2.2023 – 31.07.2023
Erklärung:

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Der Universität Erlangen-Nürnberg, vertreten durch den Lehrstuhl für Systemsimulation (Informatik 10), wird für Zwecke der Forschung und Lehre ein einfaches, kostenloses, zeitlich und örtlich unbeschränktes Nutzungsrecht an den Arbeitsergebnissen der Masterarbeit einschließlich etwaiger Schutzrechte und Urheberrechte eingeräumt.

Erlangen, den 31. Juli 2023

........................................

..........................
Abstract

Schaeffler is developing an integrated thermal management system including the refrigerant circuit and its components. A challenging but necessary task is the modeling and simulation of the dynamic behavior and interaction of reed valves with the flow. This thesis explores the development and integration of an integrated thermal management system, including the refrigerant circuit and its components, with a focus on modeling and simulating the dynamic behavior and interaction of reed valves with fluid flow. The use of computational and analytical methods for predicting the valve’s flapper behavior is investigated, requiring detailed numerical simulations that consider various physical phenomena. To potentially accelerate the simulation process and mitigate cost pressures, the study examines the integration of deep-learning neural networks (DNN) with Fluid-Structure Interaction (FSI). The research employs FSI in combination with Finite Element Analysis (FEA) and unsteady Computational Fluid Dynamics (CFD) to account for "unknown effects" resulting from interactions between fluid forces and deformable structures.

The neural network undergoes training with fluid properties, boundary conditions, and selected node coordinates in the valve system to predict final velocities and mass flow rate. It is initially trained with specific inlet velocities and subsequently tested with different velocities to evaluate its performance. While good accuracy is achieved for input velocities within the training range, performance degrades when predicting for out-of-scope velocities. Various approaches are suggested to address this issue, including data expansion and model fine-tuning.

The study provides insights into fluid flow characteristics, system behavior, and neural network limitations, facilitating improvements in real-world reed valve dynamics prediction. The integration of AI techniques, particularly data-driven neural networks, offers a valuable tool for enhancing performance and design refinement while reducing computational costs. Overall, the thesis contributes to understanding the complexities of the valve system and presents potential advancements in analysis and modeling.
I had the opportunity to complete my master thesis at Schaeffler Technologies AG & Co. KG and I would like to extend my heartfelt gratitude to my supervisors, Dr.-Ing. Ulf Meerwald and Mr. Raik Prueller. Their unwavering support, guidance, and expertise have been invaluable throughout my thesis journey. Dr.-Ing. Ulf Meerwald’s vast knowledge and deep understanding of the subject matter have been instrumental in shaping my research. His insightful feedback, constructive criticism, and consistent encouragement have pushed me to explore new perspectives and approach challenges with confidence. Additionally, I am grateful to Mr. Raik Prueller for his practical insights, technical expertise, and attention to detail, which have significantly contributed to the success of my thesis. Their dedication to my academic progress and their willingness to invest their time and expertise in my work is sincerely appreciated.

I would also like to sincerely express my deepest gratitude to Prof. Dr.-Ing Harald Köstler for providing me with the opportunity to work on this captivating and challenging topic of utilizing a Deep Learning model for FSI. Throughout the project, Prof. Dr.-Ing Harald Köstler has been readily available to offer assistance, provide valuable suggestions, and guide me in making informed decisions. Working on this topic has been an enriching experience, and I have gained a profound understanding of fluid dynamics, fluid-structure interaction, and deep learning. At each stage of the project, his support and guidance have been indispensable, inspiring me to achieve my best.

Lastly, I extend my thanks to my friends and family for their unwavering love and support. Their encouragement and belief in my abilities have been a constant source of motivation throughout this academic journey. Their presence in my life has made all the difference, and I am deeply grateful for their continued support.
## Contents

1 Introduction ........................................... 5  
  1.1 Motivation ........................................... 5  
  1.2 Literature Review ..................................... 7  
  1.3 Research Goals and Objective ......................... 8  

2 Methodology ........................................... 10  
  2.1 Geometry and materials properties ...................... 10  
  2.2 Computational Domain details ......................... 13  
  2.3 Finite Element Methods .................................. 13  
  2.4 Fluid Dynamics ......................................... 14  
  2.5 Fluid-Structure Interaction .............................. 19  
  2.6 Mesh .................................................. 20  
  2.7 Deep learning models .................................... 24  

3 Implementation ......................................... 36  
  3.1 Model Setup ........................................... 36  
  3.2 Transient Structural ..................................... 36  
  3.2.1 Mesh ................................................ 37  
  3.2.2 Setup ............................................... 37  
  3.3 Fluent ................................................ 39  
  3.3.1 Mesh ................................................ 40  
  3.3.2 Setup ............................................... 45  
  3.4 System Coupling ........................................ 46  
  3.5 Data generation and Preparation for the Neural Network 48  
  3.5.1 Data generation ...................................... 48  
  3.5.2 Pre-processing of data ................................ 49  

4 Experimental Results ................................... 54  
  4.1 Analysis and interpretation of the Ansys results ....... 54  
  4.1.1 Different boundary conditions and mesh settings ... 54  
  4.1.2 Validation of the FEM/CFD/FSI results ............... 62  
  4.2 Neural Network ......................................... 62  
  4.2.1 Analysis and Interpretation .......................... 64  

5 Conclusion ............................................... 71  
  5.1 Summary ................................................ 71  
  5.2 Limitations and Future Work .............................. 73  

A MATLAB code ........................................... 75
List of Tables

2.1 Geometrical Dimensions of the Valve System where the inlet geometry is circular in
shape, and the diameter of the cylinder, represent the curvature of the upper end stop. 11
2.2 Material properties of the structural body . . . . . . . . . . . . . . . . . . . . . . . . . 12
3.1 Mesh Quality Metrics [1] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
3.2 Comparison of mesh quality: skewness . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
3.3 The dataset with various input and output parameters at different time steps . . . . 50
3.4 Software and their versions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
4.1 Comparison between LSTM and Fully Connected layers . . . . . . . . . . . . . . . . 68

List of Figures

1.1 Example of a Flapper Valve . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.1 Schematic Representation of the Valve System. The diagram illustrates the key
components of the valve system namely the valve, upper and lower end-stops. It also
highlights the inlet and outlet directions, clearly indicating the flow path of the fluid
within the system. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
2.2 A detailed representation of the dimensions of the valve. All dimensions are given in
millimeters (mm). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
2.3 A visual comparison of the structural body and volume body of the model . . . . . . 12
2.4 Types of elements in FEA [2] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
2.5 Computational Mesh of a pipe . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
2.6 Flow over a plate [3] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
2.7 Cell mesh [4] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
2.8 Two different approaches for FSI . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
2.9 2D mesh elements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
2.10 Different mesh element types . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
2.11 Comparison of high and low aspect ratio per element type [1] . . . . . . . . . . . . . 22
2.12 Comparison of high and low skewness per element type [1] . . . . . . . . . . . . . . 23
2.13 Biological neuron [5] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
2.14 Perceptron Model [5] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
2.15 Understanding backpropagation [6] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
2.16 Difference between Batch gradient, mini-batch gradient and stochastic gradient descent
   method [7] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
2.17 Fully Connected Network . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
2.18 Architecture of Recurrent Neural Network [8] . . . . . . . . . . . . . . . . . . . . . . . 31
2.19 Underfitting and overfitting of a model [9] . . . . . . . . . . . . . . . . . . . . . . . . . 33
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Mesh of the structural body</td>
</tr>
<tr>
<td>3.2</td>
<td>Contact interfaces</td>
</tr>
<tr>
<td>3.3</td>
<td>Setup of the transient structural component</td>
</tr>
<tr>
<td>3.4</td>
<td>Mesh details for the simulation with outlet pressure of 60 bar and inlet velocity of 10 m/s</td>
</tr>
<tr>
<td>3.5</td>
<td>Mesh difference with boundary layers</td>
</tr>
<tr>
<td>3.6</td>
<td>Mesh with boundary layers</td>
</tr>
<tr>
<td>3.7</td>
<td>Workbench setup</td>
</tr>
<tr>
<td>3.8</td>
<td>Coupling strategy for FSI [10]</td>
</tr>
<tr>
<td>3.9</td>
<td>Analysis Setting for system coupling</td>
</tr>
<tr>
<td>4.1</td>
<td>Comparison of CFD results for a reed valve, displaying static pressure and velocity magnitude</td>
</tr>
<tr>
<td>4.2</td>
<td>Static pressure of the inlet over time</td>
</tr>
<tr>
<td>4.3</td>
<td>Final position of the valve</td>
</tr>
<tr>
<td>4.4</td>
<td>Different outlet pressure</td>
</tr>
<tr>
<td>4.5</td>
<td>Comparison at uniform outlet 120 bar with different velocities of 5 m/s, 15 m/s, and 25 m/s</td>
</tr>
<tr>
<td>4.6</td>
<td>CFD results of reed valve with changing pressure</td>
</tr>
<tr>
<td>4.7</td>
<td>RMSE with different learning rates</td>
</tr>
<tr>
<td>4.8</td>
<td>Training progress of the neural network</td>
</tr>
<tr>
<td>4.9</td>
<td>Verbos of Training Progress</td>
</tr>
<tr>
<td>4.10</td>
<td>Performance Evaluation with Test Data</td>
</tr>
<tr>
<td>4.11</td>
<td>Z Velocity Distribution at Various Nodes of the Valve (m/s)</td>
</tr>
<tr>
<td>4.12</td>
<td>Structure of different Network Architectures</td>
</tr>
<tr>
<td>4.13</td>
<td>Z Velocity at Node 1 for Different Network Architectures</td>
</tr>
<tr>
<td>4.14</td>
<td>Network trained with 8 m/s and 9 m/s and predicted for 8 m/s and 10 m/s</td>
</tr>
<tr>
<td>4.15</td>
<td>Network trained with 8 m/s and 11 m/s and predicted for 9 m/s and 10 m/s</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

Schaeffler is developing an integrated thermal management system including the refrigerant circuit and its components. A challenging but necessary task is the modeling and simulation of the dynamic behavior and interaction of reed valves with the flow. Reed valves are critical parts of reciprocating compressors, which are important in many different sectors, including natural gas processing, air conditioning, and refrigeration. Nearly half of the compressor’s thermodynamic losses, according to Ribas et al. [11], are caused by the suction and discharge processes. Because of this, special emphasis is placed on the flexible reed valves that control the flow. The appropriate working of the reed valves is crucial for the effective operation of these compressors. Therefore, it is crucial to comprehend how the reed valves function and their interaction with the flow of fluid through them. In terms of its fundamental use, the valve is a simple one-way mechanism used to control the mass flow exchanged between two volumes distinguished by distinct pressure conditions. The pressure gradient, which acts on the surfaces of the valve, encourages valve deformation before the opening in a certain direction, resulting in a change in the cross-section. Here are some steps that can be taken to model and simulate the dynamic behavior and interaction of reed valves:

- Define the geometry and material properties: The geometry of the reed valve and any accompanying parts, such as the valve plate and end-stops, must first be established. It is also necessary to define the material characteristics of each component, such as density and elastic modulus.

- Develop the mathematical model: The next step is to develop a mathematical model that describes the dynamic behavior and interaction of the reed valve. This model should take into account the motion of the valve, the fluid flow characteristics, and the deformation of the valve and associated components.

- Implement the model in simulation software: The mathematical model can then be implemented in simulation software, such as Ansys or COMSOL Multiphysics, using CFD or FEA techniques. The software can be used to solve the equations of motion and fluid flow and to predict the behavior of the reed valve under different operating conditions.

- Analyze the results: Once the simulation is complete, the results can be analyzed to gain insights into the dynamic behavior and interaction of the reed valve. This can include evaluating the stress and strain distributions, the fluid flow patterns, and the overall performance of the valve.

- Refine the model and repeat: Based on the results of the analysis, the mathematical model can be refined and the simulation repeated to improve the accuracy and reliability of the predictions.
Overall, modeling and simulating the dynamic behavior and interaction of reed valves can be a challenging task, but it can provide valuable insights into the performance of these important components in compressors and pumps.

The fluid flow via the reed valves can be thoroughly modeled using CFD. Computational and analytical methods are very useful for predicting the flapper behavior of the valve. A classical approach relying on concentrated parameters (virtual mass, stiffness, damping, ...) requires resolved numerical simulations of the relevant effects for parametrization anyway in some form, e.g. by CFD and/or mechanical analyses. However, because of the intricacy of the valve shape and the flow mechanics, CFD models have a significant computing cost. This is due to the large number of computational elements required to accurately model the fluid flow patterns and the complex geometry of the valve.

Despite being helpful for decision-making, these high-precision 3D engineering simulations may be costly and time-consuming to execute. Computational fluid dynamics (CFD) and deep-learning neural networks might be usefully combined to potentially accelerate the simulation process. This thesis investigates the possibility of merging artificial neural networks (ANN) with CFD to quicken the simulation process in order to assist relieve the cost pressures. ANNs have demonstrated promising results in approximating complicated fluid flow issues with a relatively cheap computing cost. They accomplish this by using deep learning models that, without the need for intricate simulations, roughly mimic the behavior of fluid flow. The idea behind using neural networks is to train the network on a set of input-output pairs, where the input represents the parameters of interest (like the operating conditions) and the output represents the corresponding fluid flow behavior (like the pressure difference, velocity, or mass flow rate). Without the requirement for a thorough CFD simulation, the network may be trained to anticipate the fluid flow behavior for fresh sets of input parameters.

The benefit of using neural networks is that they can be trained using a small number of simulations, which can greatly lower the computational cost when compared to doing a large number of Fluid-Structure Interaction (FSI) simulations. The quality and representativeness of the training data are crucial for the surrogate model’s accuracy, it should be noted. Utilizing neural networks as part of a machine learning-based optimization or control technique in addition to the simulation itself is one more method for employing them to reduce the computing costs for FSI simulations. In this method, the simulation helps to validate and improve the predictions made by the neural network, which predicts the fluid flow behavior for various combinations of input parameters. This strategy can help to keep the optimization or control algorithm’s accuracy and dependability at a high level while lowering its total computing cost.

Therefore, integrating CFD with ANNs can be an effective tactic for modeling the operation of reed valves. This paper investigates a novel approach for modeling reed valves with ANNs based on CFD data. The proposed method aims to provide computationally efficient predictions of the reed valve performance that will help in the design and optimization of reciprocating compressors.
Chapter 1. Introduction 1.2 Literature Review

The fluid-structure interaction (FSI) problem associated with reed valves presents a challenging task due to its complexity. Previous studies have extensively utilized computational fluid dynamics (CFD) to predict flow fields and investigate the impact of fluid flow and pressure fields on structural deformations, and vice versa. FSI problems have been explored in various fields, including biology and bio-medicine, such as studying fluid structure interaction in the cardiovascular system [12], and simulating flapping elastic wings in birds and insects [13].

Several commercial programs, such as Ansys, ADINA, and COMMSOL, offer FSI capabilities that leverage advancements in computational fluid dynamics (CFD), computational structure mechanics (CSM), and numerical algorithms. These programs provide realistic and time-efficient numerical FSI analysis. In the context of valve systems, previous studies have utilized CFD simulations to predict laminar flows through valves [14, 15]. These simulations employed different approaches, including one-dimensional models and predefined motion, and yielded promising agreements with experimental data [14].

The accuracy of CFD simulations is influenced by the mesh resolution, as well as other factors such as system geometry, boundary conditions, and mesh quality. Achieving high accuracy with smaller mesh sizes increases computational costs, making the process time-consuming. To address this, machine learning (ML) techniques, particularly deep learning neural networks, have shown promise in predicting solutions for partial differential equations. By integrating deep learning with CFD, simulation processes can be accelerated. Two approaches have been proposed: compressing 3D simulation results to 1D and creating a neural network workflow, or developing surrogate models for system components to significantly speed up 3D simulations.

Researchers have applied parameterized CFD simulations and deep neural network (DNN) models to predict fluid behavior in pipelines, resulting in accurate predictions with significantly reduced inference times compared to traditional CFD simulations [16]. This approach requires parameterizing the CFD simulations to train the DNN model for predicting outputs based on inputs from new data sets. An investigation into the mesh dependence of simulation accuracy and its impact on computational cost is necessary [17].

Studies have also focused on utilizing ML techniques, such as artificial neural networks (ANNs), for predicting pressure drop in two-phase flow systems [18]. These studies demonstrated good precision in predicting gas-liquid two-phase flow patterns and pressure drop using CFD and ANN models. ML-CFD hybrid approaches have been explored, where ML models initialize CFD simulations to achieve faster convergence, reduced computational costs, and accurate predictions of time-based fluid flow patterns [17]. Additionally, researchers have utilized ML techniques, such as convolutional neural networks (CNNs), to efficiently estimate velocity fields in steady laminar flows. The CNN models demonstrated significantly faster estimation speeds compared to traditional CFD methods while maintaining low error rates, allowing for lightweight flow performance feedback to improve design interactivity and optimization [19].

Although there have been successful attempts to apply neural networks to predict fluid flows and optimize turbine blade profiles, challenges remain when extending neural networks to 3D simulations due to increased complexity, data acquisition difficulties, higher computational requirements, model design challenges, and interpretability issues. Future research should aim to explore the applicability of ML approaches to different fluid flow scenarios and optimize ML models and CFD simulations for improved accuracy and efficiency in 3D simulations.

In conclusion, previous studies have utilized CFD simulations, ML techniques, and hybrid ML-CFD approaches to predict fluid flow behavior, pressure drop, and other parameters in various flow systems. These approaches have shown promise in terms of accuracy, computational efficiency, and prediction capabilities. However, further research is required to address the challenges associated with extending ML and deep learning approaches 3D simulations and to optimize the performance of ML models and CFD simulations for improved accuracy and efficiency in complex flow scenarios.
1.3 Research Goals and Objective

The research goals of this study are twofold. The first goal is to model and simulate the dynamic behavior and interaction of reed valves with fluid flow. This involves investigating the intricate dynamics between the valve and the fluid flow, considering various factors such as different boundary conditions, fluid flow patterns, and corresponding valve responses. The aim is to gain a comprehensive understanding of the behavior and dynamics of reed valves in different operating scenarios.

The second goal is to generate a comprehensive dataset through Fluid-Structure Interaction (FSI) simulations. These simulations will capture the behavior of reed valves and the generated dataset will serve as a valuable resource for training and validation purposes. Based on this dataset, the study aims to develop a neural network model capable of accurately predicting the final position of the valve in response to different operating conditions. This involves preprocessing and analyzing the dataset to ensure its quality and relevance for training. The neural network model will be designed to take relevant input parameters, such as inlet velocity, pressure, and fluid properties, and provide accurate predictions of the valve’s final position. To achieve these goals, the study will validate and evaluate the performance of the developed neural network model using test data and appropriate performance metrics. The model’s ability to generalize to unseen operating conditions and its predictive capabilities for real-world valve systems will be assessed. The outcomes of this study have the potential to contribute to the optimization and design improvement of valve systems in various industries.
Chapter 2

Methodology

One of the primary objectives of this thesis is to model and simulate the dynamic behavior and interaction between reed valves and the flow within the system. In this chapter, the focus is on the key aspects related to the analysis and modeling of the valve system. The geometry of the valve and the materials used are described, providing essential information for accurate modeling. The computational domain is discussed, covering the fundamentals of the finite element method (FEM) and computational fluid dynamics (CFD), which are crucial for simulating the behavior of the system. The concept of fluid-structure interaction (FSI) is introduced, highlighting its significance in capturing the dynamic interaction between fluid flow and structural deformations. Meshing techniques and quality metrics are explored to ensure accurate simulation results. Deep learning models are introduced, including the fundamentals of neurons and multi-layer perceptrons, as well as the backpropagation algorithm and stochastic gradient method for training neural networks. Different network architectures and their accuracy metrics are also discussed. This chapter provides the necessary groundwork for the subsequent analysis and modeling of the valve system in the thesis.

2.1 Geometry and materials properties

The performance of a reed valve is significantly impacted by its geometry. Factors such as the dimensions, size, shape, and spacing between the components play a crucial role in determining the amount of gas or liquid that can pass through the valve, the pressure drops across it, and the response time of the valve. When designing and operating the valve, consideration must be given to the pressure differential between the two volumes, the desired flow rate, the fluid properties (such as viscosity and density), and the operating conditions (such as temperature and pressure). By carefully considering these factors, engineers can develop valves that are reliable, efficient, and effective in controlling fluid flow across various applications.

The structure of the flapper valve, shown in Figure 2.1, mainly consists of the upper end-stop, valve, and lower end-stop. For the valve, in figure 2.2, a rectangular T-shaped plate, fixed at one end and free at the other, is used for the model simulations.
Chapter 2. Methodology

2.1. Geometry and materials properties

Figure 2.1: Schematic Representation of the Valve System. The diagram illustrates the key components of the valve system namely the valve, upper and lower end-stops. It also highlights the inlet and outlet directions, clearly indicating the flow path of the fluid within the system.

Figure 2.2: A detailed representation of the dimensions of the valve. All dimensions are given in millimeters (mm).

<table>
<thead>
<tr>
<th>Dimension (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radius of inlet</td>
</tr>
<tr>
<td>Cylinder diameter</td>
</tr>
</tbody>
</table>

Table 2.1: Geometrical Dimensions of the Valve System where the inlet geometry is circular in shape, and the diameter of the cylinder, represent the curvature of the upper end stop.

An austenitic steel alloy is employed for the valve material, while the upper and lower end-stops are made of the 16MnCr5 alloy. The valve plate is typically located in a housing or cylinder head, and the valve flexes in response to changes in pressure. Figure 2.3 consists of two subfigures highlighting their significance in depicting the internal structure and external geometry of the model. The first subfigure 2.3a illustrates the structural body of the geometry, providing a visual representation of the key components. The second subfigure 2.3b showcases the volume body of the model.
Chapter 2. Methodology

2.1. Geometry and materials properties

(a) A visual representation of the structural body of the valve, along with the lower and upper endstops.

(b) Volume body

Figure 2.3: A visual comparison of the structural body and volume body of the model

The material constants of the valve and the end-stops are given in table 2.2a and 2.2b.

<table>
<thead>
<tr>
<th>Material Properties</th>
<th>Value</th>
<th>Material Properties</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Material name</td>
<td>Austenitic Steel</td>
<td>Material name</td>
<td>16MnCr5</td>
</tr>
<tr>
<td>Fluid</td>
<td>False</td>
<td>Fluid</td>
<td>False</td>
</tr>
<tr>
<td>Density</td>
<td>0.0079 g/mm³</td>
<td>Density</td>
<td>0.0077 g/mm³</td>
</tr>
<tr>
<td>Ultimate Strength</td>
<td>650000000 Pa</td>
<td>Ultimate Strength</td>
<td>7500000000 Pa</td>
</tr>
<tr>
<td>Elastic Modulus</td>
<td>2E+11 Pa</td>
<td>Elastic Modulus</td>
<td>2E+11 Pa</td>
</tr>
<tr>
<td>Poisson’s Ratio</td>
<td>0.3</td>
<td>Poisson’s Ratio</td>
<td>0.3</td>
</tr>
<tr>
<td>Thermal Conductivity</td>
<td>150 W/m-K</td>
<td>Thermal Conductivity</td>
<td>25 W/m-K</td>
</tr>
<tr>
<td>Specific Heat</td>
<td>500 J/kg-degC</td>
<td>Specific Heat</td>
<td>460 J/kg-degC</td>
</tr>
</tbody>
</table>

(a) Material properties of the valve

(b) Material properties of the valve

Table 2.2: Material properties of the structural body

This section has analyzed an overview of the valve geometry, model design, and the materials utilized for the components. In the case of this project, the valve material of choice is an austenitic steel alloy, known for its excellent mechanical properties and suitability for various operating conditions. Understanding the interplay between these variables enables the development of optimal valve designs tailored to specific fluid dynamics requirements and operating conditions. Moving forward, the next section will delve into the methodology of the computational domain employed in this work, providing insights into the techniques and approaches used to investigate and analyze the
2.2 Computational Domain details

This section explores fundamental ideas of Finite Element Models (FEM) and Computational Fluid Dynamics (CFD), theoretical approaches utilized in Fluid-Structure Interaction (FSI) simulations, and various types of FSI algorithms. The objective is to provide a comprehensive understanding of the simulation techniques and methodologies employed in this study.

In numerical simulations and computational modeling, two commonly employed computational domains are structural and fluid domains. Structural domains involve analyzing the mechanical behavior of solid materials, whereas fluid domains focus on studying the behavior of fluids. These domains are characterized by distinct governing equations, computational methods, and types of simulations. They find applications across various fields, including civil engineering, biomechanics, aerospace, and oceanography.

Structural simulations aim to predict the response of materials under different loads, encompassing factors such as stress, strain, deformation, and failure. The computational domain is typically defined by a solid geometry, which is discretized into small elements (such as tetrahedrons or hexahedrons) for numerical analysis. The governing equations for structural simulations are typically derived from classical mechanics, such as the equations of statics, dynamics, and elasticity. Examples of simulations involving structures include analyzing the response of a bridge to wind loads, studying the deformation of a bone under mechanical stress, and investigating the behavior of a car chassis during a crash event.

When it comes to fluid simulations, the computational domain is often referred to as the area that is filled with fluid and is being affected by external factors like pressure gradients or velocities variations. Predicting the behavior of the fluid flow, which includes variables like velocity, pressure, temperature, and turbulence, is the goal of fluid simulations. The Navier-Stokes equations and other fluid mechanics concepts are frequently utilized to create the governing equations for fluid simulations. These equations indicate how a fluid moves while accounting for both internal and exterior forces at play.

2.3 Finite Element Methods

The finite element method (FEM) is a numerical method used to perform a finite element analysis (FEA) of any given physical phenomenon to predict the behavior of a structure. It bases itself on fundamental ideas including Newton’s laws of motion, the laws of thermodynamics, and the conservation of mass and energy.

The description of nature and the laws of physics for space- and time-dependent problems are usually expressed with partial differential equations (PDEs). These equations are solved in an approximate manner by the FEM which is based on equations of classical methods such as the Theory of Elasticity.

\[
\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} = 0
\]  

(2.1)

In order to solve this equation 2.1, it must be subjected to so-called initial conditions and boundary conditions. Boundary conditions (BCs) are constraints necessary for the solution of a boundary value problem.

A boundary value problem is a differential equation (or system of differential equations) to be solved in a domain on whose boundary a set of conditions is known. A characteristic feature of the finite element method is that instead of seeking the approximation over the entire region, the region is divided into smaller parts, so-called finite elements, and the approximation is then carried out.
over each element. When the type of approximation has been chosen (is to be applied over each
element), the corresponding behaviour of each element can then be determined. Once the behavior
of individual elements has been determined, they can be interconnected through matrix assembly.
This process allows for obtaining an approximate solution for the entire domain, providing insights
into the behavior of the entire body.

Computational Mesh: Structural

FEA involves simulating the response and behavior of a component or assembly under specified
conditions in order to evaluate it using the FEM. The fundamental concept behind FEA involves
performing calculations within a restricted region (surface or volume) rather than the entire domain.
By utilizing discretization or meshing (nodes and elements), the Finite Element Method effectively
decreases the infinite degrees of freedom to a finite number [20]. Based on the shapes, there are the
following types of finite elements:

- One-dimensional elements: Used for geometries having one of the dimensions that are very
  large in comparison to the other two. When a one-dimensional element is used for static
  analysis, the discretization phase of modeling becomes trivial
- Two-dimensional elements: Used when two of the dimensions are the dimensions are very large
  in comparison to the third one. The common 2D element shape used is the triangular element
  with 3 nodes, shown in figure 2.4
- Three-dimensional elements: Used when all three dimensions are three dimensions are compa-
  rable

![Figure 2.4: Types of elements in FEA [2]](image)

Finite element approximation is particularly powerful in 2D and 3D because the method can
handle a geometrically complex domain \( \Omega \) with ease. The principal idea is, as in 1D, to divide the
domain into cells and use polynomials for approximating a function over a cell. Two popular cell
shapes are triangles and quadrilaterals. The locations on the mesh where the data is mathematically
determined are known as nodal points, and they typically cluster around boundaries or areas of
significant design variations within an object.

2.4 Fluid Dynamics

A literal definition of computational fluid dynamics might be the prediction of fluid motion and
pressures (which can be used to get forces) by computation using numerical analysis. Numerical
analysis offers various methods and algorithms well-suited for CFD. These methods, such as finite
volume, finite element, and finite difference, calculate property distributions (e.g., pressure, velocity,
temperature) across typically stationary regions of space. For several decades, the finite volume
method has been widely adopted as the preferred technique in the most commonly used and versatile
CFD codes.
To get a better understanding, a basic example can be examined [4]. Consider the scenario where CFD is utilized to simulate fluid flow in a pipe. To do this, a problem description must be provided consisting of:

- The fluid’s domain, which is the internal area of the pipe.
- Equations that describe the fluid’s behavior, based on variables such as pressure $p$ and velocity $v$.
- Initial and boundary conditions for the fluid’s properties within the domain.

In CFD, this description is represented as:

- A computational mesh for the fluid.
- "Discrete" equations and algorithms that compute the pressure $p$ and velocity $v$ equations.
- Initial and boundary conditions for the pressure and velocity equations.

Figure 2.5: Computational Mesh of a pipe

Matrix equations for the mesh discretization of the pipe looks like the following:

$$
\begin{bmatrix}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{m1} & a_{m2} & \ldots & a_{mn}
\end{bmatrix}
\begin{bmatrix}
u_{x1} \\
u_{x2} \\
\vdots \\
u_{xn}
\end{bmatrix}
=
\begin{bmatrix}
b_1 \\
b_2 \\
\vdots \\
b_m
\end{bmatrix}
$$

Similarly, the same set of equations is solved for each component of $u$, namely $u_y, u_z$, and for pressure $p_n$.

Euler’s equation for fluid dynamics 2.3 is a fundamental equation that describes the motion of an inviscid (non-viscous) fluid. The equation relates the changes in velocity and pressure of a fluid at a given point in space and time.

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = p \cdot u + \nu \cdot (\nabla \cdot (\mu \cdot \nabla u))$$

While it provides valuable insights into the dynamics of ideal fluids, it does not account for phenomena such as turbulence and boundary layer effects, which require more complex models, such as the Navier-Stokes equations.

**Governing equations**

The fundamental framework of fluid examination revolves around the governing equations derived from the conservation laws of fluid’s physical properties. These equations serve as the basis for understanding and analyzing the behavior of fluid systems. The core equations that form the foundation of fluid analysis are the three laws of conservation:
• Conservation of Mass: Continuity Equation
• Conservation of Momentum: Newton’s Second Law
• Conservation of Energy: First Law of Thermodynamics or Energy Equation

The principles of mass, momentum, and energy conservation dictate that these quantities remain constant within a closed system. In other words, the total mass, total momentum, and total energy of a system must be conserved. These principles form the fundamental basis for understanding the behavior and interactions of fluids in various applications.

\[
\frac{\partial (\rho u_i)}{\partial t} + \frac{\partial (\rho u_i u_j)}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j} (\mu \frac{\partial u_i}{\partial x_j}) + f_i[21] \tag{2.4}
\]

The equation 2.4 represents the conservation of momentum for a fluid, where \( \rho \) is the fluid density, \( u_i \) is the velocity component in the \( i \)-th direction, \( p \) is the pressure, \( \mu \) is the dynamic viscosity, and \( f_i \) is the volume force in the \( i \)-th direction.

Turbulence models

Many fluid flows have erratically varying flow rates. Often these fluctuations are at such small scales and high frequencies that resolving them in time and space comes at excessive computational costs. Instead of solving for the exact governing equations of turbulent flows (Direct Numerical Simulation), it is less expensive to solve for averaged or filtered quantities and approximate the impact of the small fluctuating structures. Turbulence models provide different approaches for modeling these structures.

Modeling fluid flow turbulence is important for the accurate simulation of fluid flow. Beginning with the fluid flow across a flat plate, as seen in the image 2.6 below, is a good place to start [3]. The flat plate’s leading edge is struck by the uniform velocity profile, which causes a laminar boundary layer to form. This area has extremely predictable flow. After a certain distance, the boundary layer starts to exhibit minor chaotic oscillations, which causes the flow to gradually get more turbulent until it is entirely turbulent.

Among the types mentioned below, the Fixed Turbulent Viscosity and Mixing Length models are zero-equation models that can provide fast, less computationally intensive solutions. The K-Epsilon, K-Omega, and SST (Shear Stress Transport) models are types of two-equation models commonly used in computational fluid dynamics. These models are preferred over simpler zero-equation models because they provide more accurate predictions of fluid flow behavior. As they involve solving two additional partial differential equations, this adds a layer of complexity and computational difficulty but is necessary to capture more detailed and realistic flow characteristics. [22].

• Laminar Flow: In the laminar regime, the fluid flow can be completely predicted by solving Navier-Stokes equations, which give the velocity and the pressure fields. This model may be appropriate for detailed analysis of flow between heat sink fins or for more viscous fluids. Laminar flow has a lower drag, but separates much earlier which could lead to a much higher flow resistance.

• Fixed Turbulent Viscosity: The model is robust and fast. In addition to its role in identifying overall patterns and addressing model issues, the Fixed Turbulent Viscosity model can be valuable for troubleshooting specific areas of concern. However, a notable drawback of this model is its high sensitivity to the specified turbulence scales, which can impact the accuracy and reliability of the results.

• Mixing Length: Sometimes called the Algebraic model, it provides good results in many applications. While it may show lower accuracy compared to two-equation models in certain cases, the Mixing Length model is more robust and less computationally intensive than the two-equation models. Accurate predictions can be obtained using the Mixing Length model for
thin shear layer flows, including jets, mixing layers, wakes, and boundary layers. The Mixing Length model may fail in flows with separation and recirculation, such as back eddies behind a circular obstruction. In these cases, the two-equation models are better choices.

• The K-Epsilon model is a widely recognized and extensively validated turbulence model that has demonstrated good performance in various applications. However, it may exhibit limitations when applied to certain types of flows, such as unconfined flows, flows involving rotation, and flows with significant adverse pressure gradients or very low $y^+$ values ($<1$). For these flows, consider using the SST — Shear Stress Transport or K-Omega models. The k-epsilon model is ideal for predicting flow behavior in regions away from the wall. Although the K-Epsilon model, like many two-equation models, provides a more accurate description of the effects of turbulence on the mean flow, it also adds significant computational time to the solution.

• The K-Omega model is a widely employed approach for capturing turbulent flow characteristics. It is among the commonly utilized models in the Reynolds-averaged Navier-Stokes (RANS) family, which aims to model all aspects of turbulence within the flow. By incorporating this model, the effects of turbulence can be adequately accounted for, enhancing the accuracy of predictions in various fluid flow simulations. In the k-omega model, the transport equation for the turbulent dissipation rate, $\epsilon$ (epsilon), is replaced with an equation for the specific dissipation rate, $\omega$ (omega). This model proves valuable in simulating and analyzing fluid flows within the viscous sub-layer. Furthermore, the k-omega model is good at resolving internal flows, separating flows and jets and flows with high-pressure gradients, and also internal flows through curved geometries.

• Shear Stress Transport (SST) [23] [24]: The SST (Shear Stress Transport) turbulence model combines elements of the K-Epsilon and K-Omega models. It behaves like the K-Epsilon model in regions away from walls and behaves like a K-Omega model near walls. This model demonstrates improved performance in unconfined flows and flows with strong adverse pressure gradients. Similar to the K-Omega model, the SST model solves two transport equations for turbulent kinetic energy (k) and specific dissipation rate ($\omega$).

The widely used k-omega (k-omega) turbulence model is part of the Reynolds-averaged Navier-Stokes (RANS) family of turbulence models, which account for all turbulence effects. The SST k-omega turbulence model is a popular two-equation eddy-viscosity model that solves conservation equations along with two transport equations for turbulent kinetic energy (k) and specific turbulent dissipation rate ($\omega$). These transport equations consider the historical effects, including the convection and diffusion of turbulent energy.

It can be used throughout the boundary layer, including the viscous sub-layer, without the need for extra damping functions. In the free-stream region, it behaves like a k-epsilon model, eliminating the sensitivity to inlet free-stream turbulence often observed with the k-omega model. The SST model is particularly effective in adverse pressure gradients and separating flow situations, although it may slightly overpredict turbulence levels in regions with significant normal strain. While the SST k-omega model does not include compressibility options and may not be suitable for free shear flows, it exhibits less sensitivity to free stream conditions compared to many other turbulence models. Also, it shows elevated turbulence levels in regions with strong normal strain, such as stagnation regions and areas with intense acceleration.

• Large Eddy Simulation (LES): When simulating turbulent fluid flow, a turbulence model can be employed to resolve large eddies. This particular turbulence model is highly recommended when the objective is to visualize the actual eddies within the flow and when there is an interest in studying the instantaneous flow fields and their associated statistics. Large Eddy Simulation (LES) directly solves the filtered Navier-Stokes equations without relying on Reynolds stress terms. Unlike other turbulence models, which utilize Reynolds stress terms in the Reynolds-Averaged Navier-Stokes (RANS) equations, LES focuses on resolving turbulent structures through an algebraic approach.

It is well acknowledged that all current turbulence models are only approximations of the turbulence’s physical phenomena. The kind of flow to which a specific model is applied determines
the degree of approximation. Characterizing the conditions that result in "good" and "bad" performance must mostly be based on experience. To select an appropriate model, it is crucial to have a comprehensive understanding of its strengths, limitations, and specifications.

**Computational Mesh: Fluid**

The initial step in a CFD simulation involves the definition of a solution domain, which represents the specific geometric shape in space where the fluid dynamics equations will be solved. To illustrate this, let’s consider the example of fluid flow through a pipe. In this case, the solution domain corresponds to the region occupied by the fluid inside the pipe. To discretize the solution domain, the mesh generation process is employed, dividing the domain into smaller volumes or cells. Specialized computer programs are utilized to create these cells according to the user’s specifications and specific requirements. In modern CFD software that employs the finite volume method, meshes can consist of cells with irregular polyhedral shapes. A cell may possess any number of faces ($\geq 4$), and each face can have any number of edges ($\geq 3$). These cells are contiguous, meaning that the faces of a particular cell are shared with its neighboring cells unless they constitute the boundary of the solution domain. Moreover, there are no constraints on the alignment of cells with coordinate axes.

In figure 2.7, there are two cells connected by a shared face, depicted in grey. Each face is defined by a sequence of vertices connected by edges. To establish connectivity between cells, a unique index is assigned to each cell, and the indices of the owner and neighboring cell for each face are stored.

Additionally, the faces that correspond to the boundary of the domain are organized into distinct groups, each assigned a unique name. These named groups referred to as *patches*, allow for the identification of specific regions along the domain boundary. This enables the application of specific boundary conditions to those regions during the execution of a CFD simulation.
Chapter 2. Methodology

2.5. Fluid-Structure Interaction

The process of equation discretization transforms partial differential equations governing continuous fields, such as the pressure equation $p$, into sets of linear equations for discrete fields[4].

$$
\begin{bmatrix}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{m1} & a_{m2} & \ldots & a_{mn}
\end{bmatrix}
\begin{bmatrix}
u_{x1} \\
u_{x2} \\
\vdots \\
u_{xn}
\end{bmatrix}
= 
\begin{bmatrix}
b_1 \\
b_2 \\
\vdots \\
b_m
\end{bmatrix}
$$

Equation 2.5 represents the equation discretization process where the right-hand side matrix of $b_x$ could be $u_y, u_z, p$. The matrix equations are solved in an iterative manner, where the equation for one variable, incorporates the current values of other variables into the source vector. This iterative sequence allows for the progressive refinement of the solution by considering the interdependencies between the different variables. The flow between cells is evaluated and the pressure is afterward calculated from the mass flow into the cell. This is timewise interpolated using a Runge-Kutta-scheme.

The needed time step is dependent on the ratio of cell size and the flow velocity. High aspect ratios in structured cells (hexahedral+) could be beneficial in this point of view.

2.5 Fluid-Structure Interaction

The theoretical framework presented in this chapter forms the foundation for the development of FSI simulation models within Ansys-Workbench. The components involved, namely Transient Structural, Fluent, and System Coupling, are essential for implementing FSI simulations. Having gained a fundamental understanding of the FEM and CFD, the theoretical background of FSI will now be explored, providing a deeper insight into the principles and concepts underlying this interdisciplinary field.

FSI refers to the dynamic interaction between a fluid flow and a solid structure. Examples of FSI can be observed when a wind gust rotates a turbine blade, when a boat hull encounters wavy conditions, or when air rushes over the front panel of an F1 car. Whenever a fluid and a structure come into contact, FSI takes place.

There are two types of FSI:

- one-way: In one-way FSI, the fluid flow affects the solid structure, leading to stresses and strains that may cause deformations. These deformations can vary in magnitude depending on factors such as flow pressure, velocity, and material properties of the structure.

- two-way: However, in cases where the structural deformations are significant, the velocity and pressure fields of the fluid also undergo changes. This necessitates a bi-directionally coupled multiphysics analysis, where the fluid flow and pressure fields influence the structural deformations, and vice versa.

This interaction between fluid flow and structure can be simulated within the Ansys workbench using System coupling. System coupling allows the use of data transfers to model this interaction. In one-way coupling, transfer quantities are sent from one solver to the other, but not in the opposite direction. This coupling is used when one domain has an important effect on the other domain, but not the opposite. For example, the Fluid flow in a pipeline can significantly affect the stresses in the pipeline but the deformation in the pipeline does not significantly affect the fluid flow. In this case, the one-way coupling method is usually a good approximation because there are no large deformations in the pipeline.

The developed FSI algorithm is based on a partitioned strategy that employs separate solvers for fluid and structural sub-problems and uses a coupling system to take the interaction of the domains into consideration. A tightly coupled approach is used to properly meet the coupling requirement at the fluid-structure interface and to prevent numerical instabilities. By switching back and forth between the structural and fluid solvers until the convergence criterion is satisfied, it enforces the
Chapter 2. Methodology

2.6 Mesh

A crucial component in the simulation process is the mesh. It is composed of interconnected cells and nodes, forming a network. The mesh can take various shapes and sizes, and its purpose is to facilitate the solution of Partial Differential Equations (PDEs). Each cell within the mesh represents a discrete solution of the equation, and when combined with the entire network, it yields a solution for the entire mesh. By subdividing the problem domain into smaller elements, the mesh enables efficient and accurate numerical analysis.

Meshing is a fundamental process in simulation that involves generating a grid over a given geometry, whether it is in two or three dimensions. The purpose of meshing is to discretize the geometry into smaller elements, such as cells or elements, in order to facilitate numerical analysis. The grid is created based on the complexity of the geometry and the specific requirements of the simulation. In simpler geometries, a coarse mesh with larger elements may be sufficient, while more complex geometries may require a finer mesh with smaller elements to capture intricate details. The meshing process plays a crucial role in the accuracy and efficiency of the simulation results, as it determines the resolution and representation of the geometry within the computational domain.

Figure 2.8: Two different approaches for FSI

precise coupling condition. The study of fluid-structure interaction in a reed-type valve involves considering changes in the geometric domains resulting from the deformation of the structure. This is done by directly moving the solid mesh and updating the fluid grid based on the new position of the solid interface.

By dividing FSI problems into three sub-problems, namely the i) fluid problem, ii) structure problem, and iii) interfacial condition, accurate simulations can be achieved [25]. The fluid and structure domains are solved separately and sequentially using individual grids, with explicit interfacial conditions facilitating the communication of information between the two solutions. This approach provides a better understanding of the interaction between fluids and structures in various fields such as aerospace, biomedical, and civil engineering.

With a fundamental understanding of the FEM, CFD, and FSI, the simulation results obtained thus far can be used. This data will be used to generate a neural network, which in turn will contribute to reducing computational costs.

By harnessing the knowledge acquired during the previous stages, the simulation results can be used as a valuable resource for training a neural network. This neural network will possess the capability to approximate complex relationships within the data, offering an efficient means of computation. This approach enables leveraging the available data and computational resources to streamline the analysis process and enhance overall efficiency.
Mesh discretization

The first step in numerically solving a set of partial differential equations (PDEs) involves discretizing both the equations and the problem domain. As previously mentioned, attempting to solve the entire problem domain as a whole is infeasible, but breaking it down into smaller components is a valid approach. An optimal grid is as fine as necessary in regions with high gradients and coarse in the rest.

The discretization of the equations is achieved through methods such as:

- Finite Difference Method
- Finite Volume Method: Finite Volume Method is a widely used numerical technique that transforms the set of partial differential equations into a system of linear algebraic equations.
- Finite Element Method: The FEM is a systematic numerical method for solving problems, more specifically PDEs. Also, the analytical nature of the solutions of these issues typically requires the solution to boundary value problems for PDEs.

These methods convert the continuous form of the equations into a system of algebraic difference equations that can be solved numerically. This process involves approximating derivatives and integrating over discrete volumes or elements.

On the other hand, the discretization of the problem domain entails dividing it into a set of discrete cells or elements. These cells, along with their associated points or nodes, cover the continuous problem domain. The choice of how to discretize the domain depends on factors such as the geometry, the complexity of the problem, and the desired accuracy of the solution.

By discretizing both the equations and the domain, the PDEs can be transformed into a solvable system of algebraic equations that can be solved using numerical methods. This allows for the analysis and simulation of complex systems that would otherwise pose significant challenges or be impractical to solve analytically.

Different mesh types

The choice of mesh elements plays a crucial role in accurately representing the geometry, capturing the physics of the problem, and obtaining reliable simulation results. There are several types of mesh elements commonly used in numerical simulations, each with its own characteristics and advantages.

In a 2D mesh, all mesh nodes are located within a specified plane. Typically, these nodes are positioned in the XY plane, although they can also be restricted to other Cartesian planes or user-defined planes as required. The most popular 2D mesh elements are quadrilaterals (also known as quads) and triangles (tris), shown below in figure 2.9.

![2D mesh elements](image)

Figure 2.9: 2D mesh elements

3D mesh nodes are not constrained to lie in a single plane. Most popular 3D mesh elements are hexahedra (also known as hexes or hex elements), tetrahedra (tets), square pyramids (pyramids),
and extruded triangles (wedges or triangular prisms), shown below in figure 2.10. It is important to mention that all of these elements are surrounded by faces that belong to the aforementioned 2D elements. Additionally, some of the current solvers also have the capability to handle polyhedral elements, which can be bounded by various types and numbers of faces. The creation of a (hexa- or polyhedral) grid in complex geometries is difficult and may not lead to a usable grid. Tetrahedral grids are much easier to handle - in this respect.

Mesh quality metrics

To achieve accurate and reliable simulations, the generation of a high-quality mesh holds paramount importance. It is essential to recognize that there exist notable differences in meshing approaches when it comes to two widely used simulation domains: CFD and FEA.

Below are definitions of some commonly used metrics[1]:

- Aspect Ratio: Aspect Ratio is the ratio of the longest side of a finite element to the shortest side. A high aspect ratio indicates that the element is elongated and can lead to numerical instability and inaccurate simulation results, especially by using tetrahedral cells.

![Figure 2.11: Comparison of high and low aspect ratio per element type][1]
• Skewness: Skewness measures the deviation of the angles between adjacent sides of an element from 90 degrees. A high skewness value can cause numerical errors in the solution.

For CFD, the mesh quality metrics slightly differ.

• Cell Volume: This metric measures the volume of a mesh element. Large variations in cell volumes can lead to inaccurate results. A good value is a growth factor lower than 10%.

• Cell Skewness: This metric measures the deviation of the angles between adjacent faces of a mesh element from 90 degrees. High cell skewness values can cause numerical errors in the solution.

These metrics can be used to evaluate and optimize the quality of a mesh used in CFD simulations to ensure accurate and reliable results.

Mesh quality measures are used to assess various meshing methods and algorithms, thus allowing the choice of the best meshing approach for a given problem. Furthermore, they are an important tool for the validation and comparison of CFD results, leading to more confident and reliable simulations [1]. The evaluation of tetrahedral meshes in three dimensions often involves the use of volumetric skewness and volumetric collapse as common metrics[26]. The quality metrics mentioned above are commonly used in automatic mesh generation methods to assist with the generation process.
However, these metrics have certain limitations and can only be used to evaluate mesh quality in a limited manner. Firstly, most metrics are applied at the element level and provide information about local features rather than regional or global features. Metrics such as element shape and aspect ratio can provide information about the quality of individual elements, but they may not be able to provide information about the mesh as a whole. Additionally, the thresholds used for these metrics are often based on subjective experiences. Secondly, some metrics may not be relevant for accurately computing results. It is worth noting that the limitations of quality metrics do not necessarily mean that they are not useful for mesh generation. However, it is important to understand their limitations and use them in conjunction with other evaluation methods to ensure the generation of high-quality meshes that will produce accurate simulation results.

Mesh quality assessment is a crucial step in achieving accurate and reliable results in CFD simulations. This assessment can serve two main purposes:

- Prior to running the simulation: To ensure the suitability of the mesh for the intended analysis, a mesh quality assessment should be performed before starting the simulation. It is still essential to assess the mesh quality to avoid potential issues during the simulation.
- After encountering issues in the simulation: Mesh quality assessment can also be used to identify problematic areas in the mesh that could be causing inaccurate or divergent simulation results. Adjusting the CAD model or refining certain regions of the mesh can often resolve these issues. Moreover, mesh quality assessment can guide users in areas where the mesh may need to be denser or more refined to capture critical fluid dynamics phenomena. Thus, mesh quality assessment is a vital tool for achieving reliable and accurate results in CFD and CAE simulations.

Assessing the geometric and topological characteristics of the mesh allows for an evaluation of its quality, including factors such as element form, size, orientation, and connection. A high-quality mesh should contain well-shaped elements, an even distribution of element sizes, and strong connectivity between adjacent pieces[27]. Finally, smoothness measures the variation in element size or shape across the mesh, with high smoothness indicating a gradual transition that can improve the accuracy and efficiency of the simulation.

A quality mesh in computational fluid dynamics should have:

- minimal cell skewness
- consistent cell volumes,
- suitable aspect ratios, and
- sufficient resolution in critical regions such as areas with sharp gradients or boundary layers.

This ensures accurate and reliable simulations.

2.7 Deep learning models

The integration of AI into CFD applications has revolutionized various industries. This integration is considered a strategic asset for companies, as it offers numerous benefits such as reducing costs and creating new differentiated values. Smart AI-driven solutions provide significant advantages to CFD engineers, designers, and analysts. One of the primary benefits is the reduction in computational, design program, and operational costs by generating more designs per simulation at a faster turnaround time. Additionally, AI-based surrogate models and smart workflows expedite process and program development turnaround time. Another significant benefit of AI in CFD is the enhancement of simulation accuracy by flagging anomalies and providing knowledge-based workflow assistance in the CFD process.
Furthermore, AI can provide knowledge-based workflow assistance in CFD. The integration of AI in CFD applications enables companies to achieve more efficient and accurate simulations, thereby achieving better results and driving the overall success of their operations. Results from several FSI simulations are utilized to predict the overall behavior of a system using artificial intelligence (AI), eliminating the need for tables or a physical model. The conventional FSI approach, known for its time-consuming and complex nature, is avoided when integrating the system simulation within the industry. By leveraging AI, a more efficient and streamlined solution is achieved to analyze system behavior.

This section will provide an in-depth exploration of neurons, neural networks, and various types of layers including fully connected and LSTM. Additionally, different types of networks such as multilayer perceptron and recurrent neural networks will be discussed.

Furthermore, the concept of backpropagation, a fundamental algorithm for training neural networks, will be examined. Additionally, various loss functions and error metrics commonly employed to assess neural network performance will be covered. Through this exploration, a comprehensive understanding and knowledge will be gained, laying the groundwork for the effective design and training of neural networks. This knowledge will serve as a solid foundation, facilitating improved outcomes and enhanced understanding in the implementation of neural networks.

**Neurons**

Neural networks have become a popular machine learning technique that takes inspiration from the complex structure and function of biological neurons in the human brain. These networks are composed of interconnected layers of artificial neurons that work collaboratively to process and analyze complex data. Each neuron receives input from the previous layer and performs a calculation based on a set of weights and biases. The resulting output is then passed on to the next layer, ultimately producing the final output. With various types of neural networks, such as feedforward, convolutional, and recurrent, each network can be designed to handle different types of data [28].

A biological neural network is made up of several nerve cells called neurons, such as the ones found in human brains. Frank Rosenblatt proposed the perceptron model in 1957 [29] as a simplified approximation of a biological neuron.

The application of deep learning involves the use of a neural network architecture to learn a mapping function from input data to a specific output or target. The model’s parameters are optimized iteratively through an interactive process that involves the use of the gradient descent algorithm. This process allows the model to learn and refine its internal representation of the data and to improve its ability to accurately predict the target output. By adjusting the weights and biases of the network’s nodes, the deep learning method can gradually improve the accuracy of its predictions, making it a powerful tool for solving complex problems in various fields.

Neural networks have found extensive applications in various domains, such as image and speech recognition, natural language processing, and autonomous vehicles. They have also demonstrated
potential in the fields of medicine and finance, where they can be utilized to diagnose diseases and predict stock market trends. Neural networks are also applied in the field of reinforcement learning, which involves training an agent to make decisions by actively engaging with its environment and receiving positive or negative feedback in response to its actions. In this approach, the neural network serves as the decision-making component of the agent, and its weights and biases are adjusted through trial and error to maximize the expected reward. One of the benefits of neural networks is their capacity to acquire knowledge from extensive datasets, making them well-suited for tasks where traditional rule-based systems would be too complex or too difficult to specify.

However, this reliance on data also means that neural networks can suffer from over-fitting, where the model becomes too specialized to the training data and performs poorly on new, unseen data. Regularization techniques, such as dropout and weight decay, can help mitigate this issue. Another challenge with neural networks is their opacity or lack of interpretability. Because the weights and biases of the nodes are adjusted automatically during training, it can be difficult to understand how the network arrived at its predictions. Researchers are actively working on developing methods to make neural networks more transparent, such as visualizing the activations of individual nodes or using attention mechanisms to highlight important features in the input data.

Overall, neural networks have revolutionized the field of machine learning and have enabled significant advancements in artificial intelligence. As research in this area continues, it can be expected to see even more sophisticated and powerful neural network architectures being developed, as well as new techniques for training and interpreting these models.

This chapter aims to outline the process of establishing an AI model. The steps involved include data generation, data pre-processing, and the creation of a neural network architecture. The data generation process will be explained in detail, highlighting how the data is generated for training the model. Subsequently, the pre-processing techniques employed to prepare the data for input into the neural network will be discussed.

Furthermore, various neural network architectures will be explored and implemented to determine the most suitable one for the given task. The performance and accuracy of each architecture will be compared and evaluated. The evaluation of accuracy will be carried out using different metrics.

**Multi-Layer Perceptron**

One of the oldest computational representations of neural networks (NNs) is the perceptron, which serves as the foundation for today’s more advanced and deep networks. The multilayer perceptron (MLP) is a more complex and layered version of the perceptron. While the perceptron consists of a single layer with a single neuron that takes inputs and produces an output, the MLP has multiple layers of neurons, including at least one hidden layer. The perceptron and MLP differ mainly in their ability to classify data. While the perceptron can only classify data that is linearly separable, the MLP is capable of classifying data that is non-linearly separable. This is primarily due to the addition of a hidden layer in the MLP, which enables non-linear transformations of the input data.

A multi-layer perceptron (MLP) is a neural network that comprises at least three layers: an input layer, a hidden layer, and an output layer. Every layer interacts with the results of the one before it:

The perceptron [5] receives m binary inputs, marked by the letters $x_1, \ldots, x_m$, which stand for the signals originating from its nearby neurons. It then produces a single binary value, designated by the letter $o$, which shows whether the perceptron is "firing" or not. In a perceptron model, the strength of the connection between each input neuron $x_i$ and the perceptron is represented by a weight $w_i$. The influence of each input on the perceptron’s output is proportional to its corresponding weight. The objective is to identify the weights $w_1, \ldots, w_m$ and threshold $\theta$ that enable the perceptron to accurately map its inputs $x_1, \ldots, x_m$ (representing the features in the data) to the desired output $y$

$$z_l = \text{activation}(w_l z_{l-1} + b_l) \quad (2.6)$$

where:
Chapter 2. Methodology

2.7. Deep learning models

Figure 2.14: Perceptron Model [5]

- $z_l$ is the output of the $l$-th layer of the MLP, which consists of a set of neurons
- $W_l$ is the weight matrix that defines the strength of the connections between the neurons in layer $l$ and the neurons in layer $l-1$
- $z_{l-1}$ is the input to layer $l$, which is the output of layer $l-1$
- $b_l$ is the bias vector that is added to the weighted sum before applying the activation function
- activation is the activation function that is applied element-wise to the weighted sum of the inputs and biases.

To calculate the weighted sum of the inputs, the perceptron multiplies each input by its corresponding weight and then sums up the products. This sum is also referred to as the "net input" and is typically represented by the variable $z$. Inputs with higher weights have a greater impact on the net input and therefore have a stronger influence on the perceptron’s output.

1. Just as with the perceptron, the inputs are pushed forward through the MLP by taking the dot product of the input with the weights that exist between the input layer and the hidden layer ($W_H$). This dot product yields a value at the hidden layer which is not pushed forward as it would happen with a perceptron though.

2. MLPs utilize activation functions at each of their calculated layers. There are many activation functions to discuss: rectified linear units (ReLU), sigmoid function, and tanh. Push the calculated output at the current layer through any of these activation functions.

3. Once the calculated output at the hidden layer has been pushed through the activation function, push it to the next layer in the MLP by taking the dot product with the corresponding weights.

4. Repeat steps two and three until the output layer is reached.

5. At the output layer, the calculations will either be used for a backpropagation algorithm that corresponds to the activation function that was selected for the MLP (in the case of training) or a decision will be made based on the output (in the case of testing).

In summary, the perceptron computation involves two steps:

The input values $x_1, \ldots, x_m$ are multiplied by their corresponding weights $w_1, \ldots, w_m$, and the bias $b$ is added, resulting in the net input $z = \sum_{i=1}^{m} w_i x_i + b$.

An activation function $f(z)$ is applied to the net input, generating a binary output of either 0/1 or -1/+1.
Backpropagation

The backpropagation algorithm is a widely employed method for training neural networks. This process adjusts the weights and biases of the nodes to minimize the difference between the predicted output and the actual output and is repeated many times until the network can accurately predict new data.

It works by propagating the error gradients backward through the network, starting from the output layer and moving toward the input layer. This allows the network to understand how much each parameter contributes to the overall error.

By repeatedly applying the backpropagation algorithm during training, the network learns to adjust its internal parameters, such as the weights and biases, to minimize the prediction error. This iterative process helps the network gradually refine its predictions and improve its performance.

Backpropagation has had a profound impact on the field of deep learning, enabling the training of complex models with multiple layers. It has made it possible for neural networks to learn from vast amounts of data and make accurate predictions in various domains, ranging from image recognition to natural language processing.

Gradient descent methods

The iterative optimization process known as Gradient Descent is widely utilized in machine learning projects to find the optimal value (minimum/maximum) of an objective function. Its primary objective is to identify model parameters that yield maximum accuracy on both training and test datasets. By following the direction of the gradient, which indicates the steepest rise of the function at a given point, the algorithm gradually moves in the opposite direction to reach the function’s minimum. Batch Gradient Descent (BGD), Mini-Batch Gradient Descent (MBGD), and Stochastic Gradient Descent (SGD) are such different optimization algorithms. BGD calculates the gradient of the cost function using the entire training dataset, updating the model’s parameters once per epoch. MBGD divides the dataset into smaller batches and calculates the gradient using a subset of examples, resulting in more frequent parameter updates. To address the computational inefficiency of traditional Gradient Descent methods when handling large datasets, Stochastic Gradient Descent (SGD) is employed. Instead of using the entire dataset for each iteration, SGD randomly selects a single training example or a small batch to calculate the gradient and update the model parameters. This random selection introduces randomness into the optimization process, leading to the term "stochastic" in stochastic Gradient Descent. The advantage of SGD lies in its computational efficiency, particularly when dealing with large datasets, as it significantly reduces the computational cost.

Figure 2.15: Understanding backpropagation [6]
per iteration compared to traditional Gradient Descent methods that require processing the entire dataset. BGD provides accurate updates but can be computationally expensive, while SGD offers faster updates at the expense of increased noise. MBGD strikes a balance between the two, providing reasonable accuracy with improved computational efficiency. The choice of algorithm depends on the specific requirements of the problem and the trade-off between accuracy and computational resources available.

![Batch Gradient Descent, Mini-Batch Gradient Descent, Stochastic Gradient Descent](image)

Figure 2.16: Difference between Batch gradient, mini-batch gradient and stochastic gradient descent method [7]

**Learning Rate**

The "learning rate" is a crucial hyperparameter in tuning a neural network for optimal performance on a specific problem, as it determines the magnitude of change to the model in each step of the search process. Deep learning neural networks are typically trained using the prior mentioned optimization algorithm called stochastic gradient descent. This algorithm estimates the error gradient for the current state of the model using examples from the training dataset and then updates the weights of the model using the backpropagation of errors algorithm. This process is commonly referred to as simply backpropagation, which is discussed later.

During training, the weights of the model are updated by a certain amount referred to as the step size or the “learning rate.” Tuning the learning rate is an essential hyperparameter in deep learning that can greatly impact the performance of the model on a given problem.

During the training process, the backpropagation algorithm calculates the error gradient for the current state of the model, and the weights are updated accordingly. However, instead of updating the weights with the full error gradient, the update is scaled by the learning rate, which determines the step size for each iteration.

For example, if the learning rate is set to 0.1 (a common default value), the weights will be updated by a factor of 0.1 times the estimated weight error during each iteration. This means that the learning rate controls the speed at which the model learns and how quickly it converges to a solution. When the learning rate is appropriately configured, a deep learning model can best approximate the function given the available resources, which include the number of layers and nodes per layer, in a specific number of training epochs. However, setting the learning rate to extreme values can lead to performance issues. A learning rate that is too large can cause weight updates that are too large, leading to oscillating performance, while a rate that is too small may not converge or result in a suboptimal solution. A higher learning rate may result in faster convergence, but may also lead to instability, while a lower learning rate may result in slower convergence.
Fully Connected Layers

A fully connected neural network is composed of multiple layers where each neuron is connected to every neuron in the adjacent layer. This design allows for a versatile and flexible network architecture without requiring any specific assumptions about the input data.

Each neuron in a fully connected layer receives inputs from all the neurons in the previous layer and computes a weighted sum of these inputs, typically followed by a non-linear activation function. The weights associated with each connection represent the strength or importance of the connection. These weights are learned during the training process, where the network adjusts them to minimize the error or loss function.

The fully connected architecture allows for complex non-linear relationships to be learned and modeled by the network. It is particularly effective in tasks such as pattern recognition, classification, and regression. The dense connectivity between layers enables the network to capture intricate dependencies and extract high-level features from the input data. However, as the number of neurons and layers increases, the number of parameters and computations also grows, leading to increased computational complexity and potential overfitting. Regularization techniques, such as dropout or weight decay, are often employed to address these challenges.

Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a type of deep learning algorithm that is designed to process sequential data. Unlike other neural networks, RNNs can maintain an internal memory state that allows them to process not only the current input but also the previous inputs in the sequence. This makes them well-suited for tasks such as language modeling, speech recognition, and time series analysis. One of the key features of an RNN is the use of feedback loops that allow information to be passed from one step in the sequence to the next. This feedback mechanism is what allows the network to maintain its memory state and process sequential data. The output of each step in the sequence is determined by the current input as well as the internal memory state, which is updated with each new input. The basic unit of an RNN is a single neuron with a hidden state that persists over time. The hidden state is updated at each time step and is used to inform the neuron’s output for that time step. The output of the RNN is typically generated by a fully connected layer that takes as input the hidden state at the final time step. One of the main challenges of RNNs is the vanishing gradient problem, which occurs when the gradients that flow back through the
network during training become very small, making it difficult for the network to learn long-term dependencies. To address this issue, variants of RNNs such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) were developed. LSTMs and GRUs are designed to selectively forget or remember information based on the current input and the past hidden state, making them more effective at handling long-term dependencies. As a result, they have become the standard for many RNN-based applications. RNNs have been successfully applied to a wide range of applications, including natural language processing, speech recognition, and image captioning.

![Architecture of Recurrent Neural Network](image)

Figure 2.18: Architecture of Recurrent Neural Network[8]

The network diagram on the right illustrates an unrolled version of the network diagram on the left. The weight parameters, namely \( W_x h, W, \) and \( W_{h y} \), correspond to the connections between the input and hidden layer, between hidden layers, and between the hidden layer and output layer, respectively. The variable ‘a’ represents the activation of each layer. The recurrent neural network processes the data sequentially from left to right, utilizing shared weight parameters for each time step.

To decide on the number of hidden layers in a neural network, it is good practice to start with a simple architecture and gradually increase its complexity based on the complexity of the task at hand. It can also be helpful to study existing models and consider the amount of data available. Ultimately, experimentation is needed to determine the optimal number of hidden layers, and cross-validation techniques can be used to avoid over-fitting and select the best model architecture.

**Long Short-Term Memory** - Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) that is specifically designed to deal with the vanishing gradient problem of traditional RNNs. As mentioned earlier, traditional RNNs suffer from the vanishing gradient problem, which occurs when the gradients of the error function with respect to the weights of the network become extremely small, making it difficult to train the network effectively. LSTMs are able to overcome this problem by using a memory cell, which is able to remember information over long periods of time. The memory cell is made up of three main components: an input gate, an output gate, and a forget gate. The input gate controls which information is stored in the memory cell, the forget gate controls which information is removed from the memory cell, and the output gate controls which information is output from the memory cell. The use of these gates allows the LSTM to selectively remember or forget information over time, making it well-suited for tasks that require the network to maintain long-term dependencies. LSTMs have been successfully used in a variety of applications, including speech recognition, language translation, and image captioning, among others.

**Loss function**

When working with neural networks, the primary objective typically revolves around minimizing the error. This is commonly achieved by using an objective function, which is also known as a cost function.
function or a loss function. The outcome computed by the loss function is commonly referred to as the "loss" itself. The role of the cost or loss function is crucial as it is responsible for condensing all the elements of a model into a single numerical value. This value serves as an indicator of the model’s performance, where improvements in the cost function signify enhancements in the model. By reducing the complex characteristics of a system into a scalar value, the cost function enables the ranking and comparison of different candidate solutions.

The methodology of maximum likelihood estimation (MLE) enables the determination of optimal parameter estimates based on historical training data. It provides a robust method for inference, allowing for the identification of the most suitable statistical estimates. In the context of maximum likelihood, a loss function is employed to assess the degree of similarity between the distribution of predictions generated by the model and the distribution of target variables observed in the training data. One advantage of employing maximum likelihood as a framework for estimating model parameters, both in neural networks and machine learning as a whole, is that the accuracy of parameter estimation tends to improve as the size of the training dataset increases. By having a larger number of examples available for learning, maximum likelihood estimation becomes more reliable in capturing the underlying patterns and relationships in the data, leading to enhanced parameter estimates.

In machine learning and neural networks, different types of loss functions are used depending on the nature of the problem being solved. Here are some common types of loss functions:

- **Mean Squared Error (MSE):** This loss function is commonly used in regression problems. It calculates the average squared difference between the predicted and actual values. The goal is to minimize the mean squared error to achieve accurate predictions.

- **Binary Cross-Entropy:** This loss function is used for binary classification problems. It measures the dissimilarity between the predicted probabilities and the true binary labels. It penalizes significant deviations from the correct class probability.

- **Categorical Cross-Entropy:** This loss function is employed for multi-class classification problems. It quantifies the difference between the predicted class probabilities and the true class labels. It aims to minimize the cross-entropy to ensure accurate classification.

- **Sparse Categorical Cross-Entropy:** Similar to categorical cross-entropy, this loss function is used for multi-class classification. However, it is suitable when the true class labels are represented as integers rather than one-hot encoded vectors.

- **Mean Absolute Error (MAE):** Also known as L1 loss, MAE calculates the average absolute difference between the predicted and actual values. It is often used in regression tasks and is less sensitive to outliers compared to MSE.

- **Huber Loss:** This loss function is a combination of MSE and MAE. It behaves like MAE for small errors and like MSE for large errors. It is robust to outliers and balances between robustness and sensitivity.

These are just a few examples of the many loss functions available. The choice of the loss function depends on the specific problem at hand and the desired behavior of the model during training and inference.

**Accuracy metric**

Accuracy metrics are essential tools for evaluating the performance of regression neural networks. These metrics provide quantitative measures to assess how well the network’s predictions align with the actual values of the target variable. By comparing the predicted values with the actual values, accuracy metrics provide valuable insights into the magnitude and direction of errors present in the network’s predictions. There are several commonly used accuracy metrics for evaluating the performance of regression neural networks:
1. Mean Squared Error (MSE): MSE measures the average squared difference between the predicted and actual values. It provides a measure of the overall magnitude of the errors.

2. Root Mean Squared Error (RMSE): RMSE is the square root of MSE and represents the average magnitude of the errors. It is commonly used as it is in the same units as the target variable, making it more interpretable.

3. Mean Absolute Error (MAE): MAE calculates the average absolute difference between the predicted and actual values. It provides a measure of the average magnitude of the errors without considering their direction.

4. R-squared (R2): R-squared represents the proportion of the variance in the target variable that is explained by the regression model. It ranges from 0 to 1, with higher values indicating better model fit.

5. Mean Percentage Error (MPE): MPE measures the average percentage difference between the predicted and actual values. It provides a relative measure of the errors.

These metrics help assess the accuracy and performance of regression neural networks, allowing for comparisons between different models or tracking improvements during training. By employing these accuracy metrics, informed decisions can be made about the performance and suitability of regression neural networks for specific tasks and applications.

**Overfitting and underfitting**

Curve fitting refers to the process of finding a mathematical function that best represents a set of data points. It can be formulated as an optimization problem, where the objective is to minimize the difference between the predicted values of the curve and the actual data points. However, there are two common pitfalls to avoid when curve fitting: underfitting and overfitting. Underfitting occurs when the model is too simple and fails to capture the underlying patterns and relationships in the data, resulting in poor accuracy. On the other hand, overfitting happens when the model is too complex and tries to fit the noise or random fluctuations in the data, leading to poor generalization performance on new data. Therefore, finding the right balance between model complexity and accuracy is crucial for successful curve fitting. Underfitting is a common problem in curve fitting, which occurs when the model fails to capture the complexity of the distribution of data in a scatter plot. Although it is often easier to visualize underfitting in two dimensions, curve fitting is usually performed in higher dimensions. The main issue with underfitting is that the model is too simple and oversimplifies the relationship between the input and output variables. As a result, the model fails to capture most of the data points and can make inaccurate predictions. For instance, an underfitted curve may only capture a few data points out of dozens, which can lead to poor performance in real-world applications.

![Figure 2.19: Underfitting and overfitting of a model](image)

The choice of units used during model training is not crucial as the main objective is to employ a heuristic that facilitates the reduction of error at each iteration. What holds significance is the
relative change in error from one step to the next, rather than the absolute magnitude of the error. In addition to units, there is another crucial consideration to take into account when evaluating the "smallness" of the error. It needs to be measured in relation to the specific characteristics of the model, such as its type, the number of available data points, and the training history it underwent prior to the evaluation for accuracy. While this may initially seem counter-intuitive, it becomes more apparent when considering the issue of overfitting. Striking the right balance is essential to ensure that the model's performance is not excessively tailored to the training data, but instead demonstrates generalization capabilities.
Chapter 3

Implementation

3.1 Model Setup

The simulation setup consists of several steps to accurately capture the transient behavior of the valve. In the transient structural analysis, fixed supports and contact interfaces are established, system coupling regions are defined, and the mesh is generated. Mesh generation in Fluent requires careful attention, particularly in defining boundary layers and refining specific areas with smaller element sizes. Quality metrics are evaluated to ensure a reliable mesh.

Once the mesh is in place, the model setup begins. This involves selecting the real gap properties (RGP) table for fluid properties and configuring the boundary conditions. The next step involves the dynamic mesh, where smoothing and remeshing settings play a crucial role in determining the fluid behavior of the valve. Each of these steps will be examined in detail to gain a comprehensive understanding of the transient nature of the valve.

The fluid in the simulation was modeled as CO2 (100% gas phase) with initialization conditions of 101325 Pa as atmospheric pressure and 393.15 K.

3.2 Transient Structural

In general, the FEA workflow can be broken down into:

1. Pre-Processing: the model is prepared by importing the geometry, generating a mesh, defining boundary conditions, and assigning material properties. This step also includes grid generation

2. Processing: In the processing stage, the FEA solver solves the finite element equations to determine displacements, strains, and stresses within the structure. Convergence is checked to ensure a reliable solution

3. Post-Processing: results are visualized through plots and animations, and forces and moments on boundaries are evaluated.

Ansys Transient Structural is a simulation tool used for studying the behavior of structures that are subjected to time-dependent loads or that undergo large deformations over time. The software allows engineers to analyze and predict the structural response of a system under various loading conditions, including thermal, mechanical, and electromagnetic loads.

Transient Structural analysis in Ansys involves creating a finite element model of the system, defining the time-dependent loads and boundary conditions, and then solving the equations of motion for the system over time. The software allows users to observe the structural response of a
system over time, including how it deforms, stresses and strains, and how it responds to various loads.

One key advantage of Ansys Transient Structural is its ability to handle complex structures with intricate geometries, such as those found in aerospace or automotive applications. It allows users to simulate and optimize the behavior of these structures, enabling them to identify potential design flaws and make improvements before the system is manufactured.

### 3.2.1 Mesh

The mesh represents the discretization of the domain into smaller elements, allowing for the numerical solution of the governing equations at discrete points.

<table>
<thead>
<tr>
<th>Mesh Quality Metric</th>
<th>Max CFD</th>
<th>Rec. CFD</th>
<th>Max FEA</th>
<th>Rec. FEA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspect Ratio</td>
<td>20</td>
<td>10</td>
<td>80</td>
<td>4</td>
</tr>
<tr>
<td>Non-Orthogonality</td>
<td>85</td>
<td>60</td>
<td>90</td>
<td>60</td>
</tr>
<tr>
<td>Tet Edge Ratio</td>
<td>100</td>
<td>10</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>Volume Ratio</td>
<td>80</td>
<td>10</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>Skewness</td>
<td>0.95</td>
<td>0.25</td>
<td>0.85</td>
<td>0.25</td>
</tr>
</tbody>
</table>

When creating a mesh for transient structural analysis, several considerations come into play. The mesh should have an appropriate element size that balances computational efficiency with accuracy. Fine mesh resolution is often required in regions of high stress or strain gradients, while coarser elements can be used in areas with less significant variations. In addition to element size, the mesh should also consider the element type. In section 2.6, the common element types used in transient structural analysis, including tetrahedral, hexahedral, and pyramidal elements, were discussed. The choice of element type depends on the geometry and complexity of the structure, as well as the desired accuracy and computational efficiency.

In table 3.1 provides recommended values for various mesh quality metrics, as discussed in section 2.6, including aspect ratio, non-orthogonality, tetrahedron edge ratio, volume ratio, and skewness. These metrics can greatly impact the accuracy and reliability of numerical simulations, so it is important to ensure that they fall within acceptable ranges.

It’s important to note that these recommended values are general guidelines and may vary depending on the specific application, solver, and geometry of the problem. Additionally, mesh quality requirements can differ based on the physics being simulated, such as incompressible versus compressible flow or structural analysis.

Figure 3.1 shows the mesh configuration for the structural body, highlighting the differentiation in element size between the valve and the endstops. In the figure, it can be observed that the valve region is represented by a finer mesh compared to the endstops. This decision is often made based on the structural characteristics and the valve, being a critical component of the structure, typically requires a higher resolution mesh to capture its intricate geometry and potential stress concentrations. This choice helps to maintain computational efficiency while still capturing the essential behavior of the structure in the vicinity of the endstops.

### 3.2.2 Setup

When setting up a transient structural analysis in Ansys, there are several key steps to consider. Now that the material properties have been defined, and the mesh has been generated for the imported geometry, the next step is setting up the boundary conditions. This involves specifying the constraints and loads that will be applied to the structure during the analysis, including fixed supports, prescribed displacements, or external forces applied to specific regions. Boundary
Chapter 3. Implementation

3.2. Transient Structural

Figure 3.1: Mesh of the structural body

conditions and contact interfaces should be properly defined within the mesh to accurately model the interactions and constraints of the structure.

Furthermore, for transient structural analysis, the time discretization scheme, such as implicit or explicit methods, has to be considered along with the time step size. The time step should be small enough to capture the dynamic behavior of the structure, especially for fast-changing phenomena, while maintaining computational efficiency.

Contact interfaces are used to simulate the interaction between different parts or surfaces of a structure. These interfaces define how the contacting surfaces behave under applied loads and deformations. It is important to note that setting up contact interfaces requires careful consideration of the physical behavior of the contact and the desired accuracy of the simulation. Frictionless contact assumes that there is no resistance for sliding between the contacting surfaces. If the consideration of friction is necessary, frictional contact interfaces can be used instead, where a non-zero friction coefficient is specified.

Along with contact interfaces, the fixed support and system coupling region are important aspects when setting up the boundary conditions. A fixed support is a boundary condition that restricts the displacement of a structure, preventing it from moving or rotating in specific degrees of freedom. In figure 3.3, the fixed supports have been selected as the upper surface of the upper-end stop, the lower surface of the lower-end stop, and the end of the valve. System coupling refers to the
interaction between different components or regions within the structure or across multiple physics domains. A system coupling region is a specific area where these interactions occur. In the figure 3.3, all the surfaces of the valve have been considered as system coupling regions.

![Figure 3.3: Setup of the transient structural component](image)

In Transient Structural, the element size must be compatible with the mesh size in Fluent. If the mesh size in FEA does not match the CFD grid, it can lead to unusual effects or even floating-point exceptions, as discussed further in Section 4.1.1. Therefore, ensuring that both the FEA and CFD meshes are appropriately matched is crucial to avoid any numerical instabilities or inaccuracies during the simulation process.

### 3.3 Fluent

Fluid dynamics simulations are employed for a variety of reasons due to their numerous advantages over experimental methods. One significant advantage is that conducting experiments with sensors can be prohibitively expensive and challenging to set up. The cost of acquiring and maintaining specialized sensors, as well as the associated data acquisition systems, can be substantial. Furthermore, the intricate setup and calibration processes required for accurate measurements often demand significant expertise and time. Moreover, building a prototype with the exact geometry involves considerable effort, time, and costs, as it needs to be prepared for construction. In contrast, fluid dynamics simulations offer a cost-effective and convenient alternative. By utilizing numerical methods and mathematical models for an ideal geometry, simulations can accurately capture the behavior of fluids and their interactions with various objects or environments. Various solvers commonly used for CFD simulations include:

1. **Finite Volume Method (FVM):** This method is widely employed due to its ability to handle complex geometries and conserve mass, momentum, and energy. Popular FVM-based solvers include OpenFOAM, Fluent, and STAR-CCM+.

2. **Finite Difference Method (FDM):** This method is suitable for simpler geometries and flow conditions. It discretizes the equations using grid-based approaches, making them straightforward to implement.

3. **Spectral Methods:** Spectral methods utilize orthogonal functions to achieve high-accuracy approximations. They are particularly effective for problems with smooth solutions and can provide rapid convergence. Spectral element and spectral/hp methods are commonly used variations of this approach.

Ansys Fluent is the industry-leading CFD software known for its advanced physics modeling and accuracy. This CFD software enables the modeling and simulation of a wide range of fluid processes, including the ability to incorporate fluid-structure multiphysics interactions. Fluent
Chapter 3. Implementation

3.3. Fluent

contains a list of different models that the users can use according to their needs. For example, if
the user needs to determine viscous flow over an object, they can choose from several different preset
formulations to generate their results. The multiphase processing, energy in the system, acoustics,
and discrete phases can all be set here. In the flow analysis, unsteady, incompressible, and turbulent
flow equations derived from the conservation laws of mass and momentum are solved numerically by
the software Fluent.

In many engineering applications, gases undergo various processes and interactions with their
surroundings that cause them to deviate from ideal gas behavior. These deviations can be significant
and have a major impact on the accuracy of CFD simulations. As a result, real-gas models are
necessary to accurately describe the properties of these gases in non-ideal conditions. Real-gas
models can be quite complex, as they must take into account factors such as molecular interactions,
non-ideal thermodynamic properties, and phase transitions. Ensuring a timely execution of CFD
calculations requires a straightforward implementation of these models. A crucial aspect of achieving
short computational times is the streamlined incorporation of gas property models tailored for
various mediums.

In Fluent, the behavior of gas-liquid flows is governed by the Navier-Stokes equations, which
describe the motion of fluids under the influence of pressure, viscosity, gravity and others.

3.3.1 Mesh

When simulating fluid systems, the first challenge one comes across is the system’s geometry.
Meshing, as priorly mentioned in section 2.6 is the process of discretizing the geometry to ensure
that each cell uniquely captures all relevant flow features. However, this process can be difficult due
to the complexity of the domain, the geometric model’s intricacies, and the desired mesh type. For
example, generating simple cells like triangles is much easier than generating cubicles or hex meshes.
To properly represent flow features such as mixing zones, flow layers, separated regions, boundary
layers, and wakes in a flow domain, the mesh must meet specific node distribution requirements.
Neglecting to do so can lead to inaccurate flow characteristics within the simulation. Inaccurate
resolution in areas such as boundary layers can also affect the simulation’s accuracy of shear stress
and heat transfer coefficient. Therefore, the accuracy of a simulation is heavily dependent on the
quality of the mesh, particularly in near-wall regions where mesh resolution can vary depending on
the fluid properties and the used wall model within the turbulence model. To minimize the impact
of changes in flow variables, it is recommended to use fine meshes in regions where there is a rapid
change in mean flow.

Structured grids offer several advantages over unstructured grids in most solvers. They typically
require less memory, offer higher accuracy, and provide better resolution of boundary layers. Addition-
ally, structured grids effectively capture sharp leading and trailing edges by utilizing cells with
a large aspect ratio, resulting in improved resolution in these regions. When generating a mesh,
it is advisable to avoid significant variations in cell sizes, aiming for a change no greater than a
factor of 1.25 between neighboring cells. Maintaining relatively continuous mesh lines and minimiz-
ing sudden changes in cell directions is beneficial for structured meshes, avoiding discontinuities.
However, adapting a structured grid can be challenging as one edge belongs to two others, requiring
a higher-order solver like OpenFOAM or the use of triangles.

A highly detailed mesh can be created by CFD simulators (Ansys Fluent in this thesis) to achieve
an accurate representation of a 3D model for high-fidelity simulation. High-quality meshing can
be ensured by appropriately choosing the mesh type and fulfilling accuracy parameters, whether
the assessment is done before running the simulation or once the issue is encountered during the
simulation.

In Table 3.2, each row represents a comparison between different values of skewness and the
corresponding number of elements in a mesh. As the skewness decreases, the elements become more
regular and less distorted, indicating a higher-quality mesh. The number of elements decreases
accordingly, reflecting a reduction in mesh density while maintaining sufficient accuracy for the
simulation. It is worth noting that the other settings, such as the element size of 0.3 mm, target
Table 3.2: Comparison of mesh quality: skewness

<table>
<thead>
<tr>
<th>Number</th>
<th>Max Skewness</th>
<th>Min Skewness</th>
<th>Number of elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8924</td>
<td>0.22469</td>
<td>1235823</td>
</tr>
<tr>
<td>2</td>
<td>0.89249</td>
<td>0.22416</td>
<td>1226989</td>
</tr>
<tr>
<td>3</td>
<td>0.87171</td>
<td>0.2245</td>
<td>1228915</td>
</tr>
<tr>
<td>4</td>
<td>0.86413</td>
<td>0.2222</td>
<td>1168435</td>
</tr>
</tbody>
</table>

skewness of 0.7, smoothing of high, and inflation (boundary layers) with a maximum of 5 layers and a growth rate of 1.05, remain constant for all four meshes. Ensuring a closed surface for the CAD geometry and removing small regions that do not significantly impact the flow are also essential considerations.

The image 3.4 depicts the mesh settings for the simulation with boundary conditions with outlet pressure of 60 bar and inlet velocity of 10 m/s. It exhibits the mesh size, which determines the dimensions of individual elements within the model. The element order, referring to the type of mesh element (e.g., tetrahedron or hexahedron), plays a significant role in capturing the geometry’s complexity. Additionally, the figure shows the sizing controls applied to various regions, influencing the level of mesh refinement. Inflation layers, visible in the image, help resolve boundary layer effects (discussed later in section 3.3.1) near solid surfaces. Lastly, the image provides insights into the overall number of elements used in the mesh, contributing to understanding the model’s complexity and computational requirements. These mesh settings are essential in preparing a robust and accurate analysis in Ansys.

**Dynamic mesh**

In certain CFD applications, it is necessary to study the effect of moving geometries. Unlike stationary geometries where the boundaries remain fixed, these cases involve boundaries that can move and change position over time. To handle such dynamic scenarios, a specialized technique known as dynamic mesh is employed. The dynamic mesh model enables the movement and adjustment of cell zone boundaries in response to the motion of the geometry. This means that as the boundaries of the geometry move, the mesh adapts accordingly to accurately represent the changing shape and position of the domain.

The dynamic mesh technique enables the relative motion of boundaries, involving both linear and rotational movements. Moreover, it supports the deformation of boundaries, allowing for stretching,
bending, and shape changes. This feature is particularly valuable when dealing with simulations involving substantial structural deformations. For instance, in a centrifugal pump, the rotating mesh alters the boundary connection between the rotating and fixed grids while keeping the grids themselves constant.

By employing the dynamic mesh technique, the CFD solver can accurately track and account for the movement and deformation of boundaries in the simulation. This ensures that the computational domain properly represents the changing physical configuration, enabling accurate predictions of fluid flow and associated phenomena. It allows for a more realistic representation of real-world scenarios where the boundaries are not fixed but can move and deform, improving the accuracy and reliability of the CFD results.

In many cases, the initial mesh generated may contain elements with skewed shapes, poor aspect ratios, or other distortions that can compromise the reliability of the results. To mitigate these issues, Fluent offers robust tools for mesh smoothing and remeshing, enabling users to enhance the mesh quality and refine the simulation setup. Within Fluent, the dynamic mesh settings play a critical role in accurately capturing the behavior of the valve throughout the simulation.

Fluent incorporates several algorithms for mesh smoothing, such as the Laplacian, Spring Smoothing, or Edge Smoothing methods. The chosen approach for this thesis problem is the spring-Laplace boundary, although an alternative option such as diffusion could also be considered. The smoothing process helps improve the mesh quality and resolves any irregularities in the mesh. On the other hand, remeshing is essential for maintaining the mesh quality during the simulation. During the remeshing process, Fluent allows users to preserve essential features or boundary conditions from the original mesh. The limits for cell skewness are carefully defined as they greatly influence the success and accuracy of the simulation. By focusing on the mesh quality metric of 'skewness', the re-meshing process ensures that the mesh remains suitable for capturing the desired phenomena.

The decision to employ re-meshing emphasizes the importance of maintaining a high-quality mesh throughout the simulation to accurately represent the dynamic behavior of the valve.

Boundary layers

One important aspect of CFD simulations is the accurate representation of boundary layers, which are thin regions along the surface of a solid body that is strongly influenced by viscosity and play a crucial role in determining the overall fluid dynamics of the system. In many cases, the behavior of boundary layers can significantly affect the performance and efficiency of the system, making it important to accurately model these regions in CFD simulations. One such application where boundary layers play a crucial role is the fluid flow via reed valves, which has been extensively modeled using CFD. Boundary layers refer to the thin regions near the surface of a solid body where viscosity dominates, affecting the overall fluid dynamics of the problem. As a result, these regions require particular attention when using CFD to simulate fluid flow for such problems.

Boundaries have been applied on the upper surface of the lower-end stop, the lower surface of the valve, and its inner walls. A different variation of boundaries has been applied to the overall volume.

The boundaries have not been applied to the upper surface of the valve and the bottom surface of the upper-end stop to avoid the squeezing effect of the cells. This decision is primarily made to limit the grid size and the number of cells in the simulation. Applying boundaries to these surfaces would introduce additional cells and grid points in those regions, which may lead to unnecessary computational overhead and increase the complexity of the mesh. By leaving these surfaces boundary-free, the mesh remains more efficient and focused on areas of interest without compromising the overall accuracy of the simulation.
Figure 3.5: Mesh difference with boundary layers

(a) Mesh with boundary layers

(b) Mesh without boundary layers
Chapter 3. Implementation

3.3. Fluent

(a) Boundary layers on the volume

(b) Zoomed in mesh with boundary layers

Figure 3.6: Mesh with boundary layers
3.3.2 Setup

Fluid material

Selecting the appropriate fluid material is an important step of setting up a simulation in Ansys Fluent. The choice of a well-matched or suitable fluid material significantly influences the accuracy and dependability of the simulation outcomes.

Ansys Fluent provides a wide range of fluid material models to choose from, including incompressible and compressible fluids, ideal and real gases, multiphase fluids, and more. To select the appropriate fluid material, the user needs to consider the physical properties of the fluid, such as density, viscosity, thermal conductivity, specific heat, and other relevant parameters. Once the fluid material and boundary conditions have been specified, Ansys Fluent can be used to solve the governing equations of fluid flow, heat transfer, and other related phenomena. The simulation results can then be used to analyze and optimize the performance of the system being studied, and to make informed design decisions based on the insights gained from the simulation.

Users in Fluent have the option to define their own material properties, such as molecular weight, viscosity, thermal conductivity, and specific heat capacity, as a function of temperature, as well as radiation properties. In terms of compressibility, all liquids are considered incompressible with respect to pressure changes, but they are still compressible with respect to temperature changes. On the other hand, gases can be treated as incompressible if the flow Mach number is less than 0.3.

In this work, Carbon dioxide in gaseous phase has been used as the fluid material in Ansys Fluent. Carbon dioxide (CO2) is a commonly used fluid in CFD simulations, particularly in applications related to chemical and process engineering. It is a non-toxic, non-flammable gas that is abundant in the atmosphere and can be easily liquefied under moderate pressure and temperature conditions.

Relaxation factors are a crucial technique for improving the stability of a calculation, particularly in solving steady-state analysis, where the first iterations are critical. They control the under-relaxation of the solution, which helps to avoid oscillations and other numerical instabilities. By reducing the change in the solution between iterations, relaxation factors can improve the convergence rate and accuracy of the simulation. Essentially, relaxation factors provide a damping effect on the solution, helping to smooth out any abrupt changes in the model and improving the stability of the calculation.

A suitable relaxation factor should be chosen to balance stability and convergence speed during the iterative process. It should be small enough to maintain stability and large enough to achieve convergence in a reasonable time. In this thesis, the under-relaxation values worked fine with the default settings.

Boundary Conditions

In CFD simulations, the choice of inlet boundary conditions plays a crucial role and depends on the specific problem at hand. Throughout this thesis, various boundary conditions were investigated, including constant pressure and velocity operating conditions. Additionally, some simulations involved altering pressure within the system. By conducting tests with various boundary conditions, a deeper understanding of how these conditions influence the behavior of the simulated system was obtained.

Using a reference pressure allows for a more meaningful representation of pressure differentials and variations in the simulation. It enables the numerical solver to focus on resolving the relative changes in pressure, which are often of greater significance in many fluid flow problems. By setting a reference pressure, the solver can accurately capture the relevant pressure gradients without being overly sensitive to the absolute magnitudes of pressure values.

Possible inlet boundary conditions:

- Velocity Inlet: In this type of boundary condition, the velocity of the fluid entering the domain
is specified. This is useful when the inlet flow rate is known. For the thesis problem, a mass flow rate of 0.1 kg/s has been determined.

- **Mass Flow Inlet**: Here, the mass flow rate of the fluid is specified at the inlet boundary. This boundary condition is useful when the flow rate is known but the velocity profile is not.

- **Pressure Inlet**: In this type of boundary condition, the pressure of the fluid entering the domain is specified. This is useful when the inlet pressure is known but the velocity or flow rate is not.

- **Inlet Turbulence Specification**: The turbulence characteristics of the fluid entering the domain can also be specified at the inlet boundary. This can be done by specifying turbulence intensity, turbulence length scale, and turbulence viscosity ratio. These settings provide a history of turbulence characteristics at the inflow boundary.

- **Temperature Inlet**: If the temperature of the fluid entering the domain is known, it can be specified using this boundary condition.

- **Fan or Blower**: In some cases, a fan or blower may be used to force the flow into the domain. The inlet boundary condition can be set to simulate this effect.

These are some common inlet boundary conditions used in CFD simulations and the selection of inlet boundary conditions in CFD simulations depends on the specific problem at hand. In this work, simulations were performed using two distinct types of inlet boundary conditions, specifically velocity inlet and pressure inlet combined with a temperature inlet. These different boundary conditions were utilized to investigate their effects on the behavior and performance of the simulated system. For the velocity inlet, a range of constant velocity magnitudes between 5 m/s and 25 m/s were used in the simulations. This means that the fluid entering the domain through the inlet had a fixed velocity magnitude within this range for all the simulations. The simulations were conducted to study the effect of inlet velocity magnitude on the flow behavior and related parameters of the system being analyzed. By adopting this approach, the fluid velocity at the inlet was determined based on flow physics rather than a prescribed magnitude, making it suitable for scenarios with uncertain or complex inlet conditions. An opening of the valve leads to a pressure drop in real world. A constant velocity inlet is not given either in the underlying compressor however a real compressor model would only increase the issues and to get a stable run, these assumptions were made.

### 3.4 System Coupling

Researchers have typically used one of two methods—monolithic or partitioned—to conduct multiphysics simulations. The monolithic technique involves solving the equations for all the physics involved in a situation as a single matrix system. This approach considers the coupled interactions between the fluid and structure within the same solver, allowing for a more comprehensive analysis of fluid-structure interaction phenomena. On the other hand, in the partitioned approach, distinct equations are solved using an appropriate coupling mechanism (usually as parts of several single physics software). The fluid and structural domains are treated as independent entities, and the information is exchanged iteratively between the solvers at specified coupling interfaces. This approach is suitable when the fluid and structural behavior can be reasonably decoupled or when existing solvers for fluid and structural problems are already available.

Due to the varying stiffness of different mathematical models, such as liquids being about 10 times weaker than steel and gas being approximately 10,000 times weaker than incompressible fluids, the matrix system in the monolithic method often becomes ill-conditioned. The data flow between several physics solvers is a common and essential aspect of a partitioned coupling application and ‘System Coupling’ in Ansys is based on the idea to combine two or even more Ansys solvers for various physical subsystems (fluids, mechanical, thermal, electromagnetics, etc.) in a single simulation (1-way coupling, 2-way coupling, stationary, transient, etc.). The separate solver programs that are to be connected do not directly communicate with one another in this method. They normally communicate with one another through an intermediary coupling application. A data transfer is the
transmission of a variable type between two participants and is specified by the source and target regions. For example, when many participants co-simulate their parts of a linked study, they may engage in both one-way and two-way data exchanges as either a source or a target. Similarly to this, participants may only use one-way data transfers as a source when they are offering access to already published findings or data.

Mapping weights are created for all data transfers at the beginning of the coupled analysis (i.e., either at the start of an original run or at its restart), and they are then utilized in each coupling iteration to interpolate data from the source mesh onto the destination mesh. Depending on the kind of connection, the data might be temperature, displacement, force, heat fluxes, or heat transfer coefficient. It is necessary to uphold both local and global conservation of metrics like the sum while transmitting things like force and heat flow. The profile between the source and target meshes must be maintained, nevertheless, while transmitting elements like displacement and temperature.

In order to establish a 2-way FSI simulation, data transfers can be created between the transient structural and Fluent components. The System Coupling settings allow for configuration of the end time of the simulation as well as the timestep size. In this study, it was found that using a timestep size of $10^{-5}$s resulted in good and considerably accurate simulation results.

The time step size is an essential parameter that influences the simulation’s accuracy and stability. It controls how frequently the solver updates the solution throughout the simulation. The optimal value is determined by various factors, including the physical parameters of the system being simulated and the numerical methods utilized. In general, a smaller time step size results in a more accurate solution, but at the expense of higher processing time. In contrast, a greater time step size will reduce the calculation time but can provide an unstable or inaccurate answer. It is important to note that the appropriate time step size may vary throughout the simulation, depending on changes in the flow properties or geometry of the system. Therefore, it is recommended to monitor the time
step size and adjust it as necessary to ensure an accurate and stable solution.

![Figure 3.9: Analysis Setting for system coupling](image)

In conclusion, after taking all the necessary steps, including defining material properties, generating a mesh, setting up boundary conditions, and configuring system coupling, the simulation is initiated and allowed to run. By initiating the simulation, the transient structural and Fluent components work together to capture the dynamic behavior of the structure and fluid flow interactions over time. The simulation progresses according to the defined time steps, with data transfers enabling the exchange of relevant information between the two components.

### 3.5 Data generation and Preparation for the Neural Network

#### 3.5.1 Data generation

Generating data for FSI simulations for deep neural networks is a critical step in developing accurate and robust models. To build a broad and representative data set for FSI simulations used in deep neural networks, several simulations with various input parameters, boundary conditions, and geometries are often conducted. This method can be time-consuming and computationally expensive, especially when creating huge data sets.

Efficient data generation can be achieved by using a design of experiments (DoE) approach. This method involves selecting input parameters and boundary conditions that cover the range of values of interest and combining them systematically to create simulation cases. To reduce the number of simulations required, statistical methods can be utilized to choose a representative subset of cases that capture the full range of variability in the input parameters.

A DoE approach could also define several single points out of the matrix and not only a time-based approach with constant input parameters as done in this study. The main benefit in this approach is the iterative usage of valid positions in the field and the faster calculation after the first stabilization of the system. Regarding pressure levels and their corresponding temperature, density, and viscosity were used instead of temperature due to their direct physical influence and non-linear behavior, especially viscosity. This approach allows for gas exchange, but it may require different pressure differences.

Using machine learning techniques to expand the data set is another way to produce data. This entails taking a small fraction of the available data to train a neural network, then using the network to produce fake data that mimics the behavior of the real data set. The generated data may be utilized to expand the data set and enhance the network’s generalization.

To generate the necessary data for training and validating the deep learning model, simulations were carried out at two distinct pressure levels: 60 bar and 120 bar. Additionally, three different velocities were considered: 8 m/s, 10 m/s, and 12 m/s. By conducting these simulations at various pressure and velocity combinations, a diverse dataset was generated, capturing the wide range of operating conditions. This dataset will serve as the basis for training and validating the deep learning model, enabling it to learn and generalize from the simulated scenarios.

For each simulation, appropriate boundary conditions were applied to the model to replicate
Chapter 3. Implementation

3.5. Data generation and Preparation for the Neural Network

The desired pressure and velocity conditions. These boundary conditions were carefully selected to accurately represent the real-world conditions that the valve may experience during operation. Once the simulations were completed, a large amount of data was generated. This data included information such as pressure, velocity, and temperature distributions within the model, as well as the dynamic response of the valve plate. The data was then post-processed to extract relevant features that could be used as inputs for the deep learning model.

After generating the dataset, it was partitioned into three distinct subsets: training, validation, and testing sets. The training set was utilized to train the deep learning model, enabling it to learn the underlying patterns and relationships within the data. The validation set played a crucial role in fine-tuning the model and preventing overfitting, as it provided an independent dataset for assessing the model's performance during the training process. Finally, the testing set was reserved for evaluating the model's performance on new, unseen data, providing a reliable measure of the model's generalization ability. This division of the dataset into separate sets allowed for robust model training, validation, and unbiased assessment of its performance.

Overall, the data generation process was a necessary and critical step in developing an accurate and reliable deep-learning model for predicting the behavior of the valve in different operating conditions. The generated data served as the foundation for the development of the deep learning model, and ensured that the model was capable of accurately predicting the behavior of the valve under a range of different operating conditions.

The provided table (3.3) contains simulated data related to fluid flow parameters. These parameters include "Time Step (s)," "Input Velocity (m/s)," "Inlet Density (kg/m$^3$)," "Static Pressure Inlet (Pa)," "Static Pressure Outlet (Pa)," "Static Temperature Inlet (K)," "Total Pressure Inlet (Pa)," "Viscosity Inlet (Pa-s)," "Z-Coord (m)," "Mass Flow Rate (kg/s)," and "Z-Velocity (m/s)." The data covers various time steps, ranging from 0 to 0.001 seconds, with a timestep size of 1e-6 seconds. Specifically, the parameters "Z-Coord (m)" and "Z-Velocity (m/s)" have been recorded for nine selected nodes on the upper surface of the valve. A detailed discussion of these parameters can be found in Chapter 4.

For the neural network, the input parameters would include "Time Step (s)," "Input Velocity (m/s)," "Inlet Density (kg/m$^3$)," "Static Pressure Inlet (Pa)," "Static Pressure Outlet (Pa)," "Static Temperature Inlet (K)," "Total Pressure Inlet (Pa)," "Viscosity Inlet (Pa-s)," and "Z-Coord (m)." The trained network would later be used to predict the output parameters: "Mass Flow Rate (kg/s)" and "Z-Velocity (m/s)."

3.5.2 Pre-processing of data

Data preprocessing is an integral aspect of machine learning and deep learning tasks and often consumes a significant amount of time during development and performance optimization. Similar to the iterative refinement of network hyperparameters like architecture, learning rate, and loss function, implementing the data preprocessing step also follows an iterative process. This involves exploring various techniques such as data normalization, augmentation, and filtering procedures. By experimenting with different preprocessing approaches, the quality and suitability of the input data can be enhanced, thereby improving the overall performance of the machine learning model[30]. Real-valued input and output variables can be scaled through the process of normalization or standardization.

The purpose of normalization is to change the input data so that it has an average of zero and a standard variation of one. This makes it easier for the neural network to learn from the data effectively. The normalization process involves subtracting the average of each input from the data and dividing it by the standard deviation. This changes the data to a standard distribution, making it simpler for the neural network to identify patterns.

One significant advantage of data normalization is that it helps the neural network avoid getting stuck in local minima during the optimization process. Local minima occur when the gradient descent algorithm used for optimizing the neural network gets trapped in a suboptimal solution. By normalizing the data, the gradient descent algorithm is less likely to get stuck in local minima, since
<table>
<thead>
<tr>
<th>Time Step (s)</th>
<th>Input Velocity (m/s)</th>
<th>Inlet Density (kg/m^3)</th>
<th>Static Press Inlet (Pa)</th>
<th>Static Press Outlet (Pa)</th>
<th>Static Temp Inlet (K)</th>
<th>Total Press Inlet (Pa)</th>
<th>Viscosity Inlet (Pa-s)</th>
<th>Z-Coord (m)</th>
<th>Mass Flow Rate (kg/s)</th>
<th>Z-Vel (m/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8.00</td>
<td>93.30</td>
<td>6000000.00</td>
<td>6000000.00</td>
<td>393.2</td>
<td>6000000.00</td>
<td>0.0000206</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1e^-06</td>
<td>8.00</td>
<td>95.62</td>
<td>6196970.81</td>
<td>6000000.00</td>
<td>393.3</td>
<td>6182597.60</td>
<td>0.00000206</td>
<td>0.000</td>
<td>0.000</td>
<td>0.018</td>
</tr>
<tr>
<td>2e^-06</td>
<td>8.00</td>
<td>96.10</td>
<td>6214044.51</td>
<td>6000000.00</td>
<td>393.3</td>
<td>6212111.74</td>
<td>0.00000206</td>
<td>0.000</td>
<td>0.000</td>
<td>0.040</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>9e^-04</td>
<td>8.20</td>
<td>95.89</td>
<td>6146843.17</td>
<td>6000023.34</td>
<td>393.1</td>
<td>614951.32</td>
<td>-0.0000207</td>
<td>-0.029</td>
<td>-0.216</td>
<td></td>
</tr>
<tr>
<td>0.001</td>
<td>8.20</td>
<td>95.88</td>
<td>6146681.97</td>
<td>6000023.16</td>
<td>393.1</td>
<td>6149190.15</td>
<td>-0.0000207</td>
<td>-0.029</td>
<td>-0.219</td>
<td></td>
</tr>
<tr>
<td>0.000</td>
<td>10.00</td>
<td>93.30</td>
<td>6000000.00</td>
<td>6000000.00</td>
<td>393.2</td>
<td>6000000.00</td>
<td>0.00000206</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>1e^-06</td>
<td>10.00</td>
<td>95.93</td>
<td>6221998.90</td>
<td>6000000.00</td>
<td>393.3</td>
<td>6206112.92</td>
<td>0.00000206</td>
<td>0.000</td>
<td>0.018</td>
<td></td>
</tr>
<tr>
<td>2e^-06</td>
<td>10.00</td>
<td>96.48</td>
<td>6241354.15</td>
<td>6000000.00</td>
<td>393.3</td>
<td>6239548.48</td>
<td>0.00000206</td>
<td>0.000</td>
<td>0.040</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>9e^-04</td>
<td>10.20</td>
<td>95.94</td>
<td>6149553.66</td>
<td>6000025.84</td>
<td>393.1</td>
<td>6153488.23</td>
<td>-0.0000210</td>
<td>-0.032</td>
<td>-0.274</td>
<td></td>
</tr>
<tr>
<td>0.001</td>
<td>10.20</td>
<td>95.92</td>
<td>6148747.86</td>
<td>6000025.61</td>
<td>393.1</td>
<td>6152682.45</td>
<td>-0.0000210</td>
<td>-0.032</td>
<td>-0.275</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3: The dataset with various input and output parameters at different time steps
the data has a standard distribution and the optimization process can proceed more smoothly.

The mean and standard deviation used in batch normalization are related to the mean and standard deviation of the entire training dataset. During training, the mean and standard deviation of each batch of data is calculated and used to normalize the input data to that batch. However, the running means and standard deviation for each layer are also updated using an exponentially weighted moving average of the batch mean and standard deviation. By normalizing the input data, the network can adapt to shifts in the data distribution, which is essential for real-world applications where the input data may change over time. Normalization is crucial to give equal importance to each feature and avoid any single feature from overpowering the model. For example in this thesis, pressure ($10^7$ Pa) and viscosity ($10^{-1}$) are orders of magnitude apart, making normalization essential. By normalizing the data, the features can be standardized to have a zero mean and unit variance. This process aids in reducing the influence of outliers in the data and enhances the convergence of the optimization algorithm during training.

Secondly, normalization helps to speed up the training process by making the input data more manageable for the neural network. Normalization reduces the amount of computational resources needed to train the model, which can save time and resources.

Thirdly, the mean and standard deviation of the data can also be used to normalize the test data. During the testing phase, the same normalization parameters can be applied to the test data to ensure that the predictions are consistent with the training data.

MATLAB provides a comprehensive Deep Learning Toolbox that offers a range of functions, algorithms, and tools for developing and deploying deep learning models. It simplifies the process of designing, training, visualizing, and deploying various types of neural networks.

Here are some key features and functionalities of the Deep Learning Toolbox in MATLAB:

- **Neural Network Architectures**: MATLAB supports various neural network architectures, including feedforward networks, convolutional neural networks (CNNs), recurrent neural networks (RNNs), and generative adversarial networks (GANs). Creation, configuration, and customization of network architectures can be achieved through the utilization of high-level functions and classes.

- **Pretrained Models**: MATLAB provides access to a collection of pre-trained deep learning models, such as popular CNN architectures (e.g., AlexNet, VGG-16, GoogLeNet, ResNet) and word embeddings (e.g., Word2Vec, GloVe). These models can be fine-tuned or used for transfer learning on specific tasks.

- **Training and Optimization**: The Deep Learning Toolbox offers functions for training neural networks using various optimization algorithms, including stochastic gradient descent (SGD), adaptive moment estimation (Adam), and more. It supports both CPU and GPU acceleration for faster training.

- **Data Preparation**: Preprocessing and data augmentation of training data can be conveniently performed using the built-in functions available in MATLAB. It includes tools for data normalization, image resizing, data augmentation (e.g., rotation, translation, scaling), and splitting datasets into training, validation, and testing subsets.

- **Visualization and Analysis**: MATLAB provides functions for visualizing neural network architectures, plotting training curves, analyzing feature activations, and exploring learned filters. These visualization tools aid in understanding the behavior and performance of deep-learning models.

- **Deployment**: Once the model is trained, MATLAB allows deployment for real-time predictions or integration into larger systems. CUDA code can be generated for GPU deployment or conversion of the model to ONNX format for interoperability with other frameworks.

- **Automatic Differentiation**: The Deep Learning Toolbox supports automatic differentiation, enabling the automatic computation of gradients during training. This simplifies the process of implementing complex neural network architectures and custom loss functions.
Deep learning toolbox also offers activation functions which introduce non-linearity, allowing the network to learn complex patterns and make predictions for various tasks. Some common activation functions include the sigmoid, ReLU, Leaky ReLU, tanh, softmax, and linear activations. Each function serves specific purposes, such as binary classification (sigmoid), avoiding vanishing gradients (ReLU and Leaky ReLU), and handling multi-class classification (softmax). The choice of activation function significantly impacts the network’s performance, and careful selection is essential for effective model training and generalization.

For the purpose of this thesis, the tanh activation function is used in Long Short-Term Memory (LSTM) networks. It helps the network control the flow of information by capturing long-range dependencies in sequential data. Additionally, tanh maintains activations around zero, making it suitable for hidden layers with standardized data and ensuring better gradient preservation during backpropagation.

Here in the table 3.4, all the versions of the software used in the thesis have been mentioned.

<table>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ansys Workbench</td>
<td>R2022a</td>
<td>FSI</td>
</tr>
<tr>
<td>Ansys Fluent</td>
<td>R2022a</td>
<td>CFD</td>
</tr>
<tr>
<td>Matlab</td>
<td>R2022</td>
<td>Deep learning Toolbox</td>
</tr>
</tbody>
</table>

Table 3.4: Software and their versions
Chapter 4

Experimental Results

In this section, the focus is placed on the analysis of the pressure and velocity contours of the simulation model within the CFD simulation. Specific boundary conditions have been defined, with an outlet pressure set at 60 bar and an inlet velocity of 10 m/s. The velocity into the system increases the pressure up to the valve opening and beyond. By examining the pressure and velocity contours, valuable insights regarding the characteristics and behavior of the fluid flow within the system are aimed to be uncovered. The effects of these boundary conditions on the flow dynamics and the comprehensive understanding of the system’s behavior are sought through this analysis.

Additionally, the investigation will encompass the analysis of the total deformation of the valve, considering the prescribed boundary conditions. This analysis aims to comprehend how the valve responds to the fluid flow and determine its final position. Through the examination of the deformation, the structural integrity and performance of the valve under these operating conditions can be evaluated.

Furthermore, comparisons will be made between different outlet pressures, specifically 60 bar and 120 bar, while maintaining a constant inlet velocity of 10 m/s. This comparative analysis will shed light on how changes in outlet pressure affect the behavior of the system. Additionally, the effects of varying inlet pressure will be explored, providing a comprehensive understanding of how different boundary conditions influence the overall fluid flow and valve performance.

4.1 Analysis and interpretation of the Ansys results

In Ansys Fluent, pressure, and velocity contours are a visualization tool that shows the distribution of static pressure and velocity over a surface or a volume. The simulation of the compression chamber with respect to changing volume and pressure increase would increase the simulation effort by size. So the system was cut at a position where the volume is constant.

4.1.1 Different boundary conditions and mesh settings

The total pressure is the sum of both static and dynamic pressure (aka Bernoulli pressure) components. It experiences a decrease when there is a momentum sink which is primarily attributed to irreversible viscous losses. Consequently, the total pressure field tends to remain uniform throughout the flow, except in regions near surfaces where boundary layers form. These boundary layers cause deviations from uniformity in the total pressure distribution. The outlet total pressure is less than the inlet total pressure resulting in a loss of energy.

1. Boundary condition: inlet constant velocity: 10 m/s, outlet pressure: 60 bar  As the valve opens, the compressed gas in the inlet chamber experiences a pressure increase due to the
compression of the gas. This increased pressure drives the flow rate, which is influenced by both the pressure and the flow area. The pressure difference between the inlet chamber and the outlet is determined by the flow rate and the drag acting on the system.

When the flow encounters a cylinder, the molecules on the left side become trapped and unable to move due to the presence of the cylinder wall (stagnation point). As a result, the static pressure at this point reaches its maximum value while the velocity drops to zero. As the fluid flow advances, it applies pressure on the incoming stream, redirecting it around the stagnation point. This results in a higher velocity (greater than the free stream velocity) and a decrease in pressure. As these molecules gain high velocity, they encounter another curvature in the wall ahead. The flow, exerting its force on the molecules, causes them to align along the wall, aligning with the direction of the flow. This alignment causes an increase in pressure (referred to as adverse pressure) and a decrease in velocity. High inlet pressure reduces to lower outlet pressure as the air flows through this orifice.

The velocity cut plot 4.1a is next, scaled in m/s. This is a plane view of the cross-section of the model and the colormap represents the velocity. The velocity in the inlet region is between 7.5 m/s to 10 m/s however, a high velocity of 37.8 m/s is observed at the opening edge of the valve and on the right side of the image, between the small gap of the valve and the lower end stop. The increased flow could be due to the relatively poor meshing of the cells at the edge of the valve.
In the simulation, a no-slip boundary condition was applied to all walls, defining that the fluid velocity at the wall is zero. In Fluent, when simulating incompressible flow, the pressure boundary condition is defined in terms of gauge pressure, which is the pressure relative to the operating pressure. The operating pressure was determined by taking the average of the inlet and outlet pressures, and the inlet and outlet gauge pressures were calculated accordingly.

Figure 4.2 defines the graph of static inlet pressure over time. The y-axis denotes the vertex average of pressure kgm$^{-1}$s$^{-2}$ (1 x $10^6$) and the x-axis denotes the flow time (x $10^3$). At the initial stages of the simulation, the graph exhibits the highest static inlet pressure value. As time progresses, the pressure gradually decreases with almost negligible fluctuations. The maximum pressure difference observed during the simulation is 4 bar, which reflects the variation in pressure within the system during the simulation period.
In Figure 4.3, the maximum opening of the valve is visible, followed by a subsequent closing of the valve to maintain a constant outlet pressure. The red line denotes the minimum value, the green line denotes the maximum value and the blue line denotes the average value of the deformations over the nodes. The valve exhibits a gradual opening process, reaching its maximum position of 0.84 mm at 0.351 milliseconds (ms). Subsequently, it experiences a slight downward movement before rising again and maintaining a constant position until the end of the simulation. The observed shift in the valve position can be for ensuring a constant outlet pressure. The dynamic response is quite fast, the peak is observed after about 0.3 ms and the response is stable after about 0.7 ms. As the conditions within the system change, the valve adjusts its position to maintain the desired pressure level at the outlet. This dynamic response helps to regulate the flow and ensure consistent pressure conditions throughout the simulation.

Thus, a transient analysis with dynamic mesh motion is performed to determine the ultimate valve position as well as the transient nature of valve movement.

2. Comparison between outlet pressure of 60 bar and 120 bar:

In Image 4.4, the figure demonstrates the deformation of a valve with different outlet pressure conditions: 60 bar and 120 bar. The simulations maintain the same inlet velocity of 10 m/s, enabling a focused analysis of the effect of outlet pressure on the valve’s behavior while minimizing other potential factors. Notably, both simulations exhibit similar behavior in the movement of the valve. The valve reaches its maximum position, of 0.88 mm and 1.006 mm respectively, at approximately 0.35 ms in both cases and subsequently maintains a nearly constant opening. This consistency in behavior but difference in maximum opening of the valve suggests that the outlet pressure significantly influences the valve’s deformation pattern. The pressure and velocity contours are not shown in the figure because all structural and Fluent mesh settings, dynamic mesh configurations, and analysis settings, such as end time and time-step size, are kept consistent in both simulations. This ensures that any observed differences in the valve’s behavior can be confidently attributed to the variation in outlet pressure.

3. Comparison at uniform outlet pressure 120 bar with different velocities of 5 m/s, 15 m/s, and 25 m/s:

The figure 4.5 shows the deformation of a valve under various inlet velocity conditions: 5 m/s, 15 m/s, and 25 m/s, while maintaining a constant outlet pressure of 120 bar. At the lowest inlet velocity of 5 m/s, the valve exhibits stable behavior throughout the simulation, with no significant distortions or deformations observed. When the inlet velocity is increased to 15 m/s, the valve’s movement becomes more pronounced, showing a noticeable deformation. However, with this increase in velocity, the valve displays an unstable behaviour, with a sudden drop in the middle of the simulation time and never opening much again. This could be due to different dynamic mesh settings. It is observed that the behavior of the valve is relatively stable at a high inlet velocity of 25 m/s, with a twist occurring at timestep 0.18 ms. As expected, the highest movement of the valve is at the highest inlet velocity of 25 m/s.

4. Inlet boundary condition of changing pressure and constant outlet pressure:

When simulating a fluid flow problem, there are several types of boundary conditions that can be applied to the edges of the computational domain. These conditions define how the fluid interacts with the boundary and are crucial for accurately simulating real-world scenarios.

In the case of constant pressure, the pressure at the boundary is held constant throughout the simulation. This can be useful for simulating problems where the fluid is being pumped into or out of the domain at a fixed pressure, or for modeling situations where the pressure is otherwise known. Changing pressure boundary conditions can be more complex to model, as they require the pressure at the boundary to vary over time. This can be useful for simulating scenarios where the pressure inside the domain is changing due to external forces or changes in flow rate. The pressure was changed iteratively throughout the simulation to observe the opening of the valve according to the following equation:
4.1. Analysis and interpretation of the Ansys results

(a) Outlet pressure: 60 bar

(b) Outlet pressure: 120 bar

Figure 4.4: Different outlet pressure
Chapter 4. Experimental Results

4.1. Analysis and interpretation of the Ansys results

(a) At 5 m/s

Figure 4.5: Comparison at uniform outlet 120 bar with different velocities of 5 m/s, 15 m/s, and 25 m/s
Chapter 4. Experimental Results

4.1. Analysis and interpretation of the Ansys results

\[ p(t) = \begin{cases} 
12000000 \text{ Pa} - 5000000000 \text{ Pa/s} \cdot t & \text{if } t < 0.001 \text{ s} \\
6000000 \text{ Pa} & \text{otherwise}
\end{cases} \]

This uses the cases environment to define two cases for the pressure function depending on the value of the time \( t \) which leads to the following graph about the absolute pressure at the outlet.

As depicted in Figure 4.6c, the examination reveals that the upper edge of the valve encounters a significant velocity of 128 m/s. This observation suggests a rapid movement of the fluid in that region, which could be attributed to a localized flow concentration. In close proximity to the walls of the lower-end stop, a swirling pattern is observable, with the fluid flowing at velocities ranging from 2.56 m/s to 6.4 m/s. The presence of this swirling motion indicates the likelihood of vortices or eddies within the fluid flow. As the upper end-stop is approached, a gradual decrease in flow speed is observed, indicating a reduction in fluid velocity. However, upon entering the outlet chamber, there is a slight increase in flow velocity, accompanied by a swirling flow pattern with a velocity of 5.5 m/s. This change in flow behavior can be attributed to the transition from the confined space of the valve to a more open area.

In Figure 4.6d, it is observed that after reaching full opening, the inlet region exhibits a pressure of 115.37 bar, whereas a higher pressure of 128.89 bar is observed on the underplate of the valve. The small gap between the upper-end stop and the valve in the middle has the highest pressure of 170.22 bar. This indicates a significant increase in pressure within this confined space, which may be attributed to flow constriction or compression in the fluid.

Potential Errors

- Negative volume error: In dynamic mesh simulations, where the mesh undergoes significant deformations or changes during the simulation, negative volume errors can occur. Dynamic mesh simulations involve continuously adjusting and adapting the mesh to account for boundary motion or deformation. However, under certain conditions, such as extreme deformation or sudden geometric changes, negative volume errors can arise. This issue can be attributed to factors such as overlapping nodes and significant mesh motion relative to the size of the cell.

   These errors can introduce numerical instabilities and inaccuracies in simulation results, as negative volumes are physically unrealistic and pose mathematical challenges.

   To address negative volume errors, several techniques can be employed. Remeshing techniques can be applied to regenerate the mesh in regions where negative volumes are detected.

   To address this error, several potential solutions can be considered:

   - Mesh refinement: Increasing the mesh resolution by adding more cells can help prevent large displacements and mitigate the occurrence of negative volumes.

   - Time step reduction: Decreasing the size of the time step used in the simulation can limit the magnitude of mesh motion per step, reducing the likelihood of encountering negative volumes.

   - Smoothing parameters: Implementing smoothing algorithms or techniques specific to dynamic meshes can help alleviate mesh distortions and improve mesh quality, reducing the occurrence of negative volumes.

   - Triangular mesh in dynamic zones: In cases where the mesh undergoes significant deformation or motion, using triangular elements instead of quadrilateral elements can provide better resistance to cell collapse and negative volume issues.

   - First-order upwind scheme: If necessary, using a first-order upwind scheme for the numerical discretization of the governing equations can enhance stability and mitigate the impact of mesh distortions on the solution accuracy.

   Ensuring proper mesh quality assessment, selecting appropriate mesh deformation methods, and considering the physical behavior of the system are essential in achieving accurate and
Chapter 4. Experimental Results

4.1. Analysis and interpretation of the Ansys results

(a) Change in absolute pressure

(b) Deformation of the valve

(c) Plane view of velocity magnitude

(d) Plane view of pressure

Figure 4.6: CFD results of reed valve with changing pressure
reliable results. By implementing these strategies, the occurrence of negative volume errors can be mitigated, enhancing the robustness and credibility of dynamic mesh simulations in computational fluid dynamics.

- A floating point exception: In Ansys Fluent, this exception occurs when there is an arithmetic error involving floating-point numbers during the simulation. This can happen due to various reasons, such as division by zero, invalid mathematical operations, or numerical instability in the solver.

It is important to identify the specific cause of the exception in order to address and resolve the problem. This may involve reviewing the input data, checking the boundary conditions, examining the solver settings, or adjusting the mesh quality.

The floating-point exception issue during the thesis was resolved by matching the element size in the Transient Structural mesh with that of the dynamic mesh in Fluent. This step is crucial for accurate and stable simulations, as inconsistent element sizes can lead to numerical instabilities and errors in the simulation results.

4.1.2 Validation of the FEM/CFD/FSI results

To validate these Ansys results additional data is needed, which could be from:

- Analytical solutions
- Experimental solutions
- Computations from similar literature

In case of no experimental data, people validate benchmark cases similar to the intended study. Due to budget constraints and the scope of this master thesis, it was decided not to pursue experimental validation of the CFD results. Conducting experimental tests would have involved considerable costs and extensive setup, including the installation of sensors and equipment, which were not deemed feasible within the available resources. The focus of this research was primarily on computational analysis, and alternative validation methods, such as benchmarking with similar studies, were considered more suitable under the given circumstances.

4.2 Neural Network

The dataset has already been discussed in the section 3.5, a training dataset was generated with specific input parameters, namely according to an input velocity of 8m/s followed by 10m/s. Each input velocity was associated with 1000 data points, resulting in a sufficient amount of training data for the neural network model.

Learning rate: Initial learning rate($\eta$) used for training, specified as a positive scalar. Typically, the learning rate is set to a value between 0.0 and 1.0. For a neural network with standardized inputs (or inputs mapped to the (0,1) interval) are less than 1 and greater than $10^{-6}$ [31]. The default value is 0.01 for the ‘sgdm’ solver and 0.001 for the ‘rmsprop’ and ’adam’ solvers.

In the optimization process, the learning rate interacts with many other factors, and these interactions may not be linear. However, as a general rule, smaller learning rates will need more training epochs, while larger learning rates will require fewer training epochs. Additionally, smaller batch sizes are more appropriate for smaller learning rates because of the noisy estimate of the error gradient. There are diagnostic plots that can be used to investigate the impact of the learning rate on the learning dynamics and rate of the model. For example, a plot of the training loss versus the number of training epochs can be used to observe the rate of learning. Another useful plot is the learning curve, which shows the training and validation loss versus the number of training epochs.
This plot can help to diagnose issues with overfitting or underfitting, which can occur when the learning rate is not appropriate for the given problem. Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it indicates how concentrated the data is around the line of best fit. A loss function is a measure of how good a prediction model does in terms of being able to predict the expected outcome.

The Deep Learning Toolbox in MATLAB offers a restricted set of training options, limiting the available visualizations to RMSE and loss function plots. As a result, the flexibility to visualize the training progress in a linear or logarithmic plot is constrained. Nevertheless, verbose output is provided, allowing for a more comprehensive investigation into the training details.

The two graphs below in Figure 4.7 show the training progress of two models with different learning rates.

![Training Progress](image)

(a) Learning rate: 0.0001

(b) Learning rate: 0.01

Figure 4.7: RMSE with different learning rates
In Figure 4.7a, the RMSE is quite high which signifies bad performance by the network. RMSE values closer to 0 indicate good performance, which is clearly not the case here. Also, the loss function also is better with a learning rate of 0.01. The validation RMSE is also quite high with a learning rate of 0.0001 with the value being 1.47 compared with the learning rate of 0.01.

### 4.2.1 Analysis and Interpretation

In this section, the focus is shifted toward the examination of neural network predictions and their implications in the study of real-world reed valves. The patterns and correlations emerging from the model’s predictions are carefully analyzed, contributing to a better understanding of reed valve dynamics.

**Figure 4.8: Training progress of the neural network**

Figure 4.8 illustrates the training progress of the neural network using MATLAB’s Deep Learning Toolbox. The figure consists of two plots displayed vertically.

- **RMSE vs. Iterations (Upper Plot):** This plot shows the Root Mean Square Error (RMSE) on the y-axis and the training iterations or epochs on the x-axis. The plot demonstrates how the RMSE value changes as the neural network goes through training iterations. As the training progresses, the RMSE should ideally decrease, indicating improved performance and better fit to the training data.

- **Loss vs. Iterations (Lower Plot):** This plot displays the loss function on the y-axis and the training iterations or epochs on the x-axis. It visualizes how the loss function changes during the training process. The goal is to minimize the loss function as the training progresses, which signifies that the model is learning to make more accurate predictions.
Figure 4.9: Verbose of Training Progress

Figure 4.9 shows the verbose output of the training progress, providing specific details about the neural network’s performance. The figure displays the RMSE and the Validation Loss of the trained neural network. These values are key indicators of the model’s accuracy and generalization capabilities. The final RMSE represents the accuracy achieved on the training data, while the Validation Loss indicates the model’s performance on unseen validation data. Lower values for both metrics signify better model performance.

The neural network was trained using various fluid properties, such as density and viscosity, along with boundary conditions, including inlet velocity, inlet pressure, total pressure and static inlet temperature. Additionally, the z-coordinates of nine selected nodes in the valves were utilized as inputs. The primary goal was to predict the final velocities of these nine nodes in the valve system, along with their corresponding mass flow rate. The network was initially trained with inlet velocities of 8 m/s and 10 m/s. Subsequently, the trained network was used to make predictions for 9 m/s and 10 m/s velocities using 2000 data points.

Results of test vs predicted data with trained neural network  When training deep learning networks, it’s important to keep an eye on the progress of the training process. One way to do this is by plotting various metrics that provide insights into the network’s performance. These metrics enable the observation of whether the network’s accuracy is improving over time and detect
Chapter 4. Experimental Results

4.2. Neural Network

signs of overfitting to the training data. By monitoring these metrics throughout the training process, informed decisions can be made regarding adjustments to the model or the application of regularization techniques. This helps to better understand how the network is behaving and ensure that the training is proceeding in the right direction. Below, the images 4.11 depict the display of the Z velocities at different nodes of the valve, plotted alongside the corresponding predictions made by the neural network. It describes how the velocity varies across the valve system.

(a) Z Velocity at node 1
(b) Z Velocity at node 4
(c) Z Velocity at node 7
(d) Z Velocity at node 2
(e) Z Velocity at node 5
(f) Z Velocity at node 8
(g) Z Velocity at node 3
(h) Z Velocity at node 6
(i) Z Velocity at node 9

Figure 4.11: Z Velocity Distribution at Various Nodes of the Valve (m/s)

Moving forward, the comparison and analysis of different neural network architectures, training and testing data will primarily concentrate on the results obtained from node 1 of the valve and the corresponding mass flow rate. This narrowed focus enables a more specific evaluation and discussion of the variations and discrepancies within the neural network models and the associated data sets.

Comparison between LSTM and FCN network architectures: In order to evaluate the accuracy of different neural network architectures for predicting valve movement, a comparative analysis was conducted. Two architectures were investigated: One comprising LSTM and Fully Connected layers, and another consisting solely of Fully Connected layers. To gain deeper insights into their performance, a visual approach was adopted. By plotting the predicted data generated by each architecture alongside the actual test data, the accuracy and effectiveness of each architecture could be directly assessed. This graphical representation allowed for a visual comparison of the predicted values with the ground truth, providing a clear understanding of how well each architecture captured the dynamics of valve movement. Through this analysis, the strengths and weaknesses of each architecture were unraveled, and meaningful conclusions regarding their suitability for valve prediction tasks were drawn.
Chapter 4. Experimental Results

4.2. Neural Network

(a) Network with only fully Connected Layers

(b) Network with LSTM and Fully Connected Layers

Figure 4.12: Structure of different Network Architectures

The architecture that includes LSTM layers offers several advantages. LSTM networks are specifically designed to capture long-term dependencies and temporal patterns in sequential data. They excel in handling time-series data by effectively modeling the relationships between past and future observations. The presence of LSTM layers allows the network to retain and propagate information over extended sequences, enabling it to learn complex temporal dynamics. This makes them well-suited for tasks that involve analyzing and predicting sequences with varying lengths and patterns.

On the other hand, the architecture with only Fully Connected layers offers simplicity and computational efficiency. Fully Connected layers connect every neuron from the previous layer to the subsequent layer without considering the sequential nature of the data explicitly. They are more straightforward in terms of implementation and require fewer computational resources compared to LSTM layers. This simplicity can be advantageous for tasks where the sequential patterns are less prominent or the temporal dependencies are not the primary focus.

However, there are some limitations to consider for both architectures. The architecture with LSTM layers tends to be more computationally demanding due to the complex operations involved in modeling temporal dependencies. Training and optimizing such networks may require more computational resources and time. In contrast, the architecture with only Fully Connected layers
Figure 4.14: Network trained with 8 m/s and 9 m/s and predicted for 8 m/s and 10 m/s

may struggle to capture long-term dependencies and complex sequential patterns effectively. It might not be able to fully leverage the inherently temporal nature of the data, leading to sub-optimal performance in tasks where sequence analysis is critical. Figure 4.13 illustrates the Z velocity at Node 1 of the valve system, showcasing the results obtained using different network architectures. The Z velocity represents the fluid flow magnitude along the Z-axis at the specific node.

By comparing the velocities obtained from various network architectures, the figure provides insights into the influence of different model configurations on the predicted Z velocity at Node 1.

Upon analyzing the image 4.12, it becomes evident that the architecture combining LSTM and Fully Connected layers yields higher accuracy compared to the architecture with only Fully Connected layers. This improvement can be attributed to the sequential nature of the data, where the movement of the valve follows a specific order over consecutive time steps. The LSTM layers in the network excel at capturing these sequential patterns, resulting in more accurate predictions.

Comparison between different training velocities: During the intermediate steps, the trained neural network model was tested against input velocities of 8 m/s and 11 m/s. However, the graph 4.14 below shows a poor fit between the predicted and actual data points, indicating a mismatch.

During the training phase, the neural network was trained using input data containing velocities of 8 m/s and 9 m/s. Subsequently, the trained network was employed to make predictions for velocities of 8 m/s and 10 m/s. Upon analyzing the latter half of the graph, which corresponds to the predictions for 10 m/s, it becomes evident that the accuracy of the predictions is compromised. This discrepancy in accuracy is expected since the trained model was not specifically trained on data with a velocity of 10 m/s. Consequently, the predictions for this out-of-scope velocity may not align well with the actual data points. This underscores the significance of training the network with a diverse range of input data to ensure accurate predictions across the entire spectrum of possible inputs.
Chapter 4. Experimental Results

4.2. Neural Network

(a) Node 1

(b) Mass Flow Rate

Figure 4.15: Network trained with 8 m/s and 11 m/s and predicted for 9 m/s and 10 m/s

To enhance the accuracy of predictions in this scenario, several approaches can be explored. Firstly, expanding the training dataset by including more data points with varying velocities, including those closer to the out-of-scope velocity of 10 m/s, can improve the model’s ability to generalize and handle a wider range of inputs. Secondly, fine-tuning the model architecture through experimentation with different neural network configurations, such as adjusting the number of layers, and hidden units, and incorporating regularization techniques or dropout, can capture more complex patterns in the data, leading to improved predictions. Thirdly, conducting hyperparameter tuning to optimize parameters like learning rate, batch size, and optimization algorithms can significantly impact the model’s convergence and performance.

To address this issue, further investigation is required to refine the network architecture and potentially incorporate additional techniques. Through the iterative process of refining the model based on the observed disparities, a more advanced neural network can be developed. This enhanced network will possess the capability to handle a broader range of velocities and provide more accurate predictions.
Chapter 5

Conclusion

5.1 Summary

This study had two main tasks: first, modeling and simulating the dynamic behavior and interaction of reed valves with the flow to investigate intricate dynamics between the valve and fluid flow, considering various boundary conditions and valve responses. Second, generating data through FSI simulations to develop a neural network model capable of accurately predicting the final valve position under different operating conditions. Both the tasks were successfully accomplished which enhanced the understanding of reed valve dynamics and associated fluid flow behavior, offering a valuable tool for performance optimization and design refinement.

This thesis has focused on the comprehensive analysis and modeling of a valve system using a combination of the FEM and CFD referred to as FSI. Factors such as the valve’s dimensions, shape, and material composition significantly impacted its performance in controlling fluid flow. The use of austenitic steel and 16MnCr5 alloys in the valve and end-stops are well-suited for the required mechanical properties. Understanding the interplay between geometry and materials is crucial for optimizing valve designs. The model’s geometry significantly influences mesh generation, especially for complex or intricate geometries with sharp edges. Careful considerations were made during the simulation setup to determine the type of mesh elements, element size, and overall mesh quality metrics.

The computational domain is essential for analysis and simulation. FEA divides structures into smaller elements to approximate solutions, making decisions on mesh element types and sizes for accurate analysis. The simulation setup included transient structural analysis, mesh generation, model setup, and defining boundary conditions, material properties, and contact interfaces in Ansys Transient Structural. A suitable element size of 0.3 mm was chosen, and the elements were generated linearly. Better results were observed in the valve mesh when it had more than one cell in height.

The governing equations of CFD, like the continuity equation, and conservation of momentum and energy, help understand fluid behavior and the need for suitable turbulence models to capture realistic flow characteristics. Therefore, the SST-K omega turbulence model was chosen due to its effectiveness in unconfined flows through curved geometries, handling strong adverse pressure gradients, and accurately resolving internal flows, making it ideal for the analysis of fluid dynamics. The selection of mesh types (structured or unstructured) and elements (triangles, quadrilaterals, tetrahedra, or hexahedra) significantly impacted the accuracy and efficiency of the simulation. For the purpose of this thesis, a tetrahedral element mesh was generated with a special emphasis on minimizing skewness as the mesh quality metric. The process of setting up a CFD simulation in Fluent focused on meshing, dynamic mesh, boundary conditions, and fluid material selection. The simulation used CO2 gas as the fluid material, and real-gas models were employed to account for non-ideal behavior. The settings of dynamic mesh, the importance of mesh quality, and its impact on the accuracy of the simulation were specifically highlighted. Various settings were explored for mesh smoothing during the simulations. If the mesh was not properly smoothed, it introduced
numerical errors and hindered the convergence of the simulation, resulting in unsuccessful outcomes. Similarly, inadequate re-meshing led to element distortion and excessive skewness in the mesh. After careful consideration, the Spring-Laplace boundary method was selected for mesh smoothing, and for remeshing, a cell skewness value of 0.95 and face skewness value of 0.9 were found to be suitable. These choices ensured a well-optimized mesh with improved simulation accuracy and stability, providing meaningful and reliable results for the analysis.

A two-way FSI simulated the dynamic interaction between fluid flow and the solid structure of the valve. This interaction considered how fluid flow affected the valve and the resulting impact of structural deformations on the fluid flow. The simulation accounted for all valve faces and fluid interactions, leading to stresses and deformations in the structure of the valve. The resulting valve movement and deformation affected flow passages, altering fluid pressure distribution and velocity field. To properly capture this bidirectional coupling, Ansys Workbench was used. The FSI algorithm utilized a partitioned approach, employing separate solvers for fluid and structural sub-problems. The final analysis was conducted with an end time of 0.01 s and a time step size of 1e-05 s.

The effects of different boundary conditions on the system’s behavior were explored. In the first scenario, the valve’s behavior was analyzed at an outlet pressure of 60 bar and an inlet velocity of 10 m/s. The valve reached its maximum position of 0.84 mm at 0.351 ms. Then, the final deformation of the valve was observed at an outlet pressure of 120 bar. Subsequently, the boundary condition was changed to varying inlet pressure, resulting in the valve opening completely and much faster compared to the constant inlet velocity condition. When comparing simulations with different velocities but the same outlet pressure of 120 bar, it was noticed that the valve’s behavior became unstable at a high inlet velocity of 25 m/s, with a twist occurring at a timestep of 0.18 ms. As expected, the valve exhibited the highest movement at the highest inlet velocity of 25 m/s.

The neural network was trained on various fluid properties (density, viscosity) and boundary conditions (inlet velocity, inlet pressure, total pressure, static inlet temperature), along with z-coordinates of nine selected valve nodes as inputs. The main objective was to predict the final velocities of the selected nodes and mass flow rate of these nine nodes in the valve system. The network was initially trained with inlet velocities of 8 m/s and 10 m/s. Later, it was used to predict velocities at 9 m/s and 10 m/s using 2000 data points. Two learning rates were experimented with during network training, and a learning rate of 0.01 was selected for its superior performance and accuracy. The two network architectures, one with LSTM and Fully Connected layers and the other with only Fully Connected layers, were compared. The LSTM-based architecture showed higher accuracy due to its ability to capture sequential patterns in the data. However, the Fully Connected architecture demonstrated simplicity and computational efficiency. The importance of training the model with diverse data to ensure accurate predictions across different input velocities was emphasized. Despite achieving good accuracy for input velocities within the training range, the network’s performance was observed to degrade when predicting for out-of-scope velocities, such as 12 m/s. Several approaches were suggested to address this issue, including expanding the training dataset, fine-tuning the model architecture, and conducting hyperparameter tuning.

Valuable insights into fluid flow characteristics and system behavior were obtained. To use these FSI data in a wider range, stand-alone or within a 1D simulation model, AI techniques, particularly neural networks, were explored to use FSI data in wider applications, reducing computational costs while maintaining accuracy. The analysis provided insights into the neural network’s behavior and limitations, guiding improvements for real-world reed valve dynamics prediction.

In conclusion, this study successfully integrated computational modeling, simulation, and neural networks to gain a comprehensive understanding of reed valve dynamics and enhance predictive capabilities. The findings contribute to the field’s knowledge by providing insights into fluid flow behaviors, pressure distributions, and structural responses within the valve system. These insights are valuable for optimizing design and enhancing performance. Additionally, the incorporation of neural networks demonstrates the potential to improve computational efficiency while maintaining accuracy.
5.2 Limitations and Future Work

While this thesis presents significant contributions and findings, it is essential to acknowledge the presence of several limitations. Firstly, the simulations and models developed in this study are based on simplifying assumptions and idealizations, which may not fully capture the complexities of real-world valve systems. CFD, FEA, and FSI simulations often demand substantial computational power and time, particularly when dealing with complex geometries and detailed meshing. Limited computing resources can restrict the size and complexity of the simulations that can be performed. Additionally, the accuracy of the simulations relies on the validity of the material properties assigned to the valve components, and variations or uncertainties in these properties could affect the results. The choice of boundary conditions also has a significant impact, and the specific selection of pressure and velocity values, as well as assumptions made regarding inflow and outflow conditions, may introduce uncertainties.

Both CFD and FEA simulations encountered recurrent convergence issues due to the complexity of geometries, mesh quality problems, and improper boundary conditions, demanding significant time and effort to resolve. Floating-point exceptions and negative volume errors were also common occurrences. Inappropriate mesh sizes led to prolonged simulation times and an unnecessarily high number of elements. Striking the right balance was challenging, as a larger time step caused unconverged solutions, while a smaller time step needlessly prolonged the run time. Dynamic mesh settings in Fluent are highly intricate, as even minor changes can significantly impact valve behavior. For instance, using a criterion 'cell skewness' of 0.85 during remeshing resulted in a final skewness of 0.93, leading to a negative volume error in the simulation. Similarly, selecting the wrong smoothing parameter caused unexpected valve deformation and closure. Such complexities require careful consideration and fine-tuning to achieve accurate and reliable results in dynamic simulations. Due to budget constraints, experimental validation of the CFD results was not pursued in this master thesis. Instead, benchmarking with similar studies was considered more suitable for validation due to its feasibility and cost-effectiveness.

The AI approach using a data-driven neural network model has shown promise in the context of valve system analysis and modeling. However, it is essential to acknowledge and address certain limitations that affect its generalization capability and overall reliability. One significant limitation of the data-driven neural network model is its struggle to predict responses beyond the trained boundary conditions. This means that the model may perform well within the range of conditions it was trained on, but its predictions may become less accurate or even unreliable when faced with scenarios outside this range, as evidenced in the section 4.2.1. This limitation is attributed to the neural network model’s insufficient exposure to diverse and extreme conditions during training, heavily depending on the quality and diversity of the training dataset. If the training data is biased, incomplete, or lacks representation of certain critical scenarios, the model may exhibit biased behavior and produce inaccurate predictions. Additionally, noise and outliers in the data can adversely affect the accuracy and robustness of the model’s predictions, making it less reliable in such practical applications. As valve systems often involve intricate fluid dynamics and structural interactions, the simulations can be computationally demanding. Adequate computational resources are necessary to ensure that the neural network model can effectively handle such complexities and provide meaningful insights. It is important to note that the findings of the neural network model are specific to the studied valve system.

Further research can explore different aspects, such as boundary conditions, material properties, and geometric configurations on the valve system’s performance. Advanced AI techniques, like deep learning algorithms and hybrid modeling, can enhance simulation accuracy and efficiency. To overcome these above-mentioned limitations and improve the analysis and modeling of valve systems, future research should focus on several areas. First, efforts should be directed towards improving the generalization capability of the neural network model by devising techniques to handle scenarios beyond the trained boundaries effectively. This might involve data augmentation techniques or incorporating domain knowledge during training. To mitigate the impact of biased or incomplete data, strategies for collecting diverse and representative datasets should be explored. Collaborations with industry partners, settings up test-benches or accessing data from various sources could help
in creating more comprehensive and unbiased datasets. Noise and outlier handling mechanisms should be developed to enhance the model’s robustness against noisy data. Robust neural network architectures or statistical methods can help filter out irrelevant noise and improve the model’s accuracy. Lastly, the neural network model should be validated under a wide range of conditions to ensure its reliability and applicability across different scenarios. Comparing the model’s predictions with experimental data and other established simulation methods will help establish its credibility.

In valve system analysis and modeling, a promising alternative machine learning method to overcome the limitations of data-driven neural network models is Physics-Based Modeling. Physics-Based Modeling incorporates known physical principles and governing equations, enhancing generalization and reliability beyond trained data. It provides valuable insights into system dynamics and interactions, making it robust and applicable even with limited or biased data. Another option is to use hybrid models, which combine data-driven machine learning techniques with physics-based modeling, leveraging the strengths of both approaches to achieve accurate predictions while considering the underlying physical behavior of the system. Bayesian optimization (BO) with Gaussian process regression (GPR) is another alternative that has been successfully applied to various computational fluid dynamics (CFD) problems. BO efficiently searches for optimal configurations with fewer simulations, reducing computational costs. GPR serves as a surrogate model, capturing system behavior and providing uncertainty estimates. This approach has shown promise in optimizing complex CFD systems with limited data, handling noise and uncertainty, and efficiently exploring large search spaces. BO with GPR offers an effective method for accelerating CFD optimization processes, improving efficiency and performance. However, they may not entirely replace traditional neural networks, and a combination of both approaches can yield the best results depending on the problem and trade-offs between data-driven learning and explicit physics incorporation.
Appendix A

MATLAB code

clear all
close all
clc
format long g

%% Boundary conditions
training_pressure=6000000;
training_velocity_1=8;
training_velocity_2=9;

test_pressure=6000000;
test_velocity_1=8;
test_velocity_2=10;

%% Training data
RV_training = read_ANSYS('input_files', '03_60bar_8_ms_', '.txt');

results = struct2table(RV_training.vectors);
results = removevars(results, {'density_outlet',
    'outlet_flow_report_def_0_outlet', 'static_temp_outlet_outlet','
    tot_press_outlet','velocity_inlet','velocity_outlet'});

%% Reading training data 2, converting it into a table and removing
unnecessary values
RV_test = read_ANSYS('input_files', '03_60bar_9_ms_', '.txt');

results1 = struct2table(RV_test.vectors);
results1 = removevars(results1, {'density_outlet',
    'outlet_flow_report_def_0_outlet', 'static_temp_outlet_outlet','
    tot_press_outlet','velocity_inlet','velocity_outlet'});

%% Create vectors for the two input velocities
input_velocity = (1:height(results))' * 0.0002 + training_velocity_1;
results = [results(:,1:2) array2table(input_velocity) results(:,3:end)];

input_velocity = (1:height(results))*0.0002 + training_velocity_2;
results1 = [results1(:,1:2) array2table(input_velocity) results1(:,3:end)];

training_results=vertcat(results,results1); %results ; % concatenating them to create the training dataset

%%

cols=width(training_results);
% 'n' is the number of input parameters.

n=19;
m=cols - n;

% Training input and output data
train_input_params=training_results(:,1:n);
train_output_params=training_results(:,n+1:end);

% Validation input and output data
valid_input_params=training_results(:,1:n);
valid_output_params=training_results(:,n+1:end);

%%

% Normalize generalized training and validation data
% Mean and std value for training input data
for i=1:n
    arr=train_input_params(:,i);
    train_x_mean(i)=mean(arr);
    train_x_std(i)=std(arr);
end

% Normalized training input data
for i=1:n
    train_xl(:,i) = (train_input_params(:,i) - train_x_mean(i))./ train_x_std(i);
end

%%

% Mean and std value for training output data
for i=1:m
    arr1=train_output_params(:,i);
    train_y_mean(i)=mean(arr1);
    train_y_std(i)=std(arr1);
end

% Normalized training output data
for i=1:m
    train_yl(:,i) = (train_output_params(:,i) - train_y_mean(i))./ train_y_std(i);
end

%%

% Training valid data
for i=1:n
Appendix A. MATLAB code

```
arr = valid_input_params(:,i);
valid_x_mean(i)=mean(arr);
valid_x_std(i)=std(arr);
end

for i=1:n
    valid_xl(:,i) = (valid_input_params(:,i) - valid_x_mean(i))./valid_x_std(i);
end

% Valid output data
for i=1:m
    arr1=valid_output_params(:,i);
    valid_y_mean(i)=mean(arr1);
    valid_y_std(i)=std(arr1);
end

for i=1:m
    valid_yl(:,i) = (valid_output_params(:,i) - valid_y_mean(i))./valid_y_std(i);
end

%%
%% define neural regression network
numHiddenUnits=1000;
layers = [featureInputLayer(n)
        fullyConnectedLayer(n)
        tanhLayer('Name', 'tanh1')
      lstmLayer(numHiddenUnits)
        tanhLayer('Name', 'tanh2')
      lstmLayer(numHiddenUnits)
        tanhLayer('Name', 'tanh3')
      lstmLayer(numHiddenUnits)
        tanhLayer('Name', 'tanh4')
        fullyConnectedLayer(n)
        tanhLayer('Name', 'tanh2')
        fullyConnectedLayer(m)
        regressionLayer];
maxEpochs = 800;
miniBatchSize = 80;

options = trainingOptions('adam', ...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'InitialLearnRate',0.01, ...
    'GradientThreshold',1, ...
    'Shuffle','every-epoch', ...
    'Plots','training-progress',...
    'ValidationData',{[valid_xl], [valid_yl]}, ...
    'ValidationFrequency',100,...
    'VerboseFrequency',100,...
    'Verbose',1);

net = trainNetwork([train_xl],[train_yl], layers,options);

%%
plot(net)
```
%% Now reading the test data, converting it into a table and removing unnecessary values
RV_training = read_ANSYS('input_files', '02_60bar_8_ms', '.txt');
results_test = struct2table(RV_training.vectors);
results_test = removevars(results_test, {'density_outlet',
  'outlet_flow_report_def_0_outlet', 'static_temp_outlet_outlet',
  'tot_press_outlet', 'velocity_inlet', 'velocity_outlet'});

RV_test = read_ANSYS('input_files', '03_60bar_10_ms', '.txt');
results1_test = struct2table(RV_test.vectors);
results1_test = removevars(results1_test, {'density_outlet',
  'outlet_flow_report_def_0_outlet', 'static_temp_outlet_outlet'});
results1_test = removevars(results1_test, {'tot_press_outlet',
  'velocity_inlet', 'velocity_outlet'});

%% Creating vectors for the input test velocities
input_velocity = (1:height(results_test))*0.0002 + test_velocity_1;
results_test = [results_test(:,1:2) array2table(input_velocity)
  results_test(:,3:end)];
input_velocity = (1:height(results1_test))*0.0002 + test_velocity_2;
results1_test = [results1_test(:,1:2) array2table(input_velocity)
  results1_test(:,3:end)];
results1_test = removevars(results1_test, {'tot_press_outlet',
  'velocity_inlet', 'velocity_outlet'});

test_results = vertcat(results_test, results1_test); % concatenating to create the testing dataset

%% Saving test input and output data
test_input_params = test_results(:,1:n);
test_output_params = test_results(:,n+1:end);

%% Normalizing the test data
test_xl = (test_input_params - train_x_mean)./train_x_std;
test_yl = (test_output_params - train_y_mean)./train_y_std;

%% tic
tic
YP = net.predict([test_xl]);
YP_2 = array2table(YP);
YP = YP.*train_y_std+train_y_mean;
YP_1 = (YP);
toc
%%

loss_log = loss();

%% for i=1:m
fig = figure('Name', train_output_params.Properties.VariableNames{i});
plot(test_output_params(:,i), 'r-');
hold on;
grid on;
plot(YP(:,i),'b-');
y_label = train_output_params.Properties.VariableNames{i};
Appendix A. MATLAB code

ylabel(y_label, 'interpreter', 'none')
xlabel('Time steps')
legend('test data','predicted data');
title(['Prediction of output number ',num2str(i)]);
axis auto;

directory = './8&9_8&10/';
filename = sprintf('%s.png',y_label);

% Generate the full file path
fullPath = fullfile(directory, filename);

% Save the figure as a PNG file
saveas(fig, fullPath, 'png');
end

%% Now the results and the plots are ready to be interpreted and used as required.
Bibliography


