POLITECNICO DI TORINO

**Master's Degree in Aerospace Engineering**


## Degree Thesis

# Development and validation physical-guided data science models for finite element applications

**Supervisors**

Prof. Maria Chierichetti
Prof. Alfonso Pagani

**Candidate**

Andrea Rotondo
Matr. 290785

Academic year 2022-2023

# Acknowledgements

First and foremost, I wish to express my deep gratitude to Professor Chierichetti for this magnificent experience. Her wise guidance and support have made the completion of this thesis possible. Her commitment and dedication to my education have been an endless source of inspiration for my current and future academic journey.

I feel compelled to extend these thanks to San Jose State University and the Department of Aerospace Engineering for allowing me to live and utilize their laboratories during these months.

My gratitude also extends to my supervisor at the Politecnico di Torino, Professor Pagani. His attentiveness and precision have been invaluable, and his trust has played a fundamental role in shaping my educational path.

Special thanks are due to all the guys at the SJSU International House who, each day, made me feel less alone in this American adventure.

.

# Abstract

Real-time stress predictions require the continuous interaction of measurements and a complex, refined finite element model that is updated real-time based on measured conditions. Previous work has shown that this interaction can be based on simplified physical relationships that allow quick changes in the model based on actual conditions. However, even when low-fidelity models are used in conjunction with experimental measurements, the resulting computational times are still too large to be feasible with real-time predictions. This complexity has therefore restricted the ability to efficiently monitor the stresses in complex systems during operations without continuous experimental monitoring.

Data science, with its ability to extract "knowledge" from large volumes of data, has the potential to be used to predict transient stresses that a structure experiences at any given time. The objective of this thesis is to create a novel interactive framework for combining scientific knowledge of finite element methods (FEM) in mechanical systems with data science methods to predict the full-field dynamic behavior of the system. The thesis will focus on developing data-driven models based on finite element models and real-time experimental data to create mid-fidelity surrogate models that can learn the structural behavior from rich finite element simulations and predict the dynamic behavior of a system based on actual measurements. The data-driven model bridges the need of accuracy given by high-fidelity finite element simulations, of low computational cost provided by low-fidelity models and of real-time stress predictions based on actual measurements. The data-driven model will provide quick and reliable predictions of the stresses and accelerations in the presence of highly non-linear, transient response and in the presence of complex couplings. At the end of this thesis, the following goals will be attained:

- development of data-driven model to determine time-varying stresses in structural components
- definition of a database using finite elements of simple components (such as a beam) and complex components (such as plate with hole)
- numerical validation of the approach
- experimental validation doing vibration testing of simple components and complex components.

# Summary

# 1 Overview

## 1.1 Introduction

Data science models have revolutionized numerous fields, including the Finite Element Method (FEM), which represents a fundamental tool in the analysis and solution of complex engineering problems [1]. Applying machine learning to FEM methods offers an innovative and promising perspective for enhancing the efficiency and accuracy of calculations and simulations in the field of engineering.

The Finite Element Method (FEM) represents a powerful and widely used numerical technique for solving engineering and scientific problems. [2] This method provides an approximate solution to partial differential equations (PDEs) or integral equations by discretizing the domain of interest into finite elements. Its flexibility and ability to handle complex geometries and variable materials make it an essential analysis technique for many sectors, from structural component design to simulating complex physical phenomena.

FEM is particularly suitable for modeling complex problems [3] where geometry and material properties can vary locally. The domain is divided into a series of finite elements, which can be as simple as triangles or quadrilaterals in two dimensions, or tetrahedra or prisms in three dimensions. Each element is defined by a set of nodal points connected by local interpolation functions.

The approximate solution is represented as a linear combination of these local interpolation functions, with coefficients associated with each function. The values of these coefficients are determined by solving a system of linear equations obtained by applying boundary conditions and partial differential equations.

One of the main advantages of the FEM is its flexibility in handling complex geometries [4] and variable material properties. Additionally, the FEM can address various types of problems, such as structural statics and dynamics, heat conduction, fluid dynamics, electromagnetism, and many others. This has made the FEM a fundamental tool for engineers and scientists in designing and analyzing a wide range of systems and components.

However, it's essential to note that the accuracy of results obtained using the FEM depends on the proper choice of finite element type, domain discretization, and the quality of the generated mesh.

Furthermore, the approximate solution can be influenced by the numerical resolution used to solve the system of linear equations.

FEM methods are widely used to model the behavior of complex structures, such as bridges, buildings, vehicles, and industrial facilities. These methods require the division of the problem domain into finite elements to approximate the system's behavior discretely. However, the accuracy of the solutions obtained heavily relies on the mesh quality, i.e., the division of the domain into finite elements. Generating an optimal mesh is a complex task and demands a substantial amount of time and computational resources.

Applying machine learning to FEM methods provides a promising solution to this challenge. Thanks to their ability to learn from large amounts of data and recognize complex patterns, machine learning algorithms can be trained to predict and automatically optimize the distribution of finite elements in the mesh. This allows for more accurate models while significantly reducing the time and effort required for mesh generation.

In this thesis, we will delve into the methodologies, techniques, and applications of machine learning applied to FEM methods. We will analyze the most used machine learning algorithms, such as artificial neural networks and regression algorithms, and discuss strategies for acquiring, preparing, and processing the data necessary for model training. Initially, we will apply these models to a small aluminum beam and subsequently to a steel plate with a hole. Additionally, we will examine the impacts and challenges associated with implementing machine learning in the context of engineering, discussing aspects such as model interpretability, adequacy of input data, and the validation of obtained solutions.

Through the study of case studies and the practical application of finite element method (FEM) machine learning algorithms, this thesis aims to provide an in-depth overview of the capabilities and limitations of this emerging discipline. It is hoped that this work can contribute to the development and application of machine learning in FEM methodologies, opening new opportunities to enhance the efficiency and accuracy of FEM methods in the field of engineering.

The state of the art in the application of machine learning to Finite Element Methods (FEM) is continually evolving and witnessing numerous exciting developments and applications. Machine learning has demonstrated its ability to make significant contributions in various aspects related to FEM methods.

Here are some of the key points in the state of the art in applying machine learning to FEM methods:

- Material structure learning: Machine learning can be used to model and predict the properties of materials used in FEM simulations [5], enabling a better understanding of their behavior and performance.
- Computational cost reduction: A highly interesting area is the reduction of computational costs associated with FEM simulations [6]. Machine learning can be employed to expedite calculations or develop reduced models that provide approximate results in shorter times.
- Design optimization: Machine learning can be applied to optimize the design of components or structures [7], allowing for the discovery of superior and more efficient solutions in terms of performance, strength, or weight reduction [8].
- Noise and vibration modeling: Machine learning can be used to model and predict the noise and vibrations generated by a system [9], enabling better design and optimization of acoustic performance.
- Data analysis and prediction: Machine learning can be employed to analyze large quantities of data generated by FEM simulations, extract useful information, identify patterns, and predict the future behavior of analyzed systems.

These are just some of the areas where machine learning is finding applications in FEM methods. It is important to note that the state of the art is continuously evolving, and new techniques and methodologies continue to emerge, offering new possibilities and challenges in the field of analysis and optimization of engineering systems.

## 1.2 Literature review

### 1.2.1 Damage Detection in the Field of Civil Structures

Currently, one of the fields where machine learning algorithms have been applied to structures is civil engineering. Some studies have focused on structural damage detection based on real-time vibration signals and convolutional neural networks, which is an advanced approach used to assess the structural health of buildings, bridges, industrial structures, and other infrastructure. This method utilizes vibration data acquired from sensors placed on the structure to identify any damages or anomalies. On the other hand, another study has concentrated on the use of artificial

neural networks for real-time prediction of structural stress through structural vibration tests. Similar to the first study, this method relies on vibration data collected from sensors positioned on the structure to estimate the stress it undergoes.

One of the objectives was to propose an alternative to visual inspection methods, which required a significant amount of time to detect surface damages. Therefore, a solution was sought through the adoption of an SSD method based on vibrations [10], using a three-dimensional steel beam as a structural model. Initially, consideration was given to using natural frequencies as an indicator of damage. However, later [11], it was deemed preferable to use modal shapes, as they could provide better results than the previous indicator. Nevertheless, there were limitations due to the influence of the measurement environment.

As a result, the direct use of vibration signals was considered a better option, integrated with values obtained from finite element analyses on the structure. This integration allows for more accurate detection of structural damage since it contains more information. However, this combination requires more signal processing.

Several classical machine learning algorithms, such as SVM, DT, and ANN, can be utilized. Choosing among them was challenging due to each having its own pros and cons. SVM (Support Vector Machine) is a classification algorithm that seeks to find the best separator to distinguish different data classes. SVM has good generalization ability and can handle nonlinear data using the kernel trick. However, in complex situations and with large-scale samples, SVMs were found to be difficult to implement. DT (Decision Tree), another classification and regression algorithm based on a decision tree structure, requires minimal data preparation and can handle both categorical and numerical data. Similarly, artificial neural networks (ANNs), as advanced tools for data analysis, can automatically extract information about structural damages from signals and represent this information through mapping structural damage states.

In their study, the researchers did not adopt ANNs as a resolution model due to limitations in engineering applications, as ANNs adapt too easily and require high resolution time. Instead, convolutional neural networks (CNNs) were chosen, not individually, as it has not been established whether a CNN trained with only numerical data can detect damages in a real structure. The idea was to associate CNNs with finite element analysis (FEA) methods, training them with samples obtained from FEA experiments. This allowed obtaining vibration and acceleration data from

numerical simulations, resulting in four numerical datasets, each corresponding to a different number of damages in beams and excitations at four points repeated five times.

Through numerical simulations, vibration and acceleration signals were obtained and used for training and testing under various damage scenarios. The CNN architecture was based on a steel frame and the positions of acceleration measurement points. Using the appropriate ANN, a sufficiently accurate resolution was achieved, but due to the extensive data processed, model performance was optimized by training with the maximum number of frequency steps possible.

In a similar context, CNNs were not employed using the FEA method. Instead, various ANNs were used to predict system responses during the test. Accelerations were converted into structural stresses, and mass operators were developed using the ANN, providing sufficiently accurate results but requiring substantial data processing.

As in the previous cases and in the ongoing thesis, acceleration, stress, and load values were developed using the finite element method with a MATLAB code. Two structures were analyzed: NIRSpec's ceramic bench and the elements it supported. While a considerable amount of data was used, only frequencies near the peaks of the direct and uncertain response were considered. Regardless of the method used, it had to meet criteria for robustness, rapid implementation, accuracy for interface stresses, training, and quick acquisition.

Four ANN models were compared: a frequency-dependent ANN, a pre-trained ANN, a nonlinear autoregressive exogenous (NARX) model, and a recurrent ANN with a bidirectional long short-term memory layer (biLSTM). After conducting the analyses, it was observed that only the biLSTM method failed to predict adequately, unlike the other three, which were more accurate. The NARX method was identified as the best for reducing the prediction time for structural data and for use during vibration tests but was negatively sensitive to frequency step division.

The use of structure vibration data to monitor structural health was mainly applied in a civil engineering study, analyzing a steel structure with one and four stories, both with bolted joints. The hybrid combination of the previously mentioned methods aimed to monitor damages in bolted joint locations, requiring fewer datasets than individual techniques. Statistical values and properties useful for bolt loosening localization effectiveness were analyzed.

Like previous studies, a major issue with machine learning was the need for extensive data, leading to longer simulations and difficulty in obtaining sufficient data for civil structures. Consequently, a

mix of methods was used: machine learning with vibration data for approximate localization and model updating to identify the exact damage location.

Strain gauges provided data, divided into healthy (sensors away from joints) and loosened (sensors near joints) sets for comparison. Unlike previous studies using ANNs, a Support Vector Machine (SVM) algorithm was employed for machine learning. SVM's primary objective is to create a model that can categorize a dataset into different classes by maximizing the distance between data points belonging to different classes. After SVM model training, it achieved an accuracy of approximately 80%.

Stochastic free vibrations were also analyzed using machine learning for functionally graded (FG) bar-type structures through the finite element method (FEM). In contrast to the SVM model used previously, an X-SVR technique was employed to estimate the governing relationship between uncertain system parameters and structural natural frequencies.

### 1.2.2 Differences between SVM and SVR algorithms and their combined approach in structural analysis

SVM (Support Vector Machine) and SVR (Support Vector Regression) are both machine learning algorithms based on Support Vector Machines, but they have slightly different purposes and applications. SVM is primarily used for classification problems [16], aiming to find the optimal hyperplane that can separate data points of different classes in the best possible way. It produces a model that assigns data points to one of the predetermined classes. In contrast, SVR is used for regression problems [17], seeking the optimal hyperplane to approximate data points in the best possible way. It produces a model that can be used to estimate numerical values based on input data. In both cases, SVM and SVR rely on the concept of support vectors, critical data points defining the decision boundary or regression function. Both algorithms aim to maximize the distance between support vectors to achieve generalized models that perform well with new data. SVM focuses on classification, while SVR focuses on regression.

The X-SVR was employed, where the addition of the prefix "X" indicates that the algorithm has been extended or modified from the standard SVR implementation [18]. In this case, the linear X-SVR uses a linear kernel instead of a more complex kernel like the polynomial or Gaussian kernels used in other SVR variants. The linear kernel calculates the dot product between two feature vectors representing independent variables without applying any transformation. The linear X-SVR can be advantageous when the relationship between independent and dependent variables is

approximately linear. Using a linear kernel, the X-SVR can be computationally more efficient than other SVR variants using more complex kernels. However, if the relationship between variables is not linear, the linear X-SVR may not accurately model the data, and a different kernel or regression technique may be needed.

Subsequently, after using the previous regression, a Monte Carlo simulation (MCS) is applied to estimate various statistical characteristics of structural natural frequencies with significantly reduced times. Moreover, as in previous cases, this theoretically allows for the extension to a structural health monitoring algorithm during the structure's operational life.

In another case [19], a CNN-based approach is adopted, but unlike case [10], it is not combined with other algorithms. Both studies aimed to derive the localization and quantification of structural damage. The studied systems include a structural beam and a mass-spring system. The approach operates on images generated from raw transmissibility functions of the structures to derive degradation process characteristics. The main advantage over previous investigations is that this approach automates feature extraction.

Raw transmissibility functions are tools used in vibration analysis to measure and describe the behavior of a dynamic system subjected to vibrations. Essentially, raw transmissibility is a measure of the relationship between the output amplitude of a system and the input amplitude at a specific frequency. This ratio is often expressed in terms of acceleration, velocity, or displacement. It proves highly effective when damage exceeds 10%, becoming a more accurate approach.

Structural state damage was monitored through the application of deep learning and FEM methods on an aluminum beam with an applied crack [20]. As in similar studies, after obtaining a database through FEM methods, the training in deep learning follows. The study's goal is to provide adequate training to ANNs, enabling the use of SHM in a real damage scenario. To achieve accurate results, multiple experimental trials are necessary. Maximum uncertainties are simulated until the point where RNA decreases its accuracy. Finally, the data is passed to CNNs trained by FEM to predict possible real states of damage.

### 1.2.3 Improving Finite Element Methods through Machine Learning

As seen in previous studies, including the one developed in this work, the application of the Finite Element Method (FEM) was necessary to obtain the results needed for training neural networks. To enhance the finite element model, it is imperative to address the challenge of the extensive time

required to compute the problem [6]. The goal was to find solutions that could improve optimization issues. Specifically, the objective was to reduce simulation times and enhance robustness in selecting the updated model using a novel algorithm that incorporates a set of algorithms, including two optimization algorithms, a machine learning one (ANN), and a statistical technique.

Finally, machine learning, particularly employing Convolutional Neural Networks (CNNs) [21], was utilized to expedite the prediction process of results obtained through the FEM method, ultimately achieving a prediction of mesh deformation 2,960,000 times faster than the initial case.

To reduce simulation times for the FEM model update process, the current trend focuses on the use of collaborative or hybrid computational intelligence algorithms. These algorithms leverage the key strengths of two or more computational intelligence algorithms, combining them into an overall new algorithm that demonstrates improved performance compared to individual algorithms.

### 1.2.4 Machine Learning Applied to Aerostructures

Machine learning has found applications in the aerospace field, particularly at the structural level, where initially maintenance intervals were predetermined based on the expected life of the system. To address this, the introduction of random forests and artificial neural networks was proposed to predict structural stresses [22], aiming to estimate more efficient maintenance schedules. Specifically, a one-dimensional structure, such as a beam, was examined, where the real-time average stress distribution was predicted after learning from data obtained through FEA simulations on beam stresses. This approach seeks sufficiently accurate solutions in various positions. The method is characterized by its considerable versatility, allowing for the use of a wide range of response variables (such as displacement, velocity, acceleration, deformation, and stress) for training and prediction purposes [23]. Although the proposed algorithm [11] primarily focuses on accelerations, which are directly correlated with stresses.

Furthermore, for detecting acceleration data on the structure, sensors are used, and machine learning has also been applied to determine the optimal sensor placement for monitoring a vibrating system, thereby detecting the structural health status [24]. This methodology can be applied to any sensor and dynamic system to enhance precision and reduce the number of redundant sensors.

## 1.3 Organization of the Document

After providing a brief introduction to the topic of this thesis, the first chapter focused on the literature review. The second chapter will delve into machine learning algorithms and the architecture of the utilized neural network. The third chapter will outline the tools employed for signal acquisition and experiment execution. In the fourth chapter, the data acquisition process and the utilized input signals will be detailed. The fifth chapter will cover the beam experiment and various comparisons, while the same procedure will be conducted in the sixth chapter with a different beam. Finally, the last chapter will be dedicated to conclusions and the obtained results.

# 2 Neural Networks

## 2.1 introduction

Artificial neural networks represent one of the most significant innovations in the fields of artificial intelligence and machine learning. To fully grasp the concept, it is essential to begin with the history and inspiration that led to their creation. They draw inspiration from the human brain, the most sophisticated information processing system known.

In the realm of artificial intelligence, neural networks constitute a fascinating class of models and algorithms inspired by the functioning of the human brain. These networks are composed of interconnected units known as artificial neurons, which collaborate to process complex information.

A distinctive aspect of neural networks is their intrinsic nonlinearity, achieved using nonlinear activation functions. This feature enables them to model intricate relationships between inputs and outputs, making them powerful tools for a wide range of tasks, from image processing to natural language recognition. Furthermore, the concept of neurons as common building blocks in neural networks promotes the sharing of theories and learning algorithms across various application areas. This uniformity streamlines the development and implementation of neural networks in diverse contexts.

The process of training neural networks relies on mapping inputs to outputs through a sequence of training data, an approach known as supervised learning. It does not make prior assumptions about the model's parameters, allowing neural networks to adapt to the presented data.

A key feature is the adaptability of neural networks. They can adjust their weights and connections in response to changes in the surrounding environment. It is possible to train them to operate in different contexts or adapt to real-time variations.

Due to their structure, neural networks can be implemented at the hardware level for high-speed computational tasks. Their intrinsic parallelism makes them suitable for applications demanding substantial computing power.

Moreover, hardware neural networks exhibit good fault tolerance. Performance gradually degrades in case of malfunctions, ensuring increased robustness in data processing.

Ultimately, the analogy with the human brain has motivated the design of neural networks. This analogy demonstrates that parallel computing, fault tolerance, and efficiency are not only possible

but also potent, paving the way for new and exciting opportunities in the field of artificial intelligence.

## 2.2 Historical Overview

Since ancient times, humans have sought to understand the functioning of the brain. In fact, Hippocrates made early attempts to study the human brain by trying to identify the locations of certain control areas, both motor and sensory, within the brain. However, it was only in the 20th century that significant progress occurred.

In 1936, the British mathematician Alan Turing proposed an analogy between the human brain and a computer. This idea alluded to the fundamental concept that a universal machine could be programmed to emulate any other computing machine, including the human brain. This concept of a universal machine, known as the "Turing machine," was a crucial step in the evolution of information theory and theoretical computer science, providing the conceptual foundations for computation and artificial intelligence. Turing's analogous proposal was a milestone in understanding the potential symbiosis between the human mind and computing machines, paving the way for AI and artificial neural networks.

One of the earliest significant steps toward creating neural networks was taken by Warren McCulloch and Walter Pitts in 1943. They proposed a mathematical model of an artificial neuron called the "McCulloch-Pitts neuron," which represented a simplification of the functioning of biological neurons. In fact, they reproduced a simple neural network using interconnected electrical circuits based on considerations about the operation of individual neurons and demonstrated that neural networks are analogous to a Turing machine, meaning that any operation performed by a neural network could also be executed by a computer.

Frank Rosenblatt developed the "perceptron" in the 1950s and 1960s, which marked another step forward in the evolution of artificial neural networks. The perceptron was capable of learning automatically from training data and performing binary classification, paving the way for the use of neural networks in pattern recognition. However, it soon became evident that the perceptron had significant limitations, as it could only handle linearly separable problems. This led to a period of skepticism toward artificial neural networks known as the "winter of artificial intelligence" in the 1970s and 1980s.

Neural networks experienced a resurgence starting in the 1990s, thanks to several key developments such as learning algorithms, increased computational power, large datasets, and advanced neural architectures.

Today, artificial neural networks underpin many applications of artificial intelligence, including speech recognition, automatic translation, image recognition, autonomous driving, and much more. Their development and use continue to evolve, paving the way for new opportunities and challenges in the field of AI.

## 2.3 Neurons

An artificial neural network is a machine designed to simulate the functioning of the human brain, implemented either physically using electronic components or simulated through software on digital computers (Haykin, 1999). Neural networks are composed of simple elements (neurons, nodes, units) that play a crucial role in the information processing process.

Information and signals will flow among these neurons through the connections. These connections are weighted to regulate the flow of information. It's as if each connection holds a different level of importance in determining the outcome of the network. This concept of weighting connections is fundamental to the adaptation of the neural network to training data and the learning process.

Information, in the form of weighted signals, accumulates within the neurons. The central body of the neuron, known as the nucleus, sums the input signals from synapses connected to the dendrites of other neurons. When this sum of signals reaches a threshold, the neuron generates an output signal that is transmitted to other neurons. This process is known as "firing" of the neuron and is a crucial step in information processing within a neural network.

Artificial neural networks draw inspiration from the functioning of biological neurons in the human brain. They use artificial neurons, weighted connections, and activation functions to process information in a complex manner and perform a variety of tasks, from classification to image processing. The process of summing signals and generating output by neurons is analogous to the "firing" in biological neurons and constitutes a key element in the ability of neural networks to learn from data.

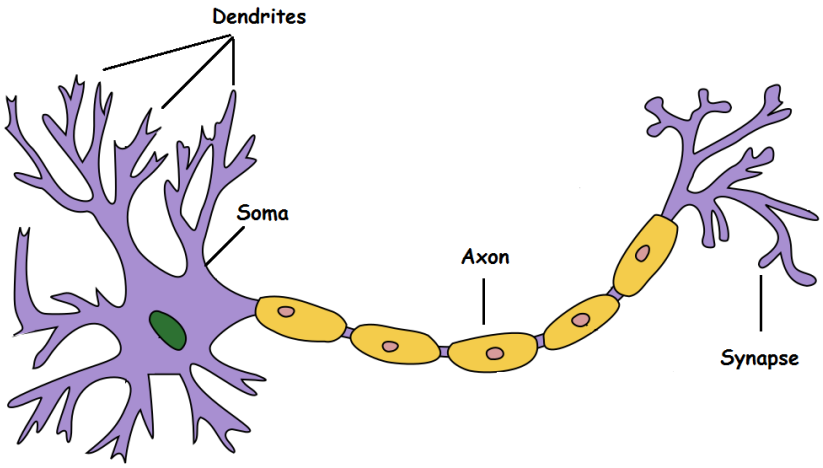



*Figure 2.1 Structure of the neuron*

The human brain is composed of an intricate network of interconnected neurons, which possess dendrites to receive incoming signals. From these inputs, neurons generate electrical signals outgoing through axons and subsequently transmit these signals via axon terminals to other neurons.

Artificial neural networks, like neurons in the human brain, are composed of interconnected units called nodes, which communicate with each other through connections called edges. Within a neural network, these nodes are organized into layers, typically with an initial broad layer. The first layer contains raw data, such as numerical values, text, images, or sounds, distributed among the nodes. Each node then transmits information to the next layer of nodes through the network's edges.

The human central nervous system comprises an incredible number of approximately $10^{11}$ neurons. The neuron is the fundamental unit of the human brain, much like the transistor is the basic unit of a CPU processor. However, neurons and transistors operate at vastly different time scales and energy consumption levels. Neurons operate on a time scale of approximately $10^{-3}$ seconds and have a low energy consumption of about $10^{-16}$ Joules per operation, while transistors operate on time scales of approximately $10^{-9}$ seconds and have a higher energy consumption of about $10^{-6}$ Joules per operation.

In the human brain, there are approximately $10^{10}$ neurons, whereas in a processor, there are approximately $10^{9}$ transistors. Furthermore, the human brain is characterized by an extraordinary number of synaptic connections, totaling around 60 trillion.

Ultimately, the human brain can be considered a highly nonlinear and parallelized information processing system that operates on a time scale and with energy consumption very different from artificial processors.

## 2.4 Mathematical Model of Neural Networks

The first layer of a neural network is known as the 'input layer.' This layer serves as the entry point for raw data into the network, including numerical values, text, images, or sounds. It is not considered an actual processing layer since it does not contribute to data processing. During the data processing, these inputs are divided into discrete units called 'nodes.' Each node in the input layer represents a specific element or feature of the input data.

One of the key features of a neural network is its layered structure. Each layer consists of a set of nodes, and information propagates from the nodes in one layer to the nodes in the next layer through the connections called 'edges' of the network. These connections play a crucial role in data processing as they carry information between nodes.

Each edge between nodes is associated with a 'numerical weight' or 'algorithm.' This weight indicates the importance of the information transmitted through that specific edge. A fundamental aspect of neural networks is their ability to learn from past experiences. These weights can be adapted and recalibrated based on how the network processes data over time and accumulated experience.

A typical neural network architecture can be divided into three main layers: the input layer, hidden layers, and the output layer. This structure reflects how data flows through the network during the processing process.

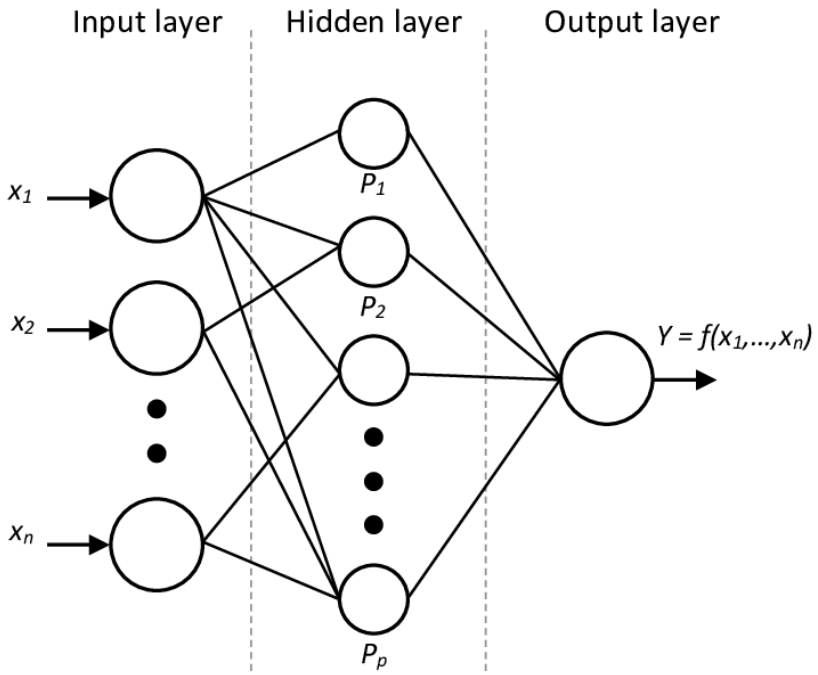



*Figure 2.2 Neural Network Architecture*

The process begins in the input layer, where initial data is received and analyzed. This layer serves as the 'sensors' of the network, detecting the basic features of raw data. This initial information detection phase is crucial for subsequent processing.

The input to the node is:

$$I_i = \sum_{j=1}^{N} w_{j,i} y_j$$

After the input phase, the data is sent to the hidden layers. These intermediate layers are called 'hidden' because they work invisibly, much like the human mind processes concepts in ways that are not immediately evident. The hidden layers perform complex analysis and calculations to process the information received from the input layer. An important aspect is that data passes through each hidden layer progressively. Each hidden layer performs computations based on the previous ones, refining and optimizing the information as it moves through the network.

The result of this processing is shown in the output layer. This layer represents the desired output or response of the network based on the input data and the processing done by the hidden layers. It is the ultimate result of data processing and represents the expected response from the neural network.

It's worth noting that the intermediate layers, the so-called hidden layers, play a crucial role. Like human perception, where we see an object as a coherent whole rather than as a series of separate elements, the hidden layers contribute to understanding the relationships between input data and the final output. This process of 'breaking down' data is fundamental to the effectiveness of neural networks.

The net input $A_i$ of neuron $n_i$ is:

$$A_i = \sum_{j=1}^{N} w_{j,i} y_j - \theta_i$$

- $y_j$ is the signal coming from neuron $n_j$.
- $w_{j,i}$ is the weight of the synapse from $n_j$ to $n_i$.
- $\theta_i$ is the threshold of node $n_i$.

The response of neuron $n_i$:

$$y_i = \phi(A_i) = \phi\left(\sum_{j=1}^{N} w_{j,i} y_j - \theta_i\right)$$

Where $\phi$ is the activation function.

The “activation function” plays a crucial role in the process of information transmission between nodes. This function determines whether a neuron in the next layer will be activated or not based on the sum of the weights of its input edges. In other words, if the sum of the weights exceeds a predefined “threshold”, known as the “activation function”, the neuron will be activated; otherwise, it won't. This mechanism leads to an “all-or-nothing” configuration, where a neuron is activated only if the input meets certain criteria.

It should be emphasized that the weights assigned to each edge are unique and specific to ensure that nodes in the network activate differentially. This means that even if two nodes receive similar input, the unique weights along their edges make them react differently and consequently produce different outcomes.

To effectively train the network, supervised learning is used. In this process, the model's output is compared with the actual output, known to be correct. The discrepancy between these two results is measured and referred to as “cost” or “cost value”. The training objective is to gradually reduce

this cost value until the model's prediction closely matches the correct output. [25] This is achieved by incrementally adjusting the weights in the network until the lowest possible cost value is reached. This training process is known as “backpropagation”.

It should be noted that, unlike the flow of data into the neural network from left to right, backpropagation is performed in reverse, starting from the rightmost output layer and proceeding towards the leftmost input layer. This process allows the network to learn from its mistakes and adjust the connection weights to improve overall performance.

However, it should be emphasized that one of the limitations of neural networks is their “black box” nature. This means that even though the network can produce accurate results, its internal structure provides limited or no information about the specific variables that influence the outcome.

Furthermore, it is possible that two neural networks with different topologies and weights may produce the same result, making it even more challenging to discern relationships between variables and the output. This contrasts with approaches such as regression techniques and decision trees, which can offer greater interpretability of variable relationships.

There are various architectures and techniques for designing neural networks, but one of the simplest is the so-called “feed-forward network”. In a feed-forward network, data signals flow in a single direction, from the input layer to the output layer. There are no cycles or loops in the structure, meaning information travels unidirectionally through the network.

The most elementary form of a feed-forward network is the “perceptron”. A perceptron consists of one or more inputs, a processor that performs computations, and a single output representing the network's final result for a given input. [26] This simple model serves as a fundamental building block in more complex neural network architectures, which may include numerous hidden layers and neurons to handle more intricate problems.

## 2.5 Keras Library

Keras is a popular open-source library written in Python for machine learning and deep learning. It has been developed to provide a user-friendly, modular, and intuitive interface for building and training neural networks. Furthermore, Keras is designed to run on various deep learning backends,

such as TensorFlow, Theano, and Microsoft Cognitive Toolkit (CNTK). This allows Keras users to leverage the specific features of each backend without needing to change the Keras API. [27]

The Keras library is designed with the goal of being easy to use, even for users with limited experience in deep learning. The Keras API is intuitive and requires only a few lines of code to create and train neural networks. This makes Keras a popular choice for developers and researchers who want to quickly prototype their machine learning models.

The library is structured in a modular way, which means that neural networks can be created by adding layers one after the other. Each layer is responsible for a specific step in the learning process of the network. Keras offers a wide range of layers, including fully connected layers (Dense), convolutional layers, recurrent layers (LSTM, GRU), normalization layers, and others.

This library supports two types of modeling:

- Sequential models: suitable for creating layered neural networks, such as feedforward or convolutional networks.
- Functional models: offering greater flexibility, allowing the creation of more complex computational graphs, including models with multiple inputs and outputs or with shared layers. It provides simple methods to compile the model with a loss function and an optimizer.

Users can customize the training by using different loss functions and optimizers, as well as specifying evaluation metrics to monitor the model's performance during training.

The learning rate, momentum, and decay are three key hyperparameters in optimization methods used in machine learning to train neural models. These parameters work together to determine how the model adjusts its weights during training. The learning rate controls the size of weight updates, momentum accelerates the training, and decay gradually reduces the learning rate during training for more stable convergence. Finding an optimal balance among these hyperparameters is crucial to achieve optimal model performance.

### 2.5.1 Learning Rate

The "learning rate" is a fundamental hyperparameter in the field of machine learning, determining how quickly an optimization algorithm updates the weights of a model during the training process. The learning rate controls the step or magnitude of weight updates based on the gradient of the cost

function with respect to the weights themselves. A learning rate that is too small may slow down training and risk getting stuck in local minima, while a learning rate that is too large may cause the training process to oscillate without converging to the global minimum.

In the context of the Keras library, the learning rate is one of the configurable options in the optimizer used to train the model. The optimizer is the algorithm responsible for updating the weights during the machine learning process. Keras offers several optimizers to choose from, such as "SGD" (Stochastic Gradient Descent), "Adam," "RMSprop," and many others.

As mentioned earlier, an excessively low learning rate can significantly slow down the model training process. Weight updates will be very small, and it may require many epochs to achieve adequate convergence. Additionally, a learning rate that is too low may cause the model to get stuck in local minima without reaching the desired global minimum. On the other hand, an excessively high learning rate can lead to unstable training. Weight updates could be so large that they cause the training process to oscillate, preventing the model from converging to the global minimum. In some cases, a learning rate that is too large may even cause the training process to diverge.

For this reason, finding the optimal learning rate is a crucial part of the training process. Typically, multiple trials are performed with different learning rate values, and the value that produces the best result in terms of convergence and model performance is observed. Techniques such as "learning rate scheduling" or the use of adaptive optimizers, such as "Adam," help adapt the learning rate dynamically and automatically during training.

An optimal choice of the learning rate helps achieve faster and more stable convergence during model training, thereby contributing to better performance on the test dataset.

### 2.5.2 Momentum

Momentum is a concept used in optimization methods in the field of machine learning to accelerate the training process and reach the global minimum of the cost function more quickly. Momentum is a mechanism that considers the directions of previous weight updates during the current weight update. This helps increase the convergence speed of the model and overcome obstacles such as local minima or plateaus more efficiently.

In the Keras library, momentum, like the learning rate, is a configurable hyperparameter in the optimizers used to train the model. Optimizers that support momentum, such as "SGD" with momentum, add this feature as an additional factor to the weight updates during training.

Momentum can be seen as a "velocity" that the model acquires as it moves through the weight space. During weight updates, momentum allows accumulating information from previous iterations and giving more weight to updates that have a consistent direction over time.

It helps avoid oscillations and "jumps" in the weight update directions. This accelerates the model training process by enabling it to follow coherent directions towards the global minimum of the cost function. It allows the model to overcome plateaus or flat areas in the cost function, which can be problematic for optimization algorithms as weight updates become very slow in these regions. Momentum enables "sliding" past these flat regions and reaching more significant regions of the cost function.

Momentum is controlled by a hyperparameter called the "momentum coefficient." This coefficient is a value between 0 and 1, determining the extent to which momentum influences weight updates. Higher values of the momentum coefficient give more weight to previous updates, while lower values make it less influential.

It can be used in conjunction with other optimization techniques, such as learning rate scheduling or learning rate adaptation. This combination of techniques helps address issues of oscillation, slowdown, or slow convergence during training.

### 2.5.3 Decay rate

Another parameter is the decay rate, which is also a hyperparameter used to gradually reduce the value of the learning rate during the training process. The decay rate is a dynamic learning rate adjustment mechanism aimed at improving training stability and model convergence.

In the Keras library being discussed, optimizers that support the decay rate add this functionality to gradually reduce the learning rate over training epochs.

This reduction rate can be expressed as a fixed percentage or as a constant value subtracted from the initial learning rate at each epoch. Oscillations and excessive variations in the learning rate can negatively affect training stability and the achievement of effective convergence.

In the Keras library, the decay rate is configurable in optimizers to enable a more stable and optimal convergence of the model during the machine learning process.

### 2.5.4 Epoch

In the context of machine learning, an "epoch" represents a single complete pass of the entire training dataset through the learning algorithm. During an epoch, all training data is used to advance the model and update the weights of its internal units. Training a model involves iterating through multiple epochs to allow the model to learn from the data repeatedly and improve its performance.

The epochs parameter (number of epochs) is a hyperparameter that indicates how many times the entire training dataset will be presented to the model during the learning process. After each epoch, the model's weights are updated based on the prediction errors compared to the training data.

During the first epoch, the model sees the entire training dataset and computes predictions for each input example. Subsequently, it calculates the prediction error (loss) relative to the expected output values. Using the optimizer and the backpropagation algorithm, the model updates the weights of its internal units to reduce the overall error.

After the first epoch, the model has already begun learning from the training data. In the subsequent epochs, the model will continue to see the entire training dataset repeatedly. This process of iterating through the epochs allows the model to gradually refine its weights and improve its performance.

Typically, with enough epochs, the model will converge to a weight configuration that minimizes the prediction error. However, it is important to find a balance in using epochs. Too many epochs may lead to overfitting, while too few epochs may prevent the model from learning enough from the data.

During training, it is common to monitor the validation error (loss) on a separate validation dataset. This helps evaluate the model's ability to generalize well to unseen data during training and avoid overfitting. In some cases, it may be necessary to stop training before all planned epochs are completed if the performance on the validation dataset deteriorates.

The decay rate can help the model converge more stably and consistently towards the global minimum of the cost function. By gradually reducing the learning rate, the model can explore the weight space more effectively, leading to better and more stable convergence.

## 2.6 Performance Evaluation Metrics for Regression Models

### 2.6.1 R2

The coefficient of determination, commonly known as R2, is a statistical index that measures the proportion of the data variability explained by the statistical model used. It is related to the fraction of the variance that the model does not account for.

R2 is a statistical measure used to determine how well a linear regression model fits the observed data. It indicates the proportion of variance in the dependent variable that can be explained by the independent variables included in the model.

The coefficient R2 takes values between 0 and 1. An R2 of 0 indicates that the model cannot explain any variance in the output data, while an R2 of 1 indicates that the model explains the entire variance in the output data. An R2 close to 0 suggests that the model does not fit the data well, while an R2 close to 1 indicates a model that is highly fitted to the data.

The formula for calculating R2 is as follows:

$$R^2 = 1 - \frac{RSS}{TSS}$$

Where $TSS = \sum_{i=1}^{n}(y_i - \bar{y})^2$ is Total Sum of Square.

It is the sum of the squares of the differences between each observed value of the dependent variable and the mean of those values. This represents the total variance of the data.

Where $RSS = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ is Residual Sum of Square.

It is the sum of the squares of the differences between the observed values of the dependent variable and the values predicted by the regression model. It represents the variance not explained by the model and is referred to as the residual error of the model.

However, it is important to note that the coefficient R2 is not an absolute measure of the accuracy of the model and has some limitations. For instance, R2 tends to increase with the number of independent variables even if they are not relevant to the model, leading to issues of overfitting. Additionally, R2 may not be sufficient to fully evaluate the goodness of a regression model, especially when dealing with non-linear models or complex relationships between variables.

Therefore, it is always advisable to use other performance evaluation measures, such as the root mean squared error (RMSE) or the mean absolute error (MAE), to obtain a more comprehensive view of the model's effectiveness.

### 2.6.2 RMSE and MAE

The Root Mean Square Error (RMSE) is a valuable metric for quantifying how much a model's predictions deviate from actual data. It is widely used in the field of data analysis and machine learning to assess the performance of predictive models and guide the choice among different models. A lower RMSE value indicates higher model accuracy, as it represents a smaller average deviation.

The formula for calculating RMSE is as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Where:

- $N$ represents the total number of samples in the evaluation dataset.
- $y_i$ is the actual value (or target value) of the sample.
- $\hat{y}_i$ is the value predicted by the model for the sample.

While the Mean Absolute Error (MAE) is, like RMSE, a metric used to assess the accuracy of a prediction or regression model, it calculates the average of the absolute differences between each model prediction and the corresponding value in the actual data.

Both MAE and RMSE express the mean value of the error made by the prediction, but the difference between the two lies in the fact that MAE is less sensitive than RMSE to extreme differences between the predicted and observed values. Furthermore, RMSE is decomposed into RMSEs and RMSEu to provide an evaluation of how much the model's predictions are affected by systematic errors. For a prediction to be considered "good", RMSEs should approach zero, while RMSEu should get closer to RMSE.

# 3 Instruments

## 3.1 Accelerometer

The accelerometer is a sensor capable of measuring the acceleration of an object or a system at a specific point in space. Accelerometers provide crucial information about linear or gravitational acceleration, enabling the monitoring of movements, vibrations, and orientation.

In this specific case, they have been employed to measure the acceleration relative to the structure of the beam, converting the acceleration into an electrical signal acquired by the acquisition system.

Accelerometers can measure acceleration along one, two, or three axes. Single-axis accelerometers measure acceleration along only one axis, while dual-axis and tri-axis accelerometers measure acceleration along two and three orthogonal axes, respectively. This allows for the determination of the total acceleration and orientation of the object in a three-dimensional space.

Accelerometers can be classified according to various criteria, including the operating principle, the technology employed, and the number of measurement axes. Now, let's explore the different types of accelerometers:

- Piezoelectric accelerometers: They exploit the principle of piezoelectricity. A piezoelectric crystal generates a voltage proportional to the acceleration when subjected to mechanical stress. These sensors are known for their high sensitivity, rapid response, and high sampling frequency. However, they can be influenced by electrical noise and require proper signal amplification.
- Capacitive accelerometers: They rely on the variation in electrical capacitance between two plates when subjected to acceleration. A movable mass inside the sensor causes a change in the distance between the plates, which in turn affects the sensor's capacitance. These accelerometers offer good linearity and accuracy, as well as low noise. However, they may require proper temperature compensation to achieve accurate measurements.
- Microstructure accelerometers: They utilize microelectromechanical systems (MEMS) structures to measure acceleration. Microscopic structures such as springs and micro-beams deform due to acceleration, generating a variation in electrical resistance, capacitance, or inductance. MEMS accelerometers are widely used due to their small size, relatively low cost, and good accuracy in everyday applications.

In this case, piezoelectric accelerometers were available, which are sensors that exploit the piezoelectric properties of materials. These materials consist of a crystalline matrix that generates an electric field when subjected to mechanical deformation. Unlike other accelerometers, piezoresistive accelerometers allow for higher sensitivity but have the disadvantage of being influenced by temperature variations.

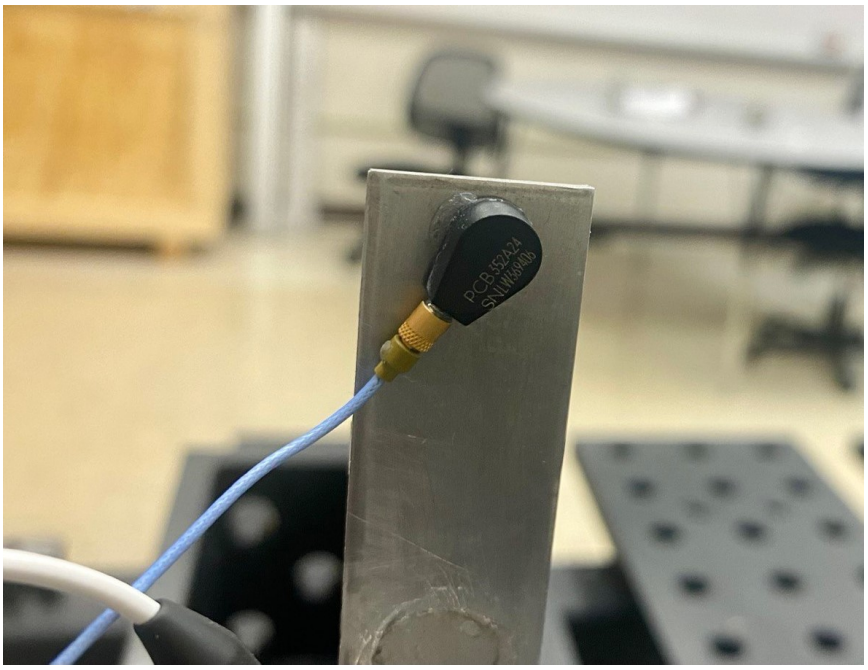


*Figure 3.1 Accelerometer*

Piezoelectric accelerometers find wide application in impact and vibration measurements. Generally, they do not provide any output in response to constant accelerations due to the fundamental principles of piezoelectric displacement measurement. However, they yield significant voltage signals, have small dimensions, and can have very high natural frequencies, a crucial characteristic for accurate measurements of impulsive phenomena.

No intentional damping is incorporated, allowing the material's hysteresis to be the sole source of energy dissipation. As a result, the damping ratio is very low (approximately 0.01), which is acceptable given the high natural frequency. The transfer function is:

$$\frac{e_0}{\ddot{x}_i}(D) = \frac{\left[\frac{K_q}{C\omega_n^2}\right]\tau D}{(\tau D + 1)\left[\frac{D^2}{\omega_n^2} + \frac{2\zeta D}{(\omega_n + 1)}\right]}$$

The high-frequency response is limited by mechanical resonance, while the low-frequency response is limited by the piezoelectric characteristic $D/(rD + 1)$. The dimensionless damping ratio $\zeta$ of

piezoelectric accelerometers is generally not provided by manufacturers, but in most practical applications, it can be assumed to be close to zero. The frequency band where the accelerometer provides accurate measurements (with a 5% tolerance upward at higher frequencies and a 5% tolerance downward at the lower end of the frequency response band) is $\frac{3}{\tau} < \omega < 0.2\omega_n$.

For accurate low-frequency response, a wide $\tau$ (time constant) is required, which is typically achieved by using high-impedance voltage amplifiers or charge amplifiers. Systems designed for a response below 1.0 Hz and subjected to temperature transients can experience errors due to the pyroelectric effect affecting most piezoelectric materials. In this case, an output charge is generated in response to a temperature input.

For systems that exhibit negligible response at low frequencies, these temperature-induced signals (as they are "slow" transients) result in a small output. Significant errors may occur if a high $\tau$ is selected to measure low-frequency accelerations or if the accelerometer is not designed to minimize thermal effects. [28]

The implementation details of piezoelectric accelerometers can be modified to emphasize specific performance aspects required for applications. There is no single configuration that is ideal for all situations due to the compromises inherent in any engineering design.

The basic design, known as the compression type, is the simplest and most robust, offering the best mass/sensitivity ratio. As the housing acts as an integral part of the mass-spring system, this type is more sensitive to spurious inputs. For piezoelectric accelerometers, these spurious inputs include temperature, acoustic noise, base bending (surface deformations induced by the bending of the mounting surface), cross-axis motion, and magnetic fields.

The spring is typically preloaded to make the piezoelectric material work in the most linear portion of its load/deformation curve. This preload also allows for measurements of both positive and negative accelerations, without ever subjecting the piezoelectric material to tensile stresses. In other words, the initial preload results in an output voltage with a certain polarity.

However, this polarity immediately decays, and the polarity associated with the potential difference subsequently produced by the acceleration, which is the subject of the measurement, will follow the direction of motion. This is because the charge polarity depends on the deformation variation and not its magnitude. The preload is chosen to be large enough to never cancel out, even in the presence of the widest input accelerations.

Microcircuit electronics have enabled the development of piezoelectric accelerometers with charge amplifiers (Integrated Circuit Piezoelectric - ICP) placed inside the instrument's enclosure. A single two-wire cable, which simultaneously transmits both the power signal to supply the amplifier and the measurement signal, connects the instrument to a simple constant current power supply.

A high-level output signal (a few volts) is directly provided to an oscilloscope or a signal analyzer. This system allows for higher sensitivity with a smaller accelerometer capable of measuring at higher frequencies while reducing the noise generated by the cable and the limitations on its maximum length, all at lower costs.

These advantages come at the expense of a reduced temperature operating range (microcircuit electronics have tighter temperature limitations compared to those of the accelerometer alone) and less versatile signal conditioning (integrated amplifiers allow for limited or no adaptation). The background noise of the accelerometer-amplifier combination might need specific attention, especially at low frequencies, where the acceleration amplitude could be small and thus masked by noise.

In the market, a wide variety of piezoelectric accelerometers can be found. The compromise between sensitivity and frequency response is evident in the common specifications provided for the instrument; an accelerometer used for impact detection may offer 0.004 pC/g and exhibit a natural frequency of 250,000 Hz, while a unit designed for low-level seismic measurements is characterized by 1000 pC/g and a natural frequency of 7000 Hz.

The response to spurious inputs of thermal nature and due to support bending, for shear isolated models, is about 200 times lower compared to models not optimized for these purposes. There are also small triaxial units, as small as a 7 mm side length cube with a mass of 1 g. Instruments that are uncooled and capable of operating in a temperature range from -40 to 815 °C typically exhibit a sensitivity variation of approximately 10% when transitioning from -40 to 815 °C and are designed to withstand typical radioactive environments found in nuclear reactors.

A system of accelerometers, wiring, and signal conditioning, designed for cost-effective multi-channel vibration testing, employs low-cost piezoelectric film transducers as plug-in modules to provide single-axis, dual-axis, or triaxial measurement points when operating in non-particularly harsh environments, such as a laboratory.

Piezoelectric accelerometers demonstrate greater cross-axis sensitivity compared to other types; however, this is typically kept around 2-4%, and it usually does not represent a critical factor. Some manufacturers indicate the axis of lesser cross-axis sensitivity, allowing the user to orient the instrument during installation to minimize this effect.

Accelerometers can be mounted using threaded studs (preferred method), adhesives, or wax, or with magnetic attachments. The primary effect of the different mounting methods is a reduction of the natural frequency to a lower value than what the accelerometer exhibits when not yet mounted, due to the elastic and inertial characteristics of the mounting system.

### 3.1.1 ICP Accelerometers

All accelerometers require an ICP (IEPE) power supply, although they can also be used with Voltage supply.

The ICP technology, which stands for Integrated Circuit Piezoelectric, is a commonly used technology in accelerometers to amplify and condition the signal generated by the piezoelectric sensors. It was introduced to simplify the reading of piezoelectric signals and reduce electrical interference.

In an ICP-based accelerometer, the piezoelectric crystal is connected to an integrated circuit within the same device. The ICP circuit amplifies the piezoelectric signal generated by the crystal. Since piezoelectric signals are generally very weak, signal amplification is essential to obtain an accurate measurement of acceleration. The ICP circuit amplifies the signal to make it suitable for further processing. This circuit provides a constant current to the piezoelectric crystal.

This constant current ensures an accurate and stable response of the crystal to mechanical stresses, as well as contributes to reducing the influence of external electrical noise and interference.

Another important function of the ICP circuit is signal conditioning. This includes converting the piezoelectric signal into a voltage or current signal proportional to the measured acceleration. Signal conditioning may also involve filtering to eliminate unwanted frequencies or noise present in the signal.

This technology greatly simplifies the interfacing of the accelerometer with the data acquisition system. In fact, an ICP interface can be directly connected to an ICP input of a measurement system without requiring additional amplification or signal conditioning circuits.

In summary, the ICP technology in piezoelectric sensors of accelerometers offers efficient signal amplification and conditioning, ensuring accurate and reliable measurements of accelerations.

Additionally, each accelerometer has its own sensitivity required to convert the Voltage value into $m/s^2$. For these experiments, accelerometers designed and manufactured by PCB Piezotronics, Inc. were used, each with a different sensitivity.

## 3.2 Load Cell

The load cell is a device used to measure the force or load applied to it. In this case, it was used for the experiments on the beam and plate to measure the tension or compression generated by the forces applied to the structures.

Load cells are typically composed of a solid block or a flexible component that undergoes deformation when a load is applied. This deformation generates a variation in electrical resistance, capacitance, or voltage, which is then converted into an electrical signal proportional to the applied load. This signal is then passed to the sensor signal to amplify it and finally to the data acquisition system.

Load cells are often strategically placed along the beam to measure the forces acting on it. They can be positioned on either the upper or lower part of the beam, depending on the type of measurement desired. For example, if one wants to measure the bending or tension at the upper end of the beam, the load cell can be positioned at that location to detect the forces acting on it.

It is important to select an appropriate load cell based on the experiment's specifications to obtain accurate and reliable results since they are available in various load capacities, ranging from grams to tons, and with different measurement accuracies.

In the case of the beam, a load cell model SN LW 55202 was used, while for the plate, the model was SN LW 55552, designed and manufactured by PCB Piezotronics, Inc., a company specializing in measurement and sensor technology. PCB's load cells are available in a wide range of load capacities, which can vary from fractions of Newtons (or pounds) to many tons, depending on the specific requirements of the application.

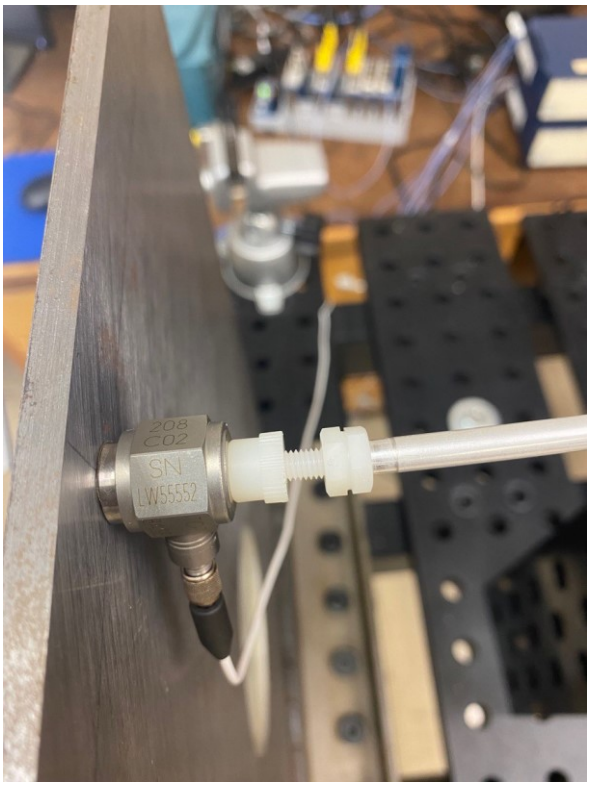



*Figure 3.2 Load cell*

PCB's load cells can be provided in various mechanical configurations to suit application needs. Some common configurations include pancake load cells (flat), basket load cells, compression or tension load cells, and other variants. They are designed to offer high precision and repeatability in measurements, ensuring reliable and accurate results.

These sensors are constructed with durable and robust materials to withstand industrial environments and harsh conditions. Additionally, many PCB load cells include overload protection systems and safeguards against accidental damage.

For the experiments conducted, the load cell has a mass of 22.7 g. It is considered a concentrated mass in the calculation of natural frequencies but was approximated to 23 g.

## 3.3 Shaker

The shaker is a mechanical exciter that generates electromagnetic vibrations by converting an electrical signal into controlled mechanical motion.

It is a crucial component in vibration tests and seismic tests for evaluating structural performance and primarily consists of three main components: an electromechanical actuator that converts

electrical energy into mechanical energy, typically composed of a permanent magnet and a movable coil subjected to a variable magnetic field generated by an electric current. The oscillation of the movable coil generates the vibration force.

It also includes a support and fixation system in which the shaker is mounted to withstand the forces generated during vibrations. It is essential for the support system to be rigid and minimize external interferences that could alter measurements or the behavior of the tested structure.

Finally, it has an electronic controller that regulates the electrical current sent to the actuator to control the intensity and frequency of the vibrations generated by the shaker. The controller can be programmed to generate various vibration profiles.

In this case, three different types of signals were generated by the shaker: harmonic cosine, pseudorandom, and chirp. Each of these signals had a specific frequency, sampling rate, and acquisition length.

For these experiments, a K2007E01 Mini SmartShaker with an integrated power amplifier was used. It is an electrodynamic exciter designed for generic vibration tests of small components and subassemblies up to 9 KHz. [29] It can also be used as an exciter for modal tests of small structures.

A new generation of ultra-compact precision power amplifiers integrated into its base eliminates the need for a separate and bulky power amplifier. The compact size of the K2007E01 shaker makes it ideal for applications such as production screening, reliability acceptance tests, and engineering evaluations.

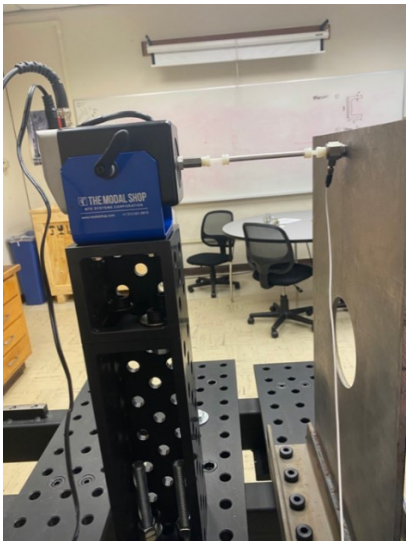



*Figure 3.3 Shaker*

The use of composite materials in the armature suspension and guidance system provides high lateral and rotational containment while maintaining maximum compliance in the direction of movement, allowing a peak-to-peak stroke of 0.5 inches (13 mm).

The body structure of the K2007E01 shaker and the pin assembly are designed to allow a variety of operational positions. The K2007E01 is generally used in a vertical orientation and can be rotated up to 90° for horizontal applications. The shaker comes with a variety of nylon tips from 10 to 32, providing electrical insulation and flexible attachments for testing items.

## 3.4 CompactDAQ

The CompactDAQ is a data acquisition system developed by National Instruments, including a series of measurement modules and a user-friendly programming interface.

The CompactDAQ system consists of several components. At its core, there is a compact chassis that houses the measurement modules. The chassis can accommodate various I/O (Input/Output) modules based on the specific needs of the application. The measurement modules may include analog inputs, digital inputs, analog outputs, digital outputs, counters/frequency meters, thermocouples, and more.

The CompactDAQ controller is a rugged, reliable, and high-performance integrated controller with standard industry certifications. It is ideal for performing waveform acquisition and online software analysis while recording data in the integrated or removable SD memory.

The controller also offers a wide range of standard connectivity and expansion options, such as USB, Ethernet, CAN/LIN, and RS232 serial. With over 60 C Series I/O modules for almost any sensor type, you can quickly design a custom hardware configuration optimized for size, cost, and performance. The C Series modules are high-quality input and output modules that provide signal conditioning and analog-to-digital conversion for the CompactDAQ system.

These hot-swappable modules connect directly to the chassis, making it easy to create a tailored system for specific testing requirements. NI provides dedicated software called LabVIEW for the configuration and management of the CompactDAQ system.

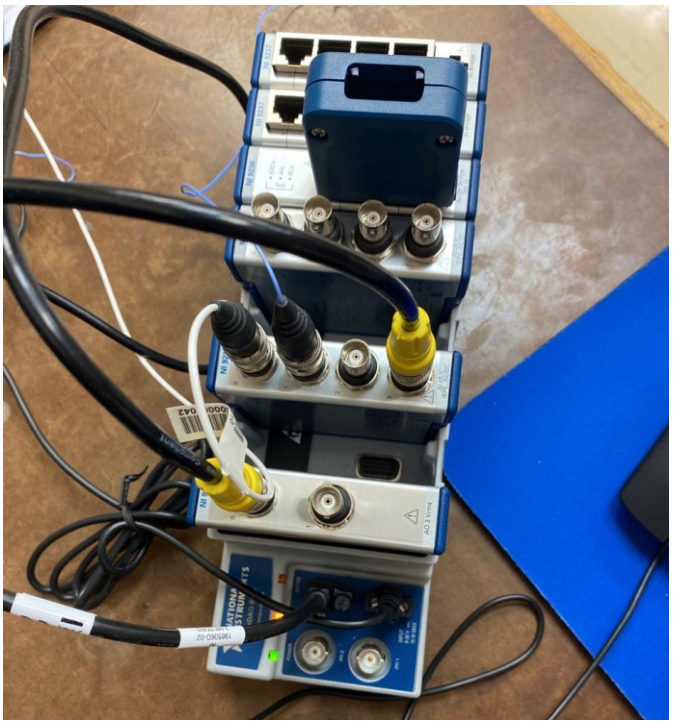


*Figure 3.4 CompactDAQ National Instrument*

In the current case, LabView software was not used; instead, Matlab was used as the interface to control and acquire data through a specific library and the "daqlist" command, enabling direct communication with NI's data acquisition devices. This usage allows for easier data acquisition and subsequent analysis, reprocessing, and visualization using various toolboxes available in Matlab.

Eight modules were used, some as outputs and others as inputs, depending on the module model.

## 3.5 Sensor Signal Conditioner

The PCB Piezotronics model 282C series sensor signal conditioners were used. They are crucial devices to ensure the accuracy and reliability of sensor measurements by amplifying and converting the signals from the sensors into a suitable format for further processing and analysis.

In the case of the sensors used on the beam, including the load cell and accelerometers, they produced very weak signals. Therefore, it was necessary to introduce this device to amplify the signals and obtain more precise measurements. This adjustment can have different gain settings, depending on how small the acquired measurement is; the correct gain is selected accordingly.

The signals from the sensors can be influenced by various environmental factors, such as electrical noise and interference. The PCB sensor signal conditioners include conditioning circuits that filter

and isolate the sensor signal from such interferences. This ensures that the signal is clean and free from disturbances, thus improving the measurement quality. Additionally, PCB signal conditioners may include analog-to-digital conversion (ADC) circuits to convert the signal into a digital format.


*Figure 3.5 PCB Piezotronics model 282C series sensor signal conditioners*

Some sensors may generate non-linear signals in response to the measured physical quantity. The sensor signal conditioners can include linearization circuits to convert the non-linear signal into a linear relationship with the measured physical quantity. This enables more accurate and reliable measurements. They may also incorporate protection circuits to safeguard the sensor and data acquisition system from overvoltage, overcurrent, and short circuits. This protection is essential to prevent device damage and ensure the safety of the measurement setup.

In the case of the beam experiment, a gain of 1x was set for the signals from the accelerometers and load cell, so that they would be visible during data reprocessing using Matlab. They are connected to the CompactDAQ modules through cables.

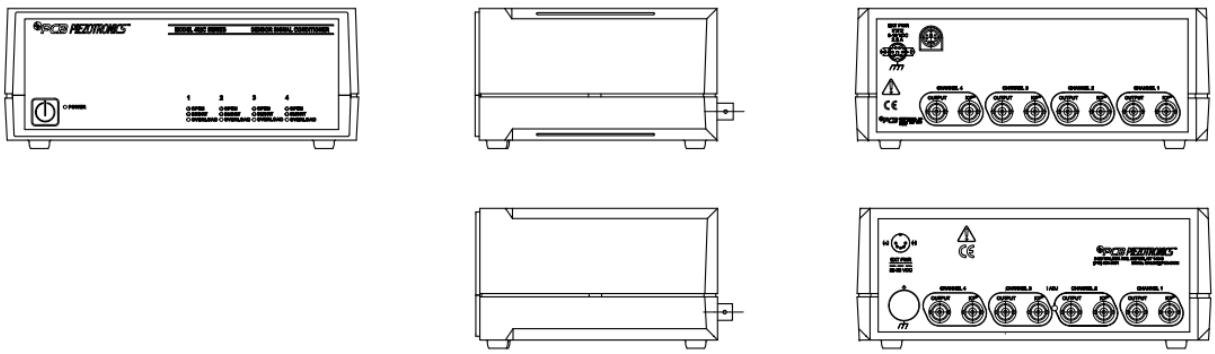


*Figure 3.6 Sensor signal conditioners diagram*

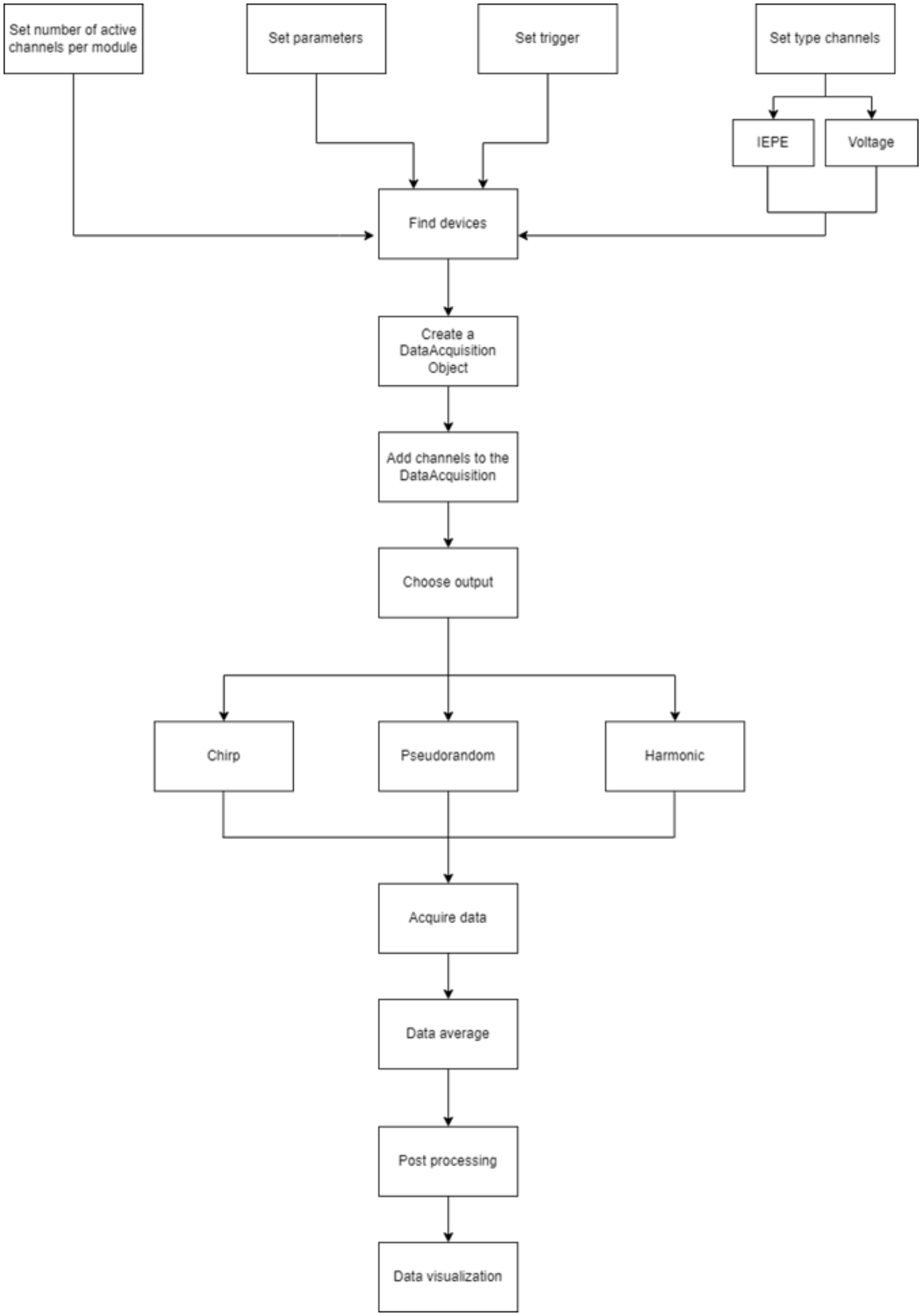
## 3.6 Other Instruments

Part of the experiment was conducted on a worktable with adjustable legs that allowed for changes in height and floor contact to avoid potential oscillations during the experimental test. Additionally, it was necessary to secure the beam and the plate to a support to prevent any undesired oscillations due to a sluggish support. This was achieved by using blocks with holes that could be bolted to the table using pins, along with one or more washers and nuts of various sizes.

# 4 Data Acquisition

The acquisition of experimental data in the beam and plate experiments was carried out using a code developed in Matlab, which allowed for the acquisition of data from the load cell and accelerometers.

In the following flowchart, an overview of the steps followed by the developed code is presented:



As evident from the flowchart, this code consisted of a series of steps. The initial step involved configuring a range of parameters based on the requirements of the specific data acquisition scenario. Notably, it was essential to set the number of active channels for each module, trigger threshold values, parameter values, and the channel type, which could be either "IEPE" or "Voltage" in the case of accelerometers.

The essential parameters that needed configuration included acquisition rate, scan length, frequency, number of averages, and signal amplitude. Setting the amplitude correctly was of paramount importance because leaving it at the default value of unity would lead to errors. This is because, after signal acquisition, it wouldn't be able to represent the entire signal curve, potentially exceeding the range and resulting in values that were not acquired. The amplitude needed to be set to small values, given that the available accelerometers had a specified acquisition range within the interval of -4.75 V to 4.75 V.

The second step involved identifying the National Instrument device through the "daqlist" command to provide a list of devices within the chassis. Subsequently, a "DataAcquisition Object" was created, and channels were added to it based on the previously selected channel type for each channel. After these steps, the desired output was selected from three available options, data was acquired, averaged, processed, and finally displayed in the requested graphs.

The developed code allowed the usage of three different output options for the signal:

- Chirp
- Pseudorandom
- Harmony

## 4.1 Chirp Signal

A chirp signal is a signal that varies its frequency over time according to a well-defined law. The term "chirp" is derived from the sound of radar signals reflected from a target, which generates an acoustic signal resembling the singing of a bird, hence the name "chirp". [30]

The chirp signal can be represented in either the time domain or the frequency domain. In the time domain, the chirp signal is characterized by a frequency that changes linearly or nonlinearly over time. A common example of a chirp signal is the cosine chirp, defined by the following equation:

$$x(t) = A \cdot \cos\left(2\pi(f_0 t + kt^2)\right)$$

Where:

- $A$ is the amplitude of the signal.
- $f_0$ is the initial frequency of the chirp at time t = 0.
- $k$ represents the steepness or rate of frequency change over time.

When k is positive, the frequency of the chirp signal increases over time (up-chirp). Conversely, when k is negative, the frequency of the chirp signal decreases over time (down-chirp).

In the frequency domain, a chirp signal will appear as an expanding or contracting frequency band. A chirp signal is best represented in its frequency spectrum form, using Fourier transform or Hilbert transform.

Chirp signals find various applications, including radar systems for distance measurement and target localization. In sonar applications, chirp signals are used for similar purposes as in radar, but in underwater environments.

Chirp signals can be employed in various signal processing applications, such as modulation and demodulation of communication signals or frequency analysis in acoustic or vibration signals.

The use of chirp signals offers significant advantages as they provide higher frequency resolution compared to signals with constant frequency. This makes them useful in applications requiring narrowband analysis and high precision in frequency determination.

### 4.1.1 Comparison between Chirp Signal Input from MatLab and Output Load Cell

The input signal sent to the shaker was generated by the following line of MATLAB code:

$$output = Amp * chirp(t, -105, length\_scan, fre)$$

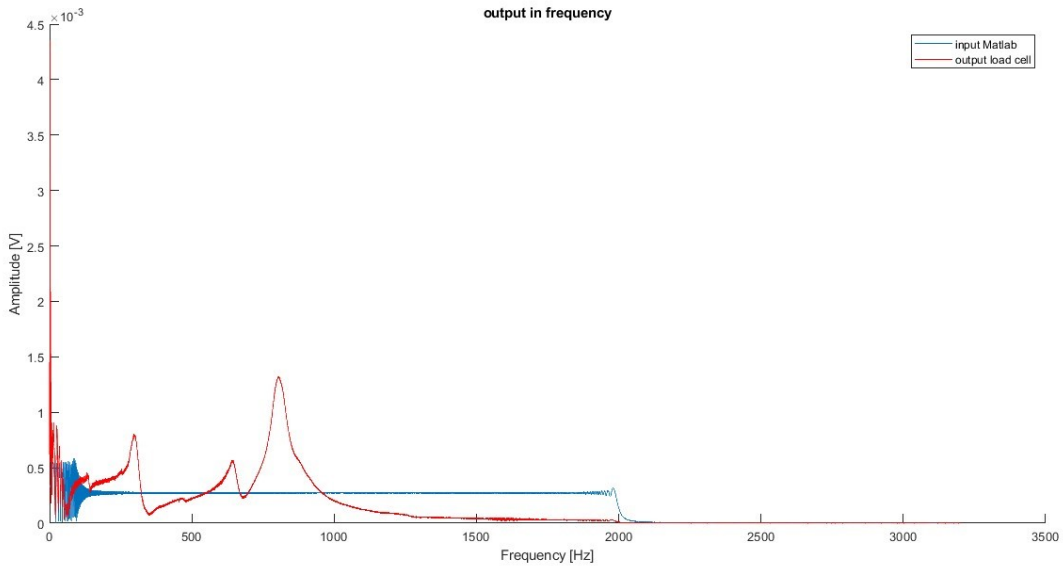



*Figure 4.1 Frequency comparison between input and output of the chirp signal in the beam*

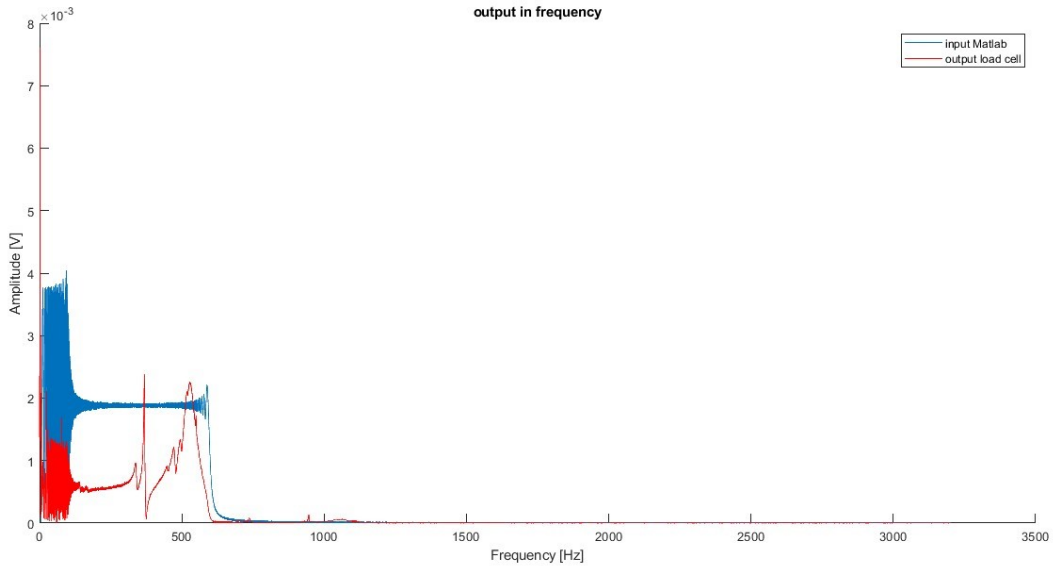



*Figure 4.2 Low frequency comparison between input and output of the chirp signal in the plate*

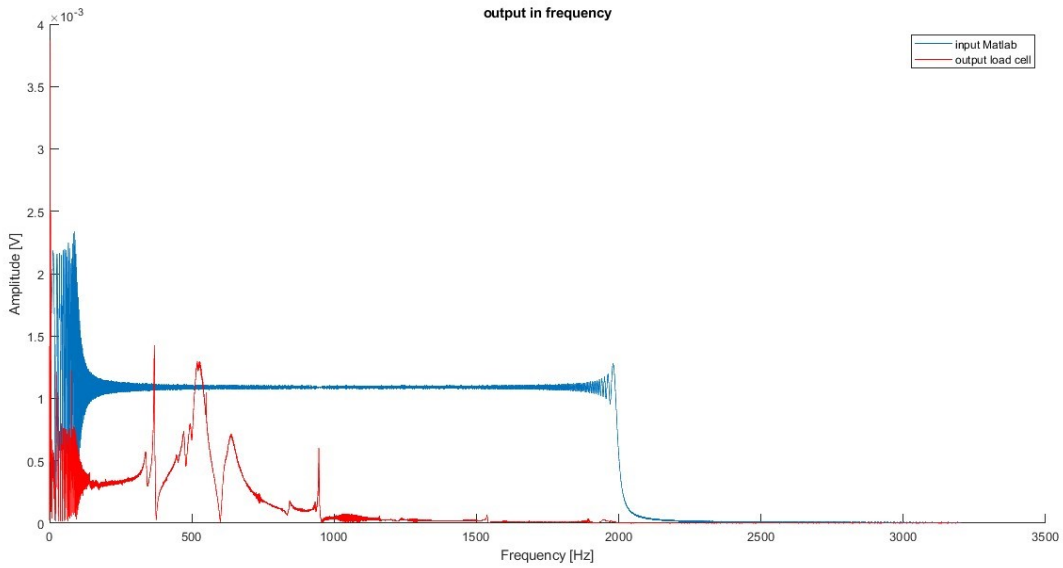



*Figure 4.3 High frequency comparison between input and output of the chirp signal in the plate*

The previous figures display a comparison, both for the beam and the plate, at low and high frequencies, between the input generated by MATLAB sent to the shaker and the actual output detected by the load cell. As evident from the graphs, there is not a strong correspondence between the input generated by MATLAB and the actual signal read by the load cell. To mitigate this discrepancy, a decision was made to use a signal with an initial instantaneous frequency at time zero equal to -105 instead of 0. This choice was made to enhance the correspondence between the input and output signals, as an initial value of 0 resulted in a less meaningful correspondence, which did not contribute to obtaining values closer to the expected ones, consequently leading to discrepancies between the experimental values and those obtained numerically.

## 4.2 Pseudorandom Signal

A pseudorandom signal, also known as a pseudo-random signal, is a signal that appears random but is generated by a deterministic algorithm. Unlike true random signals, pseudorandom signals are reproducible and predictable because their sequence is generated by a Pseudo-Random Number Generator (PRNG).

PRNGs are mathematical algorithms that produce sequences of numbers that, at first glance, appear random, but are completely determined by an initial value known as the "seed." The seed is the starting point of the algorithm, and if the same seed is used, the generator will always produce the

same sequence of numbers. To obtain different sequences, one can vary the seed or use a "jump" technique in the PRNG to move to a different position in the sequence.

In data acquisition, a pseudorandom signal is used as an input signal or "stimulus" to be sent to a system or device for conducting tests, measurements, or calibrations.

The main characteristic of a pseudorandom signal is repeatability. If you know the seed used in the random number generator, you can regenerate the exact same sequence of numbers every time. This repeatability is useful in many applications, for example, in testing and troubleshooting, as it allows reproducing the same conditions and verifying results.

A good pseudorandom number generator will attempt to produce a sequence of numbers that approximates a random distribution. This means that, even though the numbers are generated deterministically, their distribution should exhibit some properties like those of a truly random number sequence.

Although PRNGs can produce long sequences, each PRNG has an upper limit to the length of the generated sequence, known as the period. After a certain number of iterations, the generator will return to the initial sequence. This periodicity may be acceptable for many applications, but it is important to select a PRNG with a long enough period to ensure that the sequence does not repeat too frequently.

Since the pseudorandom signal is determined by the seed and the PRNG algorithm, it is fully reproducible. This repeatability is advantageous in situations where one wants to precisely reproduce the same data acquisition conditions for testing, debugging, or comparing different experiments.

Although the numbers generated by a PRNG are not truly random, a good generator will aim to approximate the statistical properties of a random sequence. This is particularly important when using pseudorandom signals in analysis and verification applications.

Pseudorandom signals produced by a PRNG have a limited duration, determined by the generator's period. After a certain number of samples, the sequence will repeat. It is important to ensure that the signal's duration is sufficiently long to meet the data acquisition requirements.

Pseudorandom signals can be designed to have a uniform spectral distribution or be modulated to follow a specific distribution. This is particularly useful when performing frequency analysis or

testing in certain frequency bands. Since the pseudorandom signal is a known sequence, it is important to ensure that it does not inadvertently contaminate the acquired data. If the pseudorandom signal were to overlap with a desired signal, it could alter the measurement results.

Pseudorandom signals are widely used in data acquisition for various applications, such as:

- Testing and verification of systems and equipment.
- Calibration of sensors and measurement instruments.
- Analysis of noise and system behavior.
- Generation of test signals for functional testing and performance evaluations.

However, it is important to note that in certain scientific or industrial applications where true randomness is required, it is necessary to use True Random Number Generators (TRNG) based on physical processes, such as thermal or quantum noise, that produce genuinely random sequences.

### 4.2.1 Comparison between Pseudorandom Signal Input from MatLab and Output Load Cell

The input signal sent to the shaker was generated by the following line of MATLAB code:

$$sv = randn(size(t'))$$
$$vs = lowpass(sv, fre, acquisition\_rate, 'ImpulseResponse', 'iir')$$
$$output = vs * Amp$$

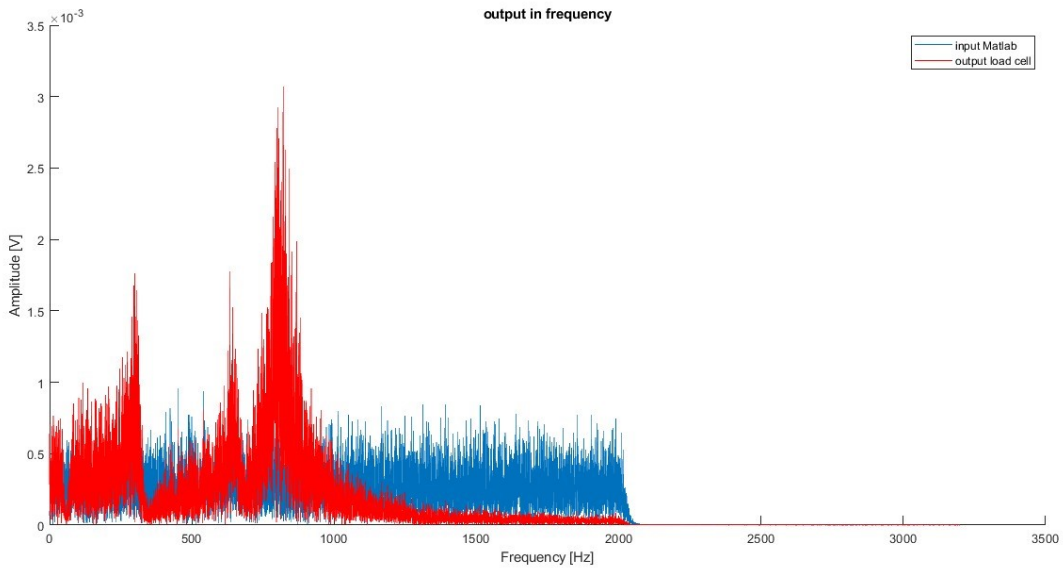



*Figure 4.4 Frequency comparison between input and output of the pseudorandom signal in the beam*

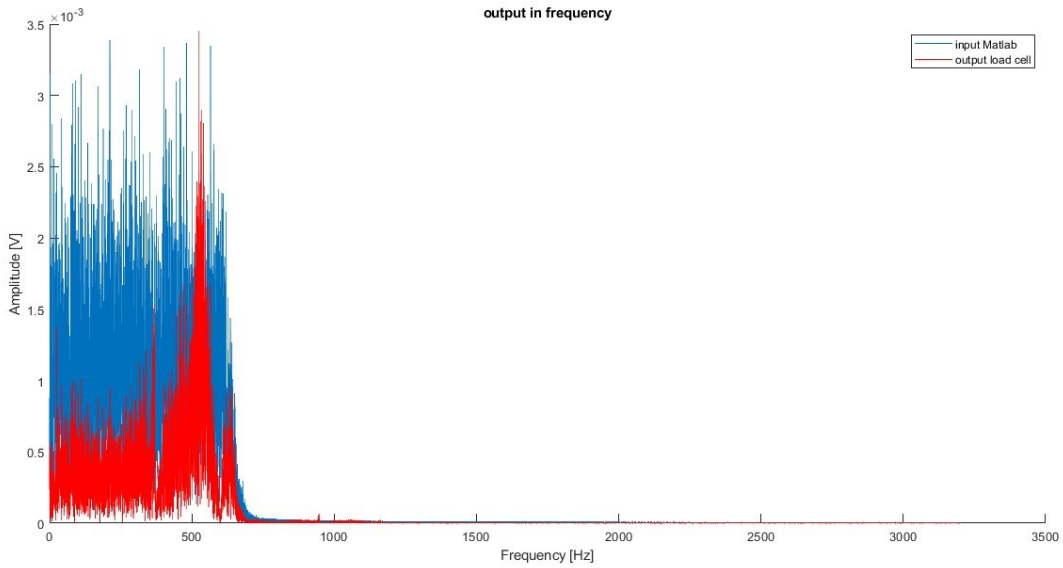



*Figure 4.5 Low frequency comparison between input and output of the pseudorandom signal in the plate*

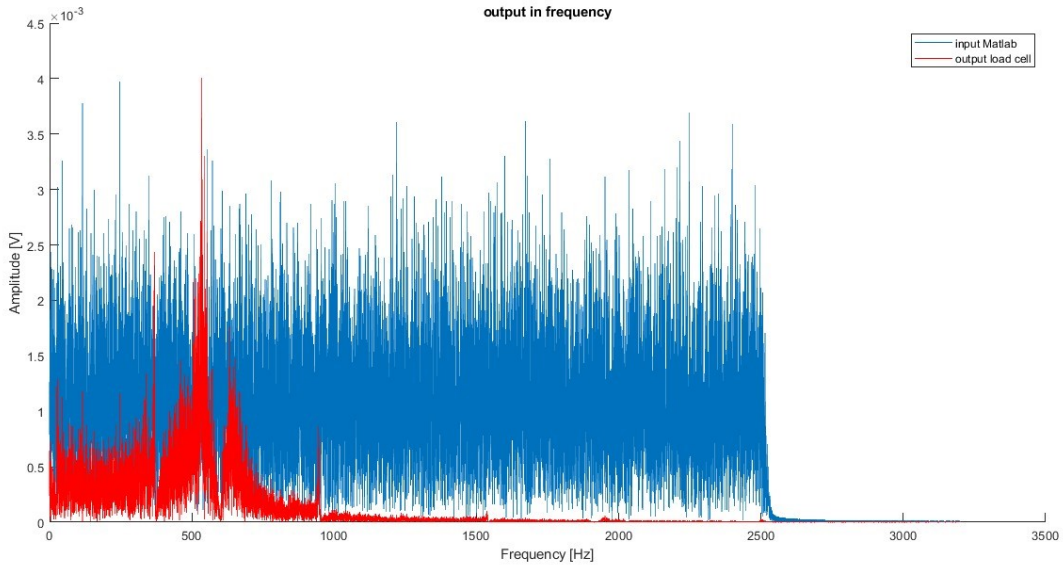



*Figure 4.6 High frequency comparison between input and output of the pseudorandom signal in the plate*

The preceding figures depict a comparison, both for the beam and the plate, at low and high frequencies, between the input signal generated by MATLAB and sent to the shaker, and the actual output registered by the load cell. As discernible from the graphs, a strong correspondence between the MATLAB-generated input signal and the actual signal read by the load cell is notably lacking. Consequently, this disparity results in experimental values that differ from those obtained numerically. Furthermore, unlike the previous case, it was not feasible to generate a signal that could closely resemble what is read by the load cell.

## 4.3 Harmonic Signal

In data acquisition, a harmonic signal is a type of periodic signal that consists of one or more sinusoidal components with frequencies that are integer multiples of the fundamental harmonic. Understanding and analyzing harmonic signals are fundamental in studying the behavior of linear, electrical, and electromagnetic systems, as well as in many other areas of engineering and physics.

A harmonic signal can be described by the following general formula:

$$x(t) = A \cdot \sin(2\pi f t + \phi)$$

where:

- $x(t)$ is the value of the signal at time t.
- $A$ is the amplitude of the sinusoidal wave, representing the signal's excursion.
- $f$ is the frequency of the fundamental harmonic.
- $t$ is the time.
- $\varphi$ is the initial phase of the signal, representing the phase shift relative to the temporal origin.

In addition to the fundamental harmonic at frequency f, the harmonic signal can also contain higher harmonics, which are sinusoidal components with frequencies that are multiples of frequency. For instance, if the fundamental harmonic is f, the second harmonic will be at 2f, the third at 3f, and so on. The amplitudes and phases of these harmonics depend on the properties of the initial signal.

The harmonic signal has a discrete and regular frequency spectrum, with peaks at the frequencies of the various harmonics. The frequency spectrum of a harmonic signal is useful for analyzing and identifying the various components of the signal.

Harmonic signals are often used in data acquisition to perform tests, analyses, or evaluations of equipment, systems, or components. For example, it is common to use harmonic signals to calibrate measuring instruments, analyze the frequency response of an electrical or electronic circuit, or characterize the behavior of a mechanical system subjected to periodic vibrations.

### 4.3.1 Comparison between harmonic signal input from MatLab and output load cell

The input signal sent to the shaker was generated by the following line of MATLAB code:

$$output = Amp * cos(2 * pi * fre * t')$$

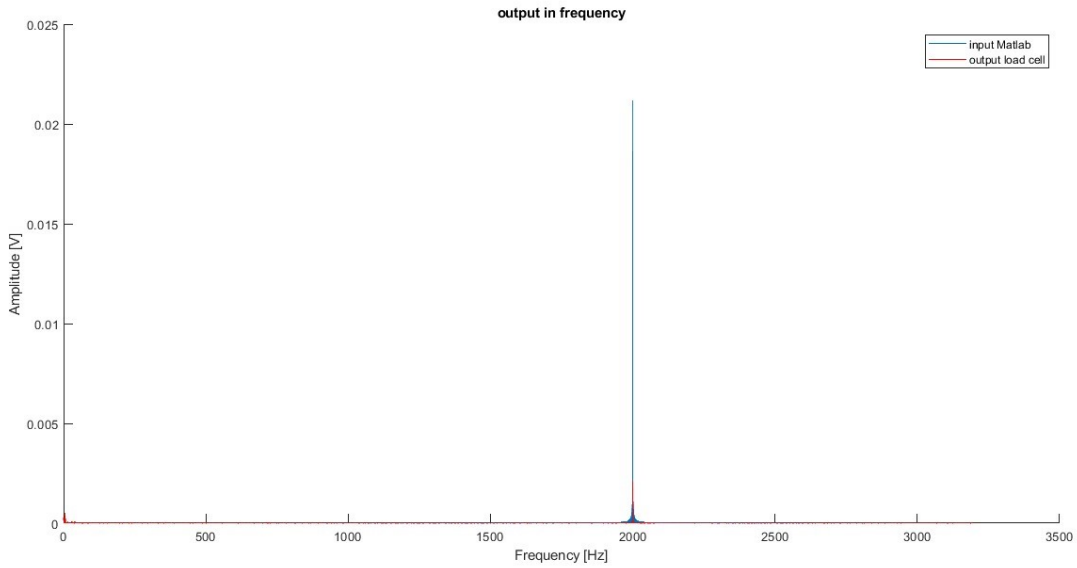



*Figure 4.7 Frequency comparison between input and output of the harmonic signal in the beam*

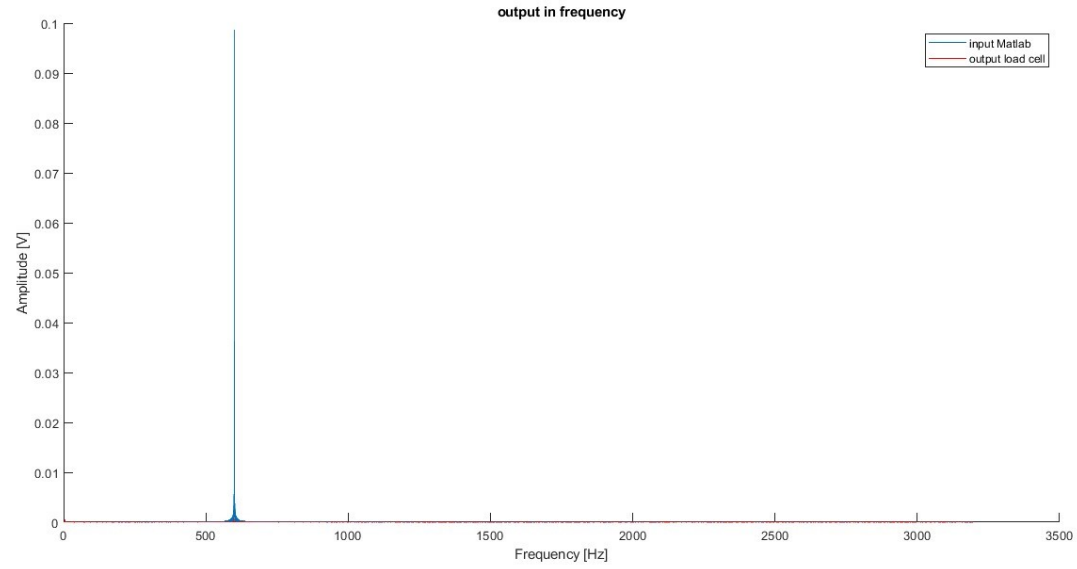



*Figure 4.8 Low frequency comparison between input and output of the harmonic signal in the plate*

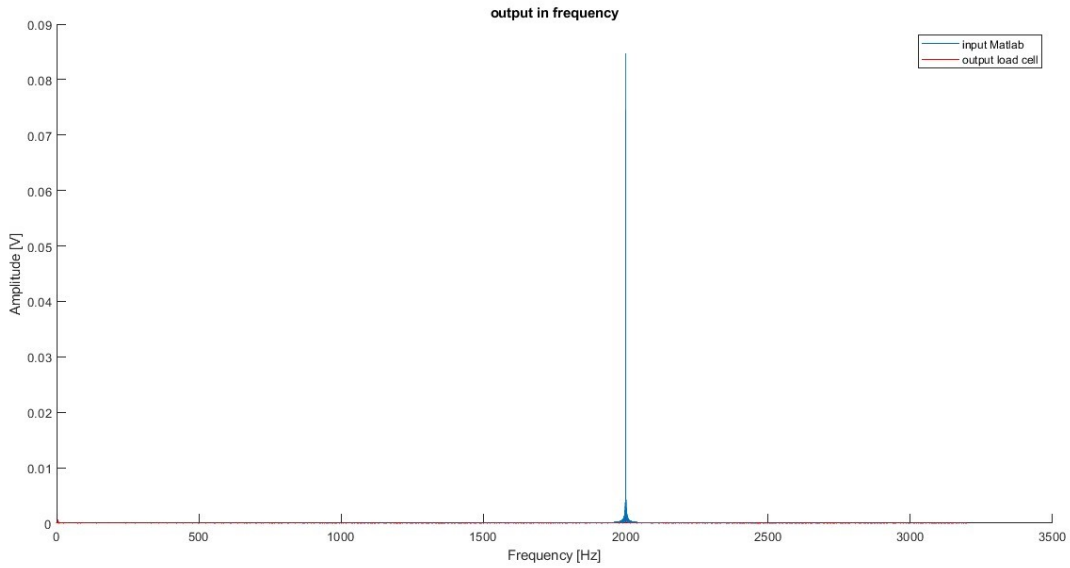



*Figure 4.9 High frequency comparison between input and output of the harmonic signal in the plate*

The previous figures illustrate the comparison, both for the beam and the plate, at low and high frequencies, between the input generated by MATLAB sent to the shaker and the actual output registered by the load cell. In contrast to the previous cases, a noticeable improvement in the correspondence between the MATLAB-generated input and the actual signal detected by the load cell can be observed in the graphs. This results in fewer errors compared to the previous cases.

## 4.4 Number Average

The "Number average" parameter allows setting the number of acquisitions to be performed before averaging the signal. The appropriate value of Number average depends on the nature of the signal and noise, as well as the application's specifications.

In general, a larger value of Number average will provide a more stable average and further reduce the influence of noise, but it will require more time to complete the acquisition and averaging process. On the other hand, a value that is too small may not significantly reduce the noise or provide a reliable estimate of the desired signal.

The choice to acquire the signal multiple times is since in most data acquisition systems, the signal we want to measure is often contaminated by noise. Noise consists of unwanted or random signals that can be caused by various sources, such as electronic interference, environmental instabilities,

or other sources of disturbance. The noise can vary randomly from acquisition to acquisition, and its level can be much lower than the amplitude of the desired signal.

As the noise varies randomly between different acquisitions, averaging multiple acquisitions tends to reduce the effect of noise, as noise has a stochastic nature. The component of the signal that is coherent among acquisitions (i.e., the desired signal) tends to accumulate in the average.

The "Number average" parameter allows setting the number of acquisitions to be performed before averaging the signal. The appropriate value of Number average depends on the nature of the signal and noise, as well as the application's specifications.

In general, a larger value of Number average will provide a more stable average and further reduce the influence of noise, but it will require more time to complete the acquisition and averaging process. On the other hand, a value that is too small may not significantly reduce the noise or provide a reliable estimate of the desired signal.

The choice to acquire the signal multiple times is since in most data acquisition systems, the signal we want to measure is often contaminated by noise. Noise consists of unwanted or random signals that can be caused by various sources, such as electronic interference, environmental instabilities, or other sources of disturbance. The noise can vary randomly from acquisition to acquisition, and its level can be much lower than the amplitude of the desired signal.

As the noise varies randomly between different acquisitions, averaging multiple acquisitions tends to reduce the effect of noise, as noise has a stochastic nature. The component of the signal that is coherent among acquisitions (i.e., the desired signal) tends to accumulate in the average.

All of this will subsequently allow for better signal processing.

## 4.5 Trigger

In data acquisition, a "trigger" is a mechanism or signal used to initiate the start of the data acquisition in synchronization with a specific event or desired condition. The trigger is crucial to ensure that data acquisition begins at the right moment, capturing exactly the relevant data and reducing unwanted noise or interference.

The desired event or condition that must occur for data acquisition to start is specified. This event can be a voltage change, a rising or falling edge, a specific value reached by a signal, or any other specific condition relevant to the purpose of acquisition.

Once the trigger is activated, data acquisition is initiated in synchronization with the detected event. This ensures that the captured data is relevant and consistent with the specific event being studied or analyzed.

In the case of the beam experiment, an initial intensity peak was observed when the input was initially applied. This was likely due to the beam's thickness being too thin, leading to data acquisition starting after the first 0.15 seconds.

## 4.6 Dataset Creation

In the development of a surrogate model aimed at learning stress distributions from simulations conducted using the Finite Element Method (FEM) to predict real-time stress distributions, the neural network algorithm relies on predicting accelerations along the structure nodes. This is achieved through accelerometers positioned along the structure, monitoring time-variable behavior. Following machine learning, the algorithm can then predict the structure's response at various time points and positions.

The dataset comprises approximately one million entries obtained from various FEM simulations based on the examined signals. These data serve as input for the algorithm. At each time step, the machine learning algorithm establishes a relationship between the desired numerical response and the calculated reference quantities, either using FEM in the numerical case or accelerometer values in the experimental case.

In the case of the beam modeled with the FEM, 50 nodes were used along its vertical axis, and the finite element simulation was executed using Matlab. For algorithm training, seven factors were employed as input: the $x$ coordinate positions of nodes, acceleration ($acc$), and $p1, p2, p3, p4, p5$, representing the five accelerometers positioned at different locations along the beam.

In contrast, for the plate, unlike the beam, the finite element simulation to obtain numerical values was performed using Ansys Workbench with the "transient structural" tool. Eighty-eight nodes were

used, and the input parameters included the $x$ and $y$ coordinates for node positions, acceleration ($acc$), and $p1, p2, p3, p4, p5, p6, p7$, representing the seven accelerometers used for the plate.

The generated data is divided into two sets: a training set and a test set. This division allows training the model on the training data and subsequently evaluating its performance on the test set. Data splitting is accomplished through a stratified sampling method. The training set comprises 80% of the data, while the test set includes the remaining 20%. Although 70%-30% simulations will also be conducted to observe potential differences. This approach is crucial for assessing the overall accuracy of the final model.

# 5 Experiment model beam

## 5.1 Experiment Preparation

During the experiment a rectangular bar was analyzed. It was a rectangular aluminum bar measuring 175 mm in length, 19.16 mm in width, and 3.10 mm in thickness.

The first step involved ensuring the cleanliness of the bar, free from any dust or dirt. Subsequently, a thorough examination of the aluminum bar was conducted to identify any surface defects such as scratches, dents, or other imperfections.

Following this analysis, the next step was to subject the bar to a surface defect removal process. Two different grades of sandpaper were selected for this purpose. During the sanding process, deliberate and consistent movements were applied. Initially, coarse-grit sandpaper was used, gradually shifting to finer grit as surface defects were removed.

Upon completing the finishing process, it was essential to clean the surface from any residual dust using a solvent and a cloth while wearing gloves to prevent contamination of the bar.

At the conclusion of these procedures, a significantly improved aluminum bar with a flawless surface was obtained.

Once the bar was prepared and cleaned, it was initially positioned vertically on a support and secured with a clamp. Three accelerometers with varying sensitivities were placed at different positions along the bar. Additionally, a load cell was installed and subsequently connected to the shaker located at the same height, allowing for perpendicular force application to the beam.

*Figure 5.1 Old beam configuration*

However, this configuration proved inadequate due to the clamp's inability to secure the bar adequately. This resulted in various errors caused by vibrations stemming from the mounting support. Consequently, the decision was made to adopt the configuration proposed in Figure 5.2.

In this case, improving the bar's fixation became necessary, as illustrated in the following figure:

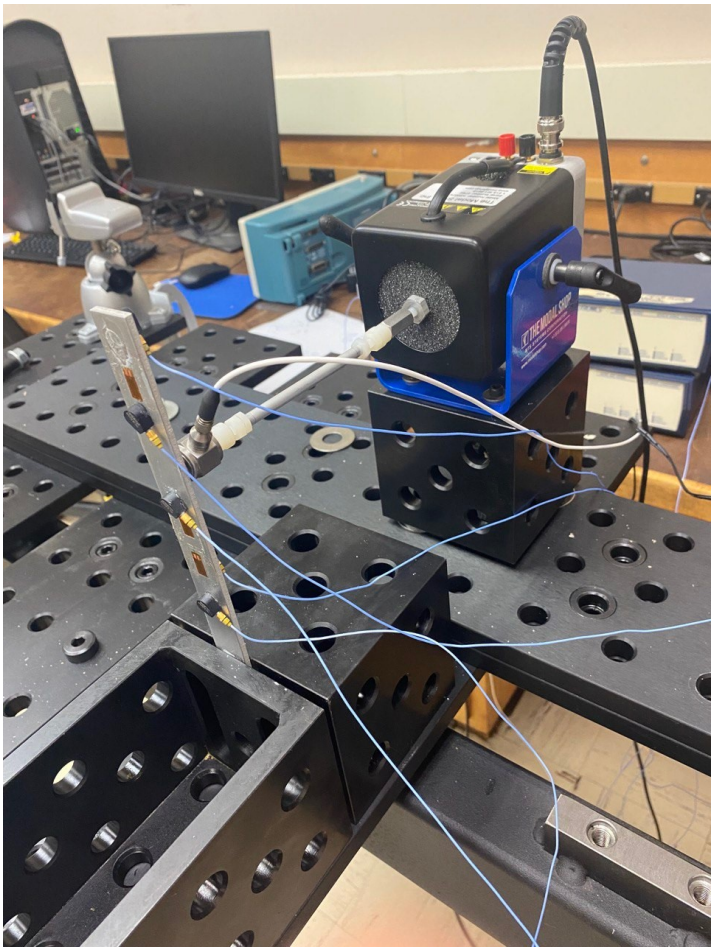


*Figure 5.2 New beam configuration*

To circumvent the problems encountered in the previous configuration, as evident in the figure, the bar's fixation was facilitated by two black blocks that were securely fastened to the bar.

Onto this properly prepared beam, five accelerometers were positioned:

- Accelerometer model 352A24 SN LW 369406 with sensitivity $9.94\ mV/m/s^2$
- Accelerometer model 352A24 SN LW 369402 with sensitivity $9.87\ mV/m/s^2$
- Accelerometer model 352A24 SN LW 339401 with sensitivity $10.00\ mV/m/s^2$
- Accelerometer model 352A24 SN LW 369399 with sensitivity $9.96\ mV/m/s^2$
- Accelerometer model 352A24 SN LW 369404 with sensitivity $9.88\ mV/m/s^2$

And a Load cell model 208C01 SN LW 55202 with sensitivity $500\ mV/lb$.

They were positioned as shown in the schematic diagram:

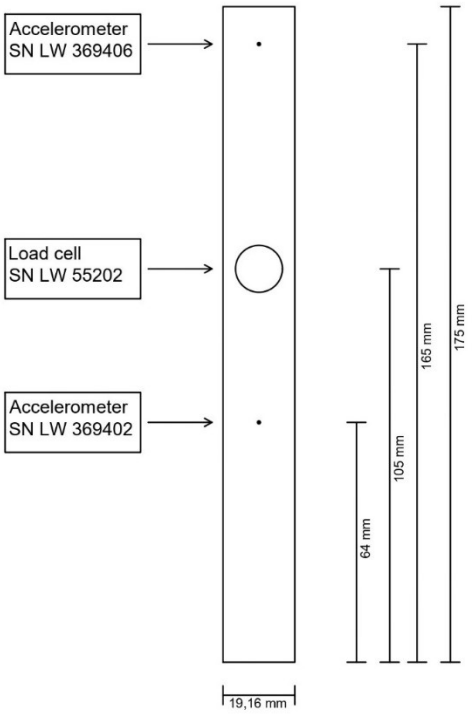



*Figure 5.3 Configuration of accelerometer and load cell placement*

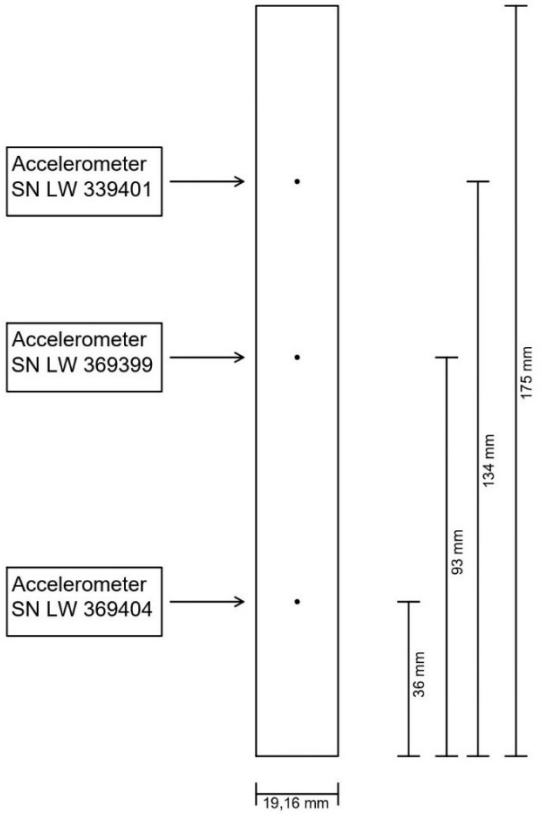



*Figure 5.4 Configuration of accelerometer placement*

As shown in Figure 5.2, the load cell was connected to the shaker, which is positioned in a structure at the same height, to apply force perpendicular to the bar.

Both the 5 accelerometers and the load cell were connected to a sensor signal conditioner. It was necessary to introduce this device so that the sensor signals, through the conditioning circuits, could be filtered and isolated from various environmental factors, such as noise and electrical interference. In this way, an effort was made to improve the signal by making it clean and free from disturbances, thus enhancing the quality of measurements.

Subsequently, the signal was passed to the CompactDAQ, connected to the computer via a USB cable. The signal was acquired using CompactDAQ and the MatLab 2021b tool.

In the MatLab code developed, it is possible to set the acquisition frequency, signal length, number of signal acquisitions for averaging, cut-off frequency, range, amplitude, and output type.

It was observed that it is not possible to set the acquisition frequency arbitrarily because the CompactDAQ module operates only within specific ranges according to the formula: [20]

$$f_s = \frac{f_M \div 256}{n}$$

Where:

- $f_M$ frequency of a master timebase
- $f_s$ data rate (fs)
- $n$ is any integer from 1 to 31.

This issue may arise due to various reasons, depending on the system configuration and the use of National Instruments (NI) driver or APIs with MATLAB. For instance, one of the issues is the inherent hardware limitations of National Instruments, which can affect its ability to acquire or generate signals at certain sample frequencies. Therefore, it's essential to check the hardware's technical specifications before proceeding with data acquisition to ensure it can operate at the desired sampling frequencies.

Another problem involves rounding errors. In some situations, there might be a rounding or approximation error in the frequencies set through the NI driver or API, resulting in slightly different actual frequencies. Alternatively, there may be an error in the configuration of the NI driver or APIs with MATLAB, preventing the system from functioning correctly at specific sampling frequencies. As a result, a decision was made to use the frequency of 8400 Hz, which was suitable for the dq.Rate.

All the signals presented in the following summary table 1 have been sampled using:

- $Acqusition\ rate\ 6400\ Hz$
- $Lenght\ scane = 4\ s$
- $N.average = 2$
- $Trigger = 0.15\ s$

| $Name$ | $Signal$ | $Frequency\ [Hz]$ | $Amplitude\ [V]$ |
|---|---|---|---|
| $CH1$ | $Chirp$ | 2000 | 0.05 |
| $CH2$ | $Chirp$ | 600 | 0.05 |
| $CH3$ | $Chirp$ | 2000 | 0.025 |
| $CH4$ | $Chirp$ | 600 | 0.025 |
| $PS1$ | $Pseudorandom$ | 2000 | 0.05 |
| $PS2$ | $Pseudorandom$ | 600 | 0.05 |
| $PS3$ | $Pseudorandom$ | 2000 | 0.025 |
| $PS4$ | $Pseudorandom$ | 600 | 0.025 |
| $AR1$ | $Harmonic$ | 2000 | 0.05 |
| $AR2$ | $Harmonic$ | 600 | 0.05 |
| $AR3$ | $Harmonic$ | 2000 | 0.025 |
| $AR4$ | $Harmonic$ | 600 | 0.025 |

*Table 1*

## 5.2 Material

The Young's modulus of aluminum can vary slightly based on the specific composition of the aluminum used, processing conditions, and measurement methods. Sometimes, to simplify analyses or consider a broader range of situations, an average or approximate value, such as 60 GPa, may be used.

In some engineering applications, it is prudent to use conservative values to ensure that the material is sufficiently safe and reliable. Using a value slightly lower than the theoretical one (e.g., 60 GPa instead of 72 GPa) can provide an additional safety margin in the design and evaluation of material performance.

The material composing the beam is aluminum, and it was not possible to ascertain the exact composition or precise alloy used. Therefore, for the purpose of analysis and data processing, it was set to $E = 60\ GPa$.

## 5.3 Damping

The differential equation that describes the behavior of a system with proportional damping is expressed as follows:

$$[M]\{\ddot{u}\} + [C]\{\dot{u}\} + [K]\{u\} = \{f(t)\}$$

- $[M]$ is the mass matrix, a diagonal n x n matrix with diagonal elements representing the masses of individual degrees of freedom.
- $[C]$ is the damping matrix.
- $[K]$ is the stiffness matrix, an n x n matrix representing the system's stiffness.
- $f(t)$ is the vector of external forces applied to the system at time t.
- $u$ is the vector of positions (or amplitudes) at time t, a column vector of size n x 1.
- $\dot{u}$ is the vector of velocities at time t, a column vector of size n x 1 representing the derivatives of positions with respect to time.
- $\ddot{u}$ is the vector of accelerations at time t, a column vector of size n x 1 representing the derivatives of velocities with respect to time.

The proportional damping model, also known as Rayleigh damping, [31] was used, where the damping matrix $[C]$ is a linear combination of the mass matrix $[M]$ and the stiffness matrix $[K]$:

$$[C] = \alpha[M] + \beta[K]$$

With $\alpha$ e $\beta$ being the Rayleigh damping constants.

The damping factor is equal to: $\xi = \frac{c}{2\sqrt{km}}$

While $\omega_i^2$ represents the square of the frequencies

The relationship that connects these three factors is:

$$2\xi_i\omega_i = \alpha + \beta\omega_i^2$$

The Rayleigh constants are determined experimentally by knowing the values of the damping and frequency factors:

$$\begin{cases} \alpha + \beta\omega_1^2 = 2\xi_1\omega_1 \\ \alpha + \beta\omega_2^2 = 2\xi_2\omega_2 \end{cases}$$

From which we derive in the case of the beam:

- $\alpha = 7 \cdot 10^{-7}\ 1/s$
- $\beta = 1 \cdot 10^{-6}\ s$

## 5.4 Comparisons of Machine Learning Simulations: Beam Case

Various signal samplings were performed through multiple experimental tests using different input signals such as chirp, pseudorandom, and harmonic. The signals listed in Table 1 were analyzed, and in the subsequent section, we will delve into the signals that yielded the most significant results.

### 5.4.1 Numerical-Experimental Chirp Signal Comparison

In this paragraph, a chirp signal with an amplitude of 0.05 V and a frequency of 2000 Hz was analyzed, comparing numerical values with experimental ones.

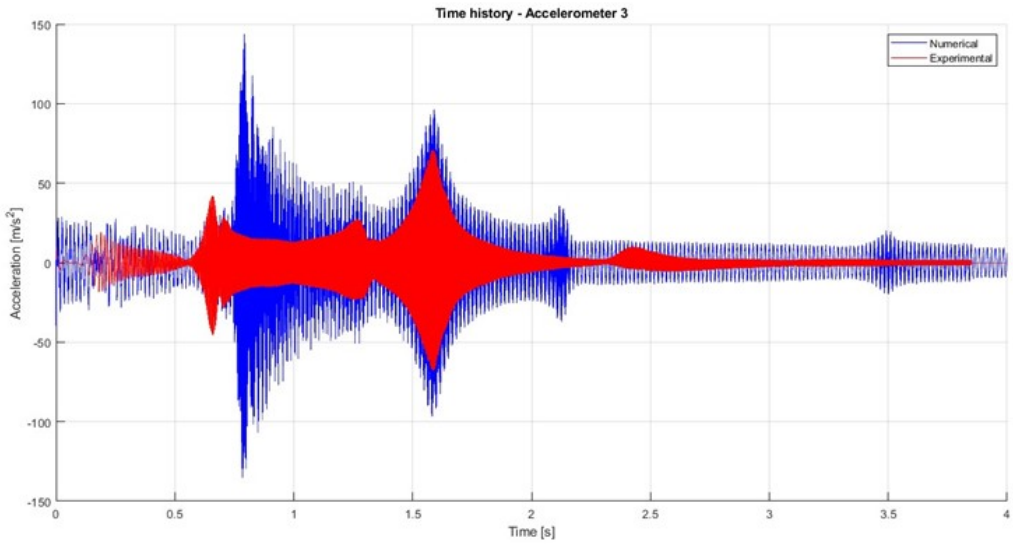



*Figure 5.5 Time history chirp signal – Accelerometer 3*

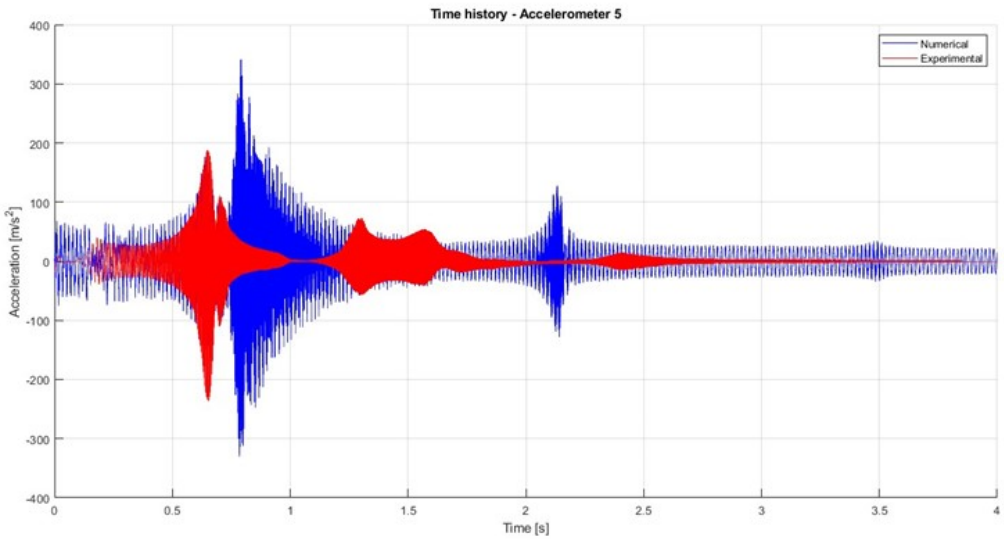



*Figure 5.6 Time history chirp signal – Accelerometer 5*

Among the five accelerometers used, the third, positioned at the midpoint of the analyzed beam, and the fifth, located at the tip of the same beam, were compared to provide an analysis at the most critical points of the beam.

From the preceding figures, it is evident that the experimental values are consistently lower than the predicted numerical values and do not entirely align with them, although the trend of the peaks is quite consistent with the numerical values obtained. This discrepancy could be attributed to several factors.

Firstly, the issue highlighted in the previous paragraph must be considered, where the load cell does not capture the same signal generated as the input to the shaker for excitation. This leads to a different acceleration value than expected because the input for numerical and experimental values differs due to the load cell's varied readings.

Additionally, the thickness and rigidity of the beam, which have significantly reduced values, can introduce irregularities in the data detected by the accelerometer. Another determining factor might be vibrations from the supports that secure the beam or an unstable table.

Other factors contributing less significantly to the disparity between numerical and experimental values include improper accelerometer calibration or incorrect installation on the beam. If the accelerometer is not positioned correctly, or if there are issues with fixing or orientation, the collected data will be adversely affected.

Other influential factors may include slight defects in the beam reducing rigidity, accelerometer limitations due to their own range, although this was addressed by reducing the amplitude of the function, and environmental conditions such as temperature and humidity that could have influenced the accelerometer.

These factors collectively help explain the observed divergences between experimental measurements and numerical predictions.

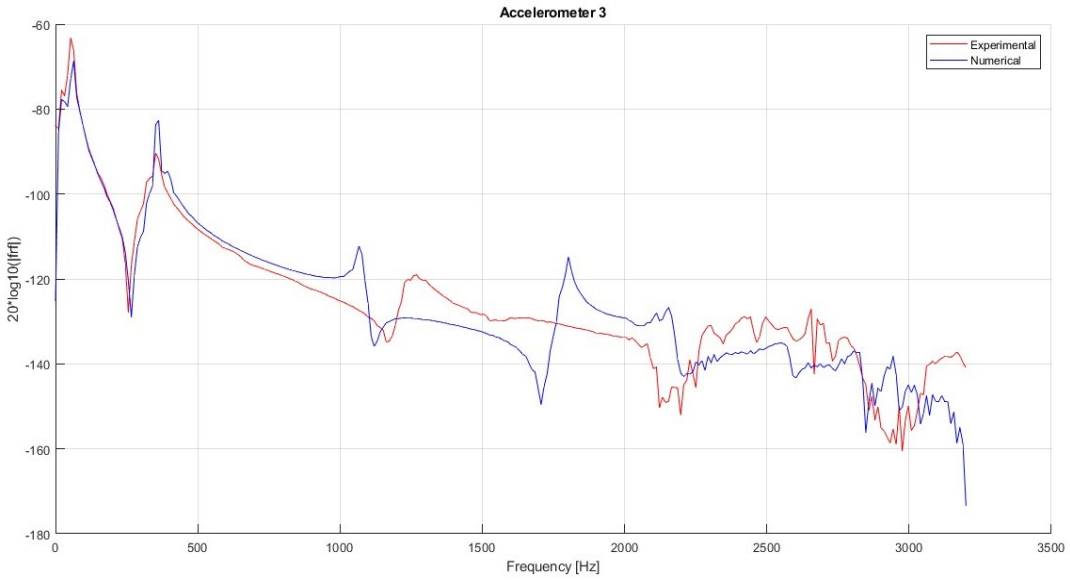



*Figure 5.7 FRF chirp signal – Accelerometer 3*

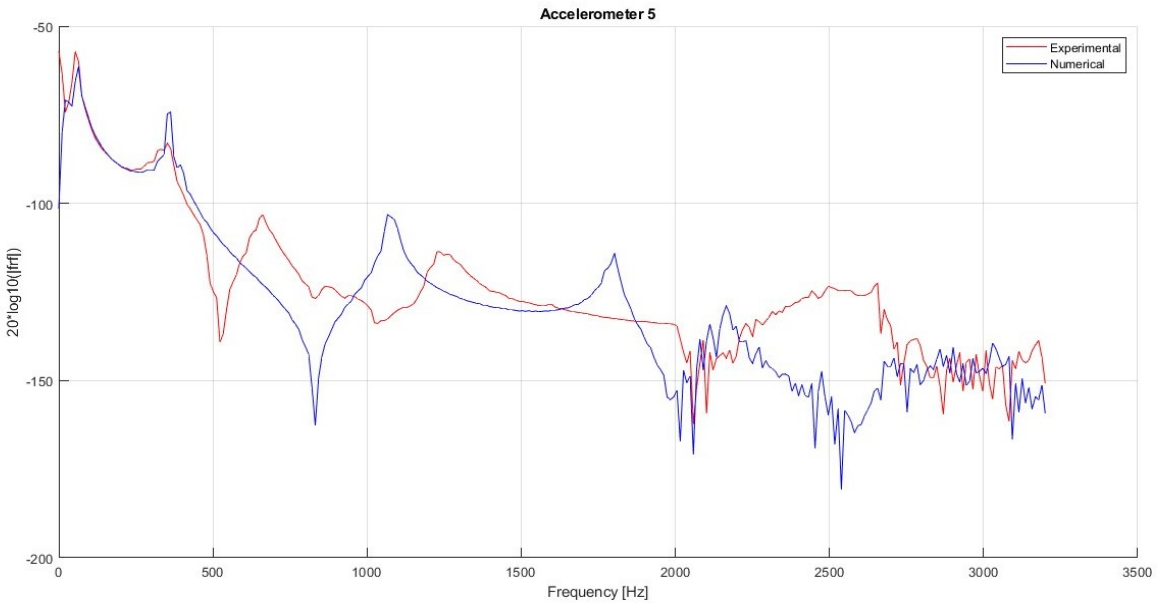



*Figure 5.8 FRF chirp signal – Accelerometer 5*

The trend of the peaks in the FRF of the two accelerometers is notably consistent between numerical and experimental values.

Subsequently, the results obtained from machine learning were compared between the numerical model of the accelerometers (on the left) and the experimental model of the accelerometer values (on the right):

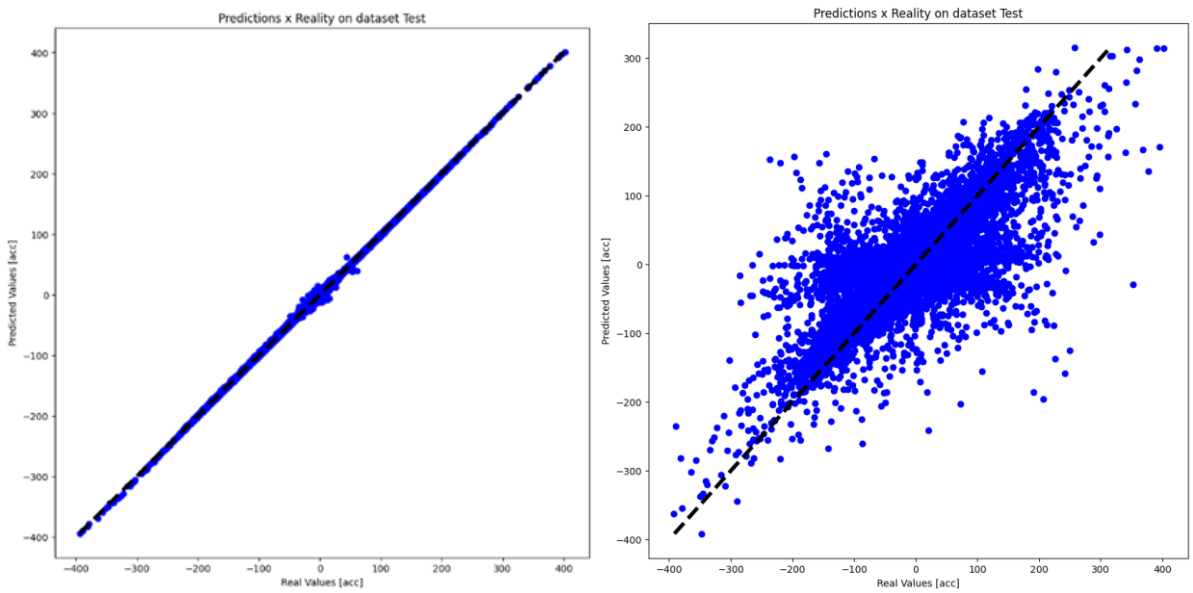



*Figure 5.9 Predictions x Reality on dataset test - L) Numerical case - R) Experimental case*

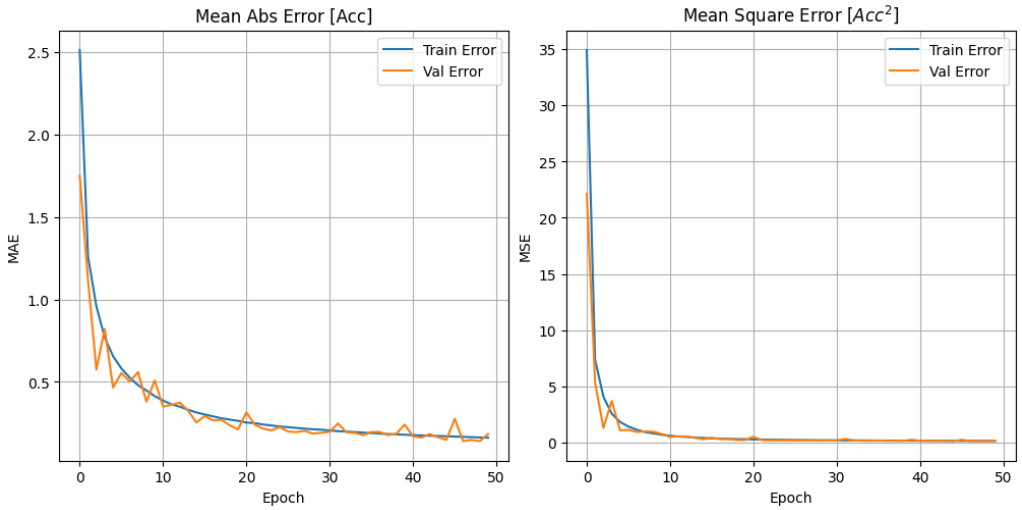



*Figure 5.10 Error evolution - Numerical case*

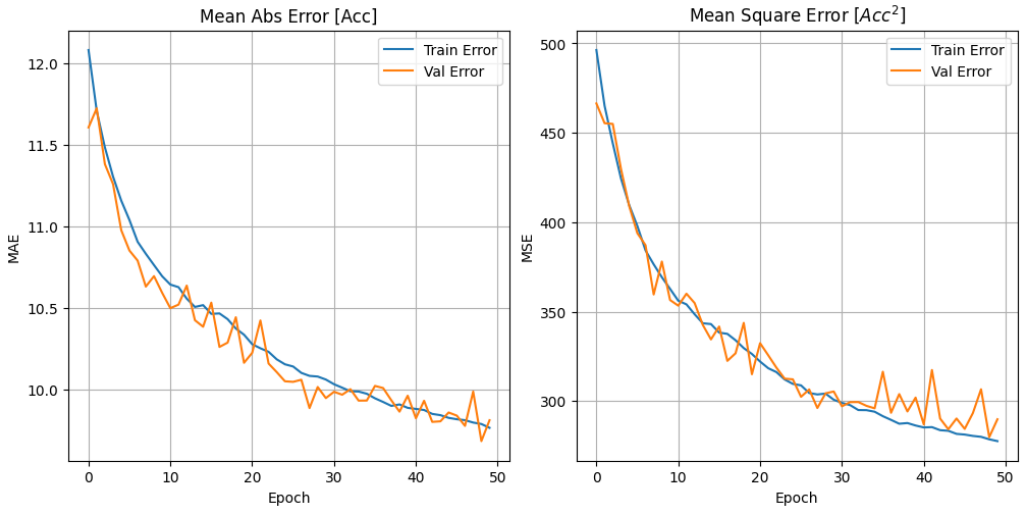



*Figure 5.11 Error evolution - Experimental case*

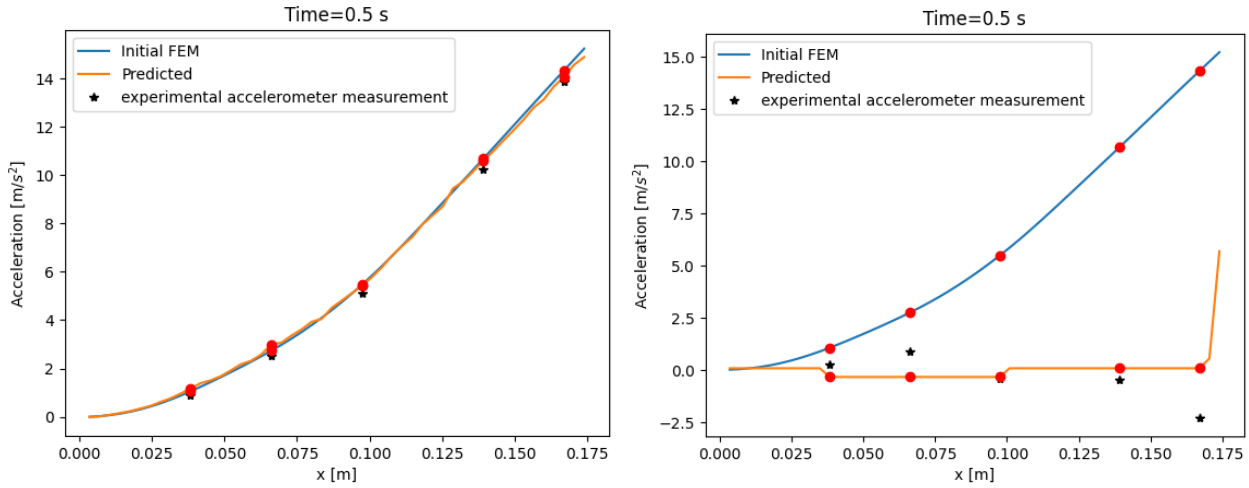



*Figure 5.12 Time=0.5 s - Beam acceleration - L) Numerical case - R) Experimental case*

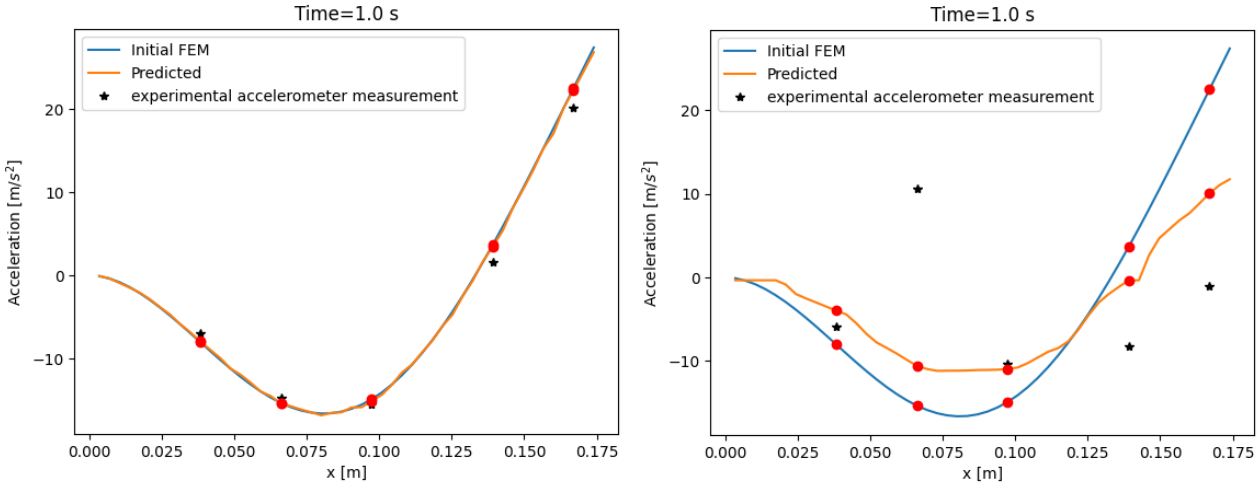



*Figure 5.13 Time=1.0 s - Beam acceleration - L) Numerical case - R) Experimental case*

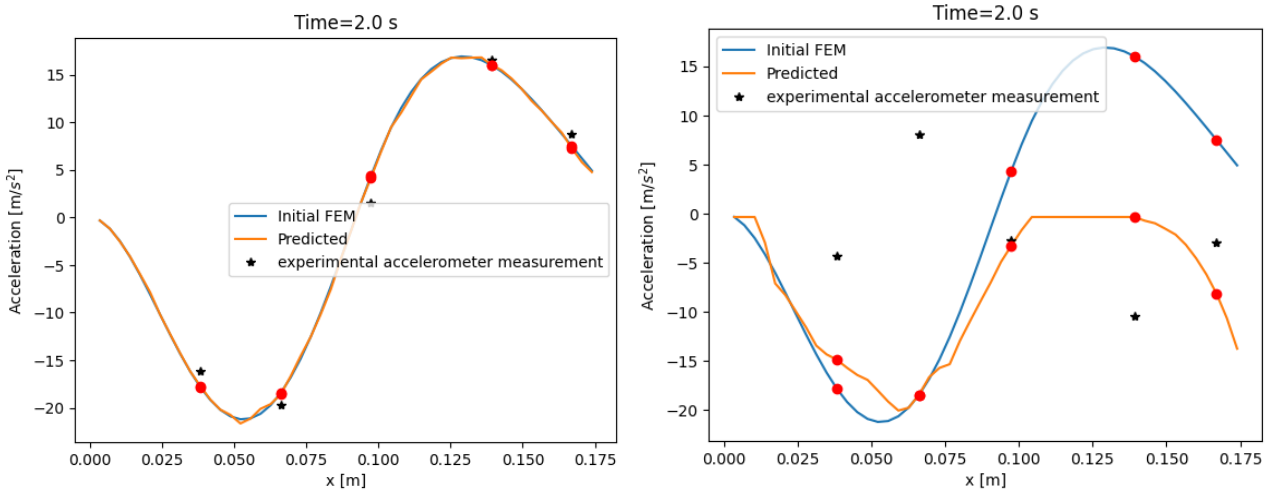



*Figure 5.14 Time=2.0 s - Beam acceleration - L) Numerical case - R) Experimental case*

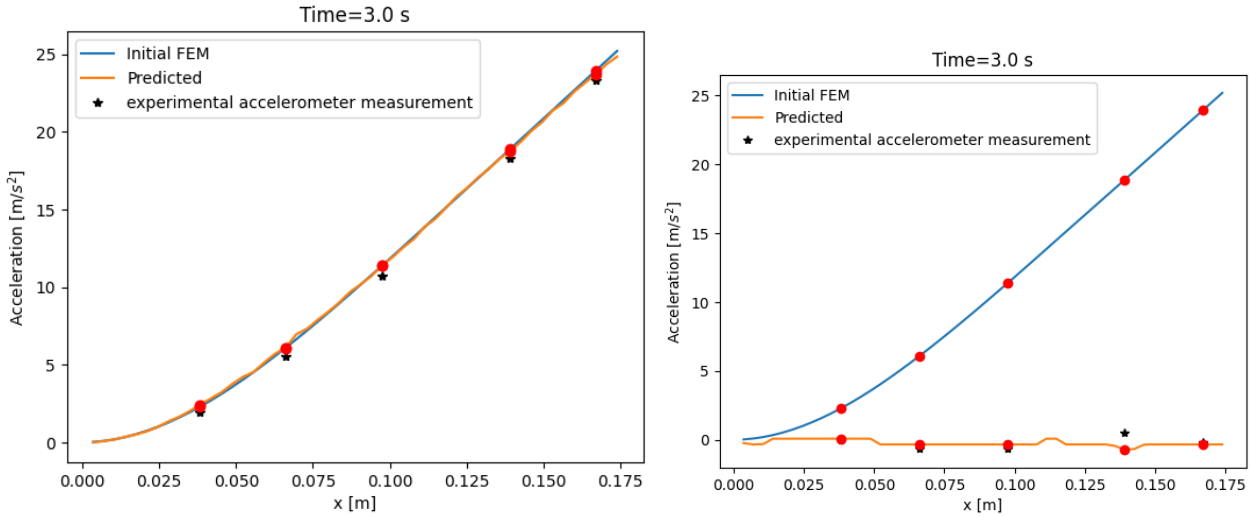



*Figure 5.15 Time=3.0 s - Beam acceleration - L) Numerical case - R) Experimental case*

In comparing the MAE and MSE graphs, it is evident that, in the case of numerical values, the line converges to zero after only 10 epochs, unlike the experimental case where significantly more epochs are required to reach zero. This is attributed to the fact that, in the experimental case, the values of the train error and value error are much higher than in the numerical case; consequently, a greater number of epochs are needed to bring the error values to convergence at approximately zero.

From all the graphs of the numerical model, it is observed that perfect values lead to the phenomenon of overfitting, supported by the obtained R2 value of 0.99. In fact, at each selected time instant, the curve of the initial FEM corresponds to that predicted by machine learning.

Regarding the experimental part, a good R2 is achieved, and the predicted values align reasonably well along the bisector. In the graphs at various time instants, particularly at 0.5 and 3 seconds, it is noticeable that the predicted acceleration curve values correspond to the data recorded by the accelerometers. This is because the experimental and numerical values at those time instants match more closely compared to the cases at 1 second and 2 seconds. However, the curve of the initial FEM and that of the predicted values are much closer than in the cases of 0.5 and 3 seconds.

### 5.4.2 Pseudorandom Signal Comparison

A pseudorandom signal with an amplitude of 0.05 V and a frequency of 2000 Hz was analyzed, comparing the numerical values with the experimental ones.

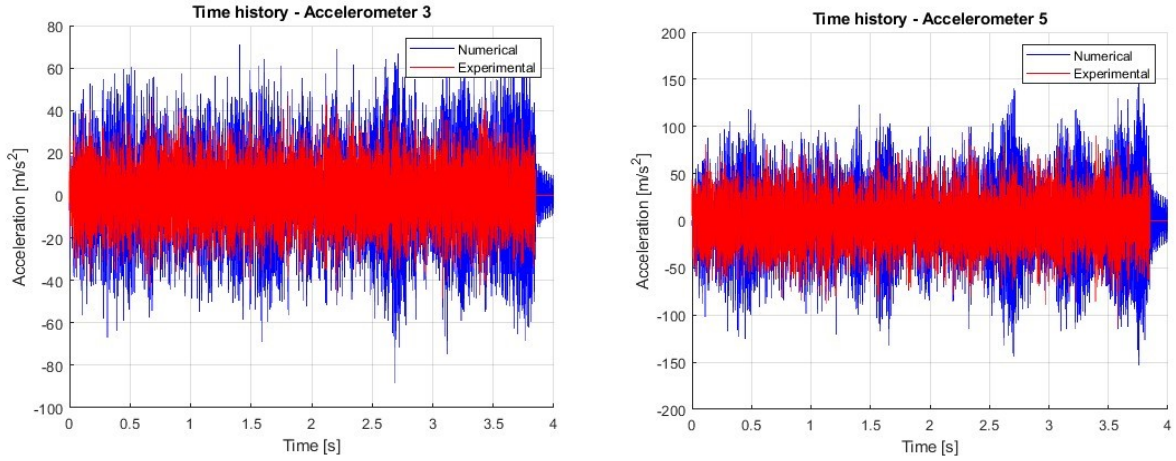



*Figure 5.16 Time history pseudorandom signal – L) Accelerometer 3 – R) Accelerometer 5*

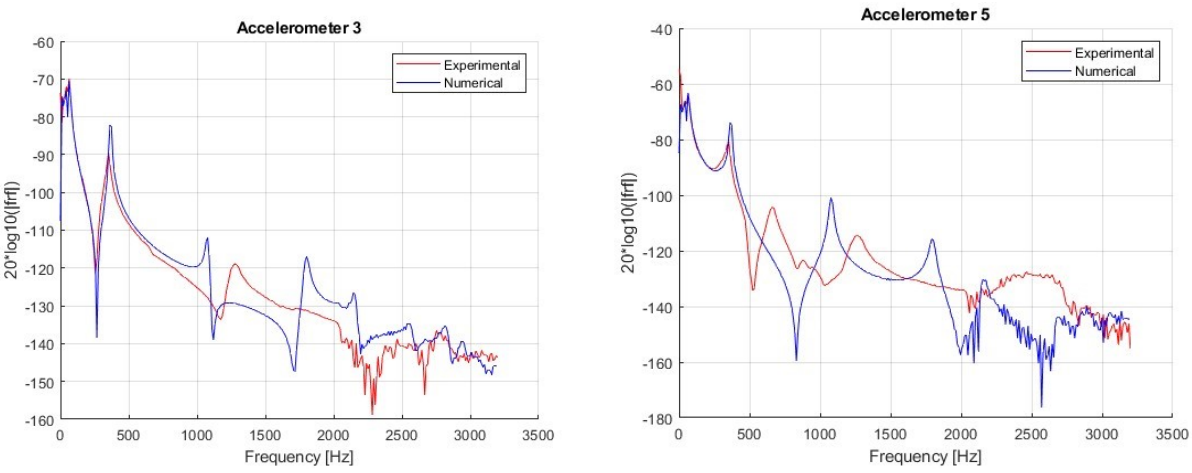



*Figure 5.17 FRF pseudorandom signal – L) Accelerometer 3 – R) Accelerometer 5*

The analysis of the comparison between experimentally obtained values and numerically calculated values for a pseudorandom signal represents a fundamental step in validating and optimizing the algorithm used. The results of this comparison provide valuable insights into the adequacy of

modeling and implementing the pseudorandom signal, as well as the ability of the experimental system to measure and acquire data accurately.

One of the most interesting aspects of this analysis is the opportunity to identify any deviations between experimental and calculated values. These discrepancies may arise from multiple sources, such as measurement errors or limitations in the system's ability to generate the pseudorandom signal. Identifying the cause of such differences is crucial to ensuring the reliability of the pseudorandom signal in the specific application. It should be emphasized that the analysis of comparison between experimental and calculated data is a standard practice in the design of pseudorandom signals, and its accurate execution is essential to ensure the reliability and effectiveness of systems that use them.

Similar to the chirp signal analysis, a difference between experimental and numerical values is observed here, attributed to plausible reasons explained earlier. It is noted that experimental values are lower than numerical values, but they fairly follow the pattern of pseudorandom peaks in both analyzed accelerometers. In fact, the FRF trends of the experimental and numerical lines coincide quite well, especially at low frequencies.

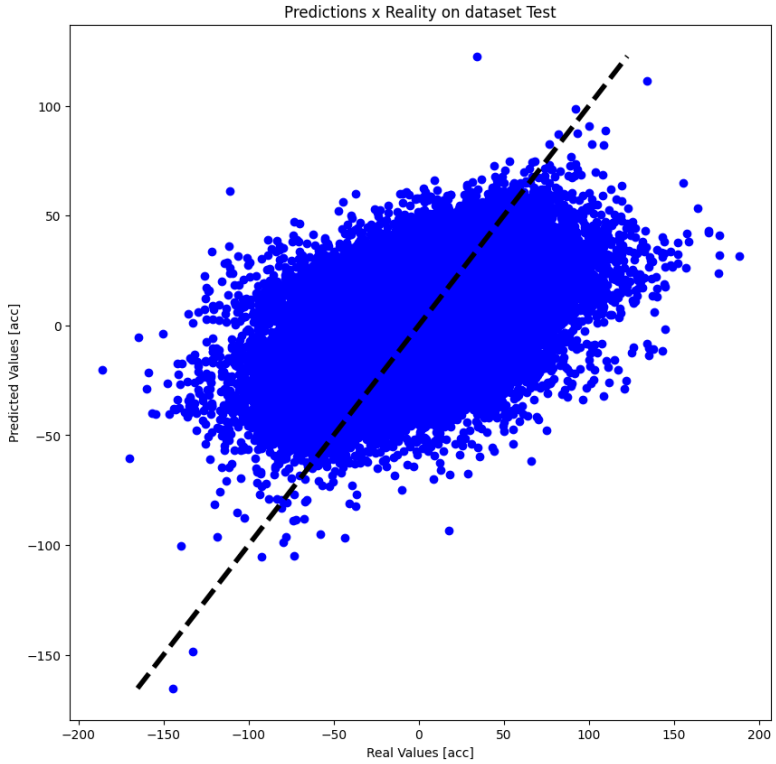



*Figure 5.18 Predictions x Reality on dataset test - Experimental case*

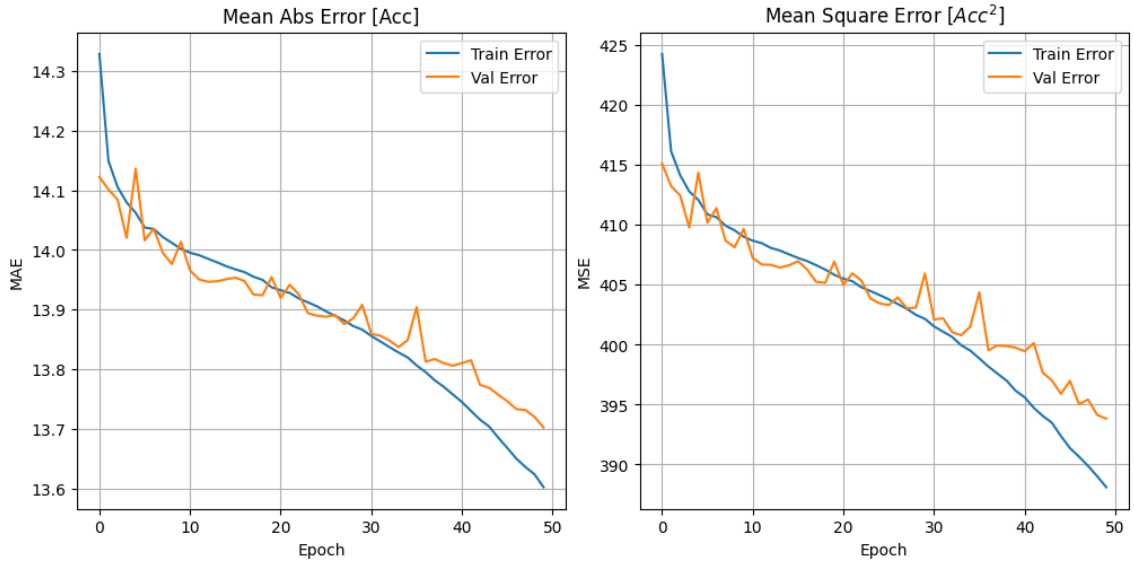



*Figure 5.19 Error evolution - Experimental case*

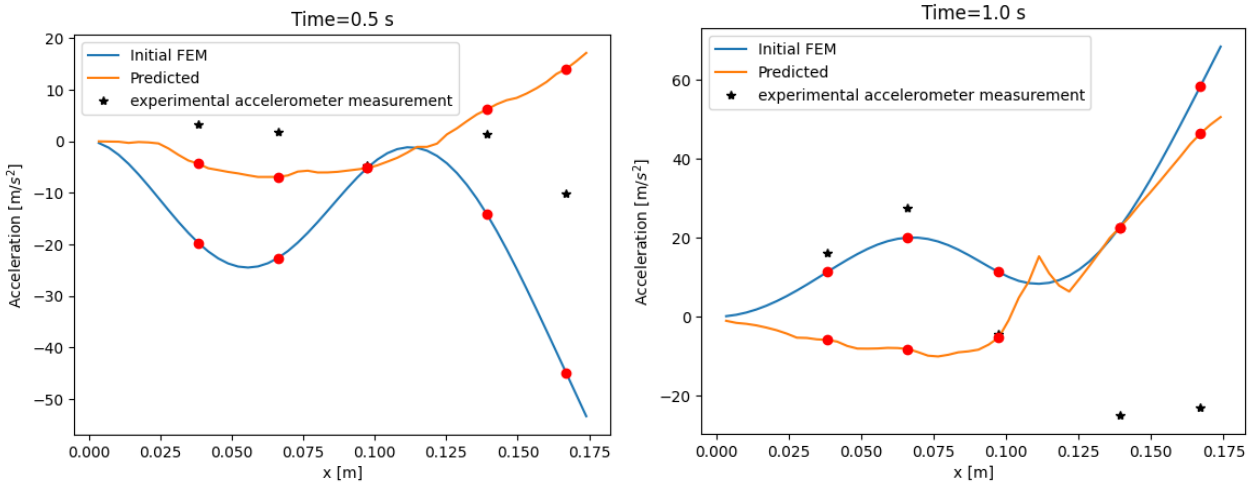



*Figure 5.20 Beam acceleration - Experimental case - L) Time=0.5 s - R) Time=1.0 s*

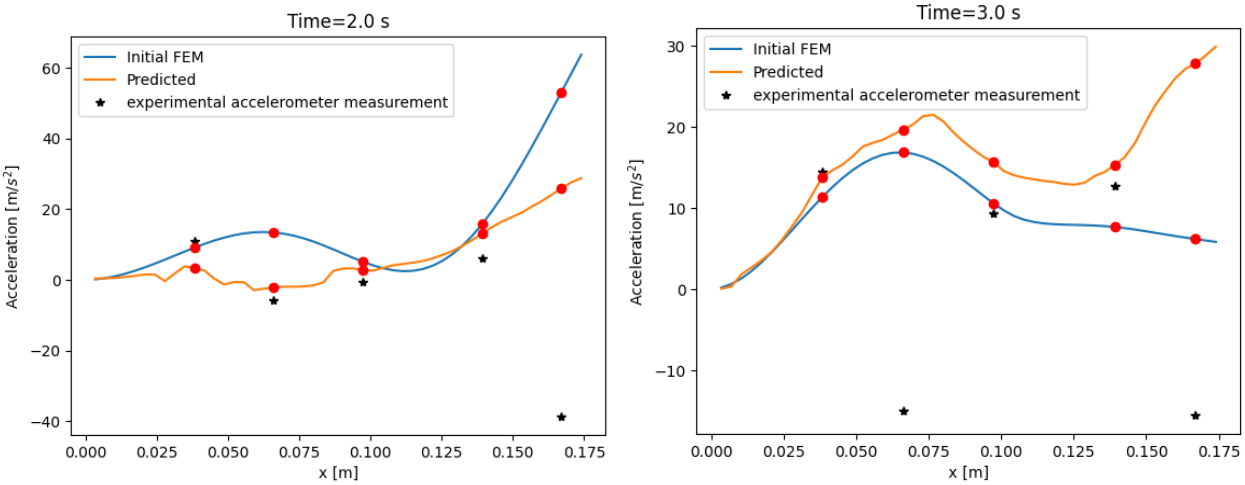



*Figure 5.21 Beam acceleration - Experimental case - L) Time=2.0 s - R) Time=3.0 s*

The Figure 5.18 should illustrate how the predicted values and the actual values are distributed uniformly along the bisector. In this case, most values concentrate in the central part of the bisector and less along its entire length. For this reason, as evident from Table 2, the R2 value is very low. Particularly near the tip of the beam, the predictive models do not closely match the actual values of the accelerometers.

From figure 5.19, it is evident that the 50 epochs used are not sufficient to achieve an error close to zero. The MAE values are still high, and the MSE values are elevated after 50 epochs, unlike the experimental chirp case where 50 epochs were enough to reach a value close to zero.

### 5.4.3 Harmonic Signal Comparison

A cosine signal with an amplitude of 0.05 and a frequency of 2000 Hz was analyzed, comparing numerical values with experimental ones.

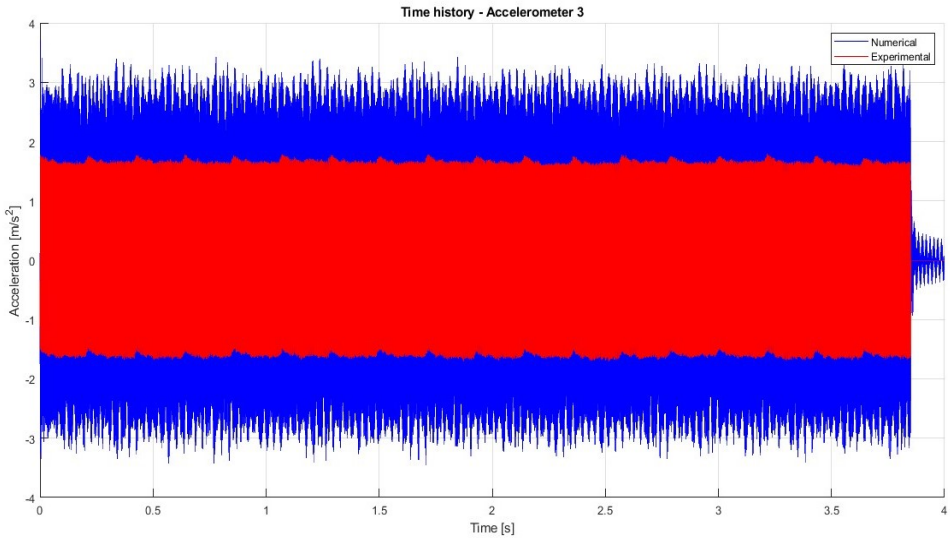



*Figure 5.22 Time history harmonic signal – Accelerometer 3*

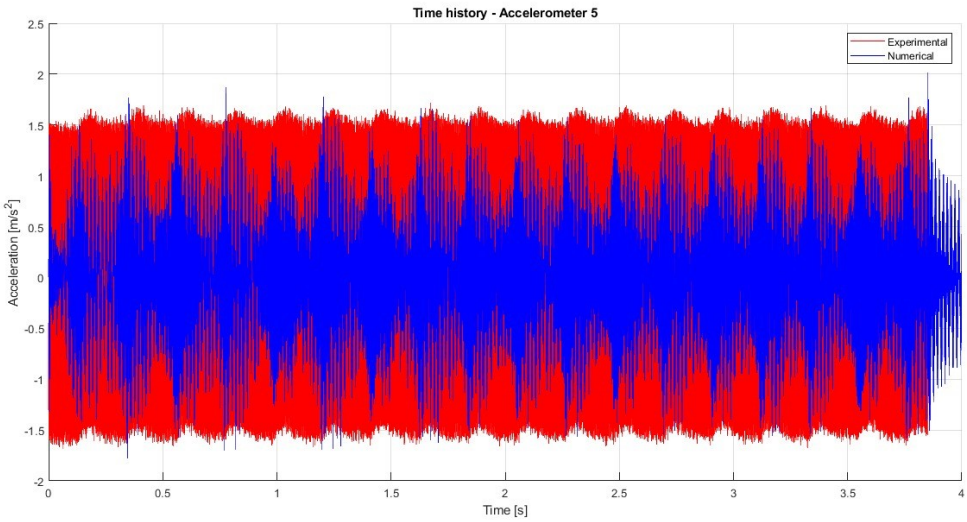



*Figure 5.23 Time history harmonic signal – Accelerometer 5*

In the recently examined case of a harmonic sine wave signal, the analysis reveals a good agreement between experimental and calculated data. This suggests that the employed model can accurately represent the behavior of the examined harmonic signal. This outcome is highly positive, confirming the robustness of the model and the reliability of experimental measurements.

Overall, a well-executed comparative analysis between experimental and calculated data for a harmonic signal contributes to an enhanced understanding of the phenomena under examination, strengthening the foundation for future decisions and developments based on such data.

In the case of the third examined accelerometer, experimental values are slightly lower than numerical values, but the peaks and the curve are in phase throughout the signal's course. Conversely, for the accelerometer at the tip, reasonably comparable values are obtained.

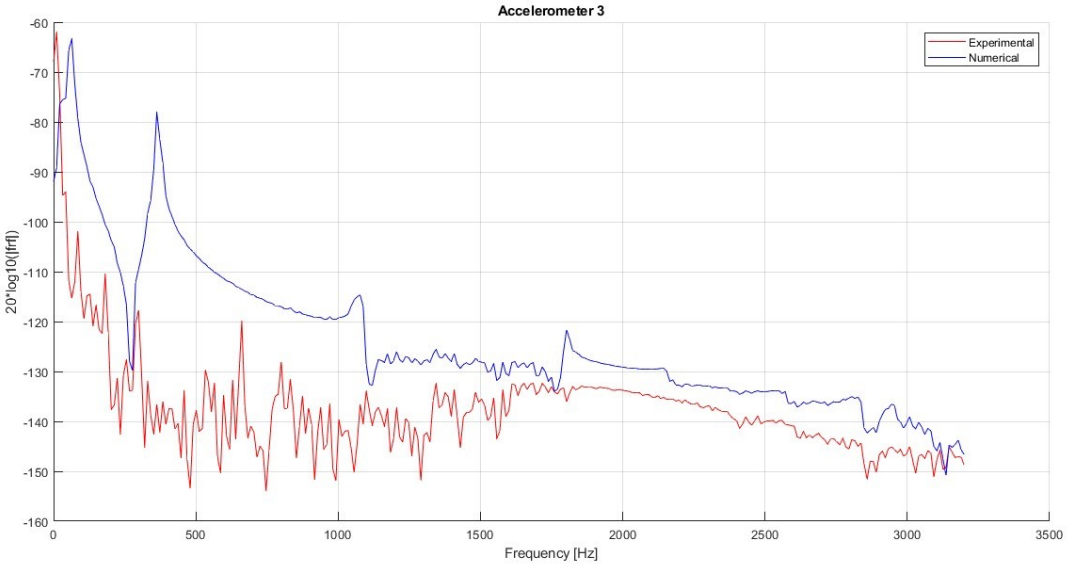



*Figure 5.24 FRF harmonic signal – Accelerometer 3*

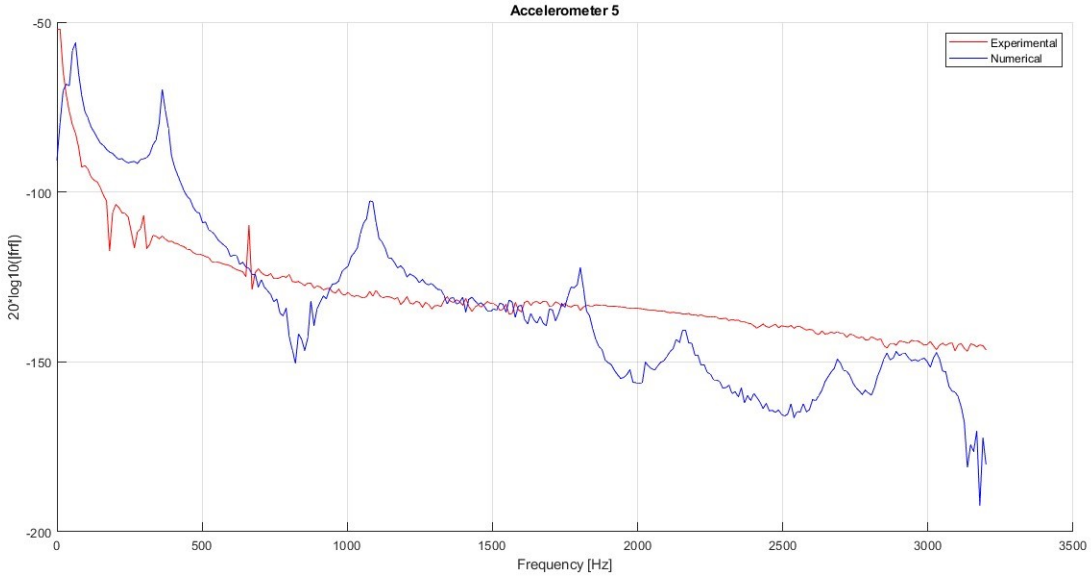



*Figure 5.25 FRF harmonic signal – Accelerometer 5*

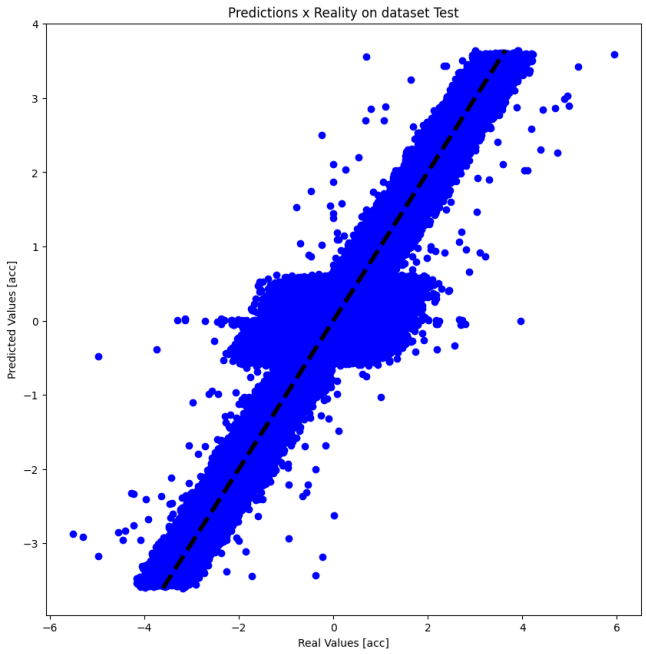



*Figure 5.26 Predictions x Reality on dataset test - Experimental case*

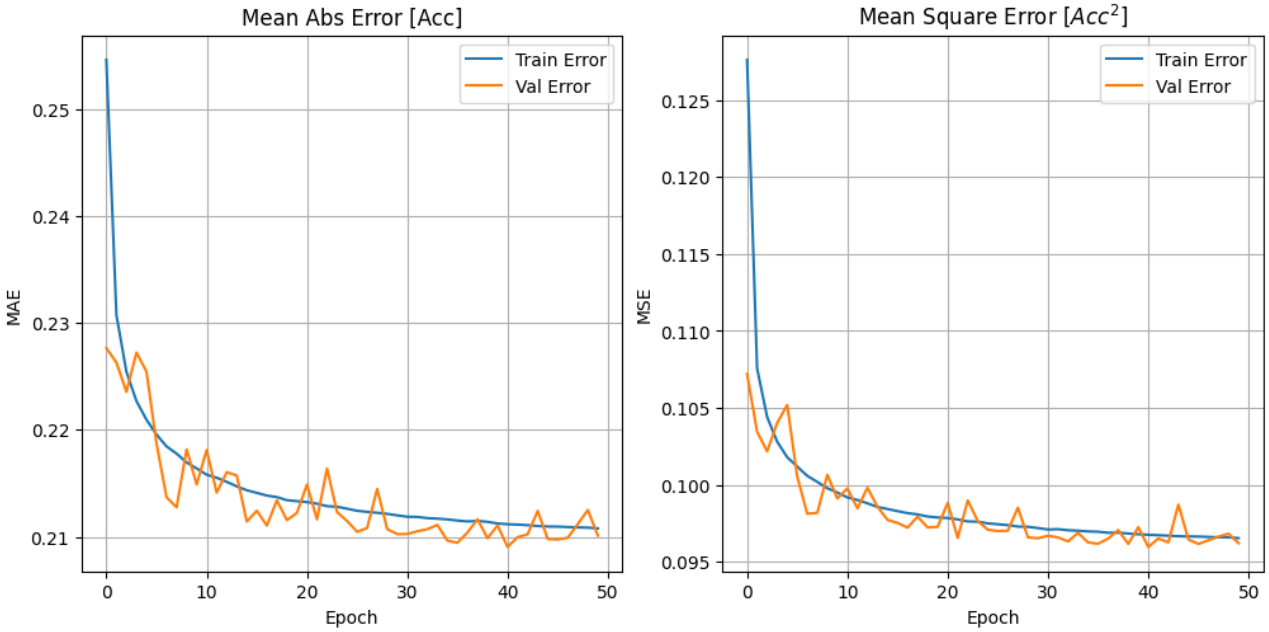



*Figure 5.27 Error evolution - Experimental case*

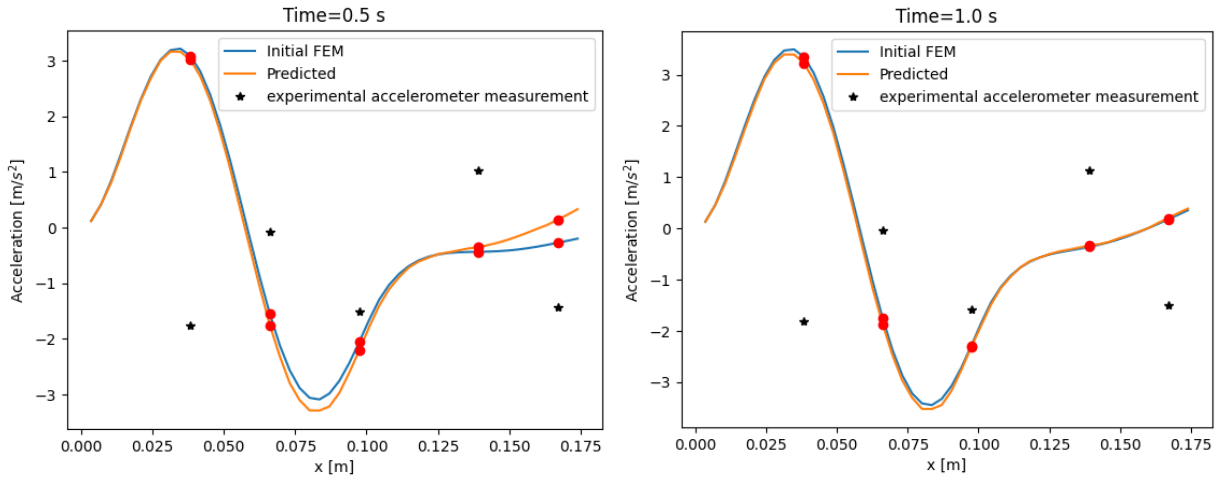



*Figure 5.28 Beam acceleration - Experimental case - L) Time=0.5 s - R) Time=1.0 s*

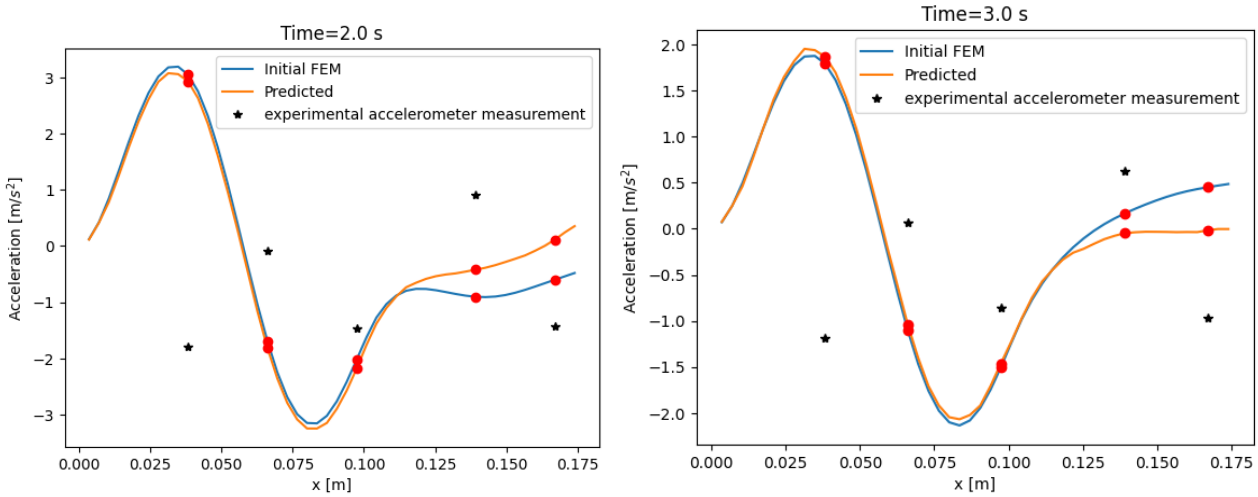



*Figure 5.29 Beam acceleration - Experimental case - L) Time=2.0 s - R) Time=3.0 s*

From the error graph, it is evident that the initial values are already quite small, and after 50 epochs, convergence to nearly zero is easily achieved, although even 25 epochs would have sufficed. As indicated by the preceding graphs and the R2 value in Table 2, this results in the initial FEM and predicted values curves being almost entirely coincident, leading to a significantly high R2, exceeding 0.90 for harmonic signals. Comparing the R2 values in the table, it is apparent that the highest R2 values are associated with harmonic signals, followed by lower values for chirp signals, and finally, pseudorandom signals.

The accuracy of the obtained results primarily stems from the near-complete coincidence between the excitation signal input sent to the shaker by the Matlab command and the signal detected by the load cell. Real and predicted values are consistently and uniformly distributed along the bisector,

enabling an accurate prediction of the acceleration curve for most values along the beam, with only a slight deviation observed at its tip.

### 5.4.4 Comparison among Harmonic Signals with Different Amplitudes and Frequencies

An analysis was conducted on a cosine signal with an amplitude of 0.05 V and a frequency of 600 Hz, comparing it with a cosine signal with an amplitude of 0.025 V and a frequency of 2000 Hz.

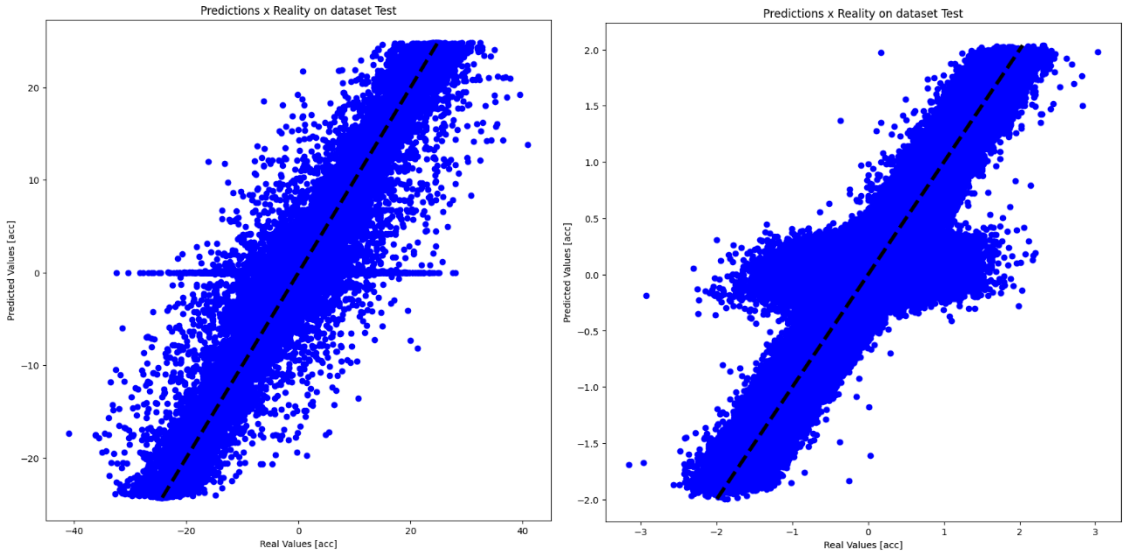



*Figure 5.30 Predictions x Reality on dataset test - L) Experimental case 0.05 V and 600 Hz - R) Experimental case 0.025 V and 2000 Hz*

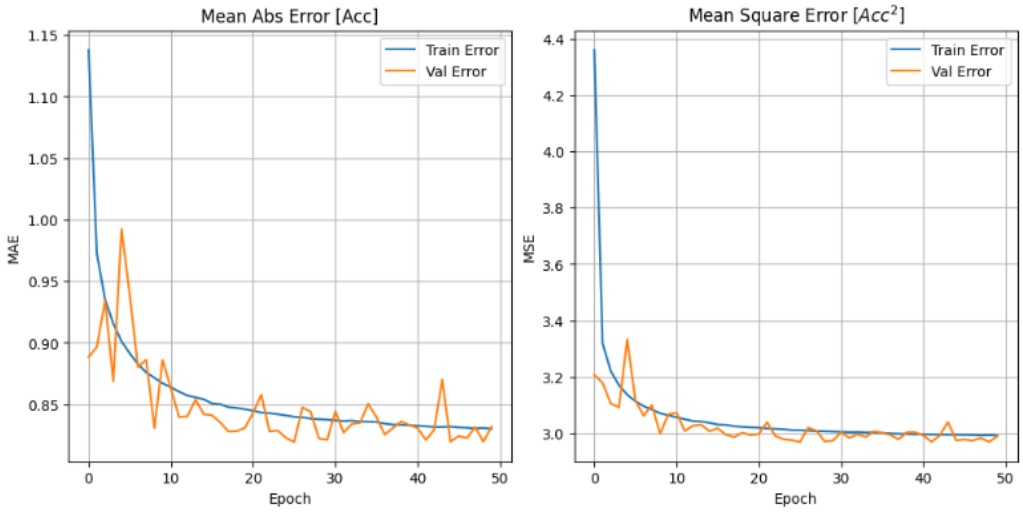



*Figure 5.31 Error evolution - Experimental case 0.05 V and 600 Hz*

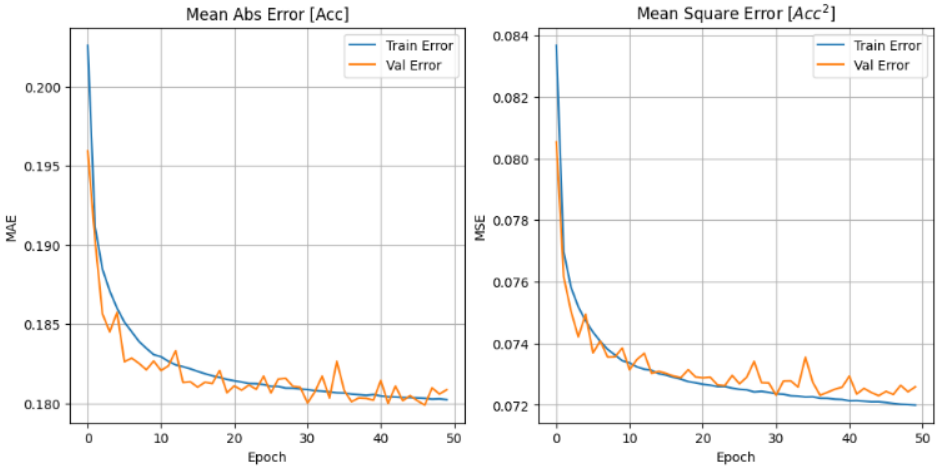



*Figure 5.32 Error evolution - Experimental case 0.025 V and 2000 Hz*

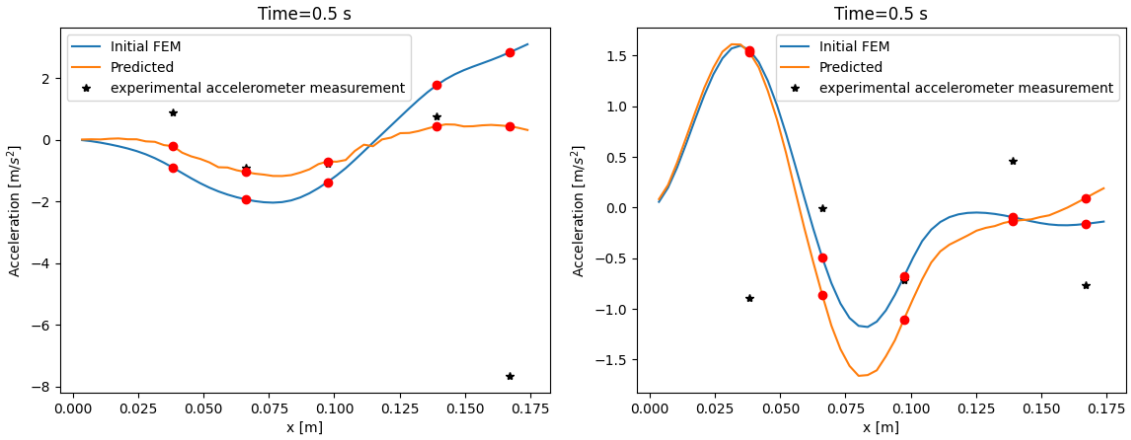



*Figure 5.33 Time=0.5 s - Beam acceleration - L) Experimental case 0.05 V and 600 Hz - R) Experimental case 0.025 V and 2000 Hz*

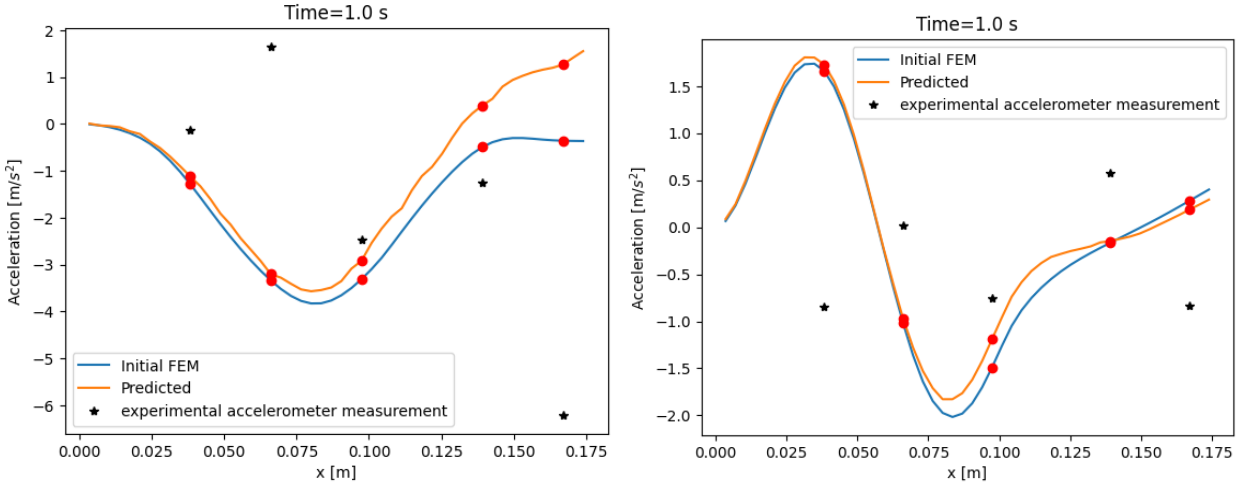



*Figure 5.34 Time=1.0 s - Beam acceleration - L) Experimental case 0.05 V and 600 Hz - R) Experimental case 0.025 V and 2000 Hz*

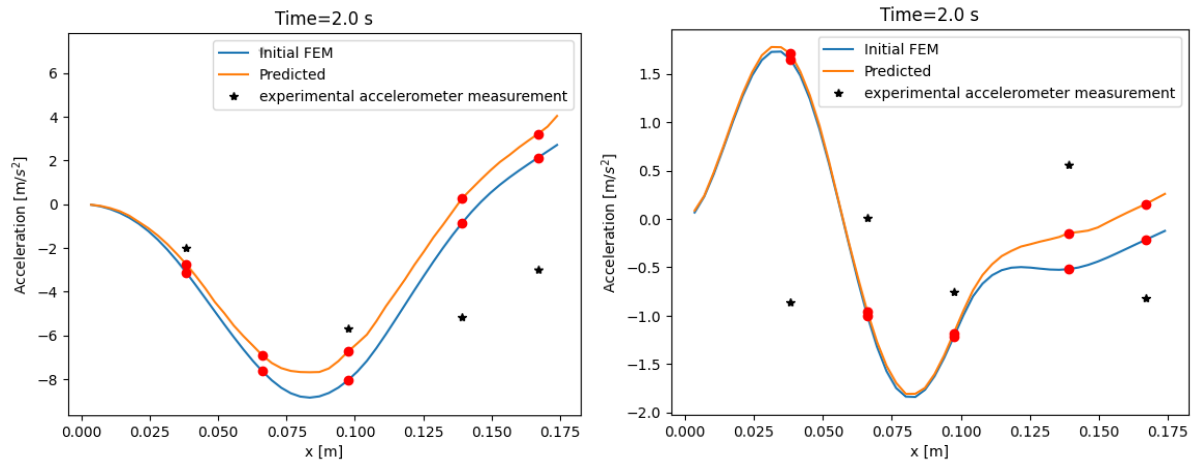



*Figure 5.35 Time=2.0 s - Beam acceleration - L) Experimental case 0.05 V and 600 Hz - R) Experimental case 0.025 V and 2000 Hz*

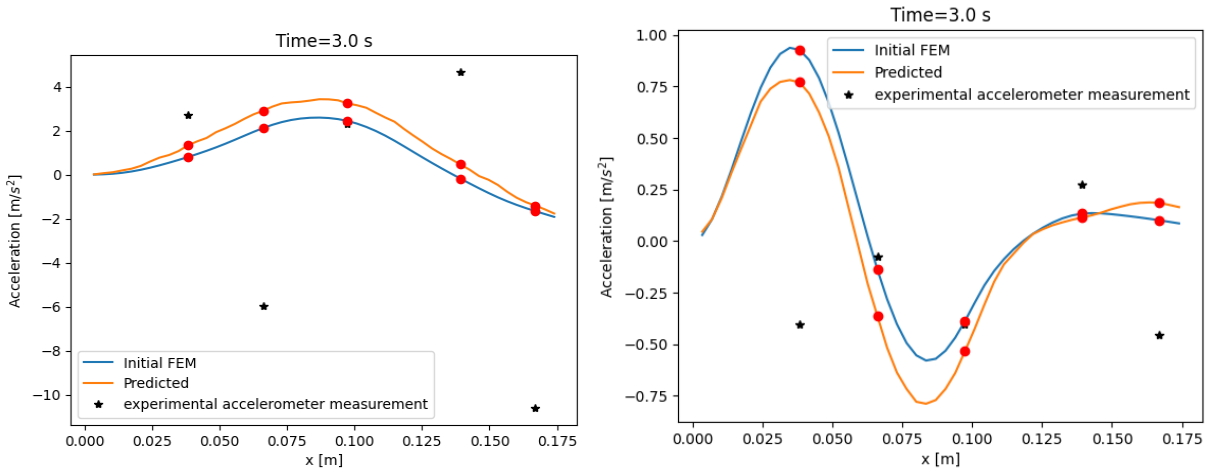



*Figure 5.36 Time=3.0 s - Beam acceleration - L) Experimental case 0.05 V and 600 Hz - R) Experimental case 0.025 V and 2000 Hz*

Between the two plots along the bisector, it is evident that in the case with lower amplitude and higher frequency, the points along the bisector are distributed quite uniformly and compactly along its entire length. In contrast, the other case exhibits numerous scattered points in the plot, far from the main bisector, and several real values predict an acceleration value of zero.

This leads to lower R2 values for the lower amplitude values, resulting in less pronounced bisector curves. The prediction curves coincide more in the case of the right-side figures, which exhibit a higher R2, especially up to about half the length of the beam, where they align completely with those of the initial FEM.

Ultimately, at none of the examined time instances in both cases, do the lines pass through all the experimental values of the accelerometer calculated at that specific time, due to the differences between experimental and numerically obtained values. Although there are time instances, such as

the plot describing the moment at 2 seconds and 3 seconds, where the lines, in some cases, pass or are very close to the experimental accelerometer values.

### 5.4.5 Comparison of Simulations using 70% Dataset and 80% Dataset

A chirp signal with an amplitude of 0.025 and a frequency of 2000 Hz was analyzed by comparing the results obtained through machine learning processing using only 70% of the dataset values and, in contrast, processing using 80% of the dataset values.

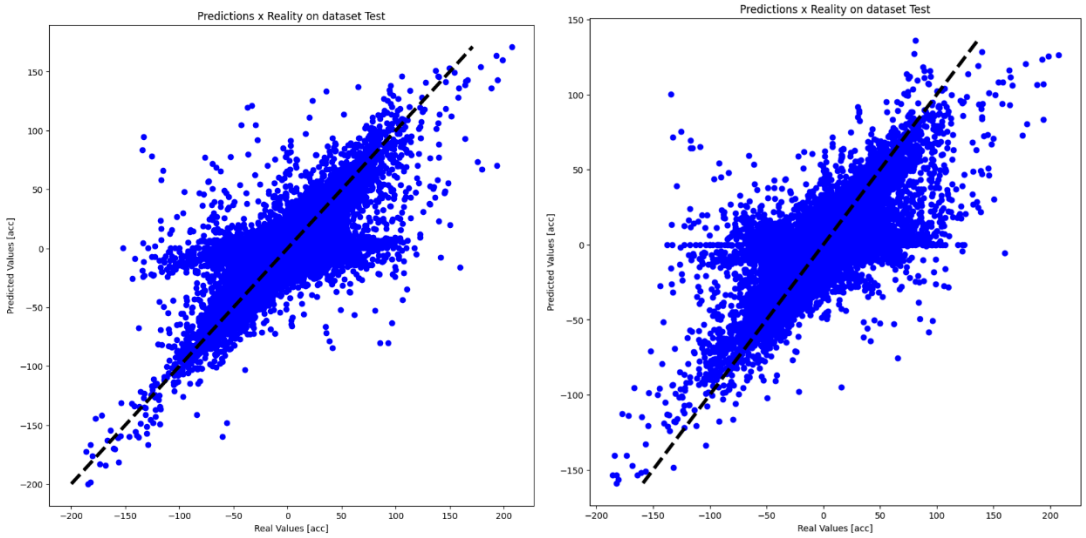



*Figure 5.37 Predictions x Reality on dataset test - L) Experimental case 70% dataset - R) Experimental case 80% dataset*

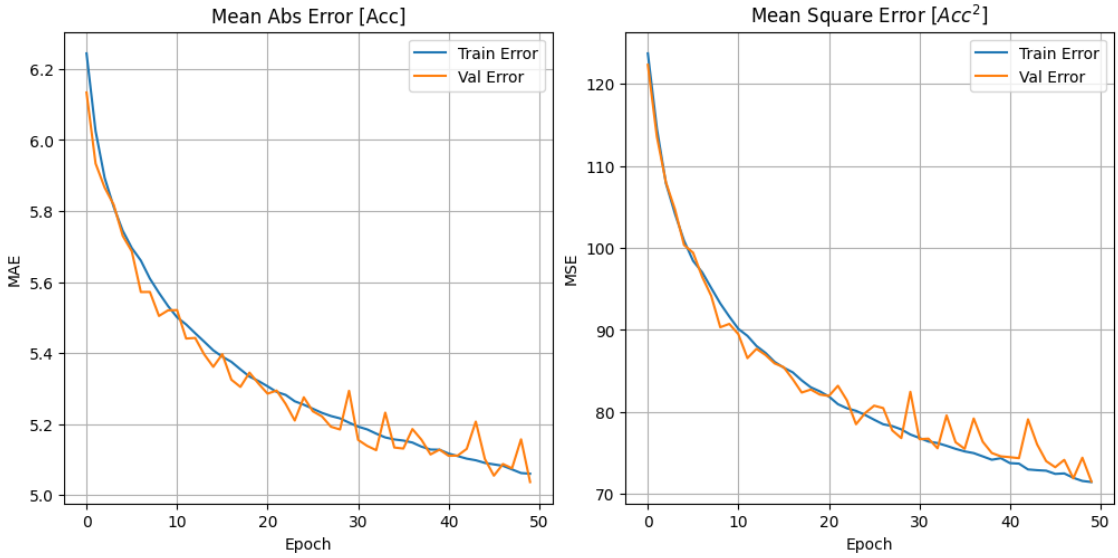



*Figure 5.38 Error evolution - Experimental case 70% dataset*

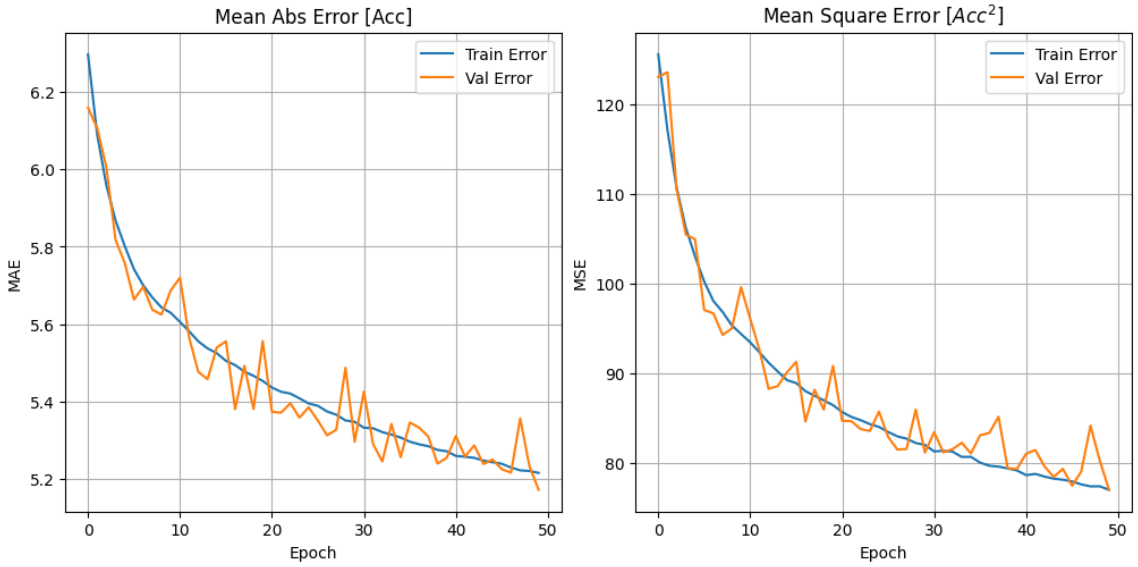



*Figure 5.39 Error evolution - Experimental case 80% dataset*

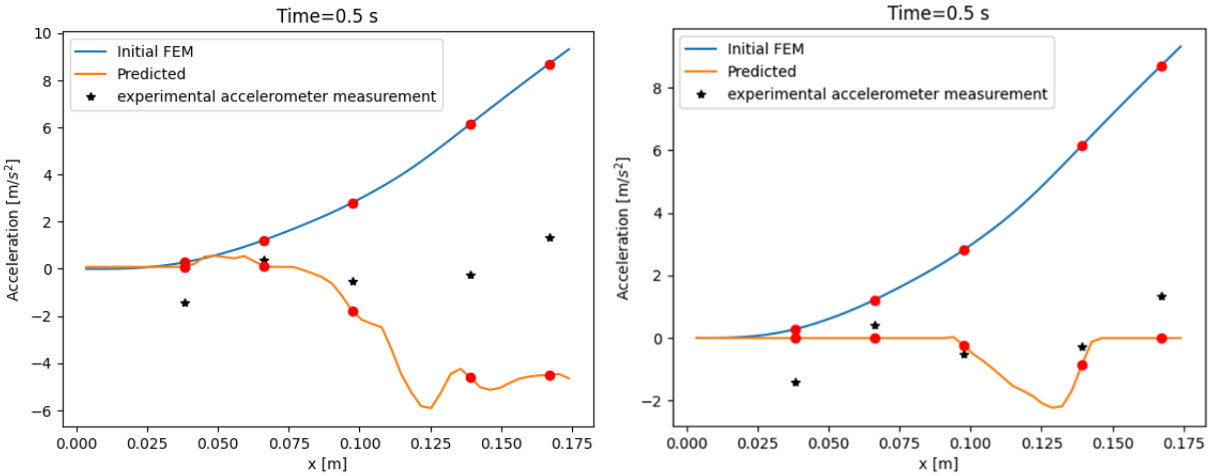



*Figure 5.40 Time=0.5 s - Beam acceleration - L) Experimental case 70% dataset - R) Experimental case 80% dataset*

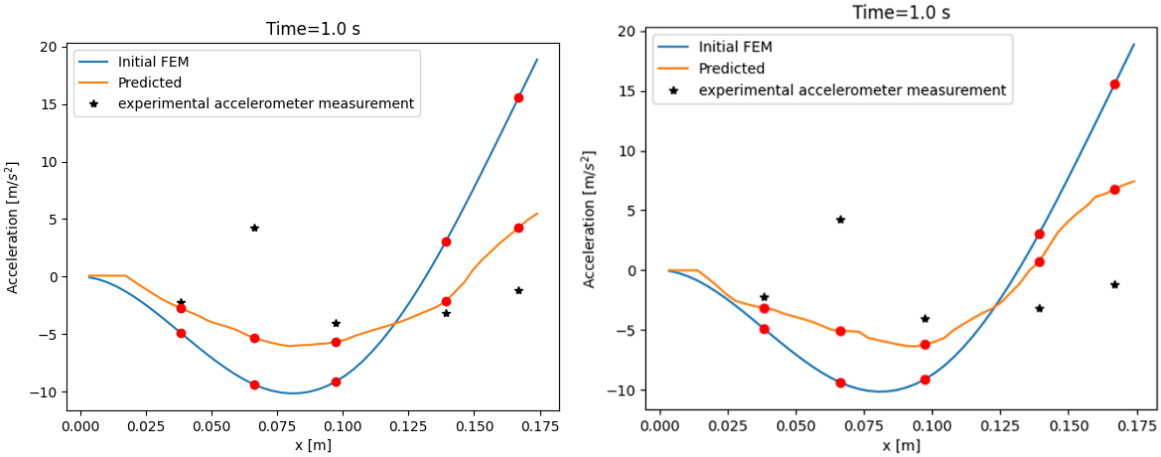



*Figure 5.41 Time=1.0 s - Beam acceleration - L) Experimental case 70% dataset - R) Experimental case 80% dataset*

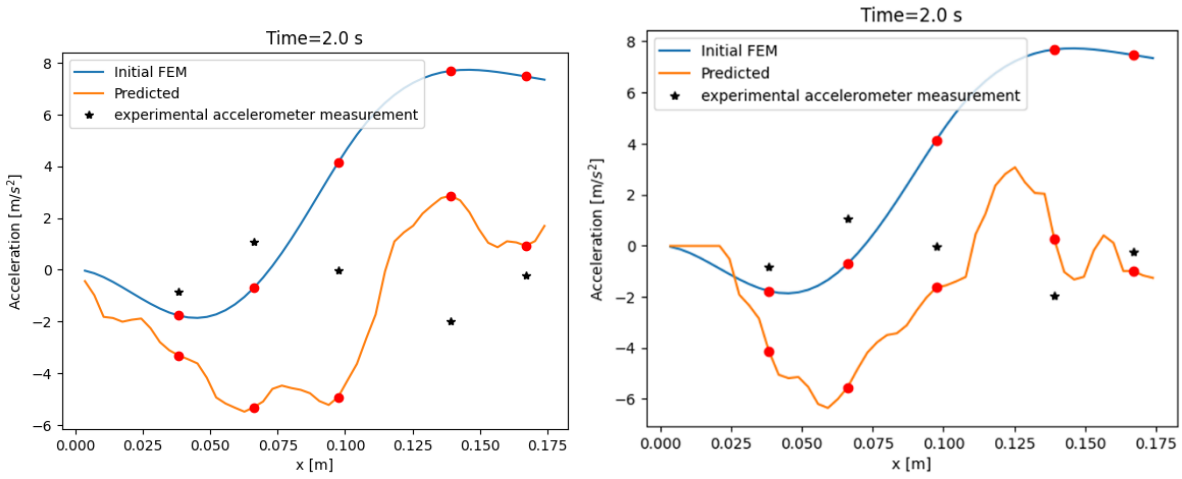



*Figure 5.42 Time=2.0 s - Beam acceleration - L) Experimental case 70% dataset - R) Experimental case 80% dataset*

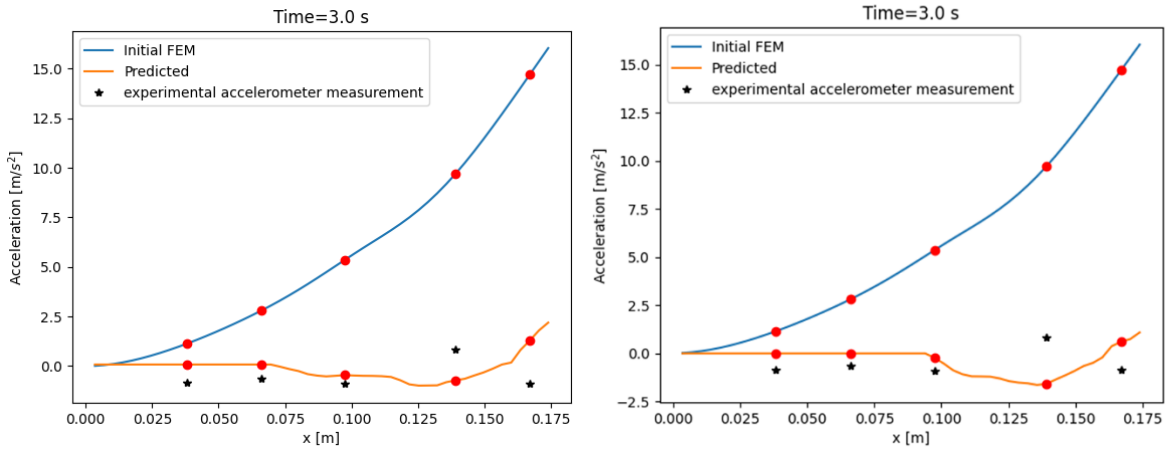



*Figure 5.43 Time=3.0 s - Beam acceleration - L) Experimental case 70% dataset - R) Experimental case 80% dataset*

From the MAE and MSE graphs, it is observed that the error values are similar, and the convergence trend is the same in both the 70% and 80% cases, as the two bisectors show data distributed in the same range. Indeed, Table 2 highlights the similarity in R2 values in both cases, suggesting that reducing the number of database entries does not significantly impact the R2 value.

Furthermore, in comparing various time instances, it is evident that the curves predicting the data, as well as error cases and the bisector, are similar in both scenarios. However, the curves predicted with 80% of the dataset are slightly better, closely approaching the initial FEM curves and the experimental values of the accelerometers. Particularly noteworthy is the case at 3 seconds, where the curve of predicted data closely aligns with the experimental values of the accelerometers in those positions.

### 5.4.6 Comparison of Simulations using 4 Accelerometers and 5 Accelerometers

In this case, an analysis was conducted with 5 accelerometers, and another analysis was performed with 4 accelerometers placed in the same positions. However, in the case with 4 accelerometers, accelerometer 3, positioned at the center of the beam, was not included as input when the machine learning algorithm analysis was conducted. The obtained values for a chirp signal with an amplitude of 0.05 and a frequency of 2000 Hz are as follows:

- Mean Absolute Error: 9.508295589619058
- Mean Squared Error: 263.4765120767734
- Mean Root Squared Error: 16.23195958831753
- r2: 0.5821782271025184

while the R2 value for the unused accelerometer is R2 = 0.53.

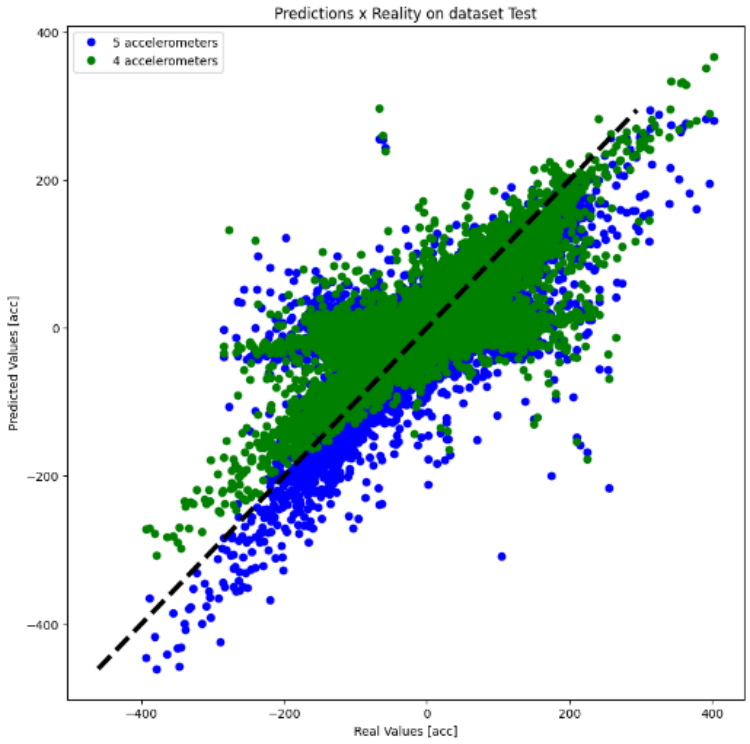



*Figure 5.44 Predictions x Reality on dataset test - Experimental case*

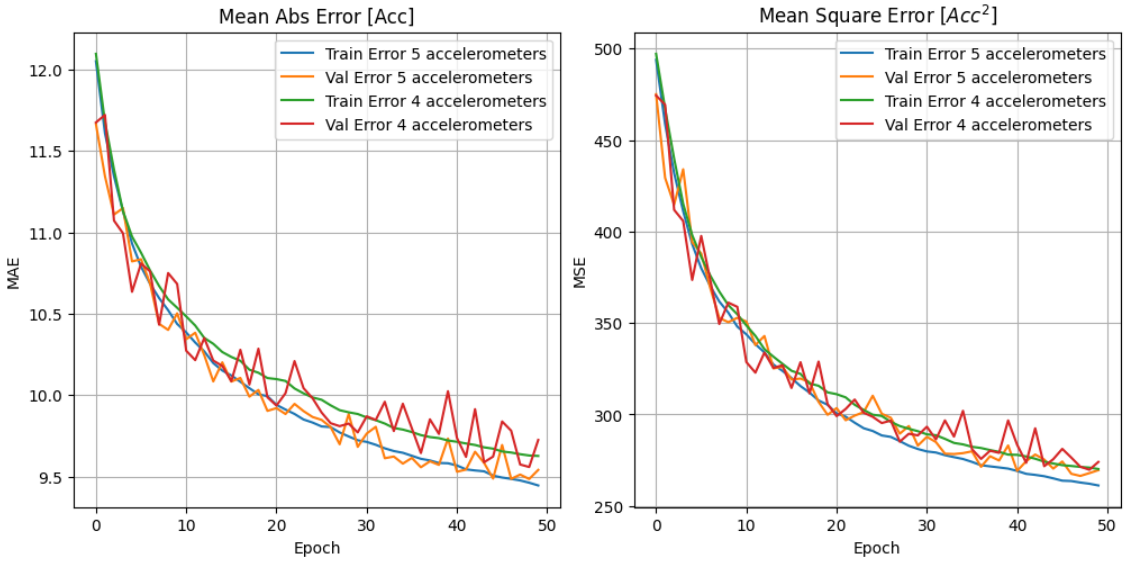



*Figure 5.45 Error evolution - Experimental case*

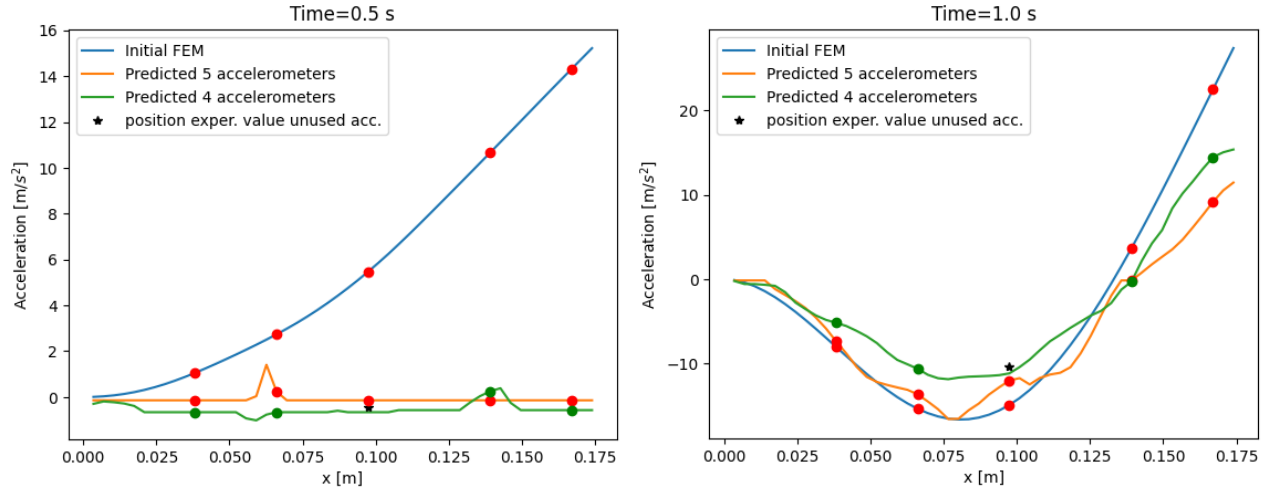



*Figure 5.46 Beam acceleration - Experimental case - L) Time=0.5 s - R) Time=1.0 s*

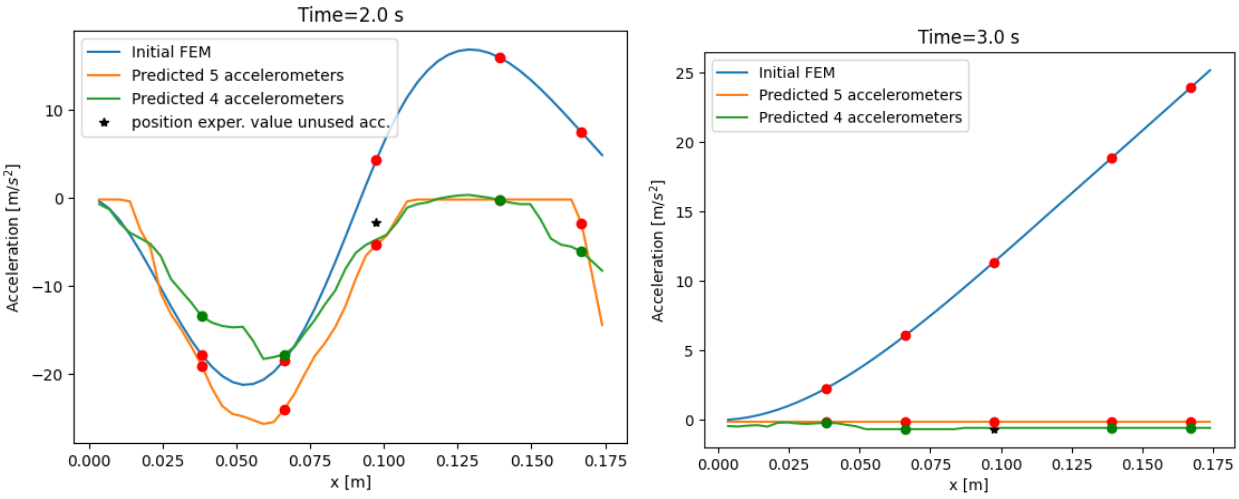



*Figure 5.47 Beam acceleration - Experimental case - L) Time=2.0 s - R) Time=3.0 s*

In this simulation, it can be observed that the curves of MAE and MSE values are almost entirely coincident, converging with the same trend and requiring the same number of epochs to reach convergence. The goodness of these results is evident in the various time instances examined, where the acceleration values completely or almost entirely coincide along the entire curve, showing no significant differences between the input with 5 accelerations and the one with 4 accelerations. It is noteworthy that in all four cases, the curves of the predicted accelerations with 5 and 4 accelerometers pass through the points of the calculated experimental values. Furthermore, at the time instances of 1 second and 2 seconds, the curves of the initial FEM, the predicted accelerations with an input of 5 accelerations, and the predicted accelerations with an input of 4 accelerations are almost entirely coincident, providing excellent results for the analyses conducted.

## 5.5 Results

In the following chapter, a detailed analysis of the previous comparisons and the results obtained will be conducted through an in-depth examination of the prediction of acceleration values along the beam using the previously proposed machine learning algorithm. Additionally, we will evaluate how the learning process has impacted the model's ability to understand and generalize such heterogeneous data.

The objective of this analysis is to understand to what extent the machine learning algorithm has been able to capture the fundamental characteristics of accelerations obtained from these different types of signals. It aims to assess its ability to predict and adapt to new data accurately and efficiently. The results obtained in this study will provide a clear overview of the model's performance and help draw meaningful conclusions about its potential use in practical applications related to acceleration analysis in variable environments.

Table 2 below summarizes the values obtained from various analyses of different signals acquired under the conditions outlined in Table 1. The first column contains the letters 'N,' indicating numerical analyses, while the letter 'E' indicates experimental analysis. Additionally, white cells indicate analyses where 80% of the dataset was used, while yellow cells indicate analyses where only 70% of the data were used to predict the remaining 30%.

| | | Evaluating Model's Performance on training data | | | | Evaluating Model's Performance on testing data | | | | Evaluating Model's Performance | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Time [s] | MAE | MSE | MRSE | R2 | MAE | MSE | MRSE2 | R2 | MAE | MSE | MRSE2 | R2 |
| CH1N | 2000 | 0.1782 | 0.1444 | 0.3801 | 0.9997 | 0.2332 | 0.21262 | 0.46111 | 0.9996 | 0.1877 | 0.1545 | 0.3931 | 0.9997 |
| CH1E | 2000 | 9.6653 | 266.9188 | 16.3376 | 0.5767 | 9.7914 | 278.6832 | 16.6938 | 0.5607 | 9.6849 | 268.7449 | 16.3934 | 0.5743 |
| AR1N | 2100 | 0.0089 | 0.0003 | 0.0186 | 0.9998 | 0.0142 | 0.0006 | 0.0257 | 0.9996 | 0.0092 | 0.0003 | 0.0193 | 0.9998 |
| AR1E | 2100 | 0.2104 | 0.0962 | 0.3102 | 0.9499 | 0.2123 | 0.0966 | 0.3108 | 0.9497 | 0.2107 | 0.0962 | 0.3102 | 0.9499 |
| PS1N | 2150 | 0.1206 | 0.0277 | 0.1667 | 0.9999 | 0.1536 | 0.0440 | 0.2099 | 0.9999 | 0.1225 | 0.0284 | 0.1687 | 0.9999 |
| PS1E | 8750 | 13.5888 | 387.420 | 19.6830 | 0.3313 | 13.7593 | 395.6257 | 19.8903 | 0.3183 | 13.6229 | 389.0574 | 19.7245 | 0.3287 |
| CH2N | 8750 | 0.1811 | 0.1323 | 0.3637 | 0.9999 | 0.2508 | 0.2075 | 0.4555 | 0.9998 | 0.1878 | 0.1386 | 0.3724 | 0.9999 |
| CH2E | 8700 | 20.0150 | 980.517 | 31.3132 | 0.4273 | 20.1749 | 994.2210 | 31.5312 | 0.4108 | 20.0242 | 980.1429 | 31.3072 | 0.4259 |
| CH3N | 2100 | 0.0723 | 0.0297 | 0.1724 | 0.9998 | 0.0843 | 0.0333 | 0.1825 | 0.9997 | 0.0736 | 0.0299 | 0.1729 | 0.9998 |
| PS2N | 2000 | 0.0813 | 0.0174 | 0.1321 | 0.9999 | 0.1164 | 0.0323 | 0.1798 | 0.9999 | 0.0860 | 0.0194 | 0.1394 | 0.9999 |
| PS2E | 7000 | 11.8409 | 333.7929 | 18.2700 | 0.2226 | 11.9681 | 339.4022 | 18.4228 | 0.2111 | 11.8661 | 334.9048 | 18.3004 | 0.2204 |
| AR2N | 2150 | 0.0378 | 0.0030 | 0.0548 | 0.9999 | 0.0517 | 0.0055 | 0.0744 | 0.9999 | 0.0389 | 0.0031 | 0.0564 | 0.9999 |
| AR2E | 2100 | 0.8340 | 2.9963 | 1.7310 | 0.9758 | 0.8377 | 3.0397 | 1.7434 | 0.9754 | 0.8343 | 3.0047 | 1.7334 | 0.9757 |
| AR3E | 2100 | 0.1807 | 0.0722 | 0.2687 | 0.8758 | 0.1812 | 0.0724 | 0.2691 | 0.8754 | 0.1808 | 0.0722 | 0.2688 | 0.8758 |
| CH3E | 2250 | 5.1308 | 73.6274 | 8.5806 | 0.5351 | 5.1856 | 76.2760 | 8.7336 | 0.5213 | 5.1399 | 74.0578 | 8.6056 | 0.5330 |
| CH1N | 9300 | 0.1341 | 0.1257 | 0.3546 | 0.9998 | 0.1626 | 0.1523 | 0.3903 | 0.9997 | 0.1424 | 0.1359 | 0.3686 | 0.9997 |
| CH1E | 8000 | 9.3914 | 252.187 | 15.8804 | 0.6000 | 9.4930 | 259.5497 | 16.1105 | 0.5909 | 9.4041 | 252.9557 | 15.9045 | 0.5993 |
| AR1N | 2150 | 0.0083 | 0.0003 | 0.0177 | 0.9998 | 0.0144 | 0.0006 | 0.0264 | 0.9996 | 0.0090 | 0.0003 | 0.0188 | 0.9998 |
| AR1E | 2150 | 0.2103 | 0.0966 | 0.3108 | 0.9497 | 0.2113 | 0.0969 | 0.3112 | 0.9495 | 0.2104 | 0.0966 | 0.3108 | 0.9497 |
| AR2N | 2250 | 0.0288 | 0.0022 | 0.0476 | 0.9999 | 0.0472 | 0.0044 | 0.0666 | 0.9999 | 0.0336 | 0.0026 | 0.0514 | 0.9999 |
| AR2E | 2250 | 0.8392 | 2.9974 | 1.7313 | 0.9758 | 0.8447 | 3.0410 | 1.7438 | 0.9754 | 0.8399 | 3.0058 | 1.7337 | 0.9757 |
| PS1N | 2350 | 0.1137 | 0.0270 | 0.1644 | 0.9999 | 0.1681 | 0.0514 | 0.2269 | 0.9999 | 0.1204 | 0.0293 | 0.1713 | 0.9999 |
| PS1E | 2400 | 13.4966 | 382.2611 | 19.5515 | 0.3402 | 13.687 | 391.6854 | 19.7910 | 0.3251 | 13.5346 | 384.145 | 19.5996 | 0.3372 |
| CH3E | 2250 | 5.0123 | 70.0446 | 8.3692 | 0.5277 | 5.1075 | 75.7657 | 8.7043 | 0.5195 | 5.0287 | 70.9968 | 8.4259 | 0.5223 |

*Table 2*

The simulations were conducted using the Google Colab platform, a cloud-based computing platform that provides free computing resources to its users. The free version was employed, so the GPU of the computer used to run the simulation was not utilized. Consequently, the runtime for each simulation could vary depending on the time of day due to server load on Google Colab. Being used by numerous users worldwide, server load can vary significantly based on the number of users using the service at a given time. If multiple users are using the servers simultaneously, longer runtimes may be experienced due to competition for available computing resources. This results in a temporal variation ranging from 30 to 50 seconds per step, depending on the dataset and amount of motion, up to 130-180 seconds per step.

For a variety of reasons, it is impractical to conduct a concrete analysis on the column related to the analysis runtime, as their consistency is compromised by a phenomenon inducing significant variations in runtimes for the same type of analysis.

If we analyze the column containing R2 values, several insights into the quality of the conducted analyses become apparent. The R2 column and the comparative analysis conducted in paragraph 5.4 reveal that, in various analyses with numerical data, the obtained R2 value is consistently 0.99, resulting in the phenomenon of overfitting. In a numerical analysis context, overfitting occurs when a mathematical model is excessively tailored to the training data, including random details or noise that do not represent true patterns in the data distribution. In other words, the model has learned the training data so well that it loses its ability to generalize to new data, compromising its predictive validity. Overfitting can occur when the model is too complex relative to the intrinsic complexity of the problem and can be addressed through regularization techniques or model complexity reduction.

As a result, it has become crucial to orient the assessments toward the validity of the analyses conducted based on the experimental values obtained. Indeed, among the three analyzed signals, the harmonic signal demonstrates the highest validity, with an R2 equal to or exceeding 0.90. Subsequently, the second signal that proves highly applicable in the conducted analyses is the chirp signal, with an R2 ranging between 0.42 and 0.57. Finally, the least favorable outcome is generated by the analyses performed with the pseudorandom signal, yielding an R2 of approximately 0.20-0.30. For this reason, the curves of initial FEM and predictive curves were markedly distant at all analyzed time points, in contrast to the harmonic case with a much higher R2, where the two curves were almost entirely coincident.

Another noteworthy observation that emerges relates to the percentage of data selected as input for the dataset in machine learning analysis. It is evident that, whether using 70% or 80% of the data, highly similar R2 values are obtained, and the predictive curves show a significant approximation to those obtained through the initial FEM in the same motion. From these results, it can be inferred that even with only 70% of the data available, valid analyses comparable to those conducted with 80% of the data could be carried out.

The same deduction can be made in the case of the analysis performed in paragraph 5.4.6, where initially, 5 accelerometer values were used and then compared with data obtained from signals coming from 4 accelerometers, excluding the accelerometer positioned at the center. The analysis

reveals that the R2 value with 4 accelerometers is slightly lower than that with 5 accelerometers and that the predicted curves pass almost along the same points. This suggests that the omitted accelerometer is relatively redundant, as valid results are obtained in the case with 4 accelerometers.

# 6 Experiment model plate with hole

After analyzing a beam in the previous chapter, we moved on to a slightly more complex two-dimensional structure, namely a perforated plate. The perforated plate was constructed using Structural Steel material with a Young's modulus equal to $2 \cdot 10^{11}\ Pa$ and a density equal to $7850\ km/m^3$, its dimensions are as shown in Figure 6.1:

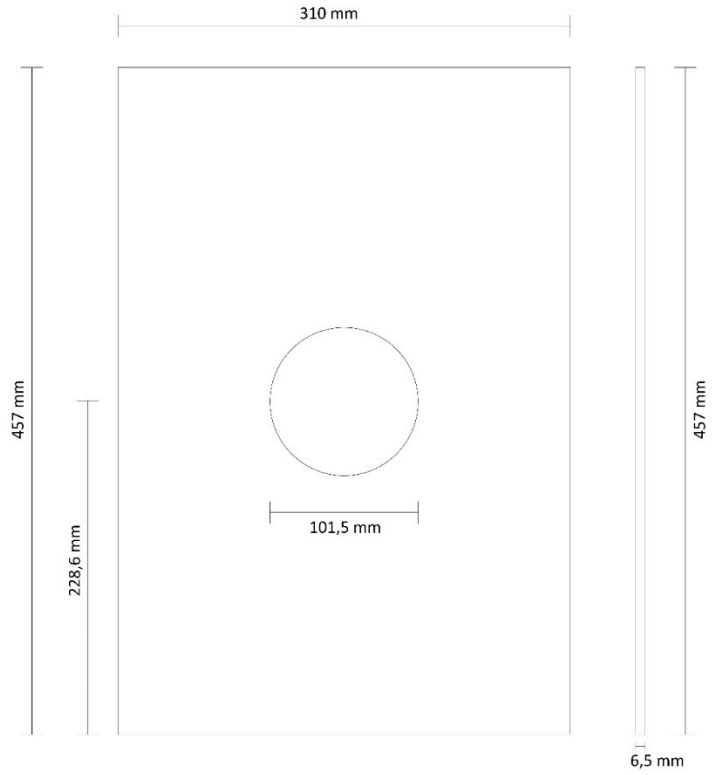



*Figure 6.1 Diagram of plate dimensions with hole*

## 6.1 Experimental Setup for Plate with Hole

As in the previous case, all the tools already used for the beam in Chapter 5.1 were employed. After thoroughly cleaning the plate from possible dust or dirt, it was placed on the table with the drilled boards, as shown in Figure 6.3. Here, the plate was carefully secured and bolted to the table using washers, nuts, and bolts to minimize errors due to vibrations caused by inadequate plate fixation during data acquisition.

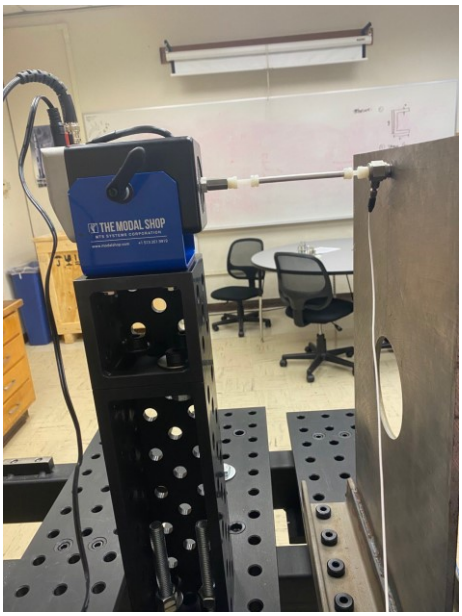



*Figure 6.2 Plate configuration*

After securing the plate, the next step involved positioning the shaker at the correct height corresponding to the point where it was necessary to apply the load. This was achieved using the supports, which were also employed to secure the beam in the previous case, adequately fixed and bolted in the same manner as the plate. On top of these supports, the shaker was placed and connected to the load cell model 208C02 SN LW 55552 with a sensitivity of $50\ mV/lb$.

On this perforated plate, using the supplied adhesive, all 7 accelerometers available were positioned:

1. Accelerometer model 352A24 SN LW 369406 with sensitivity $9.94\ mV/m/s^2$
2. Accelerometer model 352A24 SN LW 369404 with sensitivity $9.88\ mV/m/s^2$
3. Accelerometer model 352A24 SN LW 339401 with sensitivity $10.00\ mV/m/s^2$
4. Accelerometer model 352A24 SN LW 369399 with sensitivity $9.96\ mV/m/s^2$
5. Accelerometer model 352A24 SN LW 369402 with sensitivity $9.87\ mV/m/s^2$
6. Accelerometer model 352A24 SN LW 369405 with sensitivity $10.06\ mV/m/s^2$
7. Accelerometer model 352A24 SN LW 369403 with sensitivity $10.00\ mV/m/s^2$

The accelerometer values are acquired by the modules in the positions just listed. For simplicity, they will be represented in the following graphs, illustrating the placements of the various accelerometers in the proposed configurations.

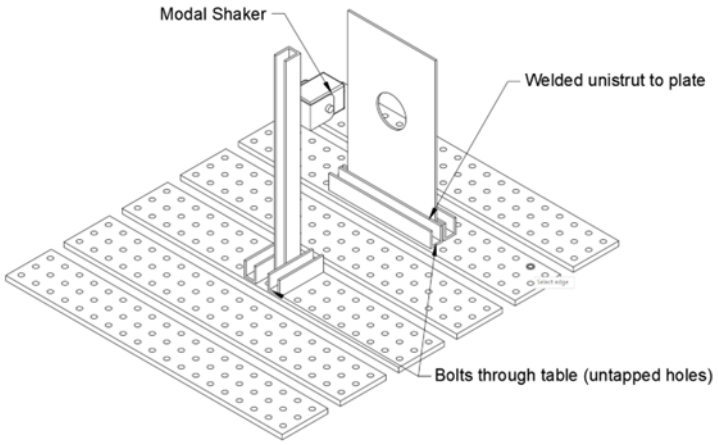



*Figure 6.3 Plate configuration with table*

## 6.2 Sensor Placement Decision

Unlike the case of the beam where 5 accelerometers were sufficient to obtain an adequate number of signals, in this scenario, the surface to be covered is much larger. For this reason, it was necessary to apply the study [32] to understand where to place the accelerometers in the most appropriate way and verify the correct method to ensure the best prediction of results. Moreover, only 7 accelerometer positions were feasible due to the experimental unavailability of additional accelerometers in the laboratory.

Six different configurations for sensor placement were examined:

- Effective Independence Method (EIM)
- RFR selecting for avg. FRF (RFR-FRF)
- RFR selecting for ODS (RFR-ODS)
- RFR selecting for normalized ODS (RFR-nODS)
- Large distributed grid (LDG)
- Small distributed grid (SDG)

### 6.2.1 Configuration Effective Independence Method (EIM)

Regarding the EIM method, it was necessary to develop a script to implement the algorithm on the analysis results produced by a modal finite element, providing it with natural frequencies and modal shapes. The goal is to maximize the independence of each accelerometer position so that the collected data vary from one accelerometer to another. Additionally, a strategy was adopted to

exclude duplicate data from the response set, allowing for obtaining the maximum amount of information with the least number of repetitions.

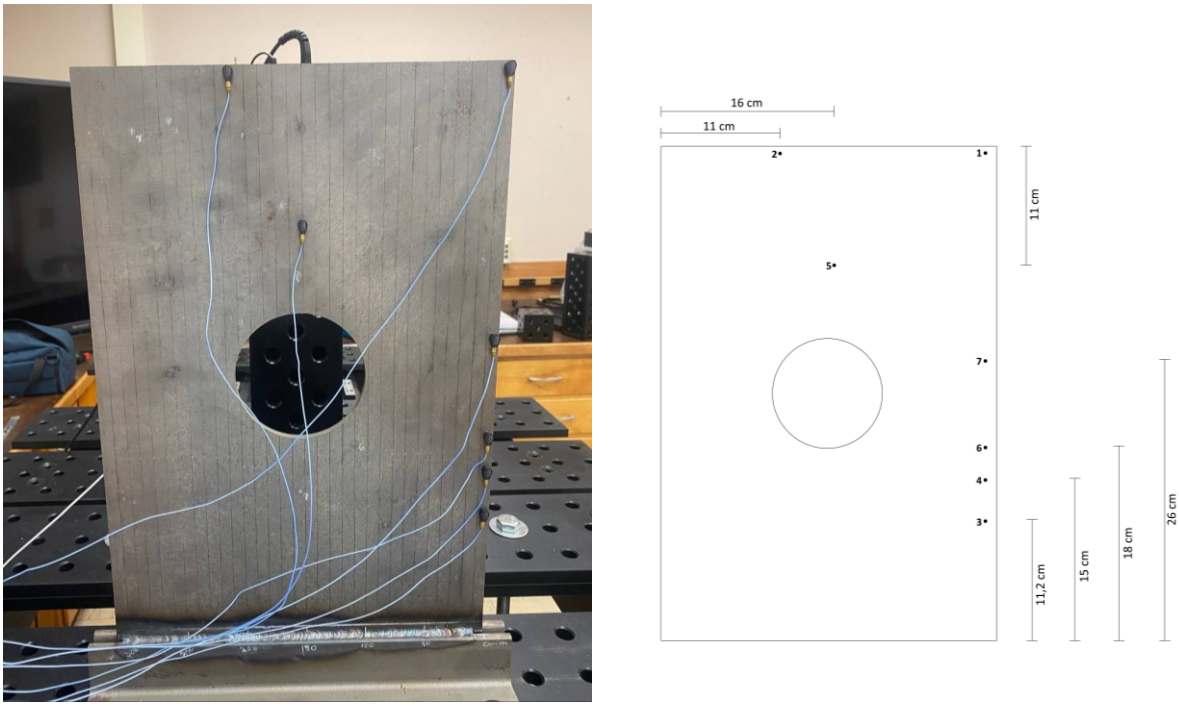



*Figure 6.4 Configuration 1 - Effective Independence Method (EIM)*

### 6.2.2 Configuration Random Forest Regression (RFR)

Another potential method for sensor selection is the Random Forest Regressor (RFR). It is a widely used machine learning algorithm for addressing regression problems. This model relies on an ensemble concept, combining various decision trees to enhance prediction accuracy. The distinctive feature of the RFR is the introduction of randomness during the construction of each tree, randomly selecting features and training data used at each step. This randomness contributes to creating diversity among the trees, thereby reducing the risk of overfitting and improving the model's ability to generalize to new data. In practice, when making a prediction with the random forest regressor, each tree contributes its prediction, and the final output is often obtained by averaging these predictions.

For creating the dataset used in the RFR for this specific application, it was necessary to extract it from the FEA models obtained from a transient analysis using the ANSYS software, where each row represents a frequency for which the FRF of each calculated node is obtained.

In this dataset, each row corresponded to the operational deflection shape (ODS) of each node in the FEA model for a given frequency. The ODS is a graphical representation of the vibration or deflection pattern of a structure during its normal operation or functionality.

Three different results obtained from:

Raw ODS, where a distinct value is obtained for each ODS at every frequency:

$$ODS_{\omega}^{T}\, ODS_{\omega}$$

To obtain a different value for each ODS at every frequency.

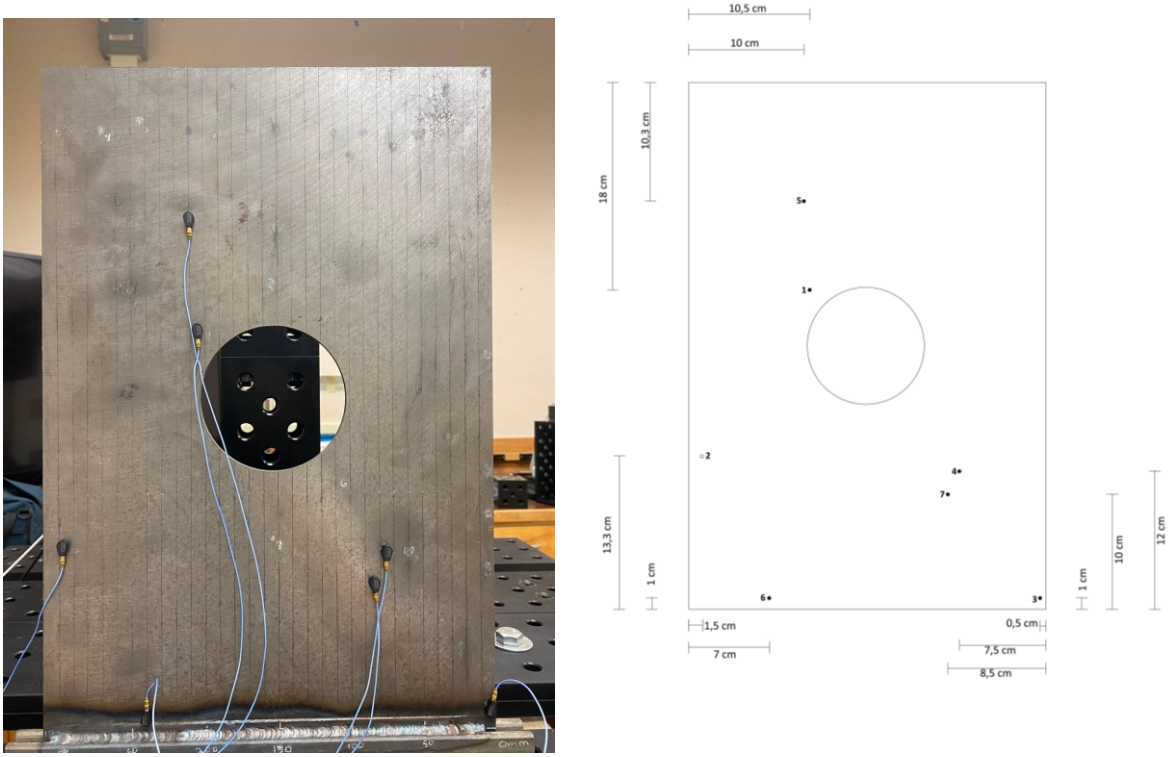



*Figure 6.5 Configuration 3 - RFR selecting for ODS*

Subsequently, a normalized form was analyzed to reduce the sensitivity of the output, thus eliminating the effect of the load magnitude:

$$\frac{ODS_{\omega}^{T}\, ODS_{\omega}}{|ODS|^{2}}$$

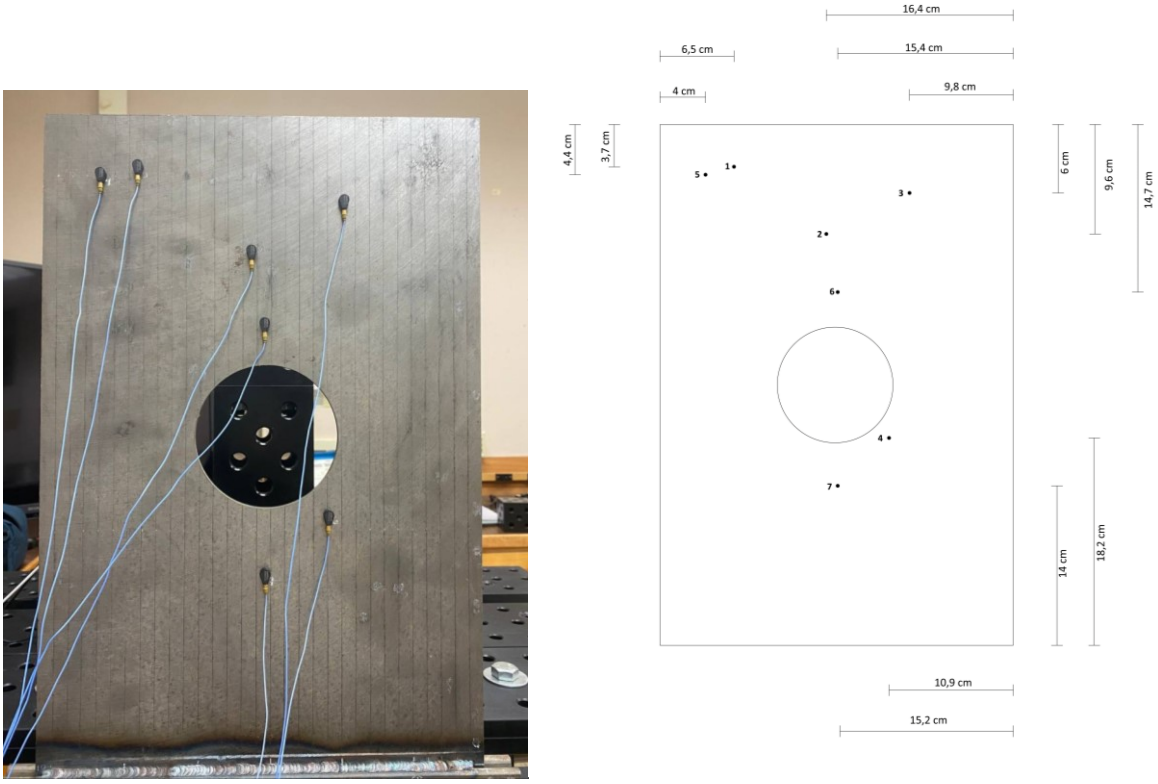



*Figure 6.6 Configuration 4 - RFR selecting for normalized ODS (RFR-nODS)*

Finally, the last output utilized was the mean FRF at each given frequency:

$$\frac{\sum_n FRF(f)}{n}$$

where *n* is the number of nodes.

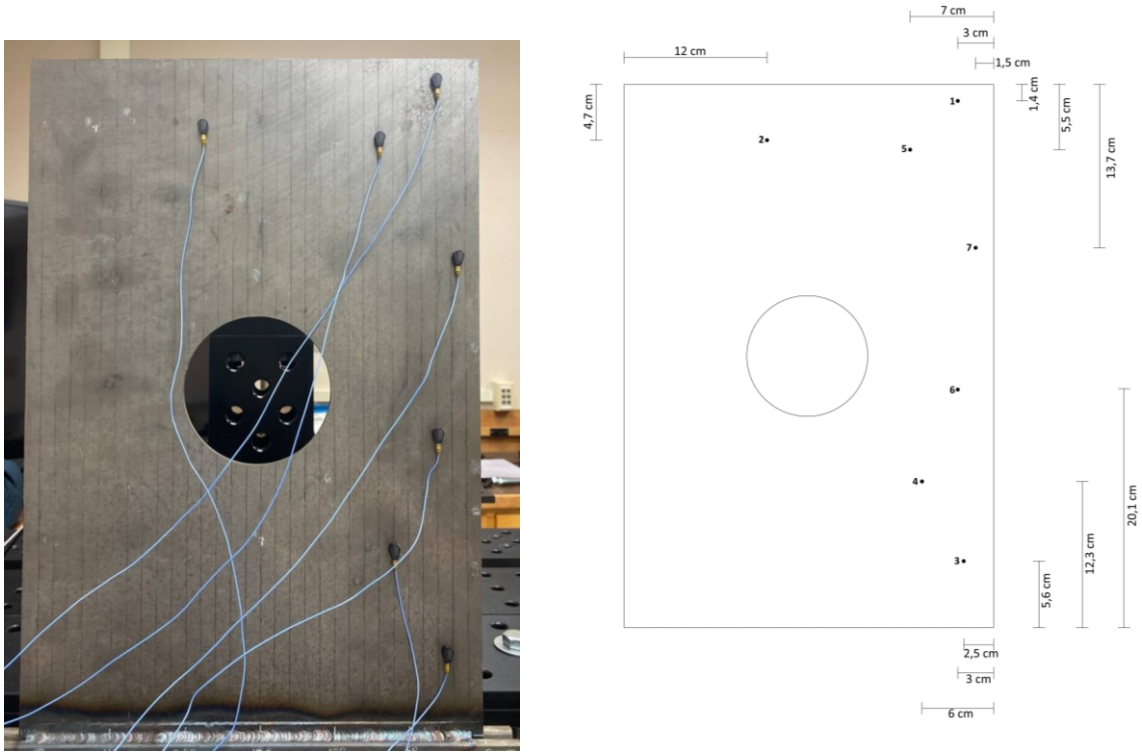



*Figure 6.7 Configuration 2 - RFR selecting for normalized ODS (RFR-nODS)*

### 6.2.3 Large Distributed Grid (LDG) and Small Distributed Grid (SDG)

Finally, the last two configurations were used without the need for a positioning technique obtained from the results of an algorithm. Accelerometers were simply placed in one half of the perforated plate in the Large Distributed Grid (LDG) configuration, while in the case of the Small Distributed Grid (SDG), accelerometers were positioned in one quarter of the perforated plate.

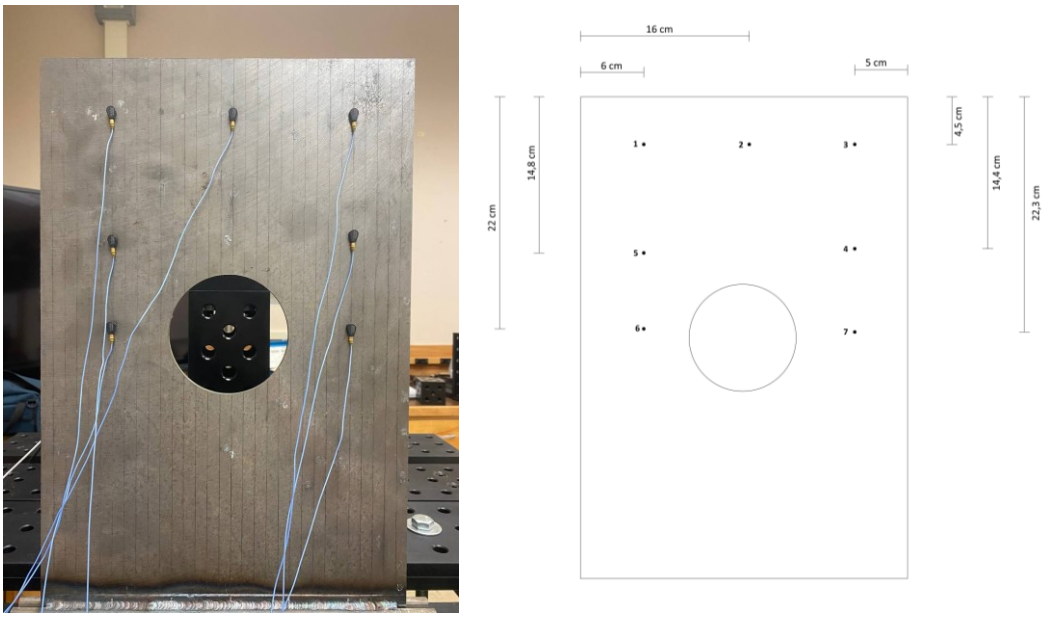



*Figure 6.8 Configuration 5 - Large distributed grid (LDG)*

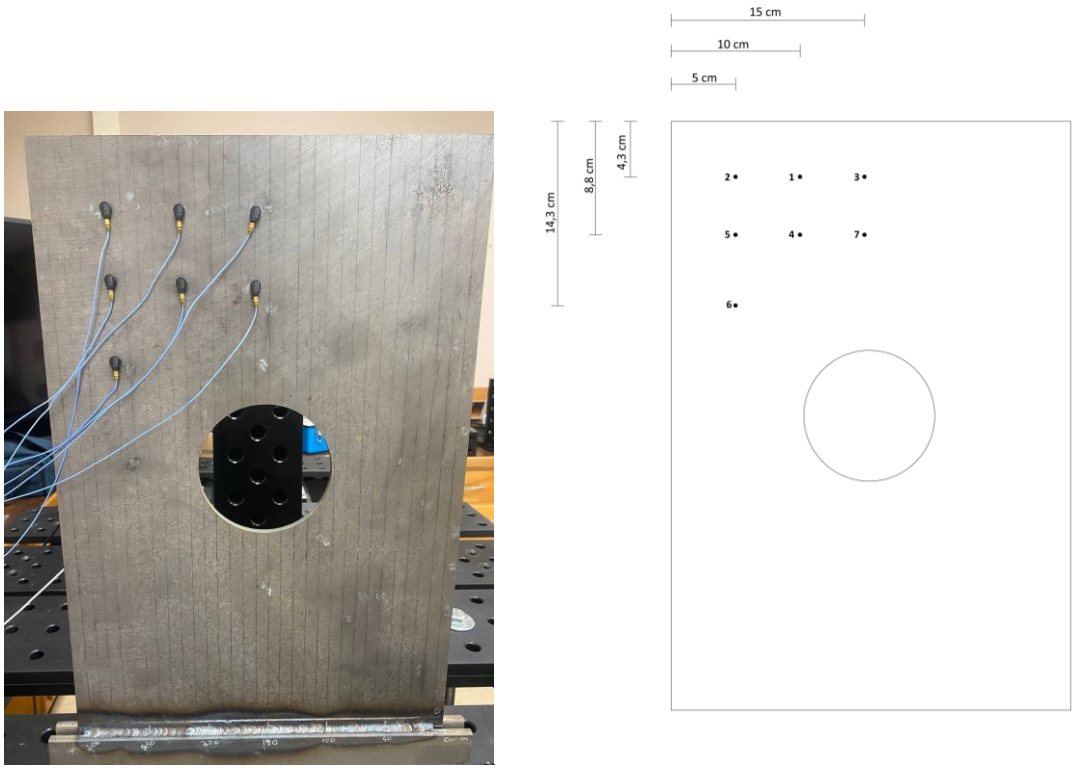



*Figure 6.9 Configuration 6 - Small distributed grid (SDG)*

## 6.3 Finite Element Method (FEM) Analysis on Ansys

The generation of a mesh in ANSYS Workbench 2023 is a crucial process in the numerical analysis of Finite Element Analysis (FEA). The mesh represents the geometric discretization of the initial model and consists of finite elements such as triangles or quadrilaterals (in the case of two-dimensional mesh) or tetrahedra and hexahedra (in the case of three-dimensional mesh).

The process began with importing the model geometry developed in SolidWorks into the ANSYS work environment. Once the geometry was imported, the software automatically generated the mesh. However, the result obtained was not entirely satisfactory, and it was necessary to refine the mesh, especially in the vicinity of the plate hole, where the mesh was densified to properly discretize the geometry. In total, the obtained mesh consists of 15,838 nodes and 7,506 elements.

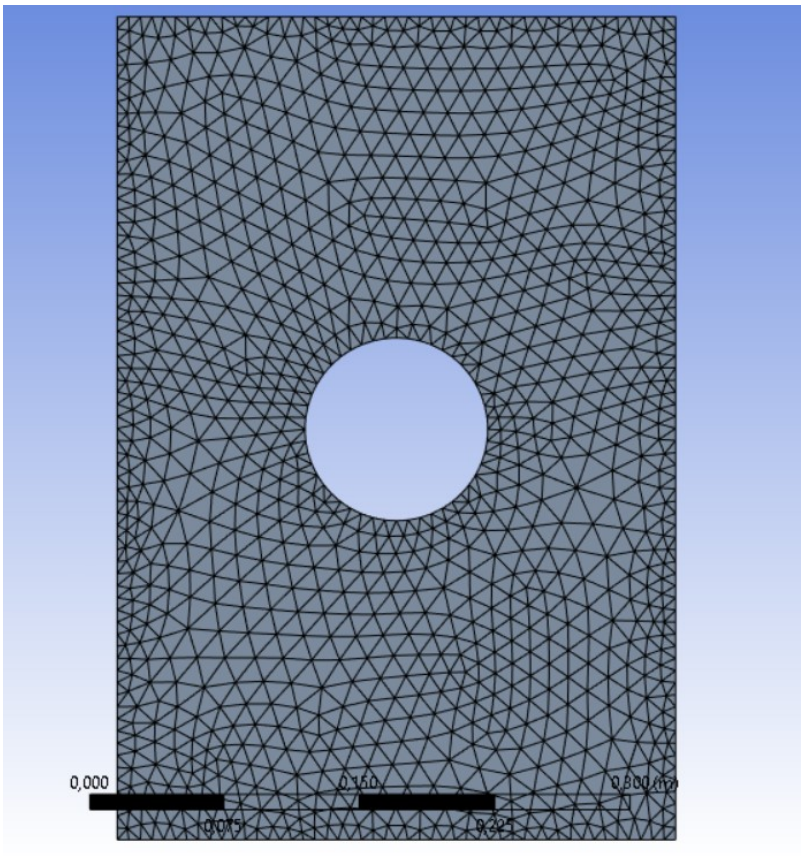



*Figure 6.10 Mesh Ansys*

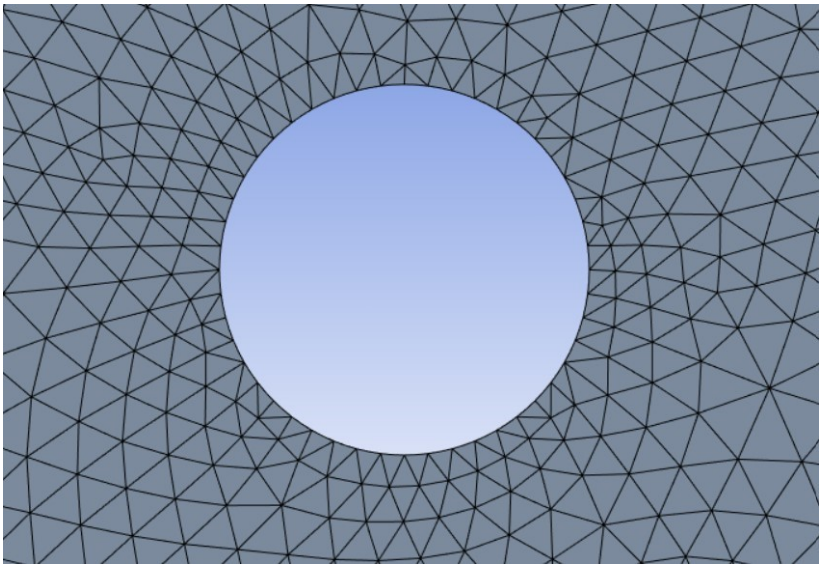


*Figure 6.11 Mesh Refinement near the Hole*

After mesh generation, it was essential to assign the material properties of "Structural Steel" to the model. Subsequently, model constraints were defined, and the force values along with the coordinates of the load vector, stored in an Excel file, were imported using the "Imported File" feature. This step enabled the specification of loading conditions, completing the problem setup for FEM analysis. In contrast to the beam analysis performed using MatLab, a "Transient Structural"

analysis was conducted at this stage. This was done to obtain accelerations and deformations at the relevant nodes, forming the dataset required for the plate algorithm, as explained in Section 4.6.

## 6.4 Comparisons of Machine Learning Simulations: Plate Case

Various signal samplings were performed through multiple experimental tests using different input signals such as chirp, pseudorandom, and harmonic. The signals listed in Table 1 were analyzed, and in the subsequent section, we will delve into the signals that yielded the most significant results.

As in the case of the beam, three different types of signals were analyzed: chirp, pseudorandom, and harmonic. All three signals were sampled in the configurations described earlier in Chapter 6.2, with the following characteristics:

- $Acqusition\ rate\ 6400\ Hz$
- $Lenght\ scane = 4\ s$
- $N.average = 2$

### 6.4.1 Numerical-Experimental Harmonic Signal Comparison in the Configuration with Algorithm

In this case, a cosine signal was analyzed with accelerometers positioned as in the previously obtained Configuration 3, comparing numerical values with experimental ones.

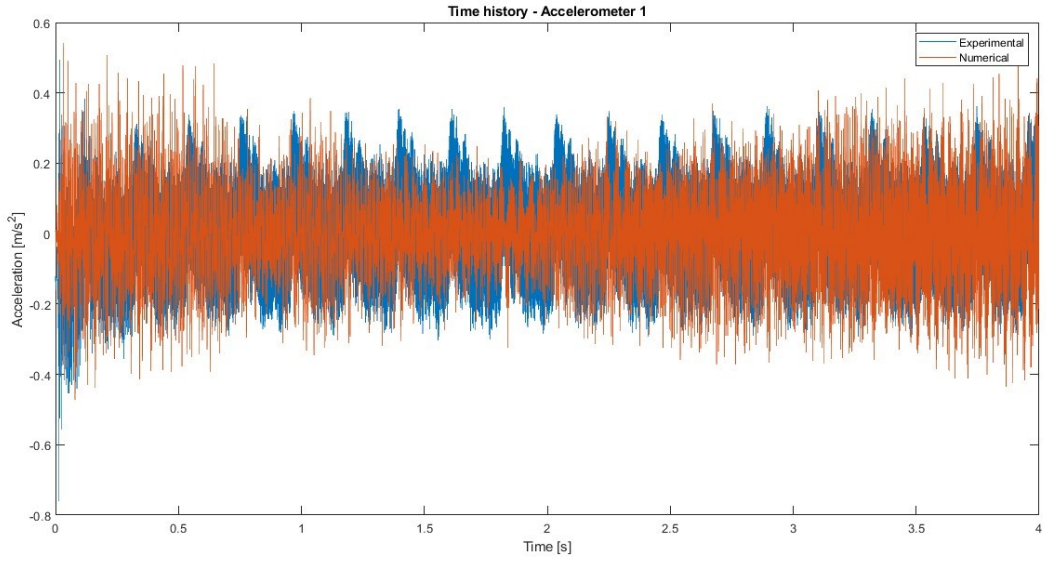



*Figure 6.12 Time history harmonic signal – Accelerometer 1*

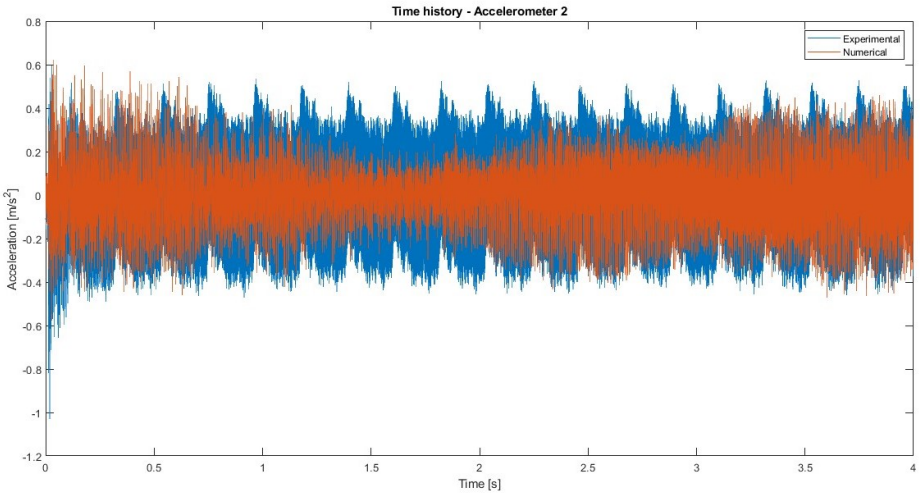



*Figure 6.13 Time history harmonic signal – Accelerometer 2*

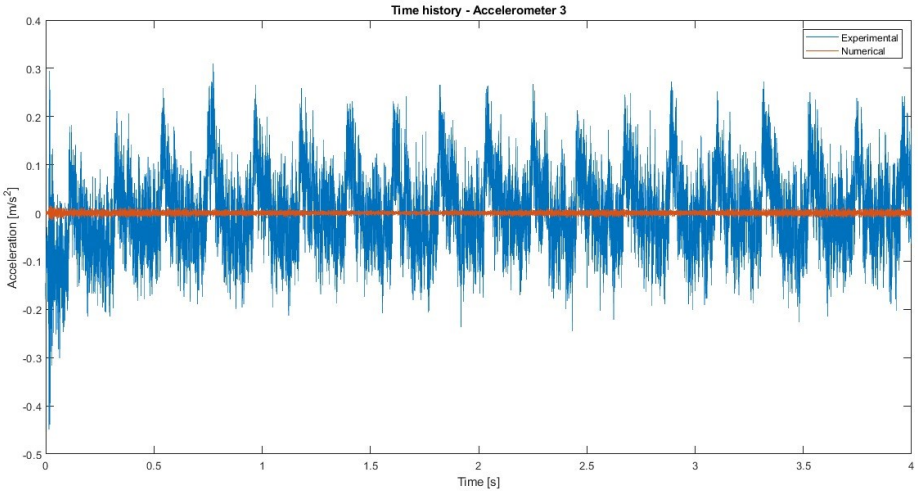



*Figure 6.14 Time history harmonic signal – Accelerometer 3*

In the recently examined case of the harmonic signal of a cosine wave, the analysis reveals good agreement between experimental and calculated data for accelerometers in positions 1 and 2. This suggests that the utilized model accurately represents the behavior of the examined harmonic signal. However, this excellent correspondence found in the first two accelerometers is not observed in the case of accelerometer 3. Unlike the first two accelerometers, which are distant from the constraint, the third accelerometer is close to the constraint, resulting in the numerical values approaching zero as expected. In contrast, the experimental results show significantly higher values. This is attributed to the fact that, despite properly bolting the plate supports to the table, a perfect constraint was not achieved, leading to movements near the connection point between the lower part of the plate and the table.

Subsequently, the results obtained from machine learning were compared between the numerical model of accelerometers (on the left) and the experimental model (on the right) of accelerometer values.

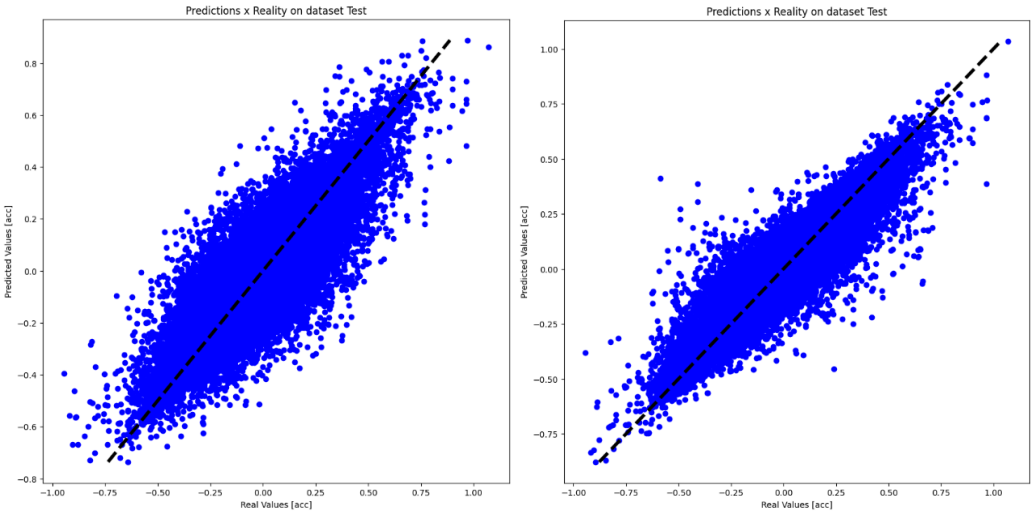



*Figure 6.15 Predictions x Reality on dataset test - L) Numerical case - R) Experimental case*

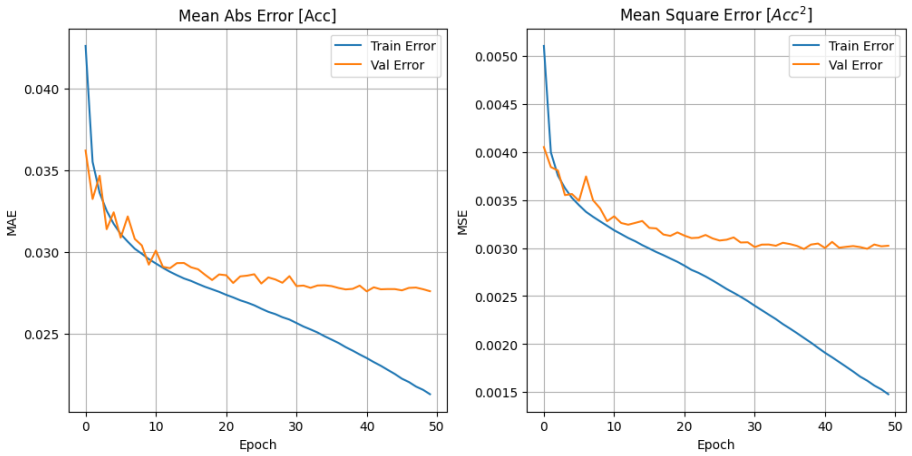



*Figure 6.16 Error evolution - Numerical case*

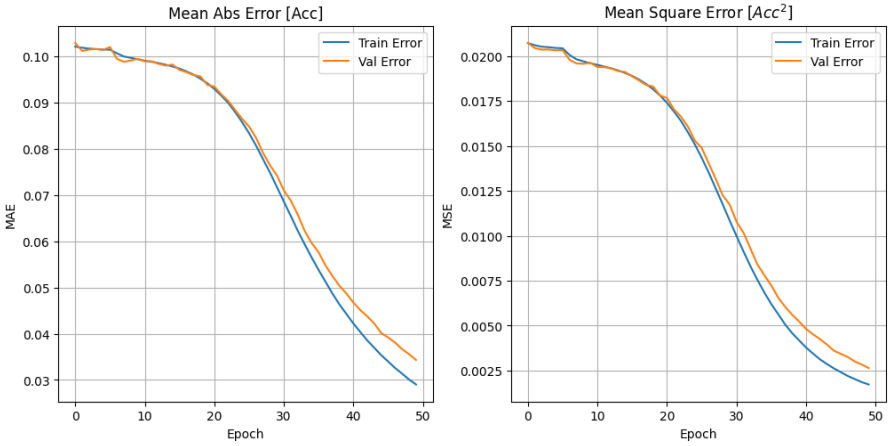



*Figure 6.17 Error evolution - Experimental case*

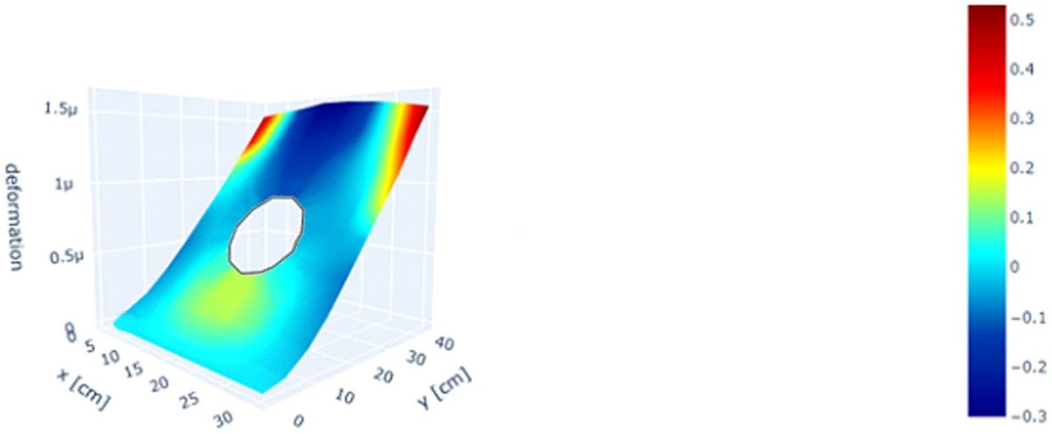



*Figure 6.18 Time=0.5 s - Initial FEM numerical case - Colorbar indicates accelerations in m/s²*

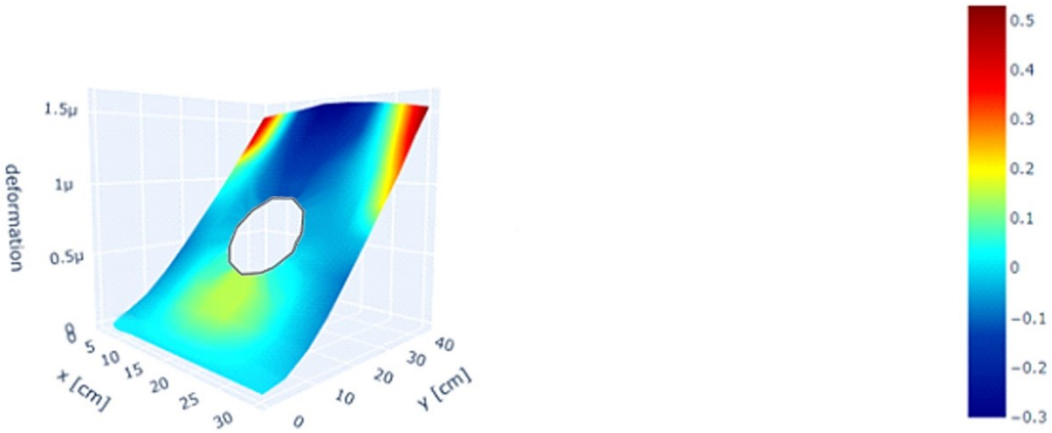



*Figure 6.19 Time=0.5 s - Predicted numerical case - Colorbar indicates accelerations in m/s²*

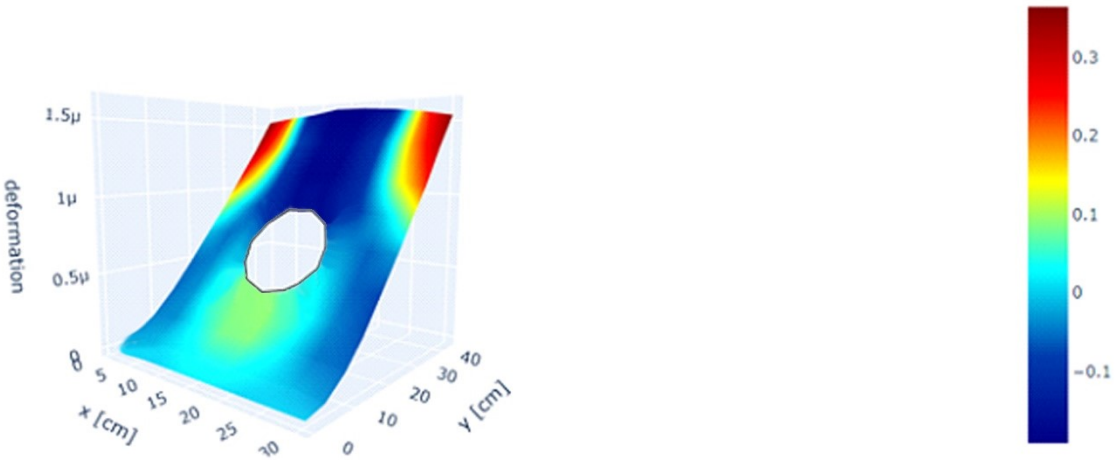



*Figure 6.20 Time=0.5 s – Predicted experimental case - Colorbar indicates accelerations in m/s²*

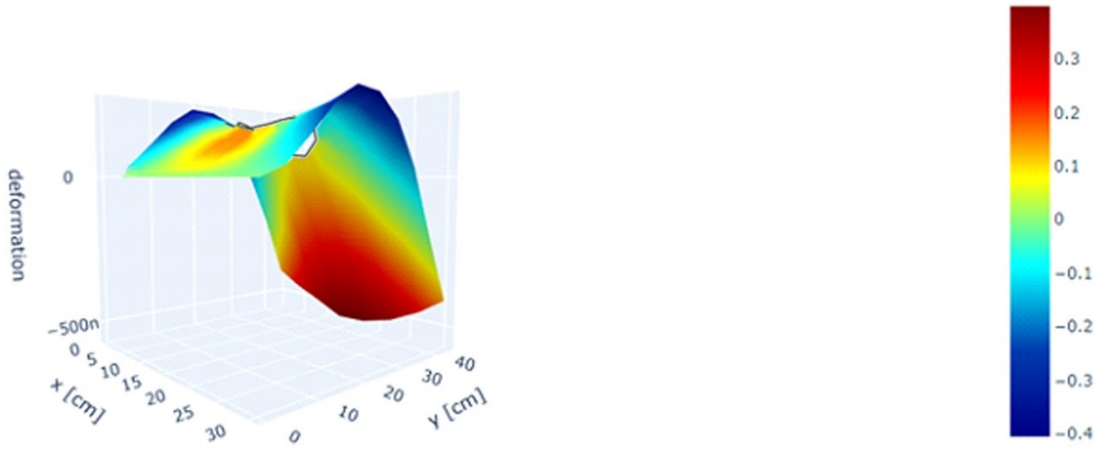



*Figure 6.21 Time=1.0 s - Initial FEM numerical case - Colorbar indicates accelerations in m/s²*

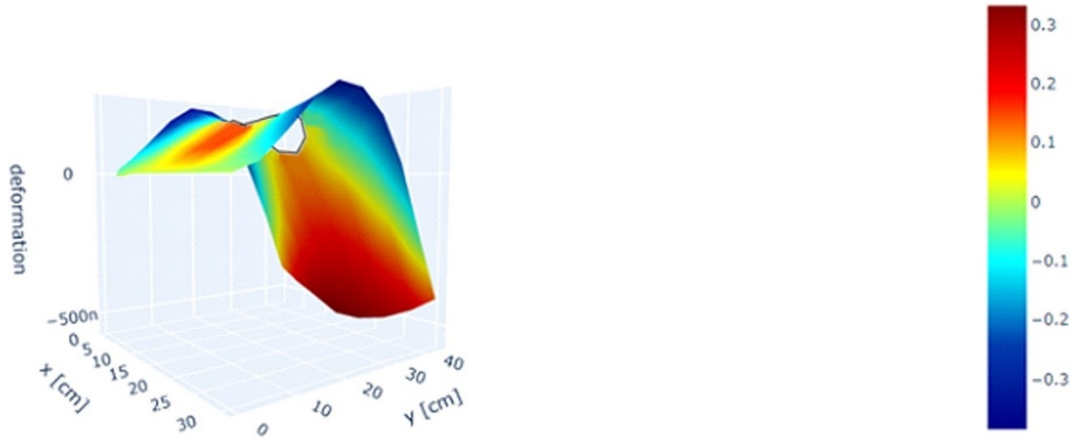



*Figure 6.22 Time=1.0 s - Predicted numerical case - Colorbar indicates accelerations in m/s²*

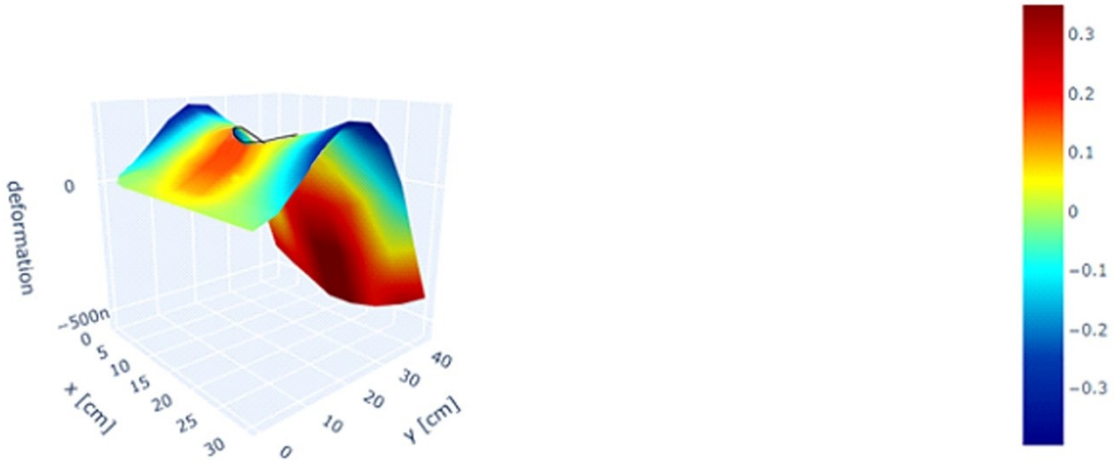



*Figure 6.23 Time=1.0 s – Predicted experimental case - Colorbar indicates accelerations in m/s²*

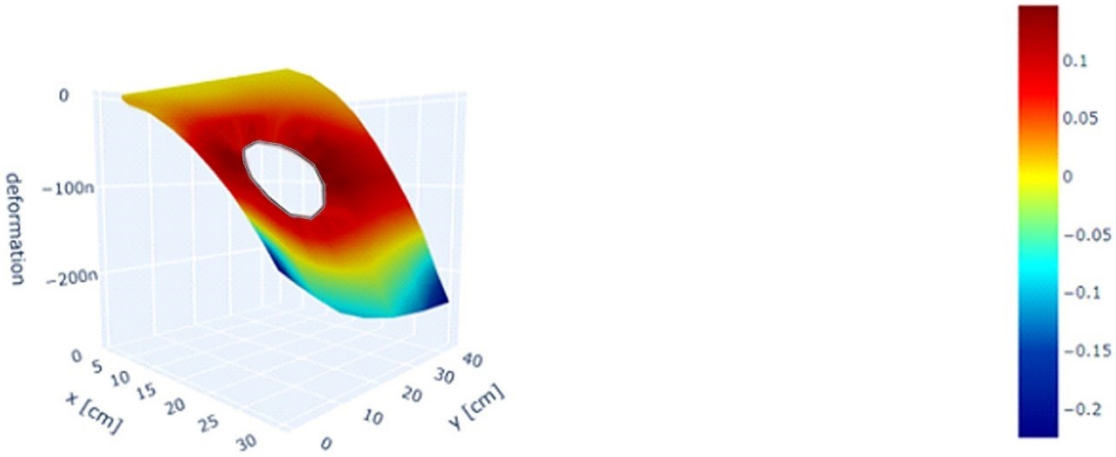



*Figure 6.24 Time=2.0 s - Initial FEM numerical case - Colorbar indicates accelerations in m/s²*

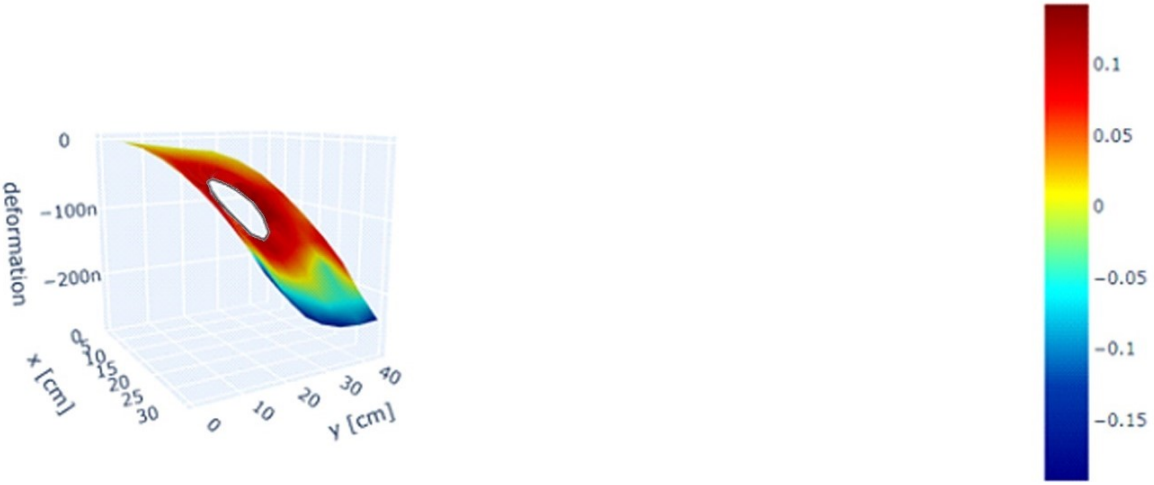



*Figure 6.25 Time=2.0 s - Predicted numerical case - Colorbar indicates accelerations in m/s²*

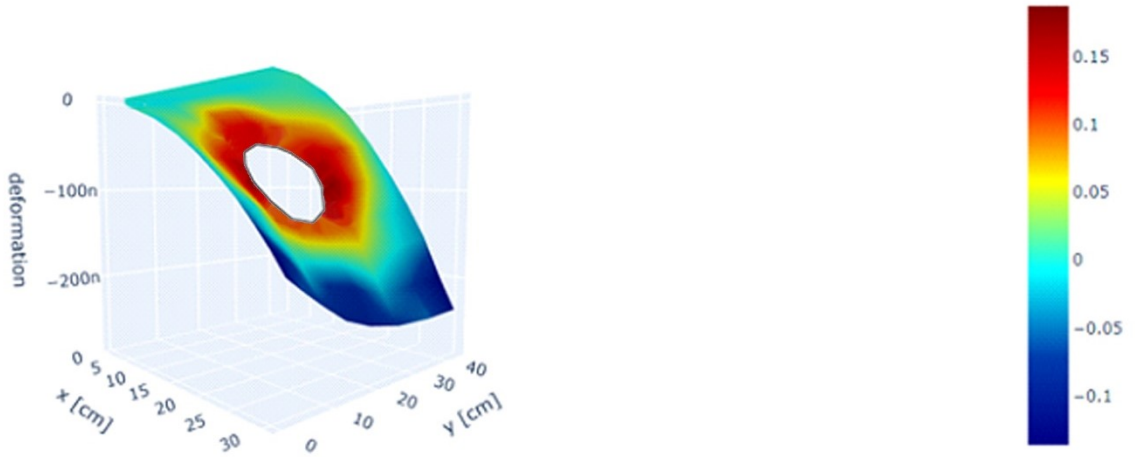



*Figure 6.26 Time=2.0 s – Predicted experimental case - Colorbar indicates accelerations in m/s²*

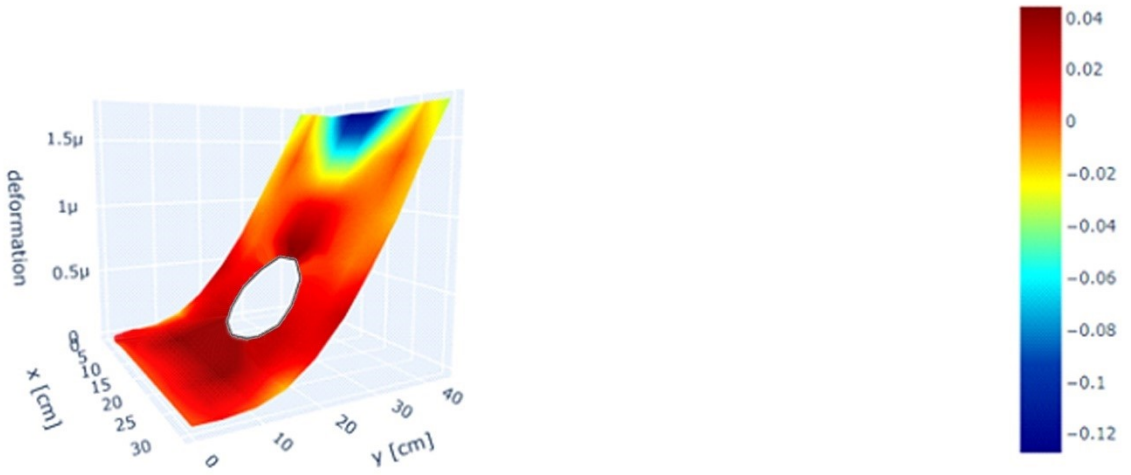



*Figure 6.27 Time=3.0 s - Initial FEM numerical case - Colorbar indicates accelerations in m/s²*

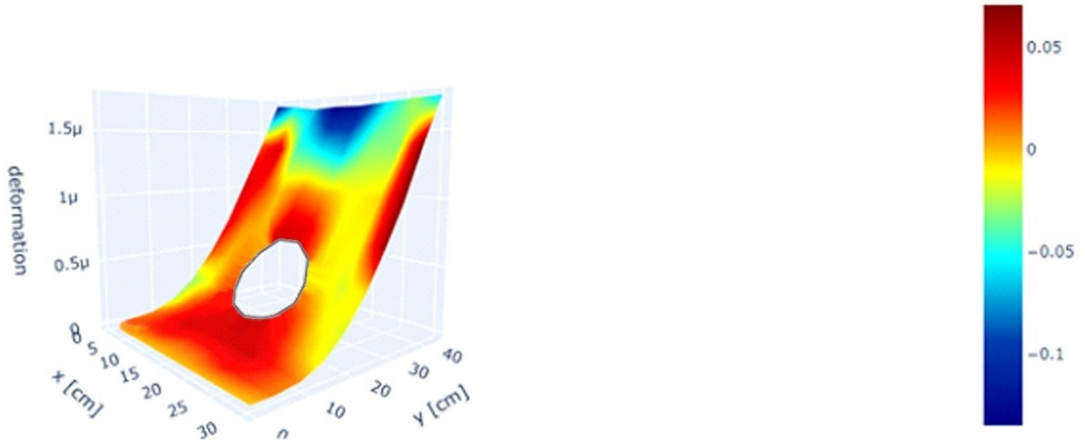



*Figure 6.28 Time=3.0 s - Predicted numerical case - Colorbar indicates accelerations in m/s²*

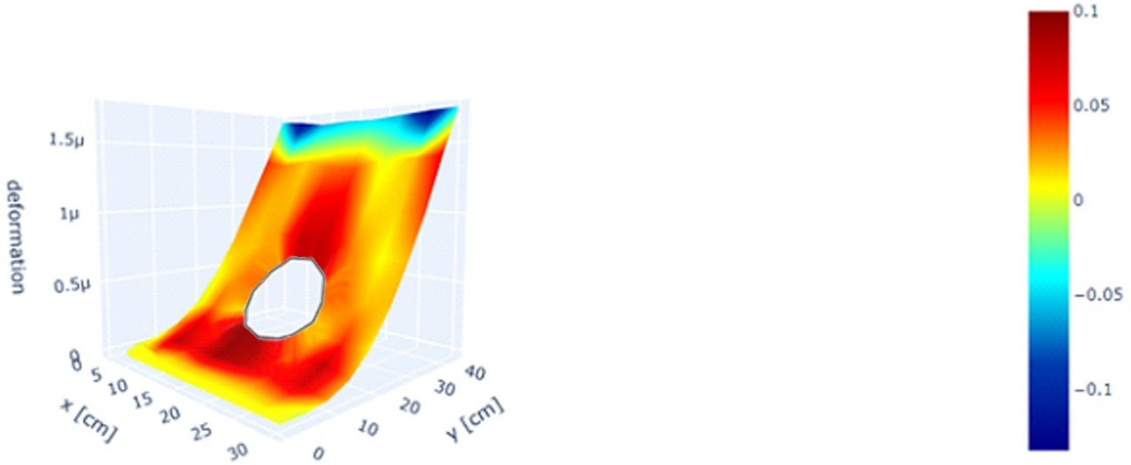



*Figure 6.29 Time=3.0 s – Predicted experimental case - Colorbar indicates accelerations in m/s²*

As in the case of the beam, 50 epochs were used for both the numerical and experimental cases. As expected, in the numerical case, there is an initially very small value for MAE and MSE; after the first few epochs, there is a rapid decrease in both train error and val error. In particular, the val error reaches a certain level of convergence after only 10 epochs, while the train error, although reaching very low values after 50 epochs, continues to decrease. In contrast, in the experimental case, the behavior of train error and val error coincides; both undergo a sharp decrease in error after half of the epochs but do not reach clear convergence after the 50 epochs used in the predictive analysis.

From the graphs in Figure representing the dataset of real and predicted values, it is evident that in both the experimental and numerical cases, the data are evenly distributed along the bisector and in the same positions in both cases. This results in a very similar R2, as observed in Table 3.

Regarding the graphs at various time points, we decided to use time instants at 0.5 sec, 1.0 sec, 2.0 sec, and 3.0 sec. The z-axis represents the calculated deformation along this axis and remains the same in the predictive case since experimental values for deformation were not available, and the algorithm could only predict accelerations, not deformations. Acceleration values are represented with isocolor lines that provide this result over the entire perforated plate represented, with the color bar on the right indicating acceleration values in $m/s^2$.

Thanks to the excellent R2 value, it is noticeable that at 0.5 seconds, both the numerical and experimental cases show excellent prediction compared to the real case, with a slight discrepancy in the top right and top left corners. This is due to the insufficient accelerometers in that portion of the plate, which hindered obtaining enough data to predict and describe the upper part of the plate.

The same issue is encountered in the subsequent analyzed time instants, while in the lower part of the plate and near the hole, a much more accurate prediction is achieved in both the numerical and experimental cases due to the presence of more data and smaller values.

### 6.4.2 Numerical-Experimental Harmonic Signal Comparison in the Regular Configuration

In this case, a cosine signal was analyzed with accelerometers positioned as in configuration 5 obtained previously, comparing the numerical values with the experimental ones.

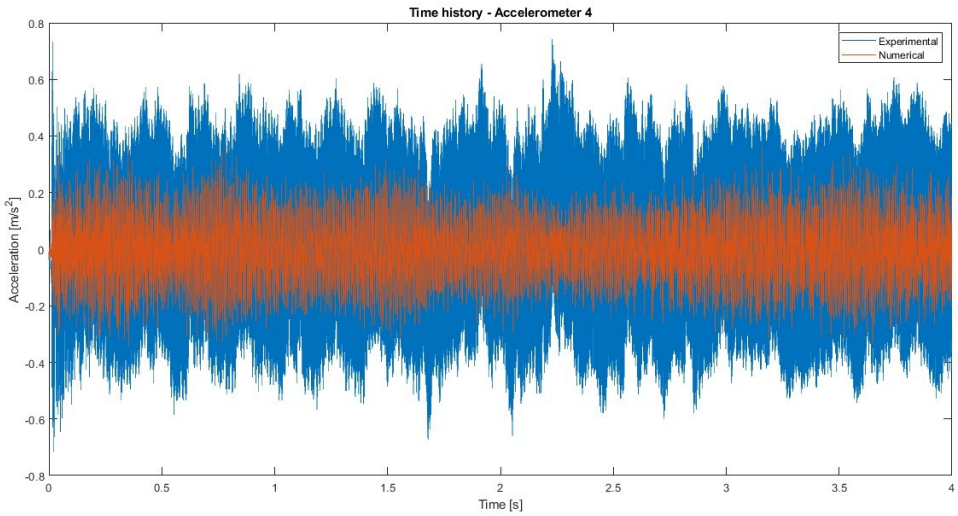



*Figure 6.30 Time history harmonic signal – Accelerometer 4*

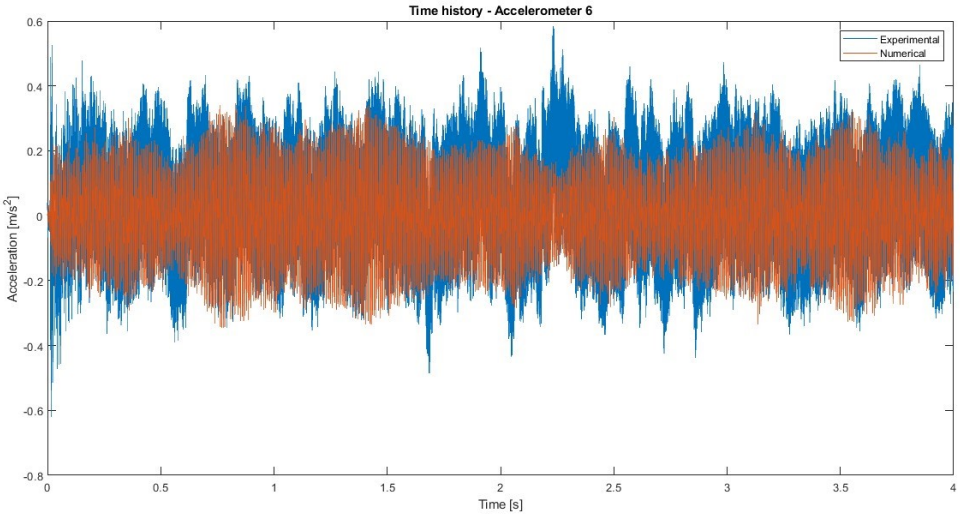



*Figure 6.31 Time history harmonic signal – Accelerometer 6*

In this examined case, the accelerometers were not positioned according to the results obtained from an algorithm but according to a regular geometry that allowed examining the central and upper part of the plate. As an example, accelerometers positioned at 4 and 6, representing the right and left sides of the plate, were illustrated. Both exhibit the same trend in both experimental and numerical signals and a significant agreement between the numerical and experimental values. These data are reflected in all the time histories of the analyzed accelerometers, presenting a consistent pattern similar to those shown in the figure. This is because the accelerometers were uniformly placed along the grid and away from the constraint, which could influence the discrepancy between the numerical and experimental parts, unlike the harmonic case represented earlier where data were obtained from accelerometers positioned according to an algorithm.

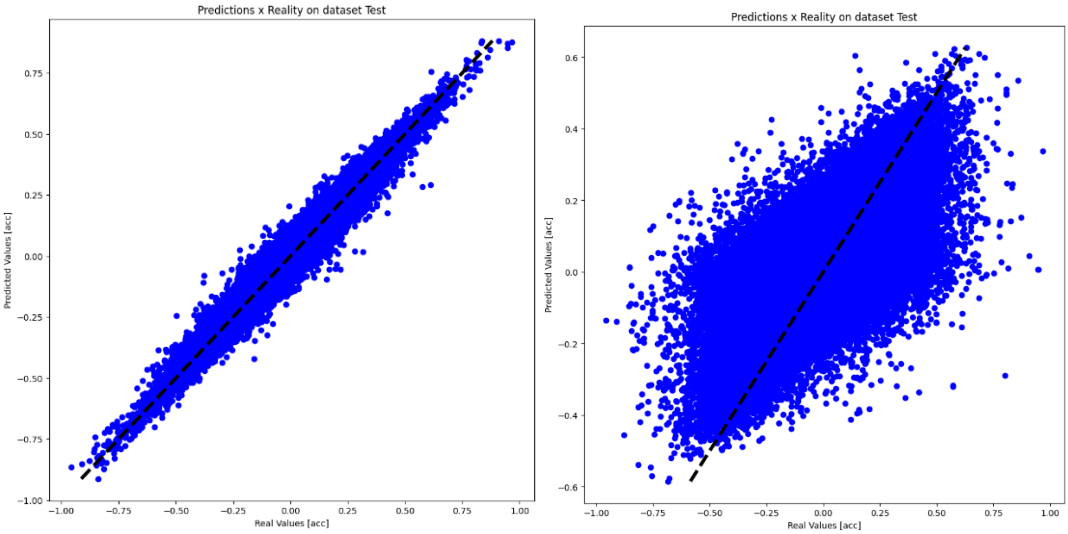



*Figure 6.32 Predictions x Reality on dataset test - L) Numerical case - R) Experimental case*

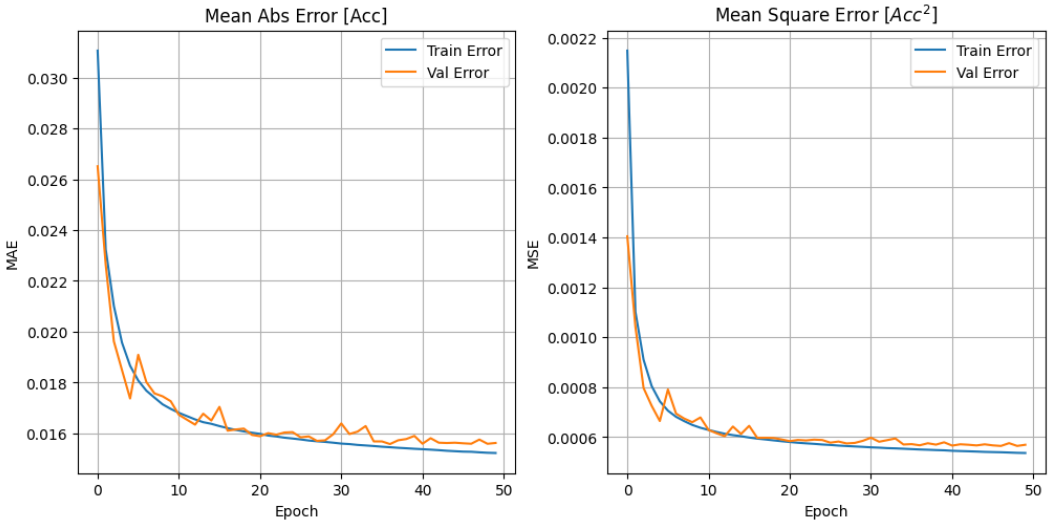



*Figure 6.33 Error evolution - Numerical case*

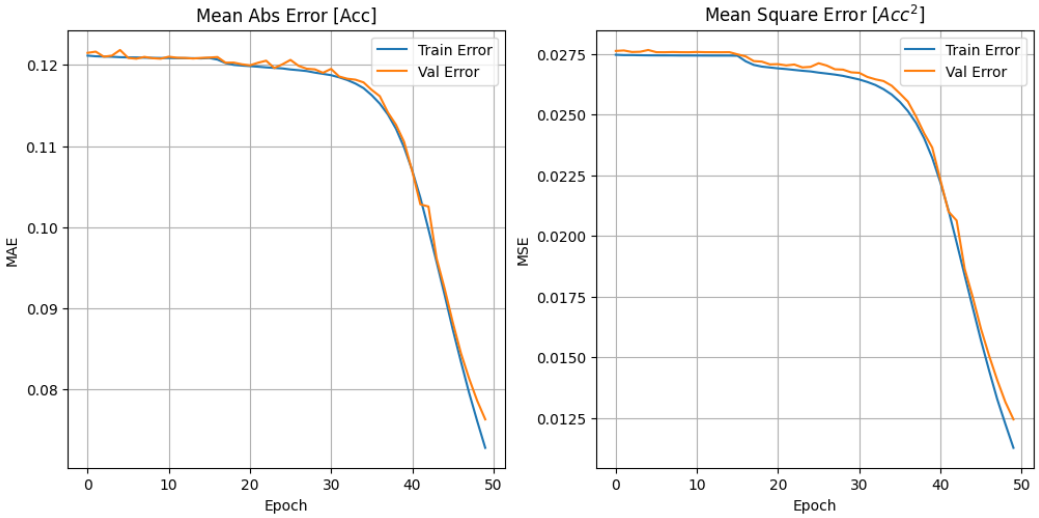



*Figure 6.34 Error evolution - Experimental case*

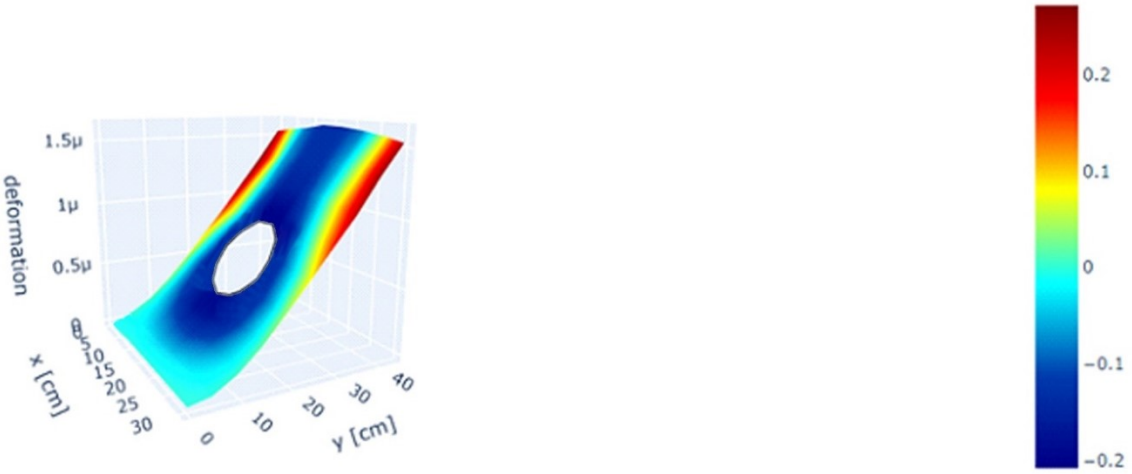



*Figure 6.35 Time=0.5 s - Initial FEM numerical case - Colorbar indicates accelerations in $m/s^2$*

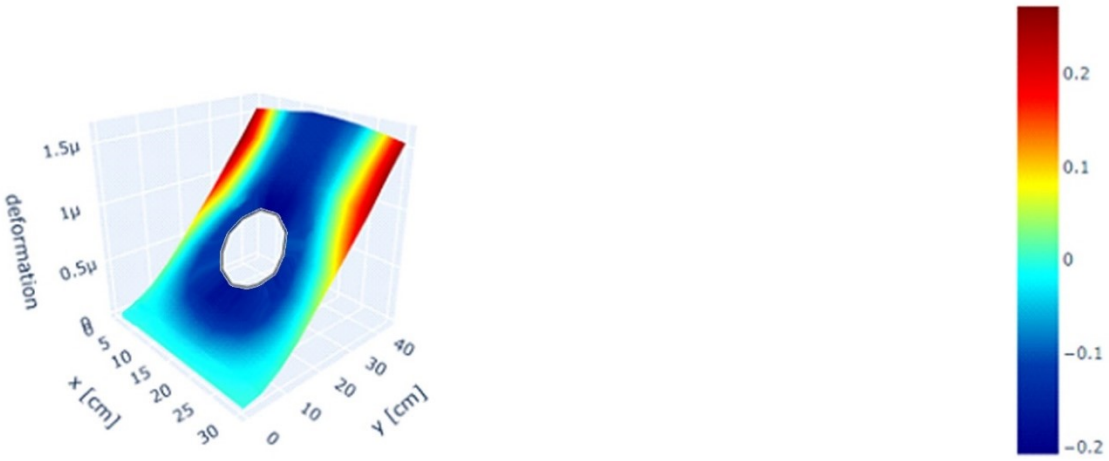



*Figure 6.36 Time=0.5 s - Predicted numerical case - Colorbar indicates accelerations in $m/s^2$*

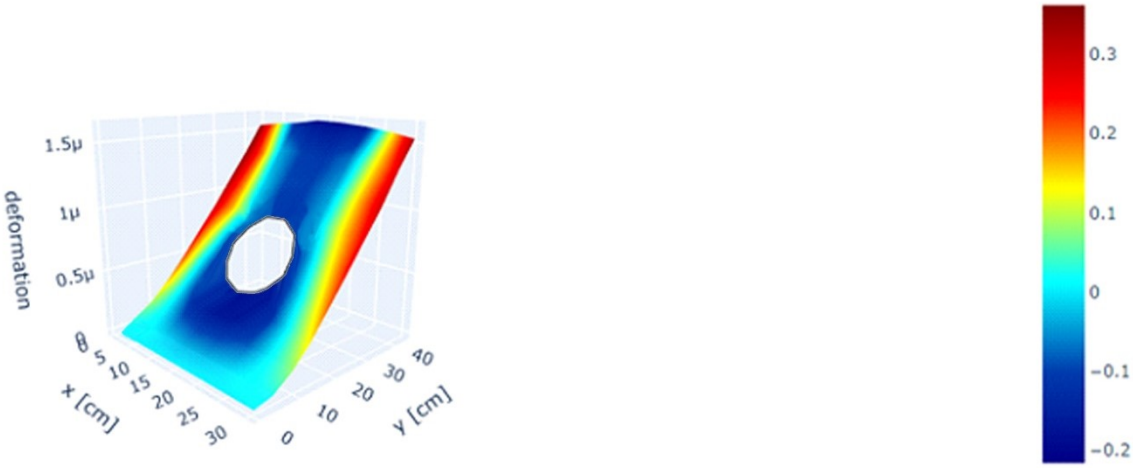



*Figure 6.37 Time=0.5 s – Predicted experimental case - Colorbar indicates accelerations in $m/s^2$*

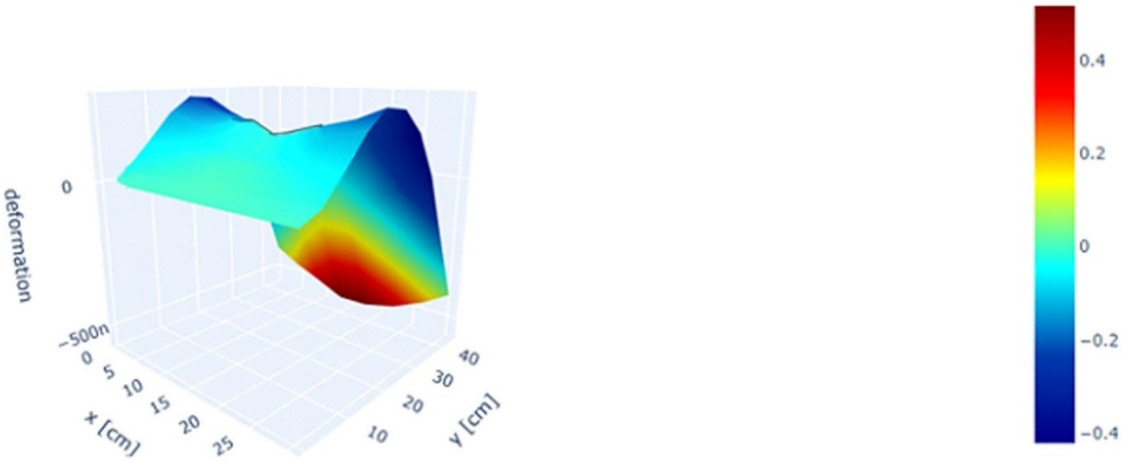



*Figure 6.38 Time=1.0 s - Initial FEM numerical case - Colorbar indicates accelerations in m/s²*

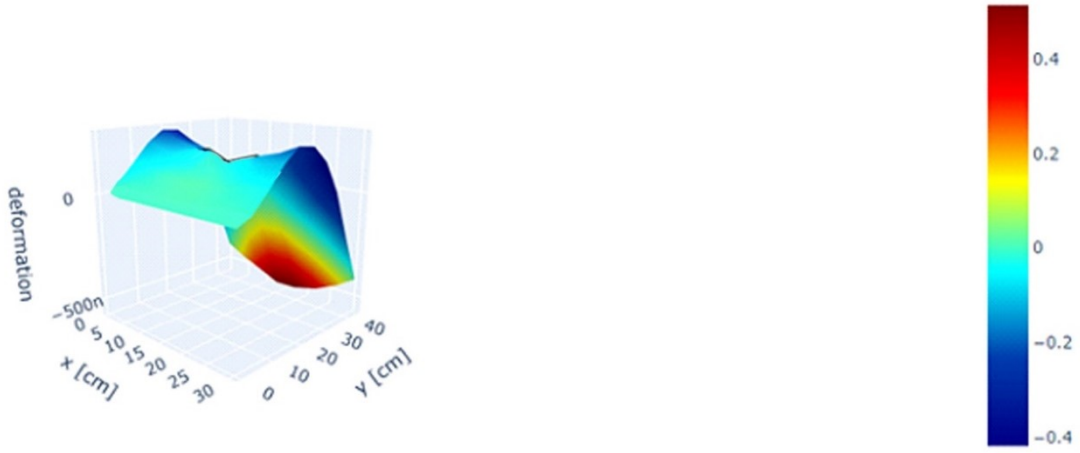



*Figure 6.39 Time=1.0 s - Predicted numerical case - Colorbar indicates accelerations in m/s²*

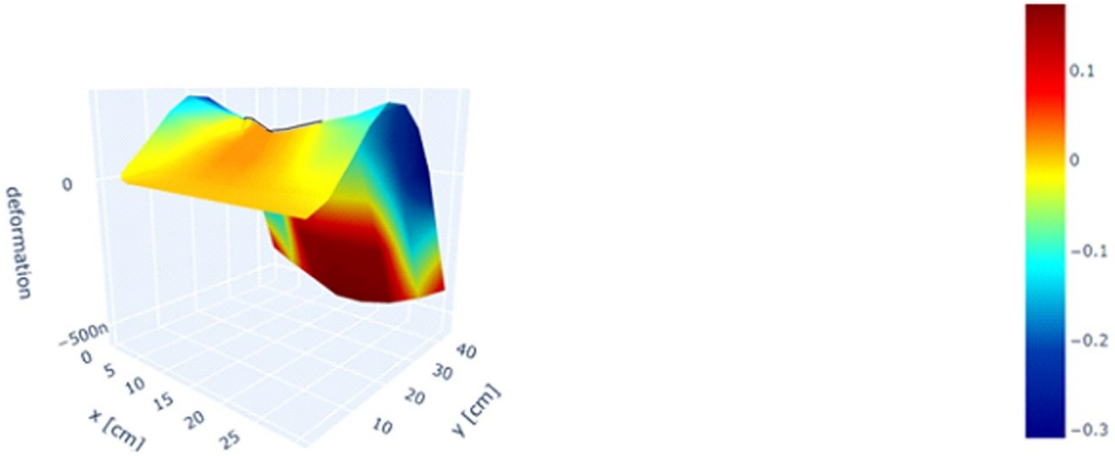



*Figure 6.40 Time=1.0 s – Predicted experimental case - Colorbar indicates accelerations in m/s²*

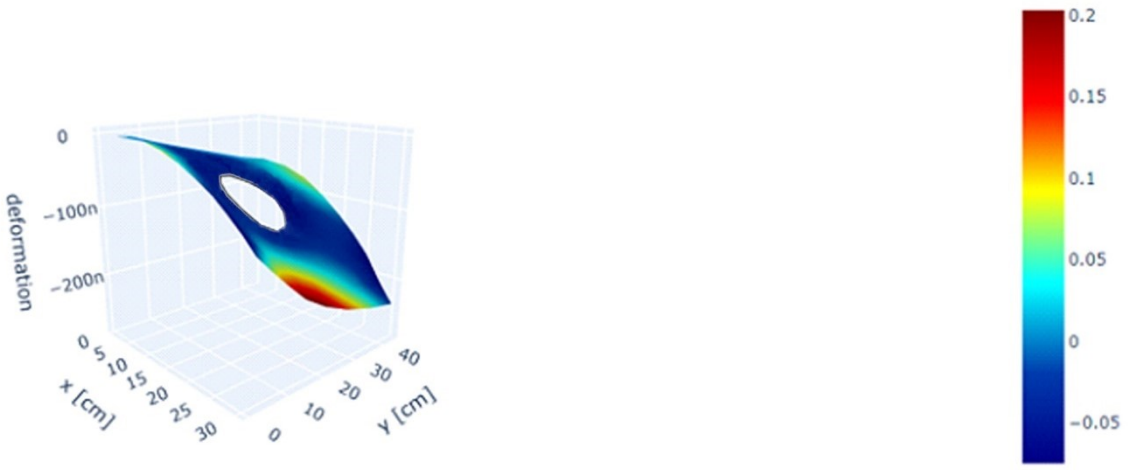



*Figure 6.41 Time=2.0 s - Initial FEM numerical case - Colorbar indicates accelerations in m/s²*

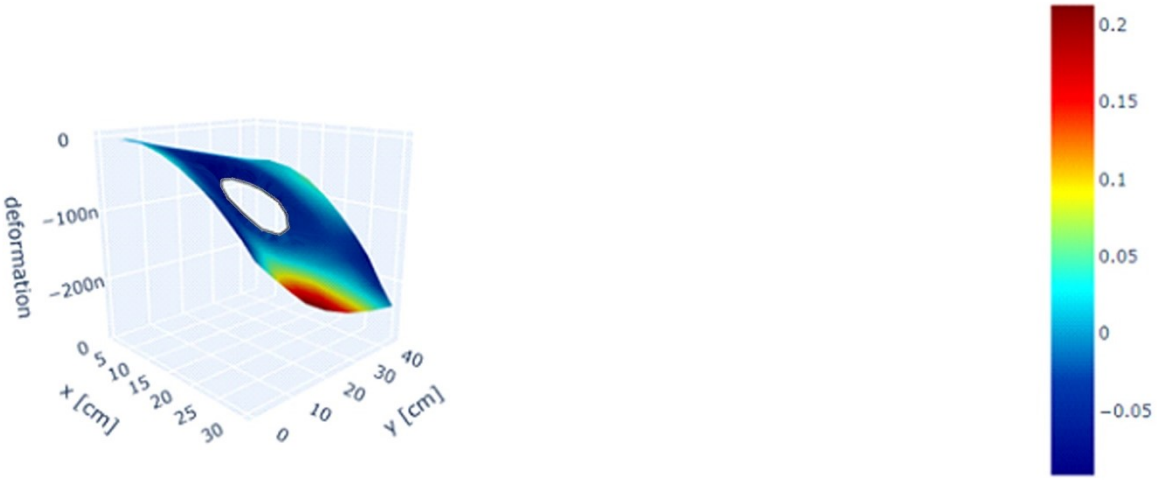



*Figure 6.42 Time=2.0 s - Predicted numerical case - Colorbar indicates accelerations in m/s²*

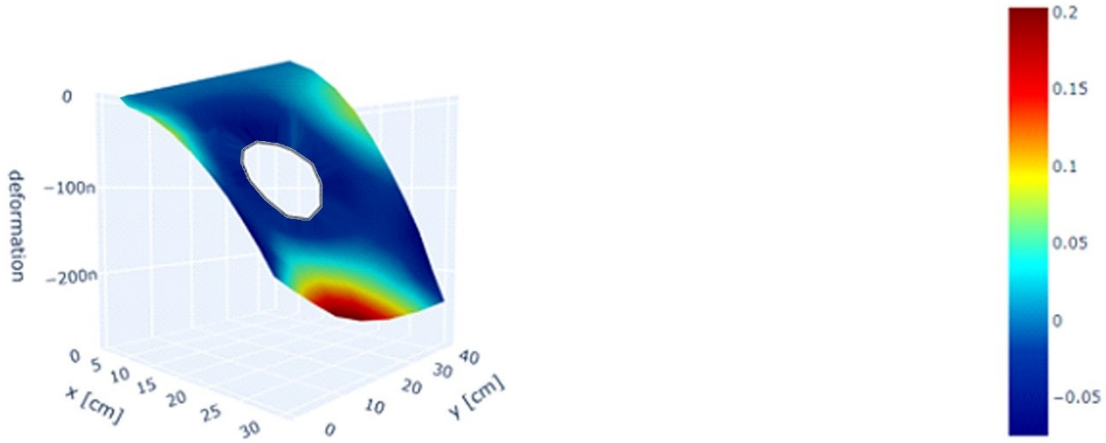



*Figure 6.43 Time=2.0 s – Predicted experimental case - Colorbar indicates accelerations in m/s²*

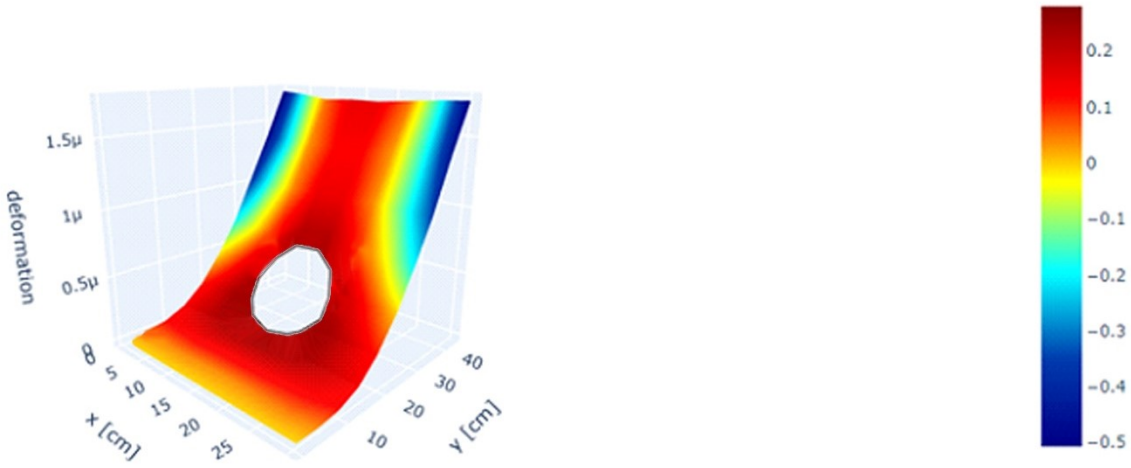



*Figure 6.44 Time=3.0 s - Initial FEM numerical case - Colorbar indicates accelerations in m/s²*

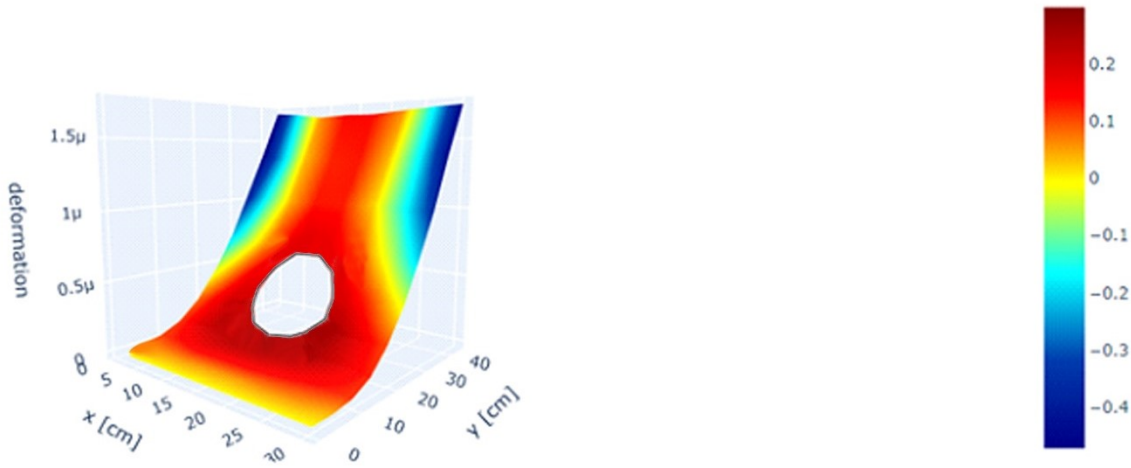



*Figure 6.45 Time=3.0 s - Predicted numerical case - Colorbar indicates accelerations in m/s²*

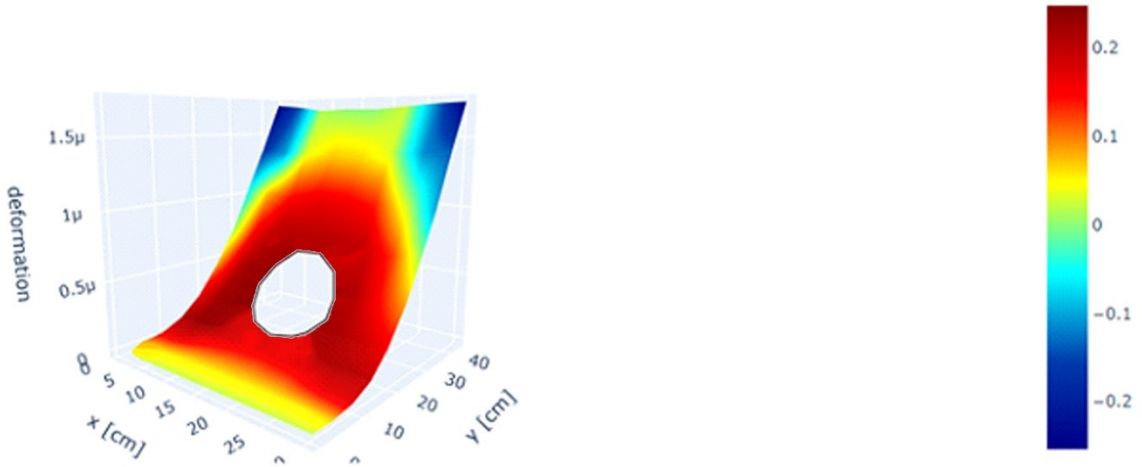



*Figure 6.46 Time=3.0 s – Predicted experimental case - Colorbar indicates accelerations in m/s²*

Unlike the previous case in Section 6.4.1, where the values obtained after the 50 epochs of the neural network algorithm were very similar, here some differences are noticeable. The first substantial difference is observed in the MAE and MSE graph, where in the numerical case, after only 10 epochs, the curves quickly converge to a value very close to zero, while in the experimental case, the curves remain more or less constant for half of the learning and then undergo a drastic decrease after half of the learning period but without converging after the 50 imposed epochs.

This difference between experimental and numerical values is also evident in the figures representing the distributions of real values with predicted ones. In the numerical case, which has a very high R2 of approximately 0.98, there is a very linear distribution along the bisector, while in the experimental case, there is a linear distribution along the bisector, but also affecting the adjacent part due to an R2 of approximately 0.60.

In the predicted time instants, numerically, there is excellent prediction of acceleration values along the various analyzed time instants, as in the case at 0.5 seconds and 2.0 seconds, where there is an excellent correspondence between Initial FEM and numerically and experimentally predicted values despite a low R2 value. However, at 3.0 seconds, it is noticeable that along the lower edges, below the hole in the plate, there is not a good prediction of experimental acceleration values.

These experimental results can be compared to the case of Configuration 3 presented earlier, which, despite having a much higher R2, had prediction errors, especially in points not covered by accelerometers. In contrast, in this case, despite having a much lower R2, the distribution of accelerometers favored more accurate acceleration predictions.

### 6.4.3 Numerical-Experimental Pseudorandom Signal Comparison

In this case, a pseudorandom signal was analyzed with accelerometers positioned as in Configuration 5 obtained previously, comparing numerical values with experimental ones.

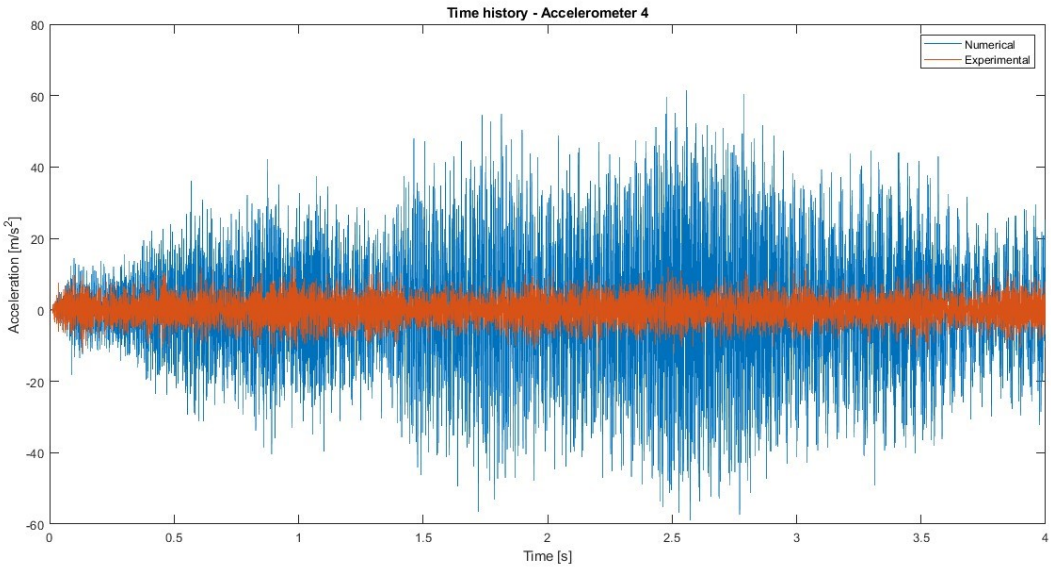



*Figure 6.47 Time history pseudorandom signal – Accelerometer 4*

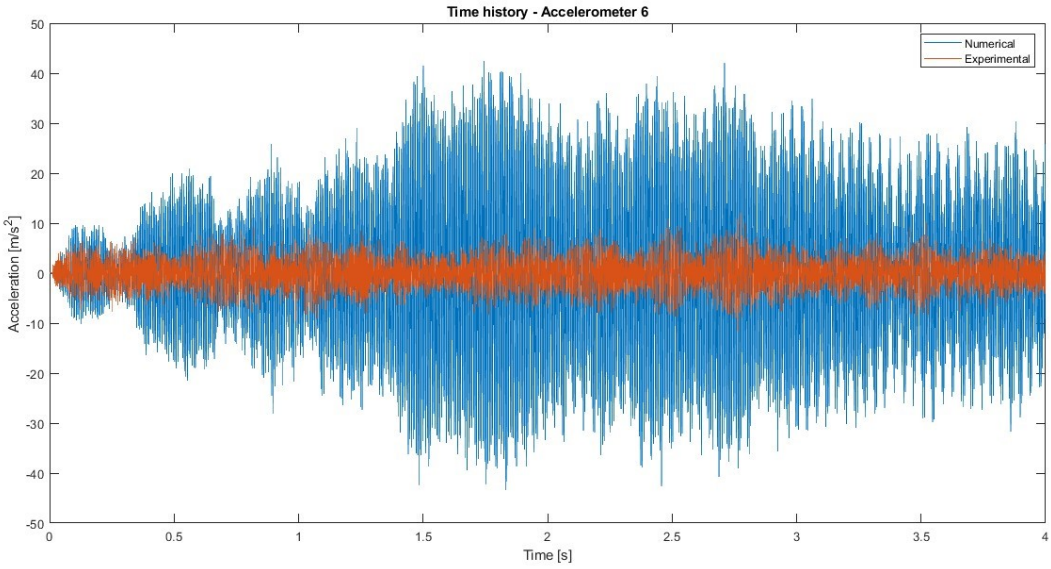



*Figure 6.48 Time history pseudorandom signal – Accelerometer 6*

The fifth configuration was analyzed similarly to the previous harmonic case. However, unlike the harmonic case where experimental and numerical values were very similar and exhibited a similar trend, the same correspondence in the signal is not found here. Indeed, after the first 0.5 seconds, the numerical values become very high, reaching unrealistic values. Due to the uniform distribution, a uniform signal is obtained along all signals acquired by the accelerometers.

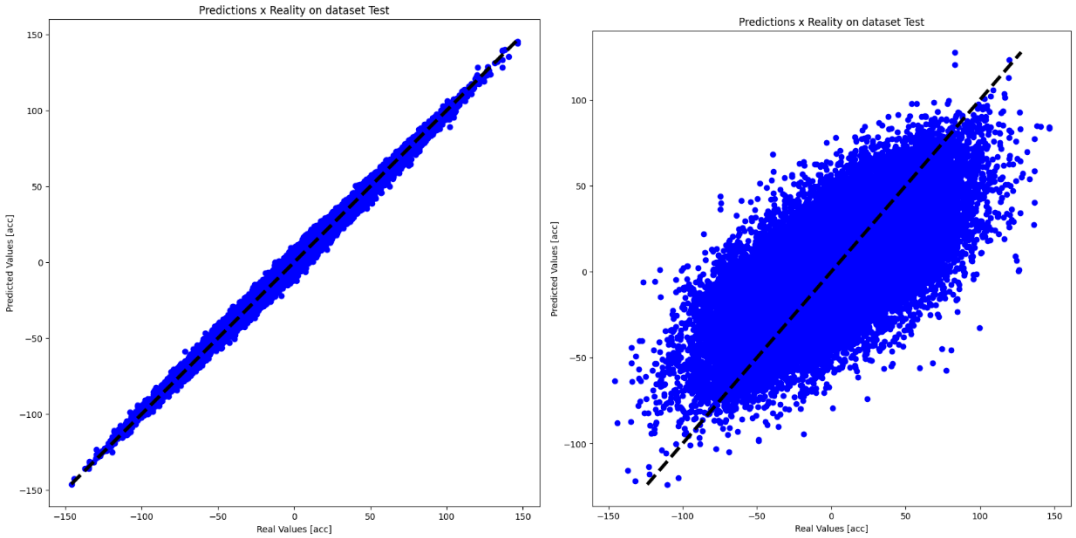



*Figure 6.49 Predictions x Reality on dataset test - L) Numerical case - R) Experimental case*

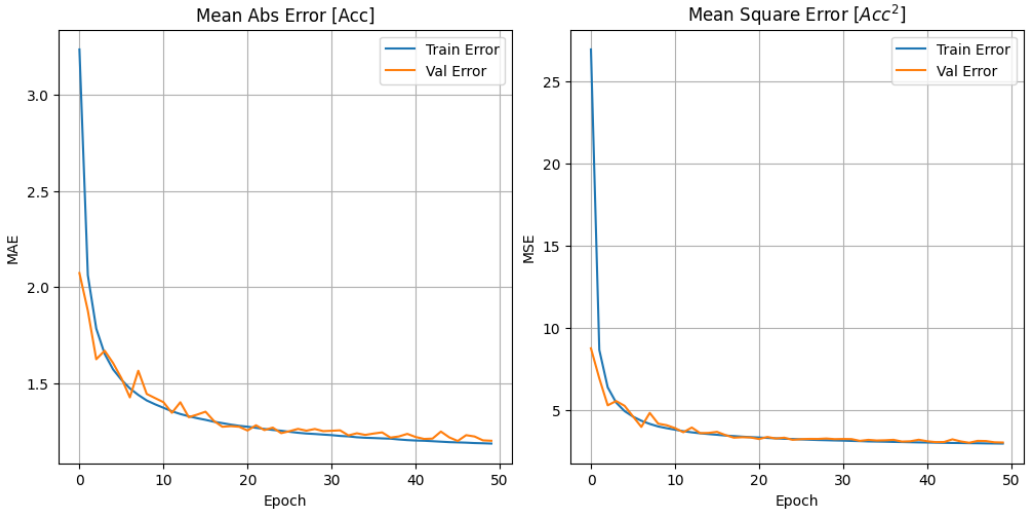



*Figure 6.50 Error evolution - Numerical case*

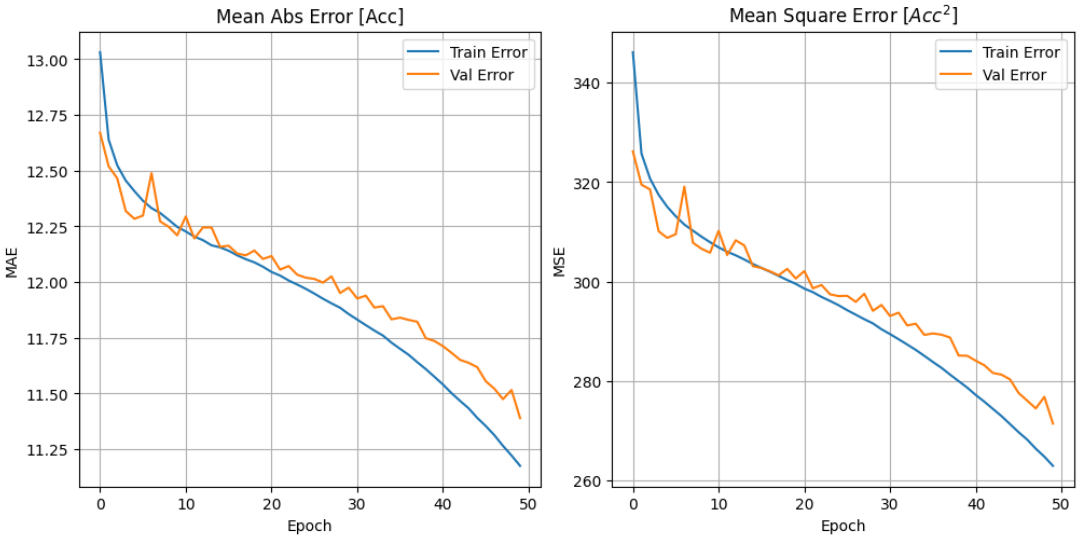



*Figure 6.51 Error evolution - Experimental case*

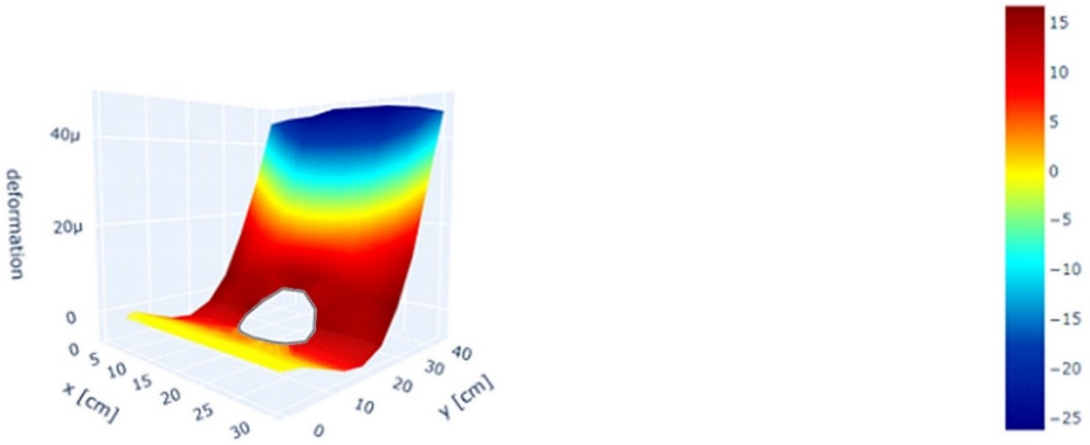



*Figure 6.52 Time=0.5 s - Initial FEM numerical case - Colorbar indicates accelerations in m/s²*

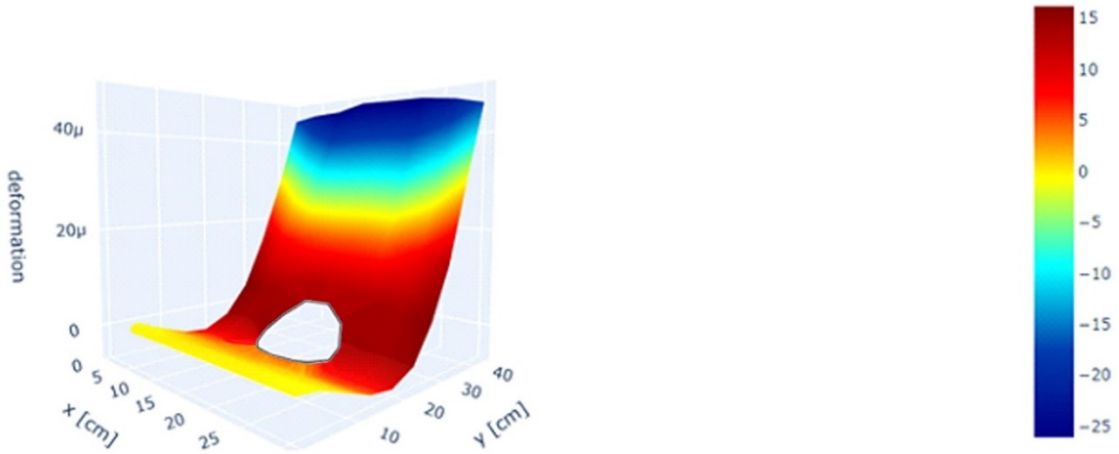



*Figure 6.53 Time=0.5 s - Predicted numerical case - Colorbar indicates accelerations in m/s²*

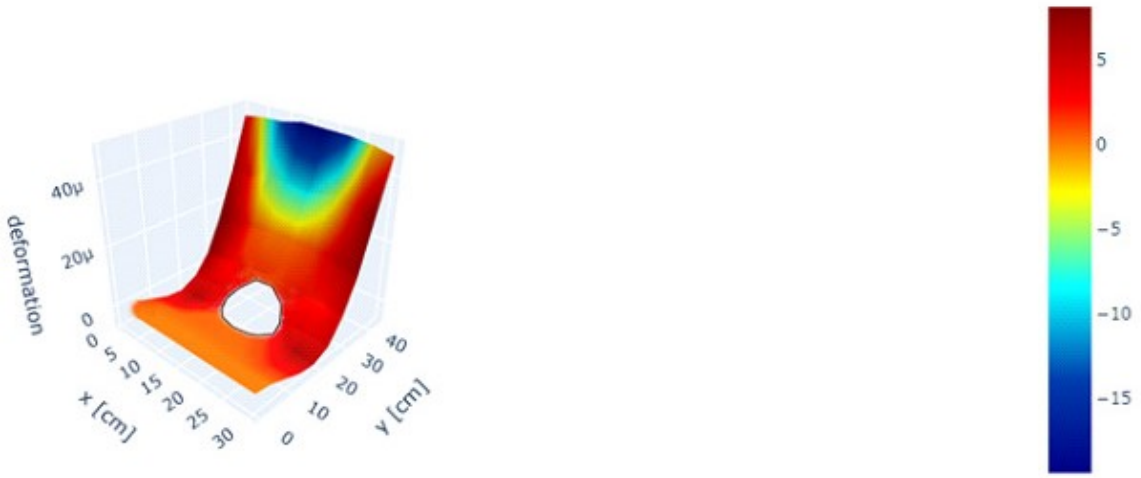



*Figure 6.54 Time=0.5 s – Predicted experimental case - Colorbar indicates accelerations in m/s²*

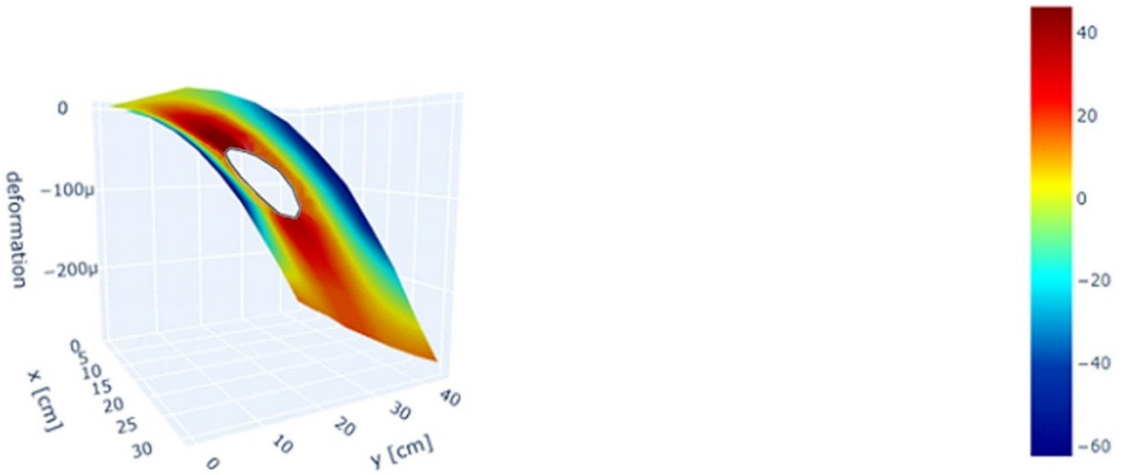



*Figure 6.55 Time=1.0 s - Initial FEM numerical case - Colorbar indicates accelerations in m/s²*

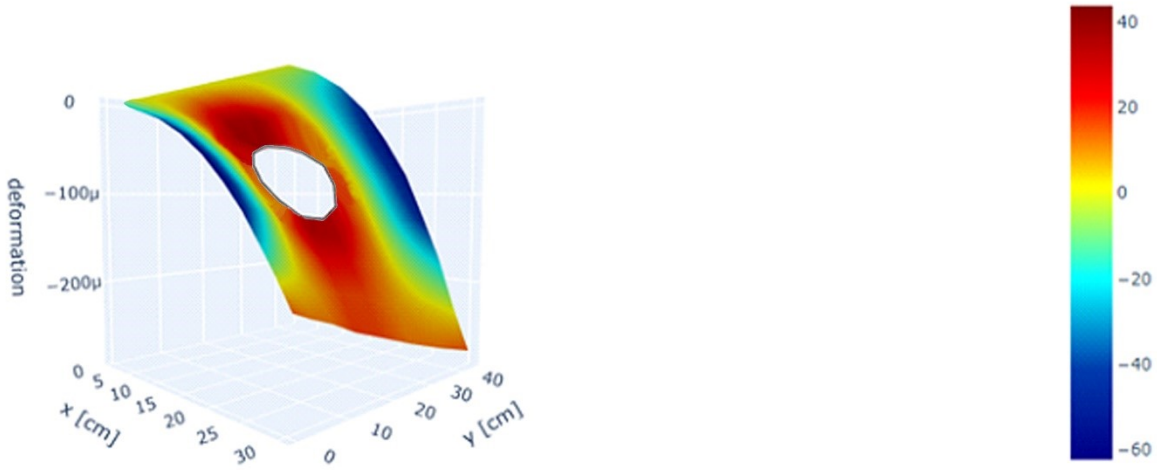



*Figure 6.56 Time=1.0 s - Predicted numerical case - Colorbar indicates accelerations in m/s²*

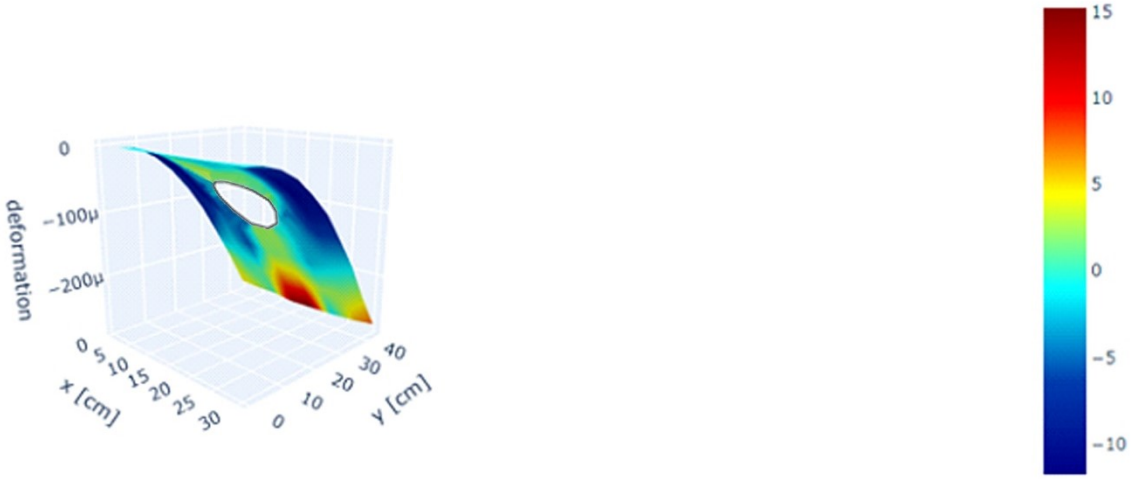



*Figure 6.57 Time=1.0 s – Predicted experimental case - Colorbar indicates accelerations in m/s²*

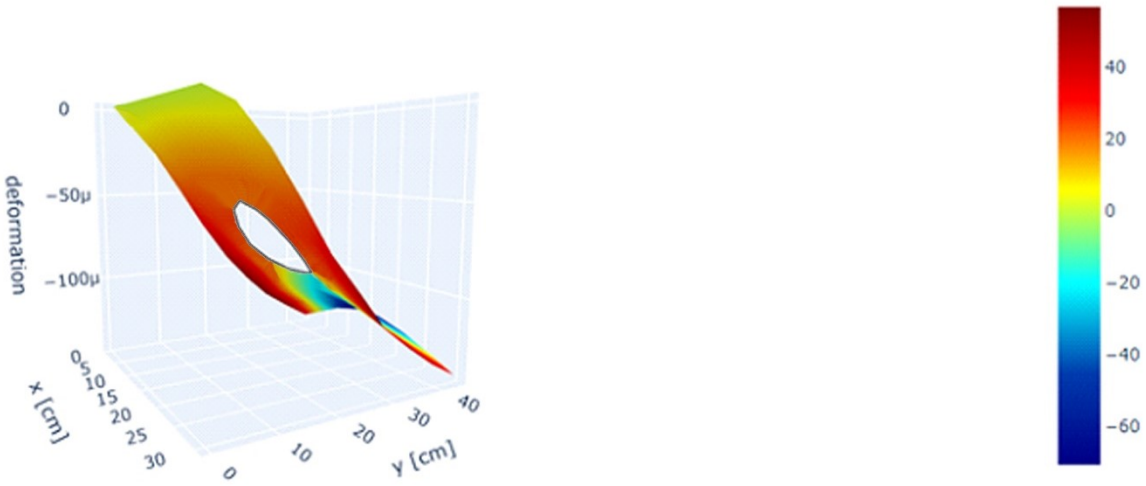



*Figure 6.58 Time=2.0 s - Initial FEM numerical case - Colorbar indicates accelerations in m/s²*

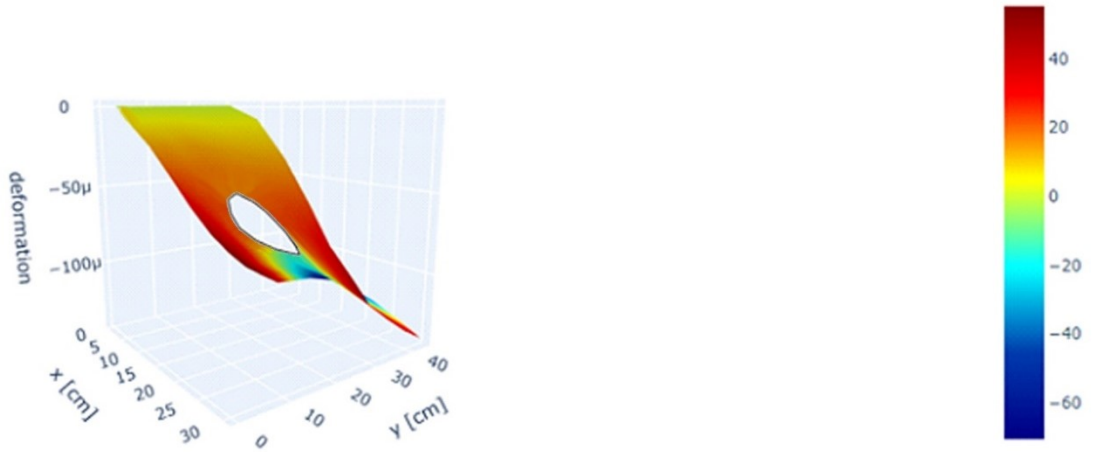



*Figure 6.59 Time=2.0 s - Predicted numerical case - Colorbar indicates accelerations in m/s²*

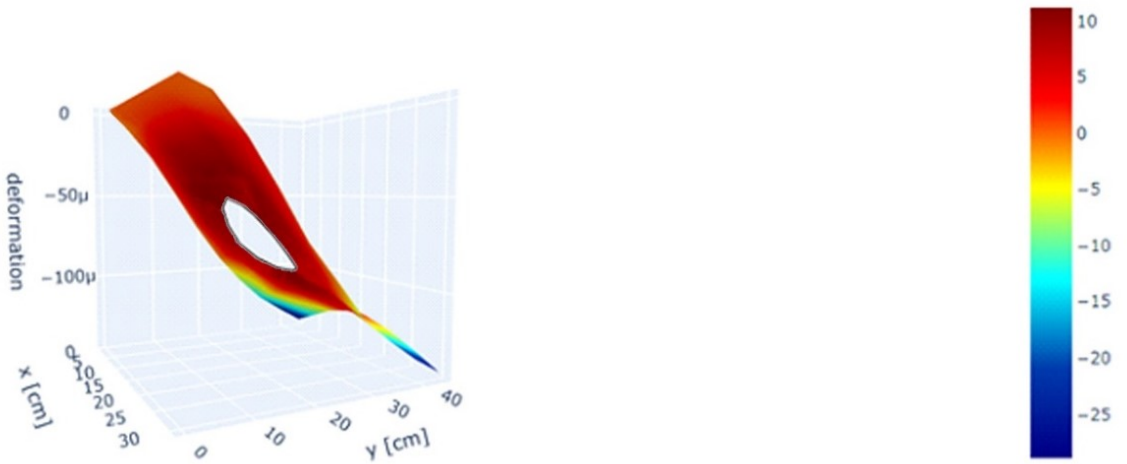



*Figure 6.60 Time=2.0 s – Predicted experimental case - Colorbar indicates accelerations in m/s²*

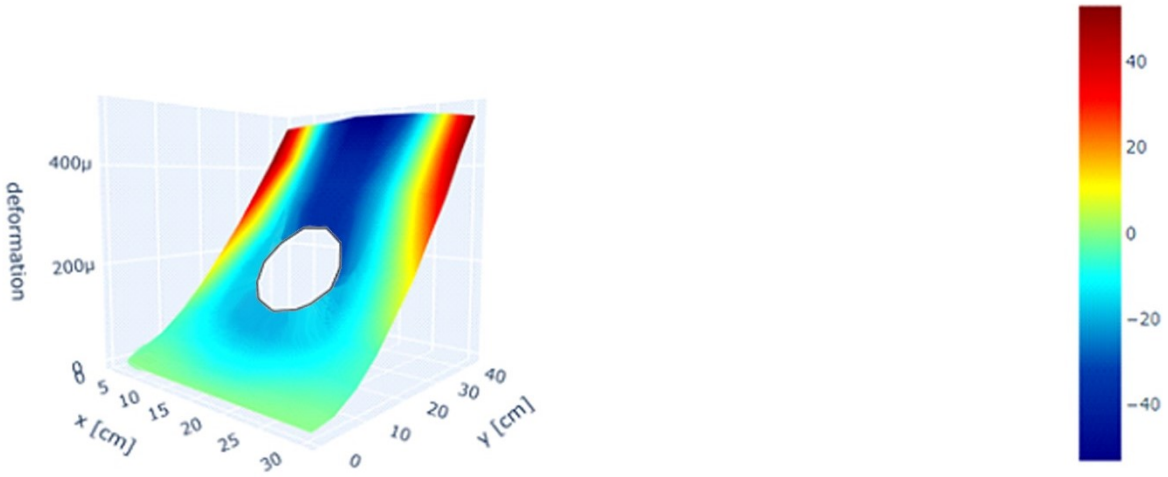



*Figure 6.61 Time=3.0 s - Initial FEM numerical case - Colorbar indicates accelerations in m/s²*

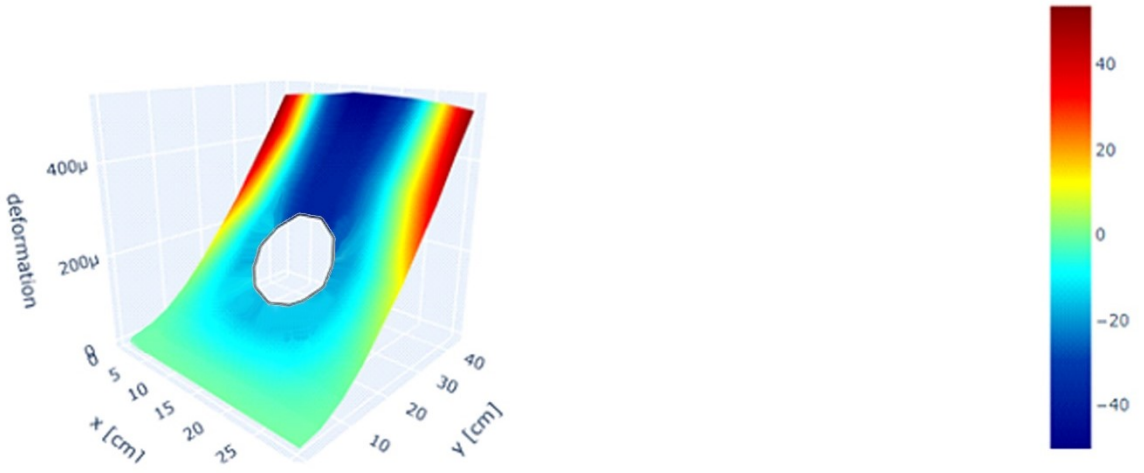



*Figure 6.62 Time=3.0 s - Predicted numerical case - Colorbar indicates accelerations in m/s²*

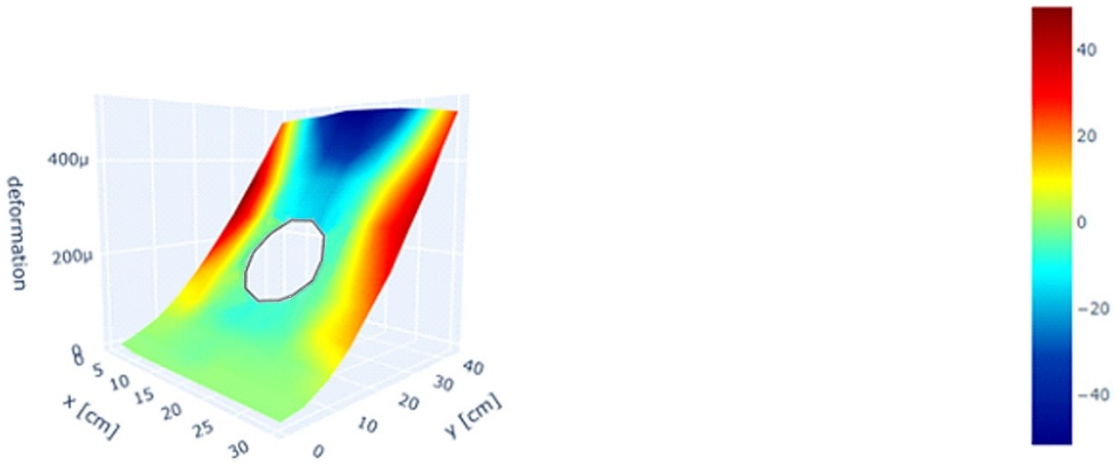



*Figure 6.63 Time=3.0 s – Predicted experimental case - Colorbar indicates accelerations in m/s²*

As evident from Figure 6.49, the real values of the Initial FEM and the predicted values are uniformly distributed along the bisector, forming a regular bisector. In contrast, in the experimental case, while the values also distribute along the bisector, they form a much broader distribution cloud. This, along with the significant difference between numerical and experimental values, results in very high MAE and MSE errors for the pseudorandom case. In the numerical case, the curve reaches convergence after just 10 epochs but with a value much higher than the almost zero value in the harmonic case. However, in the experimental case, the curve consistently decreases after 10 epochs, but unlike the numerical case, it continues to decrease without reaching convergence during the 50 epochs used.

Given the very high R2 value in the numerical case, it is evident that the predicted acceleration results correspond to those of the Initial FEM, leading to an excellent match between the obtained acceleration values and triggering the phenomenon of overfitting at all examined time points. Although the results in the numerical case are overly optimal, this is not the case for experimental data where the R2 value is relatively low compared to the examined cases, even though among all configurations, this one has the best R2 value.

The best time point is at 0.5 seconds because experimental and numerical values are very similar at that point, unlike the rest of the signal where numerical values are much higher. In fact, in the remaining examined time points in the experimental case, an excellent prediction is not achieved, especially with incorrect results in the lateral and lower areas of the plate where there are no values acquired by the accelerometers.

### 6.4.4 Numerical-Experimental Chirp Signal Comparison

In this case, a chirp signal was analyzed with accelerometers positioned as in configuration 1 obtained previously, comparing numerical values with experimental ones:

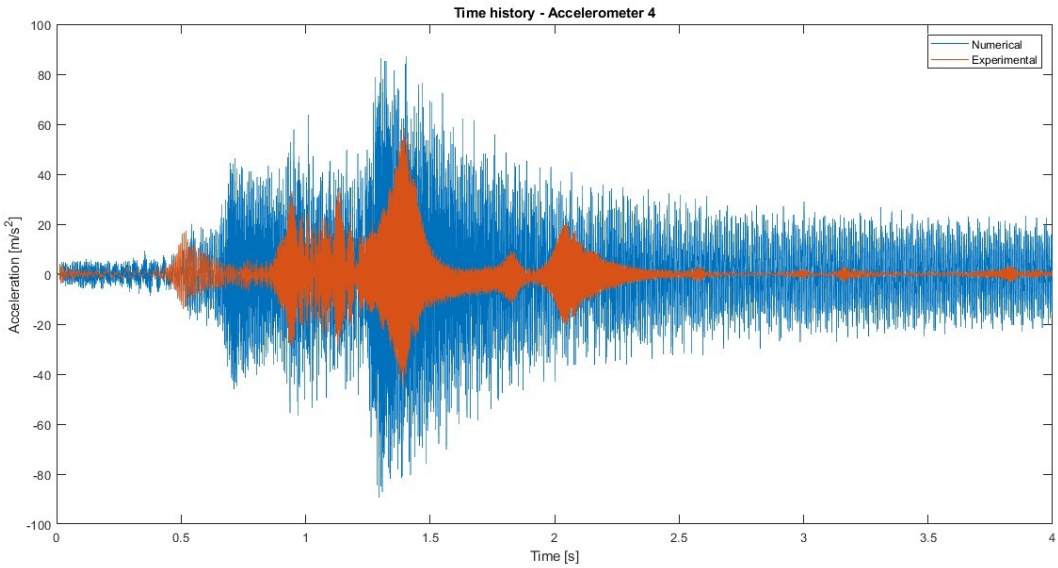



*Figure 6.64 Time history chirp signal – Accelerometer 4*

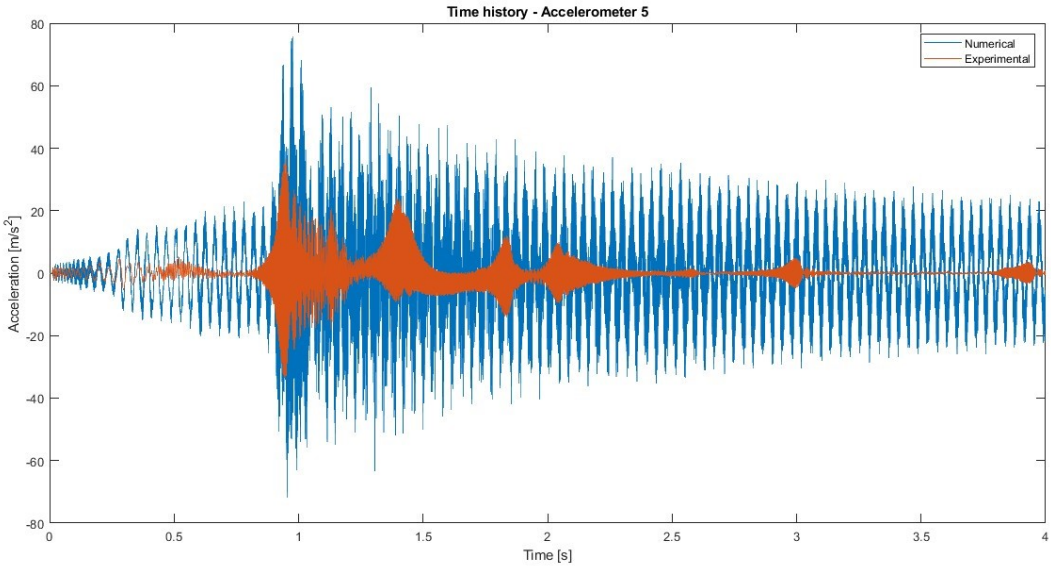



*Figure 6.65 Time history chirp signal – Accelerometer 5*

In the following case, configuration 1 obtained with the EIM algorithm was examined, where the position of four accelerometers is in the lower right part of the plate along its edge. Among these four accelerometers, accelerometer 5 is positioned, where the trend of the experimentally acquired signal is very similar up to 1 second, after which there is a deviation with numerically higher values compared to the experimental case. For accelerometer 4, the experimental signal trend is much more similar to the numerically obtained signal compared to the previously analyzed case. The numerical values of accelerations are also very similar in the first two seconds, while in the subsequent two seconds, the numerically obtained values are much higher.

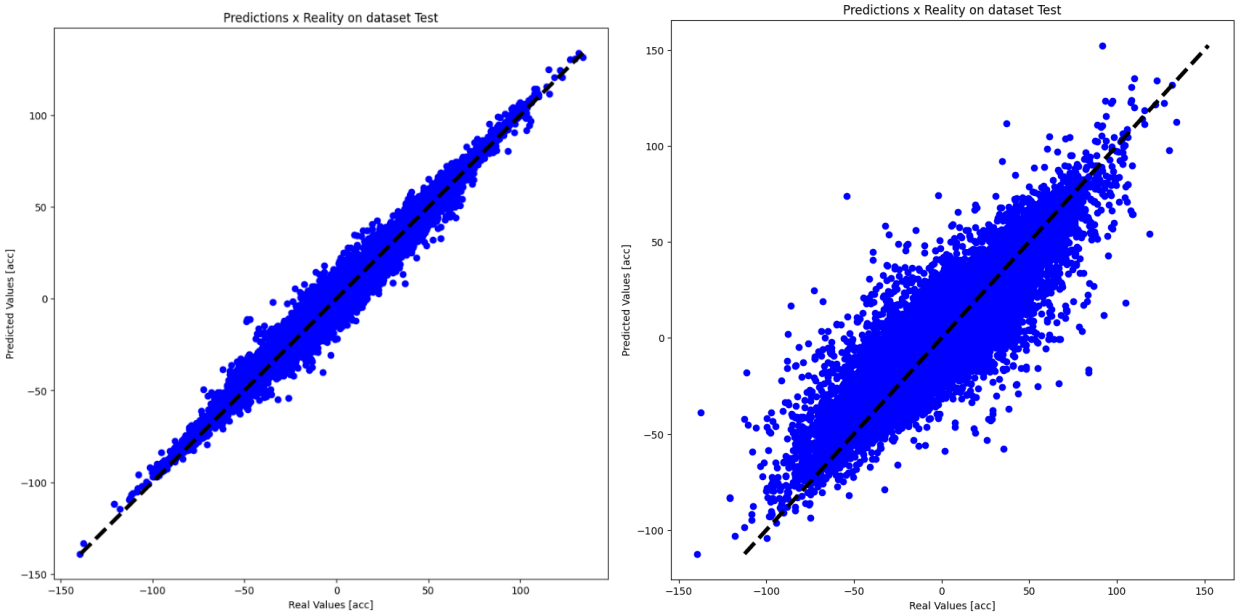



*Figure 6.66 Predictions x Reality on dataset test - L) Numerical case - R) Experimental case*

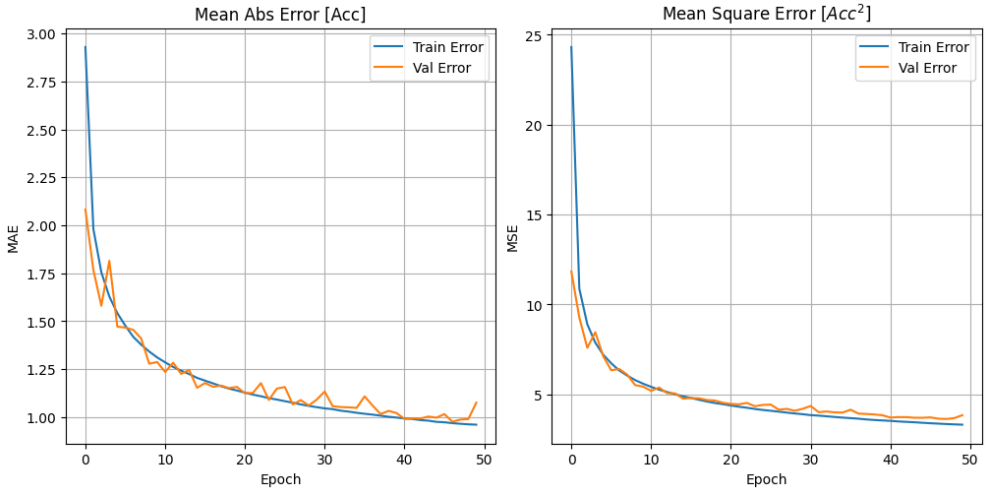



*Figure 6.67 Error evolution - Numerical case*

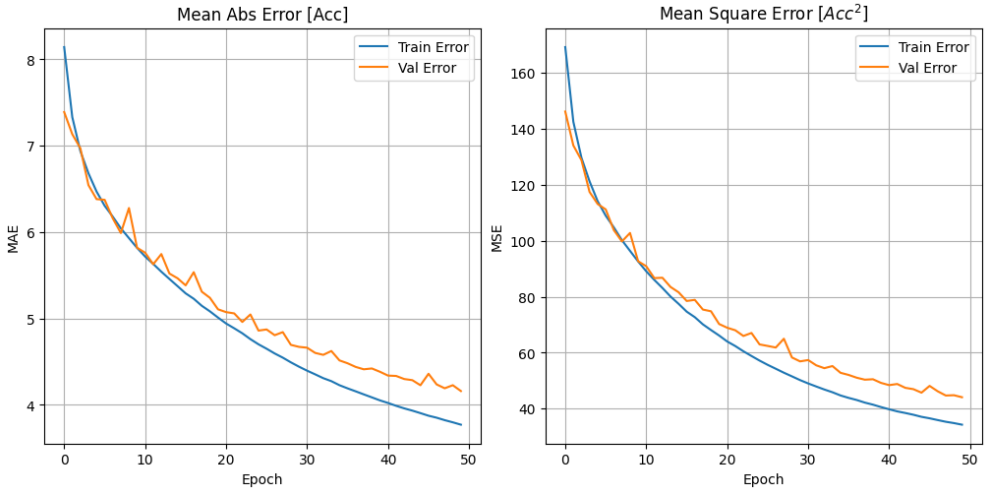



*Figure 6.68 Error evolution - Experimental case*

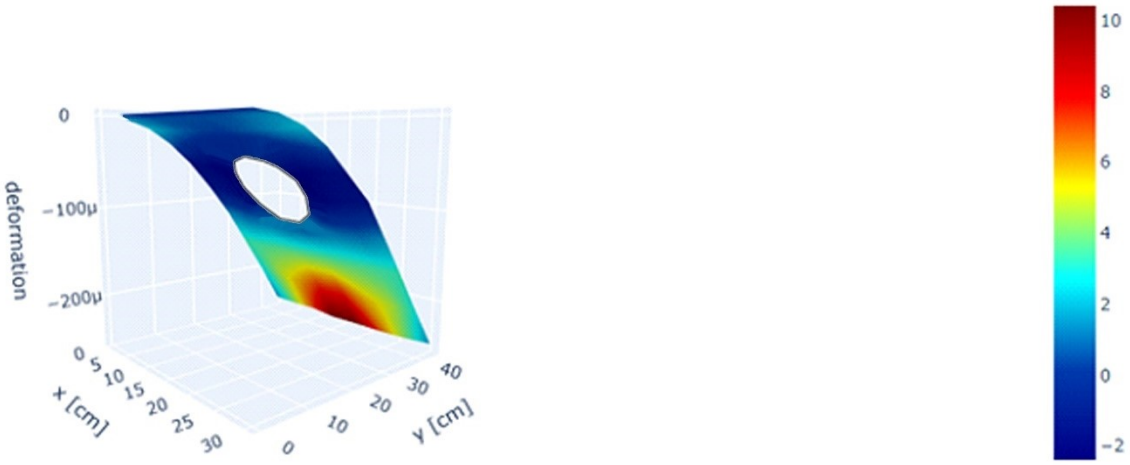



*Figure 6.69 Time=0.5 s - Initial FEM numerical case - Colorbar indicates accelerations in $m/s^2$*

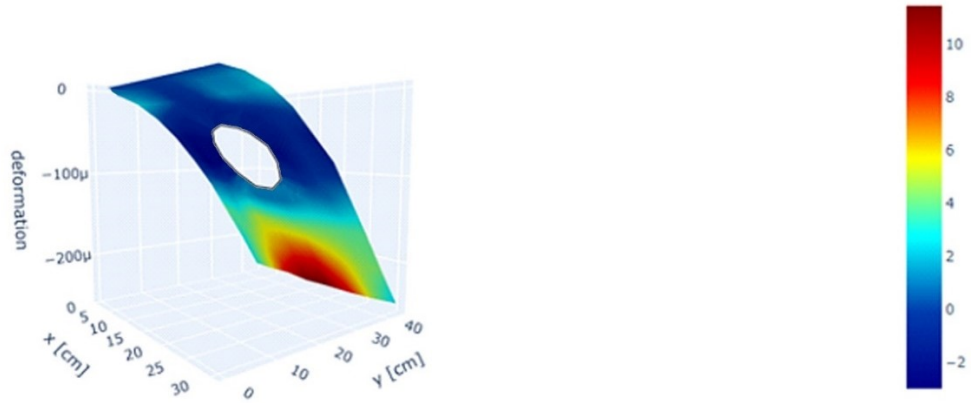



*Figure 6.70 Time=0.5 s - Predicted numerical case - Colorbar indicates accelerations in $m/s^2$*

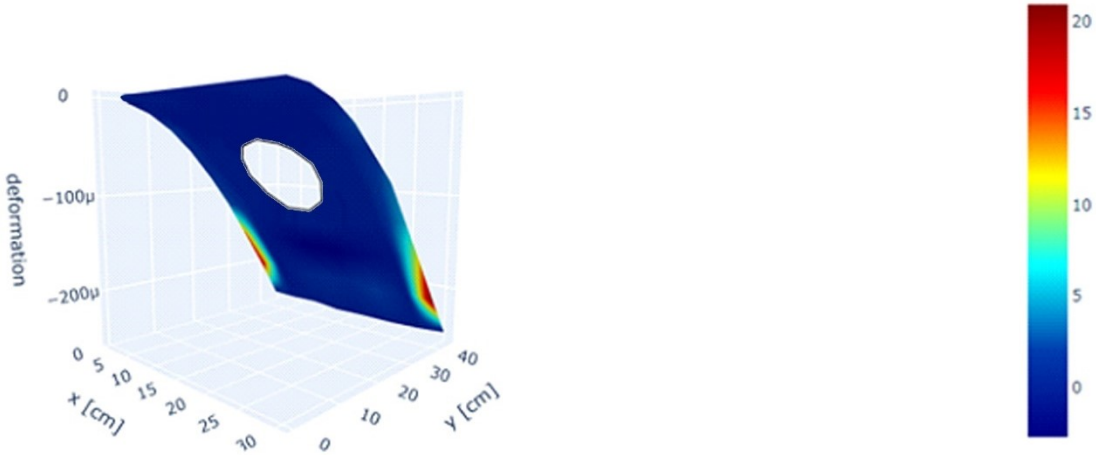



*Figure 6.71 Time=0.5 s – Predicted experimental case - Colorbar indicates accelerations in $m/s^2$*

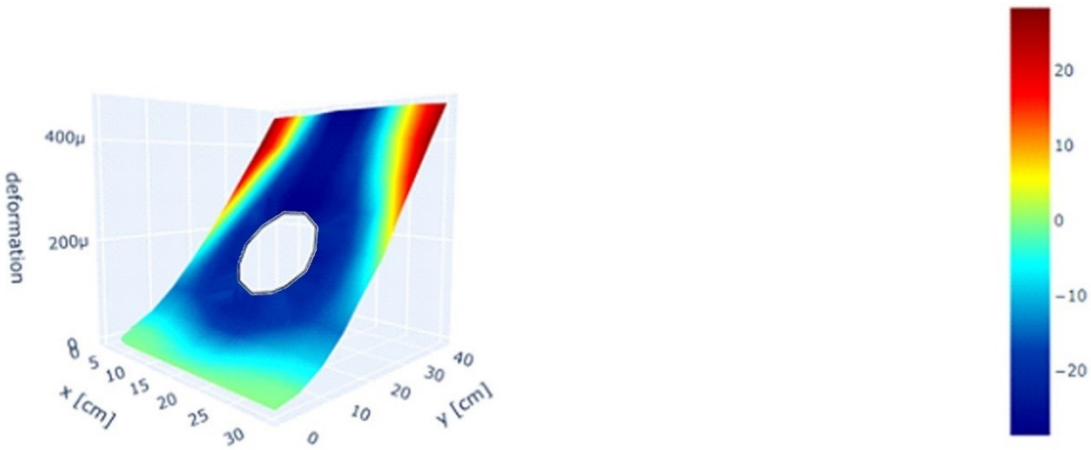



*Figure 6.72 Time=1.0 s - Initial FEM numerical case - Colorbar indicates accelerations in m/s²*

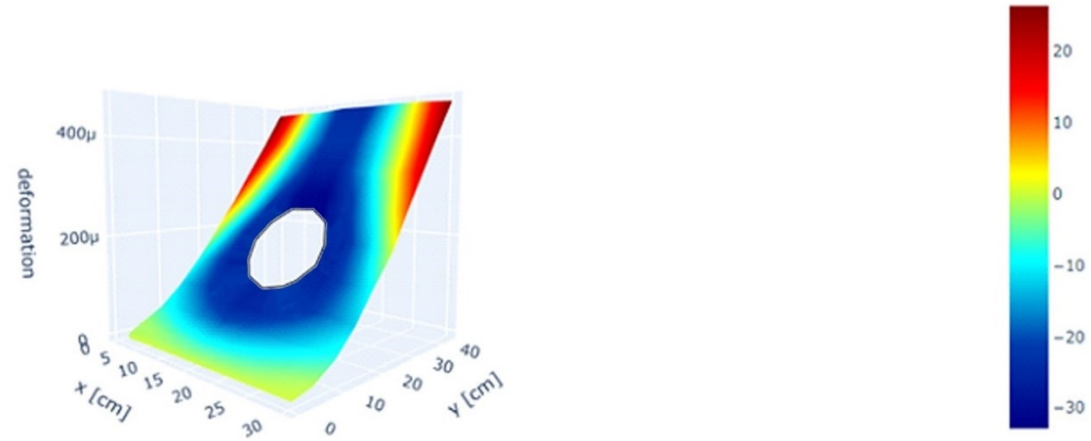



*Figure 6.73 Time=1.0 s - Predicted numerical case - Colorbar indicates accelerations in m/s²*

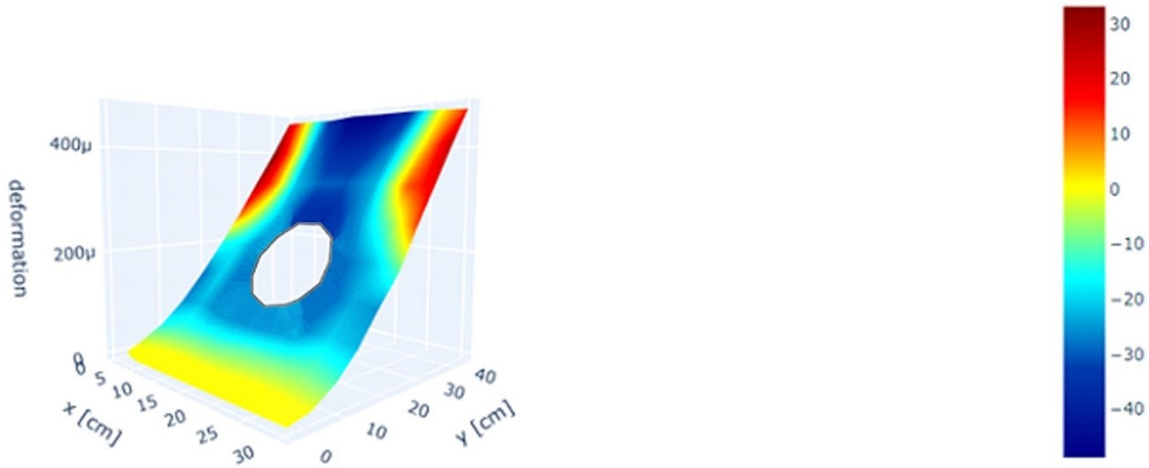



*Figure 6.74 Time=1.0 s – Predicted experimental case - Colorbar indicates accelerations in m/s²*

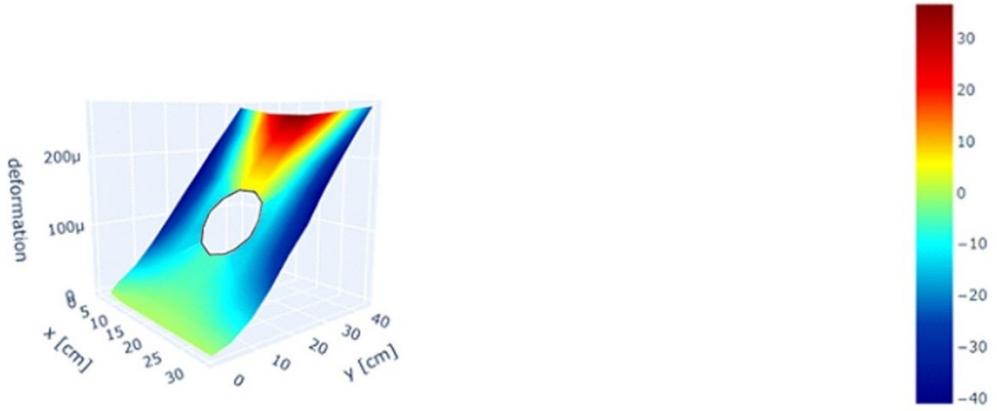



*Figure 6.75 Time=2.0 s - Initial FEM numerical case - Colorbar indicates accelerations in m/s²*

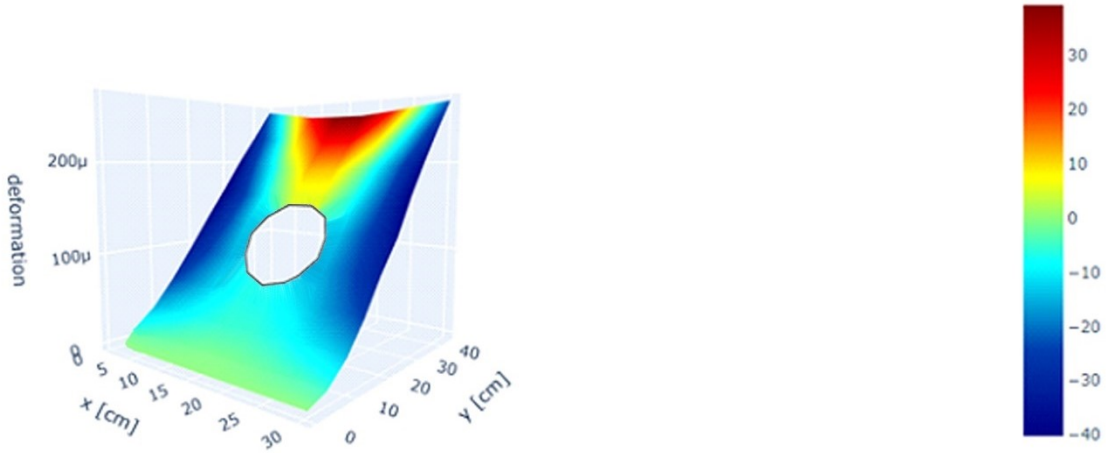



*Figure 6.76 Time=2.0 s - Predicted numerical case - Colorbar indicates accelerations in m/s²*

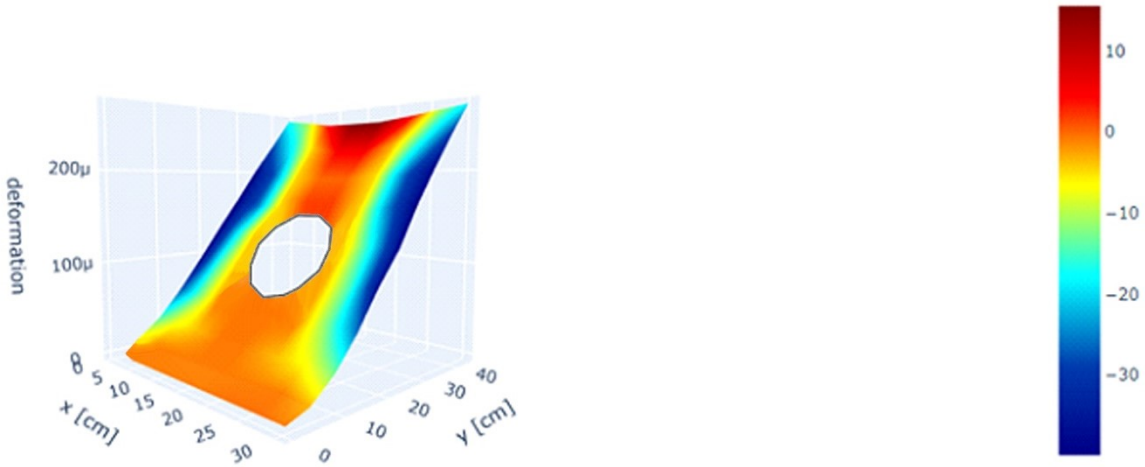



*Figure 6.77 Time=2.0 s – Predicted experimental case - Colorbar indicates accelerations in m/s²*

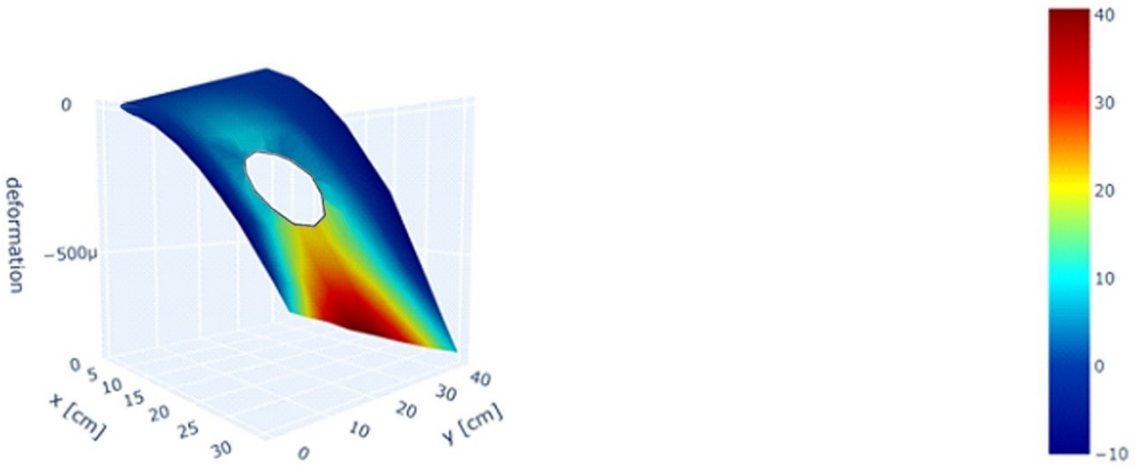



*Figure 6.78 Time=3.0 s - Initial FEM numerical case - Colorbar indicates accelerations in m/s²*

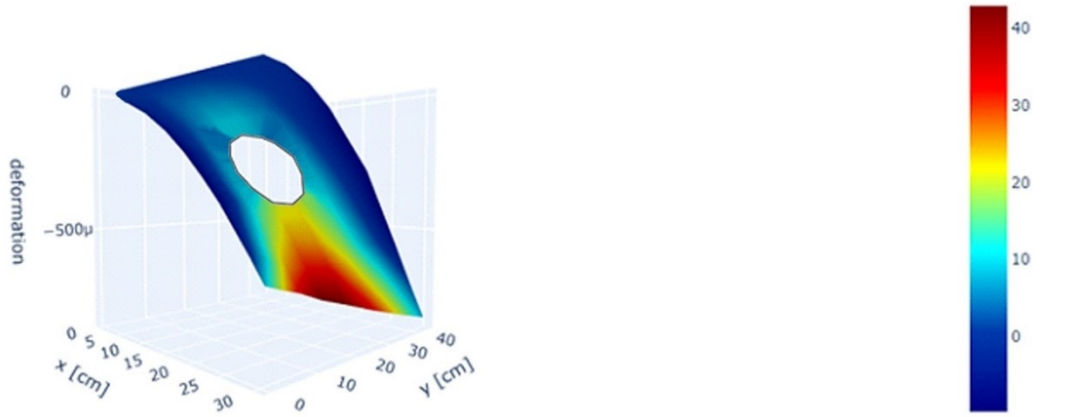



*Figure 6.79 Time=3.0 s - Predicted numerical case - Colorbar indicates accelerations in m/s²*

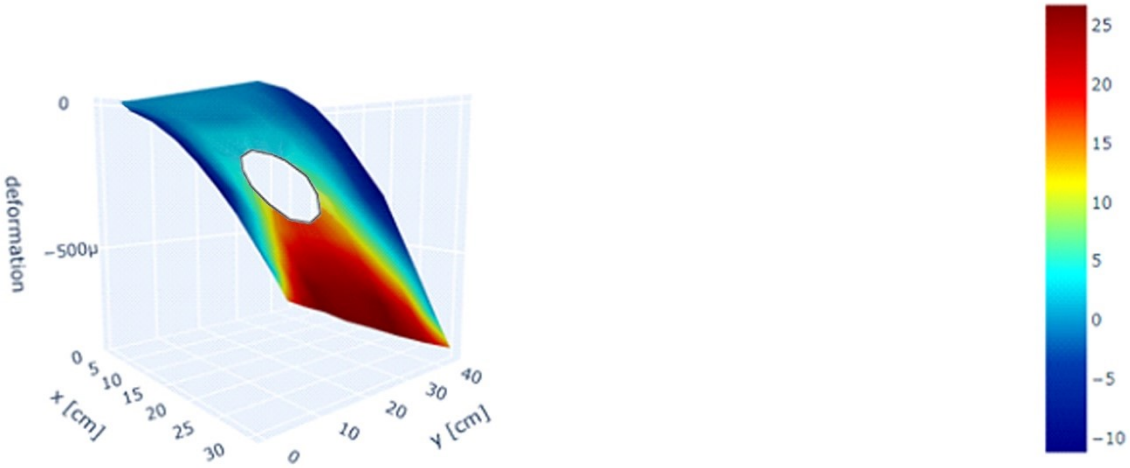



*Figure 6.80 Time=3.0 s – Predicted experimental case - Colorbar indicates accelerations in m/s²*

Like the two cases analyzed previously, the predictive analysis values of the dataset obtained with the numerical acceleration values are distributed along the bisector uniformly, producing data distributed in the same manner as the numerical case of the pseudorandom and harmonic signals. In contrast, concerning the dataset of the distribution of real and predicted values for the experimental case, they are consistently distributed along the bisector in a uniform and regular manner but with a less extensive cloud of points compared to the experimental case of the pseudorandom signal.

As mentioned in the previous cases, the same observations can be applied to the MAE and MSE curves, where, in the case of numerical values, there is an immediate decrease in error curves followed by convergence. However, unlike the previous cases, the experimental case exhibits a much smoother trend than the previous cases and reaches a convergence value after 50 epochs.

As expected from the high numerical R2 value, it is evident from the colors that the acceleration values of the Initial FEM and the predicted acceleration values are very similar, particularly this excellent prediction is observed at 1.0 and 3.0 seconds. At the same time, in the other two analyzed time instances, a slight difference is noted in the lower-left part of the plate, attributed to the absence of useful data for the dataset in that part of the plate. The same holds for the experimental case because the algorithm used to select positions in that area does not anticipate the placement of accelerometers that would have been useful for predicting results comparable to numerical values. Although there are not excellent results in the lower-left part, the remaining sections are adequately represented with a suitable prediction compared to the Initial FEM data, as evidenced by the excellent R2 value obtained, which is 0.87 in the experimental case.

## 6.5 Results

In this section, we will delve into a detailed analysis of the previous comparisons and the results obtained through the in-depth examination of the acceleration value predictions along the plate using the proposed machine learning algorithm. All six accelerometer placement configurations and the three signals used will be scrutinized.

The following Table 3 provides a summary of the values obtained from various analyses of the different acquired signals. The first column features the letters N, indicating numerical analyses, and E, indicating experimental analysis. The acronyms CF denote the configuration along with its

corresponding number, CH represents the chirp signal, AR signifies the harmonic signal, and PS denotes the pseudorandom signal.

| | Evaluating Model's Performance on training data | | | | Evaluating Model's Performance on testing data | | | | Evaluating Model's Performance | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | MAE | MSE | MRSE | R2 | MAE | MSE | MRSE2 | R2 | MAE | MSE | MRSE2 | R2 |
| CF1CHN | 1.0453 | 3.5345 | 1.8800 | 0.9868 | 1.0954 | 3.9471 | 1.9867 | 0.9852 | 1.0546 | 3.6144 | 1.9011 | 0.9865 |
| CF2CHN | 0.6842 | 1.7555 | 1.3249 | 0.9934 | 0.7027 | 1.8966 | 1.3772 | 0.9929 | 0.6866 | 1.7811 | 1.3345 | 0.9933 |
| CF3CHN | 1.8901 | 17.4618 | 4.1787 | 0.9351 | 1.9802 | 20.1978 | 4.4942 | 0.9247 | 1.9077 | 18.008 | 4.2435 | 0.9330 |
| CF4CHN | 0.8824 | 3.4094 | 1.8464 | 0.9873 | 0.9345 | 3.9758 | 1.9939 | 0.9851 | 0.8920 | 3.5206 | 1.8763 | 0.9869 |
| CF5CHN | 0.7041 | 1.9075 | 1.3811 | 0.9929 | 0.7275 | 2.0441 | 1.4297 | 0.9923 | 0.7074 | 1.9322 | 1.3900 | 0.9928 |
| CF6CHN | 1.2458 | 5.8798 | 2.4248 | 0.9781 | 1.2994 | 6.4458 | 2.5388 | 0.9759 | 1.2554 | 5.9892 | 2.4472 | 0.9777 |
| CF1CHE | 3.7506 | 34.3697 | 5.8625 | 0.8724 | 4.0937 | 42.7239 | 6.5363 | 0.8407 | 3.8185 | 36.0204 | 6.0017 | 0.8661 |
| CF2CHE | 3.9296 | 38.7332 | 6.2236 | 0.8562 | 4.2455 | 47.6329 | 6.9016 | 0.8224 | 3.9920 | 40.4955 | 6.3636 | 0.8495 |
| CF3CHE | 4.2498 | 45.1744 | 6.7211 | 0.8323 | 4.6104 | 55.6710 | 7.4613 | 0.7925 | 4.3205 | 47.2368 | 6.8729 | 0.8245 |
| CF4CHE | 3.7706 | 35.95716 | 5.9964 | 0.8665 | 4.0789 | 43.8463 | 6.6216 | 0.8365 | 3.8315 | 37.5173 | 6.1251 | 0.8606 |
| CF5CHE | 3.6637 | 33.3231 | 5.7726 | 0.8762 | 4.1111 | 44.6536 | 6.6823 | 0.8335 | 3.7521 | 35.5675 | 5.9638 | 0.8678 |
| CF6CHE | 2.8434 | 18.5914 | 4.3117 | 0.9309 | 3.2112 | 24.5133 | 4.9510 | 0.9086 | 2.9156 | 19.7576 | 4.4449 | 0.9265 |
| CF1ARN | 0.0215 | 0.0012 | 0.0349 | 0.9763 | 0.0223 | 0.0013 | 0.0365 | 0.9740 | 0.0216 | 0.0012 | 0.0352 | 0.9759 |
| CF2ARN | 0.0148 | 0.0005 | 0.0241 | 0.9886 | 0.0153 | 0.0006 | 0.0248 | 0.9879 | 0.0149 | 0.0005 | 0.0243 | 0.9885 |
| CF3ARN | 0.0219 | 0.0016 | 0.0411 | 0.9180 | 0.0273 | 0.0029 | 0.0542 | 0.8557 | 0.0230 | 0.0019 | 0.0440 | 0.9057 |
| CF4ARN | 0.0451 | 0.0010 | 0.0316 | 0.9725 | 0.0205 | 0.0011 | 0.0341 | 0.9678 | 0.0194 | 0.0010 | 0.0322 | 0.9678 |
| CF5ARN | 0.0152 | 0.0005 | 0.0232 | 0.9802 | 0.0157 | 0.0005 | 0.0240 | 0.9788 | 0.0153 | 0.0005 | 0.0234 | 0.9799 |
| CF6ARN | 0.0280 | 0.0020 | 0.0447 | 0.9613 | 0.0291 | 0.0021 | 0.0465 | 0.9577 | 0.0282 | 0.0020 | 0.0451 | 0.0451 |
| CF1ARE | 0.0382 | 0.0032 | 0.0572 | 0.9367 | 0.0455 | 0.0048 | 0.0697 | 0.9054 | 0.0396 | 0.0035 | 0.0597 | 0.9308 |
| CF2ARE | 0.0194 | 0.0008 | 0.0292 | 0.9835 | 0.0273 | 0.0019 | 0.0438 | 0.9625 | 0.0208 | 0.0010 | 0.0324 | 0.9796 |
| CF3ARE | 0.0289 | 0.0017 | 0.0418 | 0.9150 | 0.0345 | 0.0026 | 0.0517 | 0.8690 | 0.0300 | 0.0019 | 0.0440 | 0.9061 |
| CF4ARE | 0.0600 | 0.0077 | 0.0879 | 0.7886 | 0.0632 | 0.0086 | 0.0932 | 0.7610 | 0.0606 | 0.0079 | 0.0890 | 0.7832 |
| CF5ARE | 0.0720 | 0.0109 | 0.1048 | 0.5993 | 0.0763 | 0.0123 | 0.1113 | 0.5459 | 0.0728 | 0.0112 | 0.1061 | 0.5888 |
| CF6ARE | 0.0343 | 0.0026 | 0.0515 | 0.9487 | 0.0405 | 0.0038 | 0.0618 | 0.9255 | 0.0354 | 0.0028 | 0.0535 | 0.9444 |
| CF1PSN | 1.9048 | 9.1531 | 3.0254 | 0.9826 | 1.9357 | 9.4159 | 3.0685 | 0.9820 | 1.9101 | 9.2011 | 3.0333 | 0.9825 |
| CF2PSN | 1.2332 | 3.2922 | 1.8144 | 0.9937 | 1.2619 | 3.4438 | 1.8557 | 0.9934 | 1.2379 | 3.3185 | 1.8216 | 0.9937 |
| CF3PSN | 4.4476 | 73.9834 | 8.6013 | 0.8598 | 4.5126 | 75.9546 | 8.7151 | 0.8554 | 4.4603 | 74.3714 | 8.6238 | 0.8589 |
| CF4PSN | 2.2198 | 14.1644 | 3.7635 | 0.9731 | 2.2453 | 14.5287 | 3.8116 | 0.9723 | 2.2241 | 14.2325 | 3.7726 | 0.9730 |
| CF5PSN | 1.1832 | 2.9791 | 1.7260 | 0.9943 | 1.2034 | 3.0779 | 1.7544 | 0.9941 | 1.1863 | 2.9957 | 1.7308 | 0.9943 |
| CF6PSN | 2.7419 | 18.7649 | 4.3318 | 0.9644 | 2.8153 | 19.8130 | 4.4511 | 0.9622 | 2.7557 | 18.9679 | 4.3552 | 0.9640 |
| CF1PSE | 12.7961 | 338.5020 | 18.3984 | 0.3585 | 12.9217 | 344.8375 | 18.5698 | 0.3437 | 12.8211 | 339.7614 | 18.4326 | 0.3556 |
| CF2PSE | 11.8390 | 296.6713 | 17.2241 | 0.4378 | 12.0690 | 307.3540 | 17.5315 | 0.4151 | 11.8849 | 298.8016 | 17.2858 | 0.4333 |
| CF3PSE | 9.9447 | 227.7232 | 15.0905 | 0.5684 | 10.7156 | 265.0752 | 16.2811 | 0.4955 | 10.0976 | 235.1329 | 15.3340 | 0.5540 |
| CF4PSE | 9.7647 | 216.7185 | 14.7213 | 0.5893 | 10.5411 | 250.3192 | 15.8214 | 0.5236 | 9.9193 | 223.40518 | 14.9467 | 0.5763 |
| CF5PSE | 11.1310 | 261.6332 | 16.1750 | 0.5042 | 11.4183 | 275.5312 | 16.5991 | 0.4756 | 11.1883 | 264.4048 | 16.2605 | 0.4985 |
| CF6PSE | 10.5734 | 223.1238 | 14.9373 | 0.5772 | 10.6936 | 228.3058 | 15.1097 | 0.5655 | 10.5973 | 224.1549 | 14.9718 | 0.5748 |

*Table 3*

Unlike Table 2 used for summarizing simulations in the beam case, here, the time column has been omitted. This decision stems from the fact that the simulations were consistently performed on the Google Colab server, and, as explained in the previous chapter on the beam, the server did not consistently provide the same computation time for simulations with identical data. Nevertheless, given the numerous simulations conducted in this case, it can be stated that the average time for each epoch was around 40-50 seconds. However, in this case, it is not possible to analyze simulation times based on various datasets.

Instead, a highly interesting column to analyze is that related to MAE and MSE. Here, it is notable that the smallest errors, almost zero, are in the numerical harmonic case and subsequently in the experimental harmonic case. Following this, slightly higher values were observed in the numerical chirp case, the numerical pseudorandom case, and the experimental chirp case. Finally, much higher values were found in the case of pseudorandom signals obtained from experimental data.

While valuable insights can be gleaned from these two elements, the most crucial parameter to be analyzed is R2. As in the case of the beam, the R2 values in simulated numerical cases are very high, resulting mainly in overfitting phenomena, especially in the harmonic and chirp signals in almost all examined configurations. In the chirp case, the worst R2 is found in configuration 3, both in the experimental and numerical cases, while the best configurations for the numerical case are the second and the fifth, and for the experimental case, the first and the sixth. Although all configurations achieve excellent R2 values, some are so good that they reach the overfitting phenomenon explained earlier. The third configuration appears to be the worst even in the simulation performed with numerically obtained harmonic values, while surprisingly, in the experimental case, the lowest R2 values are obtained in the fifth configuration. Unlike other configurations where the fifth is one of the best or the fourth, which, along with the sixth, turns out to be the best for the pseudorandom signal in the experimental case. Finally, in the simulation using datasets obtained from the numerical pseudorandom signal, it is observed that, in this case too, the worst configuration is the third. In contrast to all other cases, which obtained good results, the worst configuration for the experimental data of the pseudorandom signal is the first, obtaining a very poor R2 value.

The result obtained, as in the case of the beam, shows excellent numerical outcomes across all three signals. However, from the perspective of machine learning simulations derived from experimental signal datasets, it is noticeable that in most configurations, the harmonic signal emerges as the best.

This is attributed to its outstanding R2 results, low MAE and MSE values, and the alignment of values and trends along the time history of various accelerometers between experimentally acquired accelerations and those obtained from numerical analysis using the FEM method. The chirp signal also achieves excellent forecasting results, significantly surpassing the beam case where the R2 value was lower. Finally, similar to the beam case, the pseudorandom signal yields the worst result.

A crucial observation from the results pertains to the best and worst accelerometer positioning configurations. The third configuration, where most accelerometers are in the lower part of the plate quite close to the constraint, emerges as the least effective. This positioning does not allow for an optimal and comprehensive mapping of plate accelerations with a hole, as there are not enough accelerometers on the upper part, resulting in suboptimal predictions. Conversely, the second configuration, obtained through an algorithm that distributes accelerometers across a significant portion of the plate except for the lower-left part, which has a lesser impact compared to the upper parts, yields excellent results. The fifth configuration consistently provides the best outcomes in most cases, where accelerometers are positioned regularly and distributed along the central and upper part of the plate, allowing for excellent analyses and predictions, thanks to the crucial contribution of predictions obtained from neural networks.

# 7 Conclusion

The conclusions of this study represent a critical synthesis of the analyses conducted by applying Machine Learning (ML) methods to datasets obtained from experimentally acquired values and numerical values obtained from finite element analysis (FEM) methods applied to 1D geometries, such as a beam, and 2D geometries, such as a plate with a hole, subjected to various loadings. The main objective of the thesis was to assess the effectiveness of machine learning models in predicting accelerations in structural contexts under dynamic loads, with particular attention to the application of finite element analysis (FEM) methods.

The results obtained indicated a significant convergence between the measured real accelerations and those predicted by ML models. The analysis of discrepancies was underscored by a substantial reduction in predictive errors, highlighting the models' ability to generalize and learn complex patterns in structural dynamics. The precision of the models was evaluated through performance metrics such as root mean square error (RMSE) and coefficient of determination ($R^2$), confirming the validity of the predictions obtained.

The robustness analysis of ML models highlighted a good generalization ability even in the presence of outlier data and variations in the parameters with which the data were acquired. As evident from the two experiments on the plate and beam, the experimental signal that achieves the best predictions is the harmonic one, even at different frequencies and amplitudes. This is due to its periodic and time-constant nature, facilitating much easier learning by the neural network. However, even a non-periodic signal like the chirp achieves excellent results in both cases, while the pseudorandom signal does not yield satisfactory results after the neural network learning.

The analyses also reveal the flexibility of the beam model to utilize only 70% of the necessary data for the dataset compared to other simulations where 80% of the data were used. Moreover, it demonstrates the possibility of achieving excellent predictions using only 4 accelerometers instead of 5, resulting in less data and faster dataset training for simple structures. For a more complex structure like the plate, however, all 7 accelerometers used are necessary, suggesting potential application to complex and diverse structural systems based on the number of available sensors.

The practical implications of the conclusions reached extend across a broad range of sectors, from civil engineering to aerospace structures, where accurate prediction of structural accelerations is crucial for design and safety assessment. The use of ML models could streamline the structural

analysis process, reducing reliance on computationally intensive FEM models and enabling a rapid assessment of performance in real-world scenarios.

Despite the successes achieved, it is important to highlight some limitations of this study. The limited availability of data and the complexity of structural dynamics may impact the generalization of the models. Future developments should focus on acquiring larger datasets and considering context-specific factors.

In conclusion, the application of machine learning methods in the analysis of structural accelerations has proven to be a promising perspective, opening new avenues for the design and assessment of complex systems. Integrating these approaches with traditional methodologies, such as finite elements, could represent the future of structural engineering, enhancing prediction accuracy and expediting evaluation times.

# 8 References


[1] E. Meethal, R., Kodakkal, A., Khalil, M., Ghantasala, A., Obst, B., Bletzinger K., Wüchner, R., *"Finite element method-enhanced neural network for forward and inverse problems",* 2023.

[2] Kam Liu, W., Li, S., S. Park, H., *"Eighty years of the finite element method: birth, evolution, and future",* 2022.

[3] Richmond, B. G., Wright, B. W., Grosse, I., Dechow, P. C., Ross, C. F. Spencer, M. A., Strait, D. S., *"Finite element analysis in functional morphology",* 2005.

[4] Sliseris, J., Gaile, L., Pakrastins L., *"Extended multiscale FEM for design of beams and frames with complex topology",* Riga Technical University, Latvia,2018.

[5] Liu, Y., Zhao, T., Ju, W., Shi, S., *"Materials discovery and design using machine learning"*, Shanghai University, Shanghai, 2017.

[6] Javier Naranjo-Perez, J., Infantes, M., Jimenez-Alonso, J. F., Saez, S., *"A collaborative machine learning-optimization algorithm to improve the finite element model updating of civil engineering structures",* Seville, 2020.

[7] Le Clainche, S., Ferrer, E., Gibson, S., Cross, E., Parente, A., Vinuesa, R., *"Improving aircraft performance using machine learning: a review",* Madrid, 2022

[8] Wang, Y., Soutis, C., Ando, D., Sutou, Y., Narita, F., *"Application of deep neural network learning in composites design",* 2022.

[9] Zaparoli Cunhaa, B., Drozc, C., Zined, A., Foulard, S., Ichchoua, M., *"A review of machine learning methods applied to structural dynamics and vibroacoustic",* Lyon, 2023

[10] Zhiqiang, T., Shuai T., Jiqiao Z., Gongfa C., and Fangsen C. "*Structural Damage Detection Based on Real-Time Vibration Signal and Convolutional Neural Network",* Guangzhou, 2020.

[11] Döhler, M.; Hille, F.; Mevel, L.; Rücker, W. Structural health monitoring with statistical methods during progressive damage test of S101 Bridge. Eng. Struct. 2014, 69, 183–193.

[12] Wilmes, L., Olympio, R., M. de Payrebrune, K. and Schatz M." Structural *Vibration Tests: Use of Artificial Neural Networks for Live Prediction of Structural Stress"*, Kaiserslautern, Germany, 2020.

[13] Pal, J., Sikdar, S., Banerjee, S., Banerji P., "*A Combined Machine Learning and Model Updating Method for Autonomous Monitoring of Bolted Connections in Steel Frame Structures Using Vibration Data*", Hamirpur, India, 2022.

[14] Gao, Y., Mosalam, K.M., Chen, Y., Wang, W., Chen, Y. *"Auto-Regressive Integrated Moving-Average Machine Learning for Damage Identification of Steel Frames".* Appl. Sci. 2021, 11, 6084

[15] Wanga, Q., Wub, D., Tin-Loia, F., Gaoa, W., *"Machine learning aided stochastic structural free vibration analysis for functionally graded bar-type structures",* Sydney.

[16] López, O., López, A., Crossa, J., *"Support Vector Machines and Support Vector Regression",* 2022.

[17] Awad, M., Khanna, R., *"Support Vector Regression",* 2015

[18] Amari, S., Wu, S., *"Improving support vector machine classifiers by modifying kernel functions",* Hirosawa, Saitama, Japan, 1999.

[19] Cofre-Martel, S., Kobrich, P., Lopez Droguett, E., Meruane V., *"Deep Convolutional Neural Network-Based Structural Damage Localization and Quantification Using Transmissibility Data",* Santiago, Chile, 2019.

[20] Seventekidis, P., Giagopoulos, D., Arailopoulos, A., Markogiannaki, O., *"Structural Health Monitoring using deep learning with optimal finite element model generated data"* Greece, 2020.

[21] Kohara, C., Greveb, L., K. Ellerb, T., Connollya, D. S., Inala, K., *"A machine learning framework for accelerating the design process using CAE simulations: An application to finite element analysis in structural crashworthiness"*, Waterloo, Ontario, Canada, 2021.

[22] Chierichetti, M., Davoudi Kakhki, F., Huang, D., Vurturbadarinath, P., *"Surrogated finite element models using machine learning"*, San Jose State University, San Jose, CA, USA,2021.

[23] Vurturbadarinath, P., Chierichetti, M., Davoudi Kakhki, F., Huang, D., *"A Machine Learning Approach as a Surrogate for a Finite Element Analysis: Status of Research and Application to One Dimensional Systems"*, San Jose State University, San Jose, CA, USA, 2021.

[24] Chierichetti, M., Davoudi Kakhki, F., *"Optimal sensor location along a beam using machine learning"*, San Jose State University, San Jose, CA, USA, 2022.

[25] Basheer, I.A., Hajmeera, M., *"Artificial neural networks: fundamentals, computing, design, and application"*, Kansas State University, Manhattan, USA, 2001.

[26] Gershenson, C., *"Artificial Neural Networks for Beginners Article",* Universidad Nacional Autónoma de México, 2003.

[27] Murphy, K., *"Probabilistic Machine Learning: An Introduction. Probabilistic Machine Learning: An Introduction.",* MIT Press. Retrieved 10 April 2021.

[28] Doebelin, E., *"Measurement Systems: Application and Design",* McGraw-Hill Higher Education, 2003.

[29] Vibration Control Strategies for Shaker Systems, https://www.hbkworld.com/en/knowledge/resource-center/articles/strategies-for-shaker-systems#!ref_www.bksv.com

[30] Chassande-Mottin, E., Flandrin, P., *"On the stationary phase approximation of chirp spectra, in Proceedings of the IEEE-SP International Symposium on Time-Frequency and Time-Scale Analysis"*, Pittsburgh, 1998, pp. 117-120.

[31] Adhikari, S., Phani, A. S., *"Rayleigh's Classical Damping Revisited",* United Kingdom

[32] Kelmar, T. K., *"Machine Learning Based Sensor Selection for Modal Testing",* San Jose, US, 2023.