Prediction of Stress Using Machine Learning for Aerospace Applications

a project presented to
The Faculty of the Department of Aerospace Engineering
San José State University

in partial fulfillment of the requirements for the degree
Master of Science in Aerospace Engineering

by

Nataliya Grigoryan

December 2022

approved by

Dr. Maria Chierichetti
Faculty Advisor

ABSTRACT

Prediction of Stress Using Machine Learning for Aerospace Applications

Nataliya Grigoryan

As machine learning is becoming omnipresent, applications in aerospace industry have the potential to aid standard aircraft maintenance and reduce operating costs. Data driven approaches allow for further additional utilization of past and present data within the industry. If properly implemented, machine learning has the ability to significantly decrease computational time and power when compared to traditional testing and FEA. Given input and output data from equations of motion or Finite Element Analysis in Ansys, a supervised regression learning approach is applied to several engineering systems. Systems included in the project are a spring mass damper system, a static channel beam, and static wing with internal geometry and airfoil. Linear regression, decision tree, random forest, and neural network algorithms are used for the machine learning in MATLAB. Highest performing models among all cases explored are trained with the random forest algorithm, with trained R^2 values being larger than 0.99 for all cases. Decision tree models have slightly lower R^2 values compared to random forest models but show milder overfitting. Linear regression was not ideal for any system as the models do not have consistent performance among cases and tends to overfit or underfit data. Neural network models show great potential but requires further study and fine tuning due to lack of consistency in performance.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to Dr. Maria Chierichetti for introducing me to machine learning and her guidance throughout this project.

I am also grateful to my family and friends for their support.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

1. Introduction

1.1 Motivation

The aviation industry has been striving to reduce operating costs as the demand for air travel continues to rise. Maintenance of aircraft is a crucial aspect of airworthiness and is generally a very conservative process. Globally in 2018, maintenance, repair, and overhaul accounted for about 10% of total annual operational costs of airlines [1]. Structural health monitoring (SHM) has gained popularity as airline operators are actively driving its integration in traditional maintenance procedures. Programs have introduced adaptations of SMH by utilizing comparative vacuum monitoring sensors, which present challenges during preparation and installation [2]. Finite element modeling has also been widely used for analyzing complex systems, though it requires high computational power and time. Studies have demonstrated the ability for machine learning techniques to accurately surrogate traditional Finite Element Analysis (FEA), while also drastically reducing computational time [3]. If in-flight data, along with finite element models, can provide sufficient data to train machine learning algorithms further FEA would not be necessary. Traditionally, during certification, extensive testing is conducted on wing loads to certify the design and airworthiness. Machine learning algorithms could provide much more data and insight on the structural behavior of the wing and transform maintenance procedures and timelines.

1.2 Literature Review

Engineering reliance has begun to shift from academic knowledge to artificial intelligence in the last several decades. Intelligent systems assist, and at times replace, human capabilities including, but not limited to, learning, optimization, recognition, and classification [4]. As part of artificial intelligence, machine learning is a study where computer algorithms are trained using data without relying on human mediation. Machine learning assists in data processing and allows for valuable analysis; familiar examples include search engines, spam classification, and self-driving cars. Design optimization of turbine discs utilizing surrogate models demonstrated a decrease in computational time and cost due to the decreased number of FEA required [5]. The optimal design illustrated the feasibility and validity of the method for shape optimization problems [5].Studies have demonstrated that trained algorithms can operate without compromising accuracy [6][7]. Learning algorithms with co-training and self-training approaches for planetary exploration rovers with terrain classification showed an error reduction of close to 8% when compared to a supervised approach [8]. Collaborations with aerospace manufacturers have presented approaches of using machine learning to assist engineers during the design process that could lead to savings in time and resources [9].

Machine learning has many categories and types of algorithms. Supervised learning is a data driven regression approach that involves training an algorithm with collected data to predict an output. Data driven approaches for analyzing higher-order beams have shown to reflect results found through previous analytical models while not requiring specific assumptions [10]. This type of approach takes advantage of previously collected data and allows engineers to significantly reduce time cost when bypassing traditional analysis methods. The amount of data necessary for a data driven approach directly correlates to the accuracy of the results obtained [10]. The necessity, along with the high computational cost, of FEA has driven the development of approaches that combine FEA with machine learning. This combination approach used for

measuring delamination damage of composite materials “allows rapid non-destructive analysis for the iterative design of composites, accelerating the development of novel delamination-resistant materials” [11]. More complex machine learning algorithms, such as deep learning or neural networks, can more accurately fit data than other regression models. Deep learning models have shown to significantly reduce computational time, obtaining stress distribution in one second compared to thirty minutes with traditional FEA [3]. The study developed a machine learning surrogate for FEA capable of estimating stress distributions with an error of less than 0.5% when compared to FEA results [3]. Finite element mesh size reduction is also possible through neural network approaches, specifically for analyzing stress concentration where errors were found to be about 0.0012% [12]. Though these algorithms are powerful and accurate, there are cases where the model lacks generalizability. Efforts to overcome this obstacle were shown in a study that presented a physics-based approach for structural health monitoring, “which involves the integration of domain knowledge into the learning process” [13]. Machine learning has already begun to be integrated into aircraft fatigue stress predictions as focus is now shifting towards automated data processing and analysis for large amounts of complicated data [14]. As it is possible to recreate stress spectrums flight by flight with machine learning techniques, other applications include detecting corrosion and damage through image processing techniques [14].

1.3 Proposal

The objective of this project is to design a program that utilizes machine learning algorithms along with finite element analysis for the prediction of stress in aerospace systems. A supervised regression learning approach, with known input and output data, will map input variables to some continuous function. Data will be collected by modeling structural dynamics with discretized equations of motion as well as FEA of a beam and wing geometry. Several types of algorithms will be tested and analyzed to determine which is able to accurately predict behavior of systems. With appropriate training, these algorithms will provide insightful analysis at a lower computational complexity.

1.4 Methodology

A three degree of freedom system is used for initial modeling. Displacement, velocity, and acceleration in the system are solved for and used to train the machine learning algorithm. A simple linear regression model is used as a starting point to predict several outputs given various combinations of inputs. The algorithm is trained with 80% of the data taken from the initial system to better assess the prediction. The accuracy is observed by the R2 value when comparing the model prediction to the given output. The R^2 value will be compared for all combinations of inputs and outputs to spot the weak points of the machine learning. It is expected that a linear regression model will not be the most accurate due to the complexity of the system, therefore other algorithms with nonlinear terms and greater complexity will be explored. Further analysis will be done on other systems such as a cantilever beam system and a wing under various loads. A similar approach will be taken by starting with a simple linear regression model followed by more complex algorithms such as decision tree, random forest, and neural network.

2. Machine Learning Concepts

2.1 Introduction to Machine Learning

Machine learning is a relatively new field of study of algorithms programmed to automatically learn and improve with experience. The development of this field is strongly tied to artificial intelligence and information theory, as well as statistics, control theory, neurobiology, etc. Algorithms have already been incorporated into everyday life and have the potential to greatly benefit various applications. With the ability to recognizes relationships in large databases, machine learning can provide valuable information in fields such as engineering or medical for example. Its ability to perform without human interference aids in areas where human knowledge does not reach. This also aids in dynamic problems with varying inputs and outputs. Though this form of artificial intelligence does not yet compare to human intelligence, it is still able to efficiently learn and perform certain tasks including predictions, classifications, image processing, regression, etc.

Machine learning problems can typically be categorized as either unsupervised or supervised [15]. Unsupervised learning only uses predictor data without a given response to predict future responses, whereas supervised learning is given a set of response data. The variables used to train supervised algorithms allow problems to also be classified as either regression or classification. Regression problems use quantitative variables that hold numerical value and create a relationship between variables by mapping predictor variables to some continuous function. Classification on the other hand follows a more qualitative approach where the predictor variables are mapped into discrete responses. For example, image processing is considered an unsupervised classification problem as there is no response data for the algorithms to use when learning. Image processing may also be supervised if there is human assistance. When beginning to design an approach, it is crucial to assess the type of problem at hand and properly categorize it. The focus of this paper is on the design of an algorithm for the prediction of stress, therefore the main approach to be discussed is supervised regression.

Data sets, including given input and output values, are used to train models. Input values are independent variables, that are also referred to as the predictors, whereas output values are dependent. The output can also be considered as the response variable, as it is compared to the function the model produces. To assess performance of an algorithm, data sets are split into two subsets: training data and test data. Training data is used by the algorithm to learn and create a fit. Once a fit is created, it is evaluated using the test subset.

Linear regression models solve for an interpretable mathematical relationship, allowing humans to understand the relationship. Decision trees and neural networks are considered black box models since the relationship is created directly by the algorithm and does not clearly present the prediction function. Black box models are used for complex problems with incomprehensible relationships between variables. For an approximation of a mathematical relationship, there are parameters associated with variables in model. Different parameters result

in different function approximations, each of which will have different errors based on the fit. Models solve a minimization model to find the parameters that result in the best fit. An iterative approach minimizes the square difference of the predicted values and the training data. This is termed as cost function, also referred to as mean squared error, and will be discussed in more detail in a further section.

2.3 Brief History of Machine Learning

Many previously developed concepts have built the foundation of machine learning. The earliest form of linear regression can be dated to the publication by Adrien-Marie Legendre in 1805 on the method of least squares. The technique introduced a procedure for using data to fit linear equations and was quickly recognized by mathematicians and physicists, such as Carl Friedrich Gauss who used the method of least squares to calculate orbits of celestial bodies. An additional approach was proposed for qualitative predictions in 1936 by Fisher, who proposed linear discriminant analysis which is used in supervised classification problems. In the late 1980's, spoken word recognition and autonomous vehicles were utilizing machine learning [16,17]. Not soon after, further development of human and animal learning models advanced algorithms and allowed for human performance of backgammon at the world champions [18,19].

The first definition of machine learning was given by Arthur Samuel in 1959 as the following: "field of study that gives computers the ability to learn without being explicitly programmed" [20]. Throughout the evolution of this field, many other definitions were introduced and there is still debate on how to fitly define machine learning. A more recent definition was given in Tom Mitchells Machine Learning textbook: "a computer program is said to learn from experience E with respect to some task T and performance measure P, if its performance at tasks in T, as measured by P, improves experience E" [21].

2.4 Assessing Quality of Algorithms

Machine learning encompasses many different algorithms that vary in accuracy depending on the problem type, therefore assessing the quality of these algorithms is crucial. There are several measures used to assess the performance and accuracy of algorithms such as mean squared error, root mean squared error, and R^2 (R squared). Mean squared error (MSE) evaluates how close the prediction is when compared to the given training data. In mathematical terms MSE is the mean of the square of the difference between the data (y_i) and the prediction ($f(x_i)$).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - f(x_i)\right)^2 \tag{2.1}$$

This measures the average of square of the residuals (distance from the data to the predicted function). Lower MSE values correlate with higher accuracy in the algorithm. Root mean square error (RMSE) is the root of MSE and returns values in the same unit as the data and predictor values. Though there is no clear correlation between machine learning methods and

Commented [NG1]: Waibel, Lee 1989- programs learn to recognize spoken words

Commented [NG2]: Pomerleau 1989- detect fraudulent use of credit cards, drive autonomous vehicles on public highways

Commented [NG3]: Models of human and animal learning, develop relationship to learning algorithms(Laird 1986, Anderson 1991, Qin 1992, Chi & Bassock 1989, Ahn & Brewer 1993)

Commented [NG4]: Tesauro 1992 – games such as backgammon at performance of human world champions

MSE, evaluating MSE for test data may demonstrate which method will result in a smaller MSE [15].

Another common measure of error is R^2, which is the percentage of variance between predicted data and independent variables. It is calculated with the residual sum of squares (RSS), variance of training and response data, and the total sum of squares (TSS), variance within the training data.

$$R^2 = 1 - \frac{RSS}{TSS} \tag{2.2}$$

Values of R^2 range between zero and one, where greater values typically indicate a better fit. Although providing insight on the how well the model fits training data, this alone does not give the necessary grounds for determining the quality of the model. No model can achieve a R^2 value of one, as this would require the model to accurately predict all variance within the training data.

Plotting the training data with the model prediction provides a visual representation of the model's performance. In an ideal case, where the model predicts all variance, the training data and model prediction will be equal. This would be represented in the plots as a positive linear trend with a slope of one.

2.5 Linear Regression

One of the most common and simple algorithms in machine learning is linear regression. This approach approximates a relationship between the predictor variables in vector $\boldsymbol{x}$ and the quantitative response vector $\boldsymbol{y}$ as a linear relationship. Most applications of linear regression are for interpolation and the prediction of future responses. The estimated linear relationship also allows to find responses for data sets not within the training set. In cases where more than one predictor is estimated, the simple linear regression model is expanded to the multiple linear regression model. Given *p* number of predictor variables, the multiple linear regression approach estimates beta coefficients so that [15]:

$$\boldsymbol{y} = \beta_0 + \beta_1 \boldsymbol{x_1} + \beta_2 \boldsymbol{x_2} + \cdots \beta_P \boldsymbol{x_p} + \epsilon \tag{2.3}$$

Unknown parameters in Eq. (2.3) include slope coefficients (β_P) for each predictor, as well as intercept β_0 and an unpredictable error function ϵ. This approach provides an equation that allows for a quantitative and analytical understanding of the relationship between variables.

Training data produces estimates of β_p, that model the slope coefficients, and $\boldsymbol{y}$, prediction of $\boldsymbol{y}$, so that [15]:

$$\boldsymbol{y} = \beta_0 + \beta_1 \boldsymbol{x_1} + \beta_2 \boldsymbol{x_2} + \cdots + \beta_p \boldsymbol{x_P} \tag{2.4}$$

Given that the model is not able to predict ϵ, the results will inherently include some error if the predictor function fits the data to the best of its ability. Coefficients are estimated using *n* observation sets of *x* and *y* measurements as training data [15]:

$$(x_{11}, x_{12}, \dots x_{1p}, y_1), (x_{21}, x_{22}, \dots x_{2p}, y_2), \dots, (x_{n1}, x_{n2}, \dots x_{np}, y_n)$$

For a given prediction of $\boldsymbol{y}$ based on an *i*th set of observations, the residual is calculated using Eq. (2.5) [15]:

$$e_i = y_i - \boldsymbol{y}_i \tag{2.5}$$

The residual sum of squares (RSS), Eq. (2.6) expanded into Eq. (2.7), is minimized using the least squares approach.

$$RSS = e_1^2 + e_2^2 + \cdots + e_n^2 \tag{2.6}$$

$$RSS = \sum_{i=1}^{n} (y_i - \boldsymbol{y}_i)^2 = \sum_{i=1}^{n} \left(y_i - \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{iP}\right)^2 \tag{2.7}$$

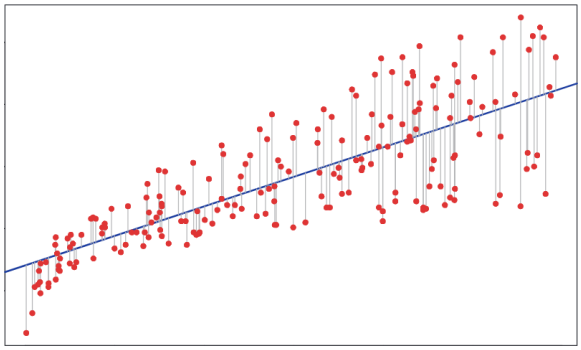

Figure 2.1 – Linear fit for a given data set minimized by RSS [15].

In Fig. 2.1, the blue line represents the linear fit determined for the observations represented as red dots. This allows to predict response values to specific points that are not included in the given data, inside or outside the domain. The error for each observation, represented as a grey line, is minimized through the residual sum of squares. The fit is influenced by the amount of given data, as well as the distribution of the data.

The complexity of multivariable regression is most easily represented through matrix algebra. The β estimates are optimized by taking the first derivative of Eq. (2.7) with respect to each estimated β variable and set to zero. This results in a system of equations that can be represented in matrices.

For example, given three sets of observations (x_{11}, y_1), (x_{21}, y_2), and (x_{31}, y_3) the prediction will be:

$$\boldsymbol{y} = \beta_0 + \beta_1 \boldsymbol{x_1} \tag{2.8}$$

The residual sum of squares for this example:

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = (y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2 + (y_3 - \hat{y}_3)^2 \quad (2.9)$$

To optimize the estimate β_0, the following derivative is taken:

$$\frac{d(RSS)}{d(\beta_0)} = 2(y_1 - \hat{y}_1)\frac{d(-\hat{y}_1)}{d(\beta_0)} + 2(y_2 - \hat{y}_2)\frac{d(-\hat{y}_2)}{d(\beta_0)} + 2(y_3 - \hat{y}_3)\frac{d(-\hat{y}_3)}{d(\beta_0)} = 0 \quad (2.10)$$

$$-2(y_1 - \beta_0 - \beta_1 x_{11}) - 2(y_2 - \beta_0 - \beta_1 x_{21}) - 2(y_3 - \beta_0 - \beta_1 x_{31}) = 0 \quad (2.11)$$

$$-y_1 + \beta_0 + \beta_1 x_{11} - y_2 + \beta_0 + \beta_1 x_{21} - y_3 + \beta_0 + \beta_1 x_{31} = 0 \quad (2.12)$$

$$3\beta_0 + \beta_1(x_{11} + x_{21} + x_{31}) = y_1 + y_2 + y_3 \quad (2.13)$$

Similarly estimate β_1 is optimized:

$$\frac{d(RSS)}{d(\beta_1)} = 2(y_1 - \hat{y}_1)\frac{d(-\hat{y}_1)}{d(\beta_1)} + 2(y_2 - \hat{y}_2)\frac{d(-\hat{y}_2)}{d(\beta_1)} + 2(y_3 - \hat{y}_3)\frac{d(-\hat{y}_3)}{d(\beta_1)} = 0 \quad (2.14)$$

$$\frac{d(RSS)}{d(\beta_1)} = -2(y_1 - \hat{y}_1)(x_{11}) - 2(y_2 - \hat{y}_2)(x_{21}) - 2(y_3 - \hat{y}_3)(x_{31}) = 0 \quad (2.15)$$

$$-(y_1 - \beta_0 - \beta_1 x_{11})(x_{11}) - (y_2 - \beta_0 - \beta_1 x_{21})(x_{21}) - (y_3 - \beta_0 - \beta_1 x_{31})(x_{31}) = 0 \quad (2.16)$$

$$x_{11}\beta_0 + \beta_1 x_{11}^2 + x_{21}\beta_0 + \beta_1 x_{21}^2 + x_{31}\beta_0 + \beta_1 x_{31}^2 = x_{11}y_1 + x_{21}y_2 + x_{31}y_3 \quad (2.17)$$

$$(x_{11} + x_{21} + x_{31})\beta_0 + (x_{11}^2 + x_{21}^2 + x_{31}^2)\beta_1 = x_{11}y_1 + x_{21}y_2 + x_{31}y_3 \quad (2.17)$$

The estimates β_0 and β_1 are simultaneously solved for through matrix algebra using Eqs. (2.13) and (2.17):

$$\begin{bmatrix} 3 & x_{11} + x_{21} + x_{31} \\ (x_{11} + x_{21} + x_{31} & x_{11}^2 + x_{21}^2 + x_{31}^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} y_1 + y_2 + y_3 \\ x_{11}y_1 + x_{21}y_2 + x_{31}y_3 \end{bmatrix} \quad (2.17)$$

From Eq. (2.17), the β_0 and β_1 coefficients are then substituted into Eq. (2.8) to determine the prediction $\hat{\boldsymbol{y}}$.

The simple implementation of this method and low complexity and interpretation of results is advantageous. This has led it to be one of the more common models with many available resources. However due to its simplicity, it is more than likely that this method will oversimplify problems. The assumption of a linear relationship between predictor and response variables may inaccurately model the complexity of most engineering applications. This model

also possesses sensitivity to outliers in data, required attention to the training data set to enhance the response prediction.

2.6 Decision Trees

Regression and classification are fundamental methods for decision trees. Compared to the previously discussed linear regression, decision trees are a much simpler non-parametric method for the approximation of discrete-valued functions [21]. The representation of a decision tree is intelligible and consists of a root node, branches, internal nodes, and terminal nodes [15]. A common example of a classification decision tree structure is the PlayTennis concept (figure 2.2), which determines whether conditions are suitable for playing tennis.

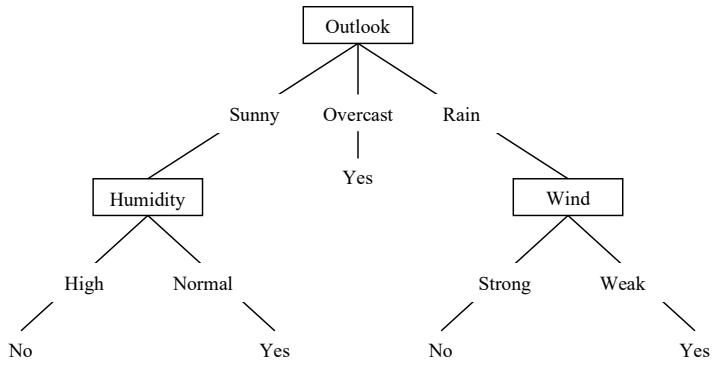

Figure 2.2 – PlayTennis decision tree representation [21].

In figure 2.2, the decision tree starts at the root node "Outlook" and branches out the terminal nodes, or leaves, "Yes" and "No". Internal node "Humidity" tests attribute values "High" and "Normal" to classify down to a terminal node. Following the branches, a sunny day with high humidity will not be suitable for playing tennis. Whereas an overcast or rainy day with weak winds would be suitable.

Regression trees follow a similar approach, but create splits based on predictor values. A simple example predicts miles per gallon of a car from displacement ($x1$), horsepower ($x2$), and weight ($x3$) predictors [22]. A simple decision tree trained with 3 splits in MATLAB using the "carsmall" sample data set is seen below [22].

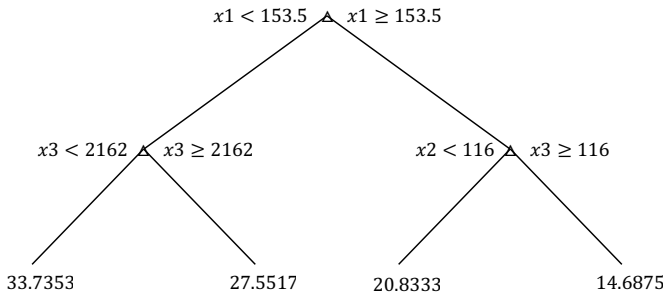

Figure 2.3 – Decision tree trained to predict miles per gallon of car.

In this example, the first split is taken at the root node based on the displacement of the car. The two resulting branches then determine the miles per gallon of the car based on weight and horsepower. This regression tree only produces four predictions which will likely not be a suitable model and require more branches.

Decision trees are created by dividing a predictor space and assigning predictions to observations that fall in specific regions. For example, a given set of values $X_1,\ X_2, \dots, X_p$ will be divided into J separate regions $R_1,\ R_2, \dots, R_p$ [15]. For the regression tree example above, the predictor values are used to create four regions which are the terminal nodes. Every value of x that falls into region R_j will have the same response. These regions are constructed as high dimensional rectangles, and are optimized for the lowest RSS by [15]:

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \bar{y}_{R_j})^2 \tag{2.18}$$

However, a recursive binary splitting approach is used instead, as minimizing the RSS is computationally infeasible [15]. This approach is considered a top-down, greedy approach that creates partitions from the root node and selects the best splits at each step [15]. Consider a splitting point s in regions R_1 and R_2 that minimize RSS. The split at s is for a predictor X_j denoted as [15]:

$$R_1(j,s) = \{X | X_j < s\}, R_2(j,s) = \{X | X_j \geq s\} \tag{2.19}$$

The following equation is minimizing to find values of j and s, where $\bar{y}_{R_j}$ is the mean response for training observations in $R_j(j,s)$ [15]:

$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \bar{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \bar{y}_{R_2})^2 \tag{2.20}$$

This process is repeated for every predictor variable each time a region is split until the terminal nodes are reached. The response for the terminal nodes is determined by the mean of training observations in each of the regions.

The accuracy of each attribute is determined by entropy (Eq. (2.21)), which measures the uncertainty of a collection of observations. Across K classes, the proportion of observations in the mth region from the kth class (p_{mk}) are used to measure the purity of the mth node [15]:

$$D = -\sum_{k=1}^{K} p_{mk} log p_{mk} \tag{2.21}$$

This is used during the decision tree process to determine the quality of a split. In an undesirable case where all observations belong to one class ($p_{mk} = 1$), the entropy will be 0 and not the training set would not be suitable for machine learning.

2.6.1 Random Forest

Improving the performance of the decision tree method is done by creating a random "forest" of multiple decision trees until reaching terminal nodes. Each tree in the forest will randomly build new data sets to use as training data, which is known as bootstrapping. The decreases the sensitivity to the data set, as every tree is trained with a different data set. The algorithm only considers subsets of predictors at each split, resulting in more reliable and consistent trees [15]. From a full set of p predictors, a random sample of m predictors is used as split candidates [15]. At each split, new samples of m predictors are taken where $m \approx \sqrt{p}$. Small values of m are beneficial for bigger data sets and tend to reduce errors. The results from each tree are averaged, which is termed aggregation. Bootstrapping along with aggregation is referred to as bagging and is a general process for reducing variance in the model. Another approach for improving predictions that can be applied to models is boosting. Unlike bagging, where trees are built parallel to each other, boosting builds trees sequentially [15]. This increases the complexity of the model, as well as the accuracy.

Consider the example from the regression tree, where the model is trained to predict miles per gallon of a car. A random forest model will consist of multiple regression trees built and trained with random sets of data and predictors. This increases diversification among the trees and decreases the dependency of the model to the data. An example of a random forest with three regression trees is seen in the figure below.

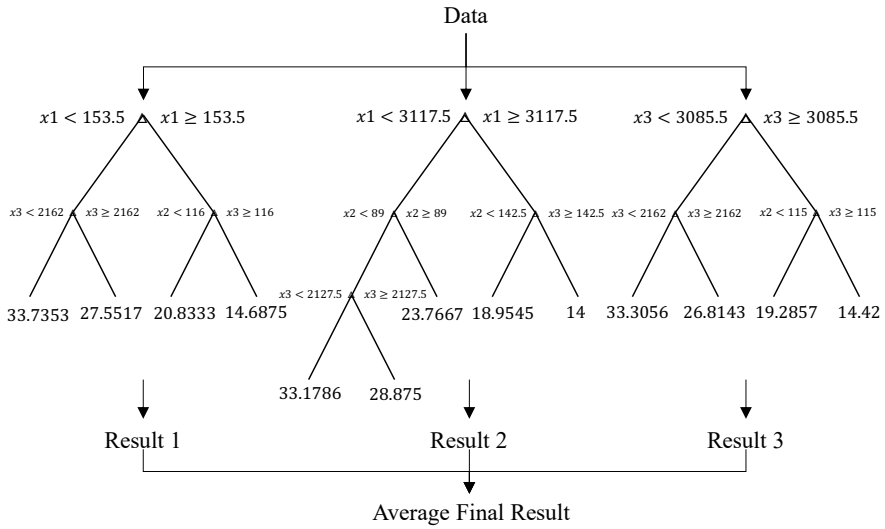

Figure 2.4 – Random Forest trained to predict miles per gallon of car.

Each individual tree in a random forest is built using the process discussed in the previous section. This method was designed to prevent overfitting therefore the number of trees used is determined on a trial-and-error basis [23]. Increasing the number of trees has shown to have no effect on the performance of the model [23].

Decision trees and random forest methods may be used for both classification and regression problems. Though random forests possess a higher computational time, they increase prediction accuracy. The bagging process in random forests also prevents overfitting for complex data sets. Preprocessing data is not necessary as null values and outliers have little effect on the performance of the method.

2.7 Neural Networks

As decision trees resemble trees, neural networks resemble the nature and behavior of the human brain. This method was influenced by the basic neuroscience finding that networks of neurons drive mental activity through electrochemical activity [24]. In machine learning, these neurons process inputs through an activation function and are connected by links to form a network.

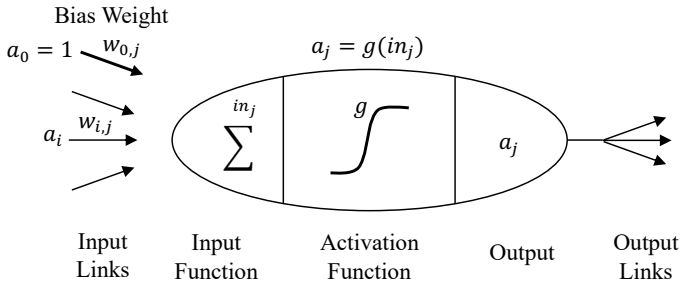

Figure 2.5 – Representation of the mathematical model for a neuron [24].

Consider neuron i linked to neuron j, and its respective activation a_i. The strength of each link is determined by its numeric weight $w_{i,j}$. The weight sum of the inputs for neuron j is defined as [24]:

$$in_j = \sum_{i=0}^{n} w_{i,j}\, a_i \tag{2.22}$$

The output is then obtained by the activation function [24]:

$$a_j = g(in_j) = g\left(\sum_{i=0}^{n} w_{i,j}\, a_i\right) \tag{2.23}$$

The activation function is typically either a hard threshold (binary step function) or a logistic function (sigmoid function), both of which allow for the representation of nonlinear functions [24]. Sigmoid functions typically work well with classifications problems.

Once the neurons are established, the connection between them is created. This network consists of multiple neurons within multiple layers. An input layer at the beginning contains as many neurons as there are predictors. The output layer typically contains one neuron, the number of response variables. Between the input and output layers are hidden layers that are defined by the user as well as the number of neurons in each hidden layer. The hidden layers are optimized by reducing the error of the model. A representation of a neural network trained to predict miles per gallon from displacement, horsepower, and weight predictors is seen below. The model has three hidden layers with varying numbers of neurons.

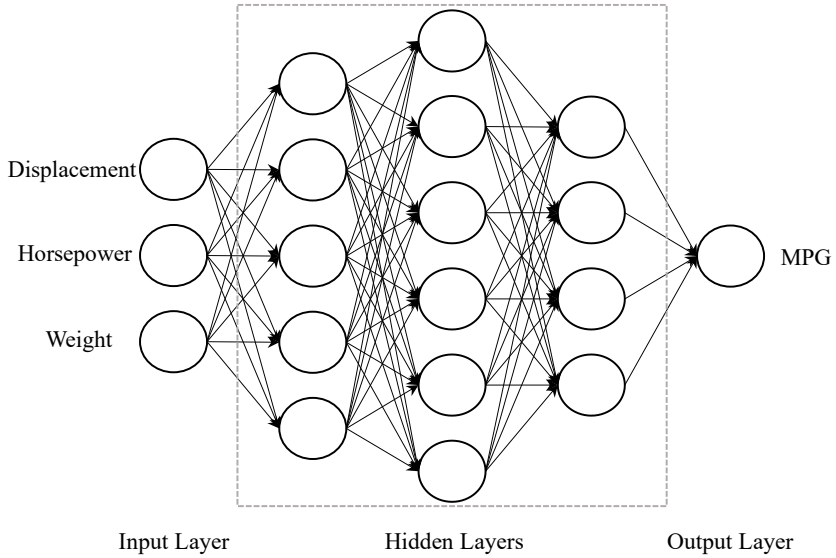

Figure 2.6 – Neural network trained to predict miles per gallon of a car.

A feed forward network connects neurons in one direction, where a neuron receives an input from previous neurons and delivers output to the following neurons [24]. In a recurrent network, outputs of neurons are fed back into its own inputs, which is more fitting for modeling the brain [24]. The number of times the data passed through a model is measured in epochs. This value is typically determined on a trial-and-error basis and is dependent on the dataset and model. It has been observed that as the number of epochs in a model increase, the test data set error will rapidly decrease then begin to increase after some point. This can be seen in Fig. 2.7 as the early-stopping point, which identifies the onset of overfitting [25]. The terms validation and test are usually interchangeable for data sets or samples that are used on a trained model for performance analysis. Low numbers of epochs correspond to an underfitted model, whereas epoch numbers after the early-stopping point correspond with overfitted models.

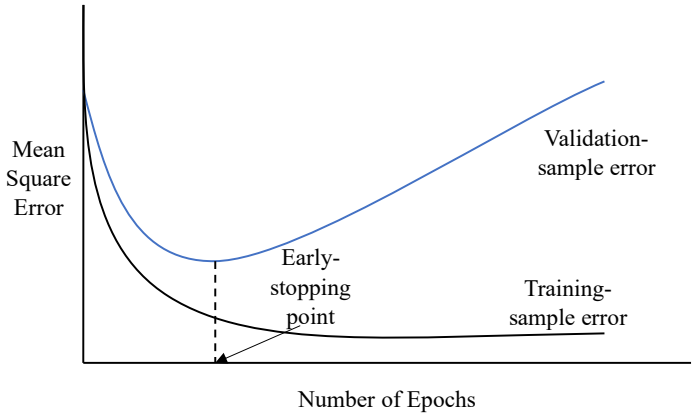

Figure 2.7 – Early-stopping point based on training and validation (test) sample errors [25].

3. Spring Mass Damper System

It is common to model various engineering problems with spring mass damper systems. A simple system will include just one spring (k), one mass (m), and one damper (c), resulting in a single degree of freedom system.

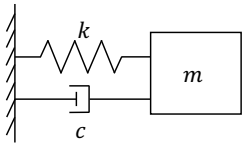

Figure 3.1 – Simple spring, damper, mass system.

However, many engineering problems possess multiple degree of freedoms and require a more complex systems for modeling. A multiple degree of freedom system is used for an initial exploration of machine learning algorithms. Equations of motion will obtain the displacement, velocity, and acceleration of each mass, which will serve as the training data. Linear regression, decision tree, random forest, and neural network algorithms will be trained to compare the capability of each algorithm. Each algorithm will be trained using displacement, velocity, or acceleration data of all masses for various cases to predict the displacement, velocity, or acceleration of a single mass. Various predictions will be analyzed to explore the implications of such a complex system and the accuracy of the algorithms.

3.1 Problem Definition

Consider a forced three degree of freedom system, as seen in Fig. 3.2 below.

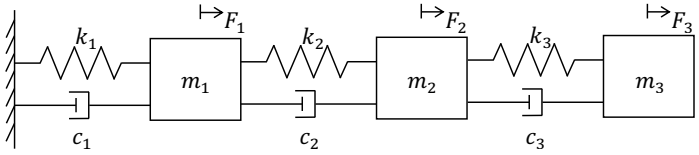

Figure 3.2 – A three degree of freedom spring mass damper system with forcing functions.

The system is characterized by:

$$m_1 = 5\ kg \qquad m = 1\ kg \qquad m_3 = 3\ kg$$

$$c_1 = 1\ {}^{Ns}/_{m} \qquad c_2 = 1\ {}^{Ns}/_{m} \qquad c_3 = 2\ {}^{Ns}/_{m}$$

$$k_1 = 1\ {}^{N}/_{m} \qquad k_2 = 1\ {}^{N}/_{m} \qquad k_3 = 2\ {}^{N}/_{m}$$

$$F_1 = 0.5 \sin(15t) \qquad F_2 = 0.7 \cos(15t) \qquad F_3 = 0.1 \sin(15t)$$

3.2 Equations of Motion

The stiffness, dampness, and mass matrices for solving the necessary variables are derived from the free body diagrams of each mass.

Newton's second law (Eq. (3.1)) is applied to each mass to derive the equation of motion for each mass.

$$\Sigma F = mx \tag{3.1}$$

The motion of the first mass is related to the spring and damper connected to the fixed support as well as the spring and damper connected to the second mass. The free body diagram represents the forces acting on the first mass.

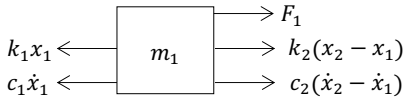

Figure 3.3 – Free body diagram for mass 1.

From the free body diagram in Fig. 3.3:

$$\Sigma F = -k_1x_1 - c_1x_1 + k_2(x_2 - x_1) + c_2(x_2 - x_1) + F_1 = m_1x_1 \tag{3.2}$$

$$m_1x_1 + k_1x_1 + c_1x_1 - k_2(x_2 - x_1) - c_2(x_2 - x_1) = F_1 \tag{3.3}$$

$$m_1x_1 + k_1x_1 + c_1x_1 - k_2x_2 + k_2x_1 - c_2x_2 + c_2x_1 = F_1 \tag{3.4}$$

$$(m_1)x_1 + (c_1 + c_2)x_1 + (-c_2)x_2 + (k_2 + k_1)x_1 + (-k_2)x_2 = F_1 \tag{3.5}$$

The identical approach is used for the second and third masses.

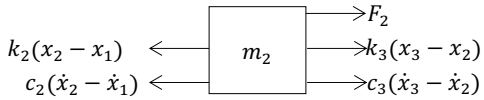

Figure 3.4 – Free body diagram for mass 2.

From the free body diagram in Fig. 3.4:

$$\Sigma F = -k_2(x_2 - x_1) - c_2(x_2 - x_1) + k_3(x_3 - x_2) + c_3(x_3 - x_2) + F_2 = m_2x_2 \tag{3.6}$$

$$m_2x_2 + k_2(x_2 - x_1) + c_2(x_2 - x_1) - k_3(x_3 - x_2) - c_3(x_3 - x_2) = F_2 \tag{3.7}$$

$$m_2x_2 + k_2x_2 - k_2x_1 + c_2x_2 - c_2x_1 - k_3x_3 + k_3x_2 - c_3x_3 + c_3x_2 = F_2 \tag{3.8}$$

$$(m_2)x_2 + (-c_2)x_1 + (c_2 + c_3)x_2 + (-c_3)x_3 + (-k_2)x_1 + (k_2 + k_3)x_2 + (-k_3)x_3 = F_2 \tag{3.9}$$

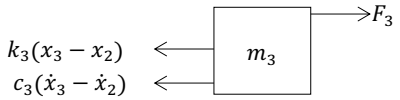

Figure 3.5 – Free body diagram for mass 3.

From the free body diagram in Fig. 3.5:

$$\Sigma F = -k_3(x_3 - x_2) - c_3(x_3 - x_2) + F_3 = m_3 x_3 \tag{3.10}$$

$$m_3 x_3 + k_3(x_3 - x_2) + c_3(x_3 - x_2) = F_3 \tag{3.11}$$

$$m_3 x_3 + k_3 x_3 - k_3 x_2 + c_3 x_3 - c_3 x_2 = F_3 \tag{3.12}$$

$$(m_3)x_3 + (-c_3)x_2 + (c_3)x_3 + (-k_3)x_2 + (k_3)x_3 = F_3 \tag{3.13}$$

The system of Eqs. (3.5), (3.9), and (3.13) may be written in matrix form:

$$\begin{bmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} c_1+c_2 & -c_2 & 0 \\ -c_2 & c_2+c_3 & -c_3 \\ 0 & -c_3 & c_3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} k_1+k_2 & -k_2 & 0 \\ -k_2 & k_2+k_3 & -k_3 \\ 0 & -k_3 & k_3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} \tag{3.13}$$

The mass, damper, and spring matrices will be used to solve this system.

$$M = \begin{bmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_3 \end{bmatrix} \tag{3.14}$$

$$C = \begin{bmatrix} c_1+c_2 & -c_2 & 0 \\ -c_2 & c_2+c_3 & -c_3 \\ 0 & -c_3 & c_3 \end{bmatrix} \tag{3.15}$$

$$K = \begin{bmatrix} k_1+k_2 & -k_2 & 0 \\ -k_2 & k_2+k_3 & -k_3 \\ 0 & -k_3 & k_3 \end{bmatrix} \tag{3.16}$$

3.3 Mathematical Modeling

Machine learning models are trained with observational data, which will in this case be solved from Eq. (3.13). The second order differential equation is converted to state space form resulting in two first order differential equations. New variables are introduced to reduce the order of the differential equation. Consider:

$$y_1 = x_1 \qquad y_4 = x_1$$

$$y_2 = x_2 \qquad y_5 = x_2$$

$$y_3 = x_3 \qquad y_6 = x_3$$

The derivative of these equations will allow for Eq. (3.13) to be rewritten in terms of the newly introduced variables.

$$y_1 = x_1 = y_4 \qquad y_4 = x_1$$

$$y_2 = x_2 = y_5 \qquad y_5 = x_2$$

$$y_3 = x_3 = y_6 \qquad y_6 = x_3$$

From Eq. (3.13):

$$M\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} - C\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} - K\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{3.17}$$

Which can be rewritten as:

$$M\begin{bmatrix} y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} - C\begin{bmatrix} y_4 \\ y_5 \\ y_6 \end{bmatrix} - K\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \tag{3.18}$$

ODE45 is used in MATLAB to solve Eq. 3. (see Appendix A), over a span of two seconds in 0.001 intervals. All initial conditions were set to zero. The resulting displacement, velocity, and acceleration are shown in the figures below for all masses. These data sets will be split for each model; 80% used as training data and 20% as testing data. A separate function is created in the code for the data splitting.

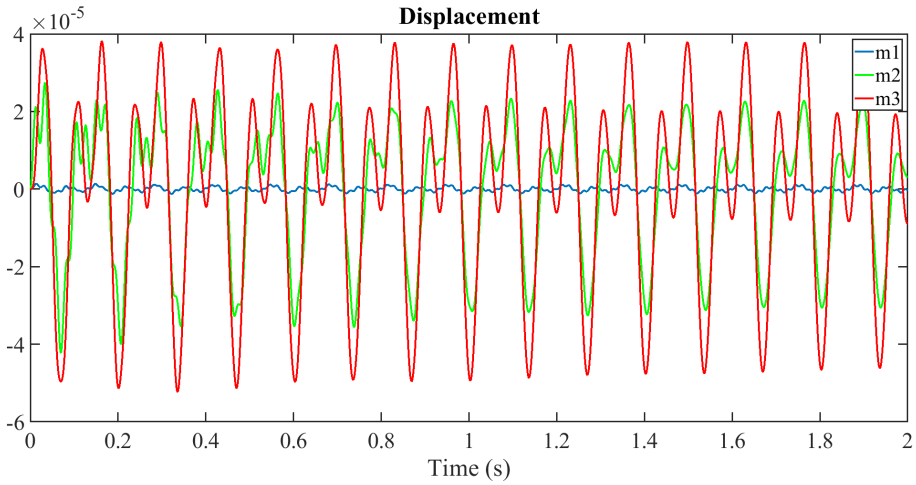

Figure 3.6 – Displacement over time for all masses.

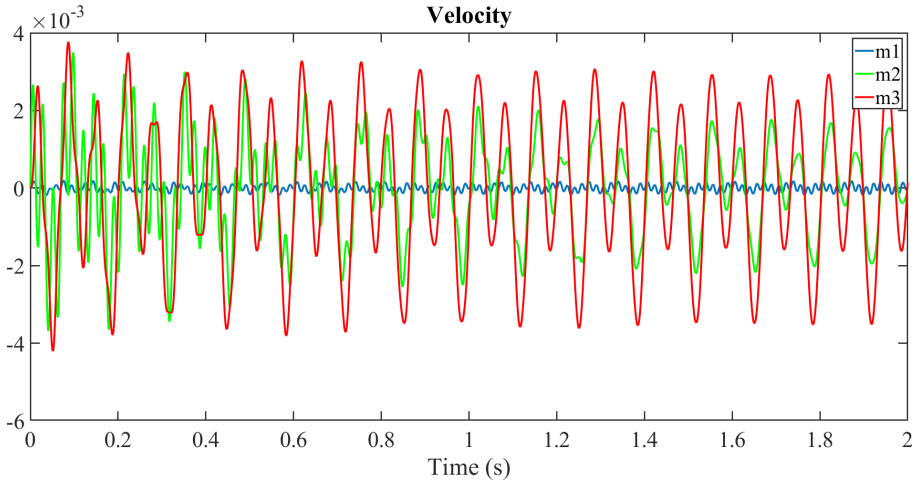

Figure 3.7 – Velocity over time for all masses.

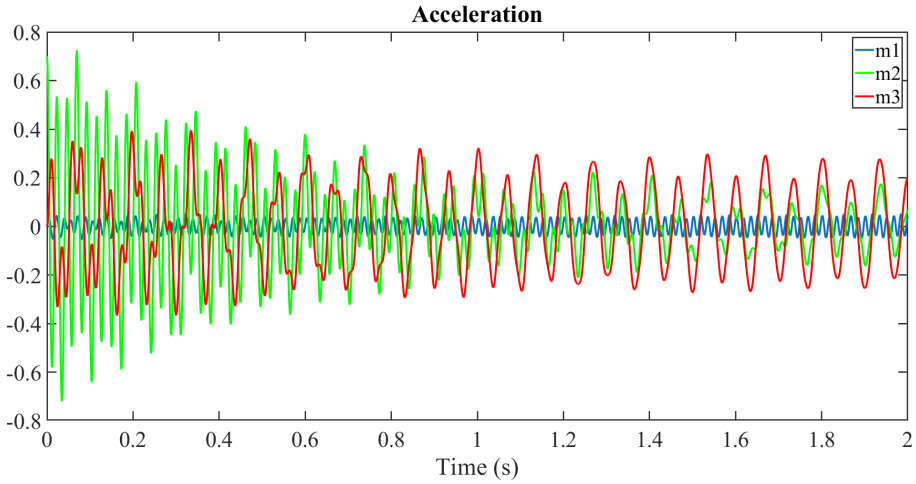

Figure 3.8 – Acceleration over time for all masses.

3.4 Linear Regression Modeling

3.4.1 Training Model

The linear regression model was trained in MATLAB using the "fitlm" function. QR decomposition is the main fitting algorithm and equations are estimated through M-estimation and solved using iteratively reweighted least squares [26]. Once the model is trained, it is evaluated using the "feval" function in MATLAB.

Models were trained with either displacement, velocity, or acceleration of all three masses to predict the kinematics of each single mass. Models were not trained with the same kinematics it was predicting, for example using displacement to predict displacement of a mass. A total of 18 models were trained for every combination of predictor and response variables.

3.4.2 Results

Performance was evaluated for each model using R^2 values.

Table 3.1 – All combinations of predictor and response variables used for training linear regression models, along with their respective R^2 values.

Predictor	Response	Trained R^2	Tested R^2
Acceleration	Displacement of M1	0.2467	0.2945
Acceleration	Displacement of M2	0.5418	0.522
Acceleration	Displacement of M3	0.7871	0.7801
Acceleration	Velocity of M1	0.2598	0.3209
Acceleration	Velocity of M2	0.0215	0.0266
Acceleration	Velocity of M3	0.0238	0.0446

Velocity	Displacement of M1	0.6266	0.6453
Velocity	Displacement of M2	0.1009	0.1179
Velocity	Displacement of M3	0.1933	0.2141
Velocity	Acceleration of M1	0.0172	0.0404
Velocity	Acceleration of M2	0.0601	0.0825
Velocity	Acceleration of M3	0.2963	0.3194
Displacement	Velocity of M1	0.2473	0.2754
Displacement	Velocity of M2	0.4346	0.3835
Displacement	Velocity of M3	0.7441	0.7495
Displacement	Acceleration of M1	0.2356	0.3154
Displacement	Acceleration of M2	0.2741	0.2152
Displacement	Acceleration of M3	0.9981	0.9981

Table 3.2 – All combinations of predictor and response variables used for training linear regression models, sorted by ascending trained R^2 values.

Predictor	Response	Trained R^2	Tested R^2
Velocity	Acceleration of M1	0.0172	0.0404
Acceleration	Velocity of M2	0.0215	0.0266
Acceleration	Velocity of M3	0.0238	0.0446
Velocity	Acceleration of M2	0.0601	0.0825
Velocity	Displacement of M2	0.1009	0.1179
Velocity	Displacement of M3	0.1933	0.2141
Displacement	Acceleration of M1	0.2356	0.3154
Acceleration	Displacement of M1	0.2467	0.2945
Displacement	Velocity of M1	0.2473	0.2754
Acceleration	Velocity of M1	0.2598	0.3209
Displacement	Acceleration of M2	0.2741	0.2152
Velocity	Acceleration of M3	0.2963	0.3194
Displacement	Velocity of M2	0.4346	0.3835
Acceleration	Displacement of M2	0.5418	0.522
Velocity	Displacement of M1	0.6266	0.6453
Displacement	Velocity of M3	0.7441	0.7495
Acceleration	Displacement of M3	0.7871	0.7801
Displacement	Acceleration of M3	0.9981	0.9981

Models with R^2 values higher than 0.6 will be considered high performance. It is first observed that no one predictor type can accurately model the response kinematics, nor are the kinematics of any one mass easily modeled. The models with the lowest R^2 values use velocity to predict acceleration and vice versa. The models using acceleration to predict the velocity of M1 and velocity to predict the acceleration of M3 have slightly higher values of R^2. The model with the lowest R^2 value, using velocity predictors to predict the acceleration of M1, is seen in Fig. 3.10. The enlarged view in Fig 3.11 shows that the model can follow the general trend of the data but not in the correct amplitude or frequency. The low correlation between the training data and the model prediction, seen in Fig. 3.9, confirms the low performance of the model.

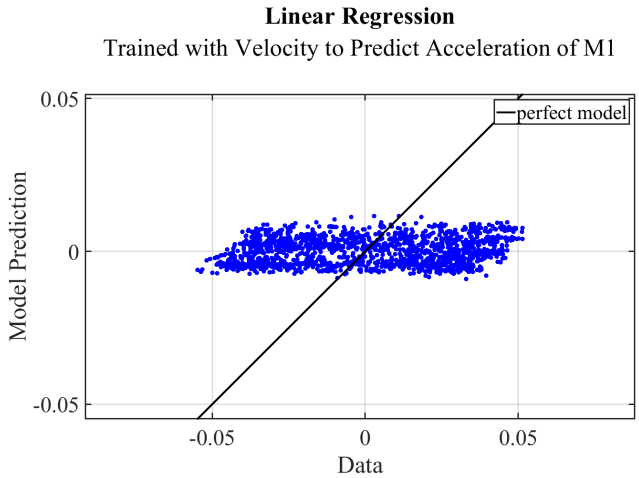

Figure 3.9 – Training data vs model prediction for linear regression model trained with velocity to predict acceleration of M1.

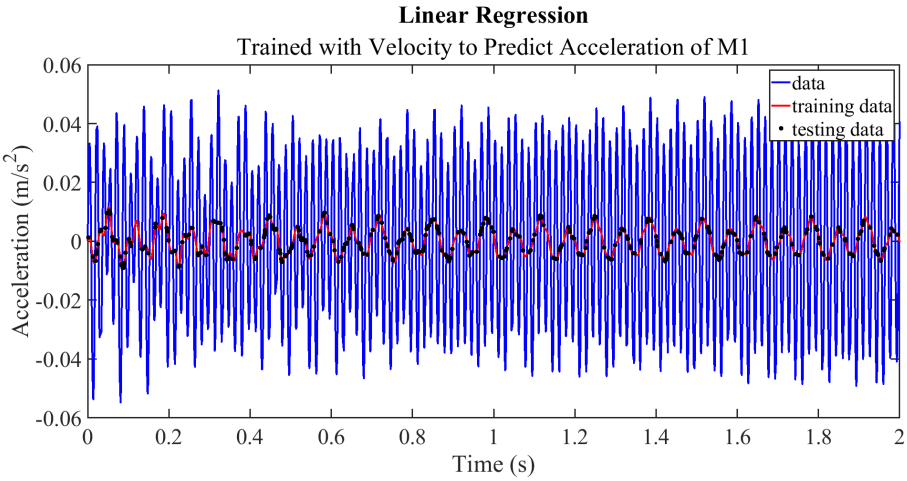

Figure 3.10 – Linear regression model trained with velocity to predict acceleration of M1.

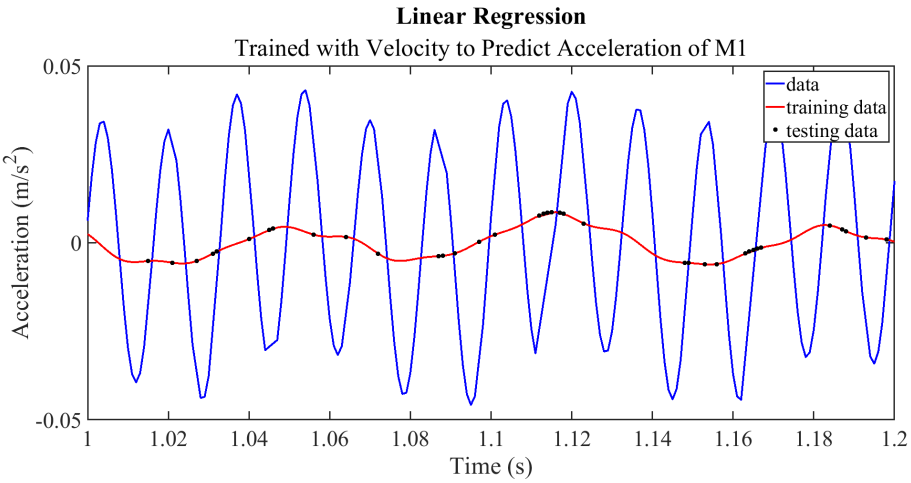

Figure 3.11 – Enlarged view of linear regression model trained with velocity to predict acceleration of M1.

Four models showed high performance and ability to model the data. Three of the highest performing models predicted the kinematics of mass 3:

- Using displacement to predict the velocity of M3
- Using acceleration to predict the displacement of M3
- Using displacement to predict the acceleration of M3

The other predicting displacement of M1 using velocity predictor variables. As R^2 increases, the observed data and model prediction have a stronger positive correlation, seen as a positive linear slope in the following figures.

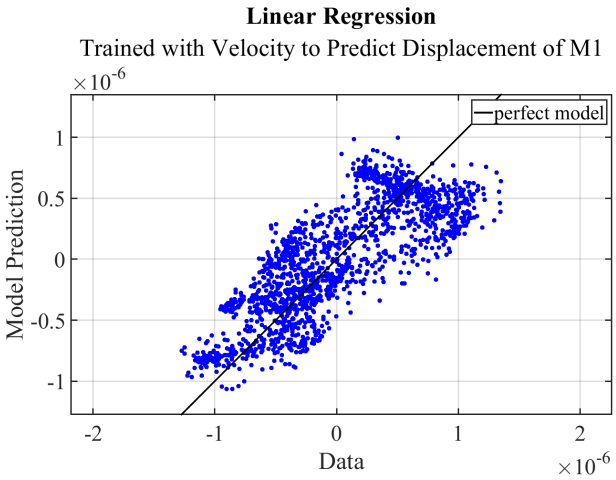

Figure 3.12 – Training data vs model prediction for linear regression model trained with velocity to predict displacement of M1.

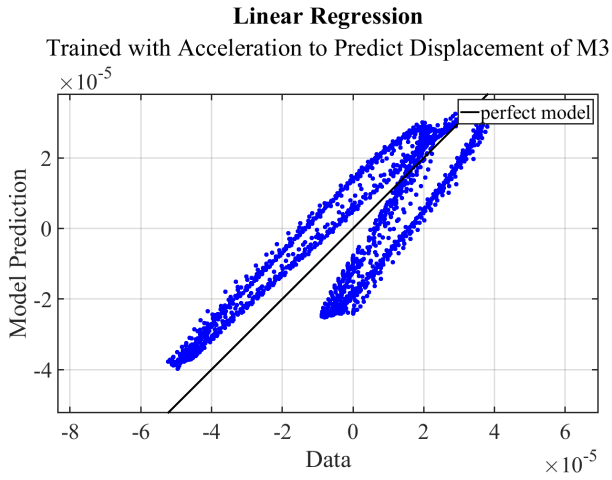

Figure 3.13 – Training data vs model prediction for linear regression model trained with acceleration to predict displacement of M3.

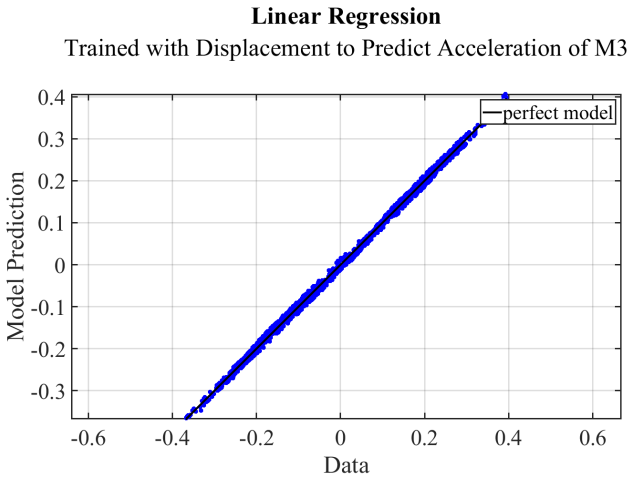

Figure 3.14 – Training data vs model prediction for linear regression model trained with displacement to predict acceleration of M3.

The near perfect linear relationship between the data and model prediction shows the high performance of the model. In Fig. 3.15, the training data is close to identical to the data with small discrepancies. The discrepancies are shown in an enlarged view in Fig. 3.16.

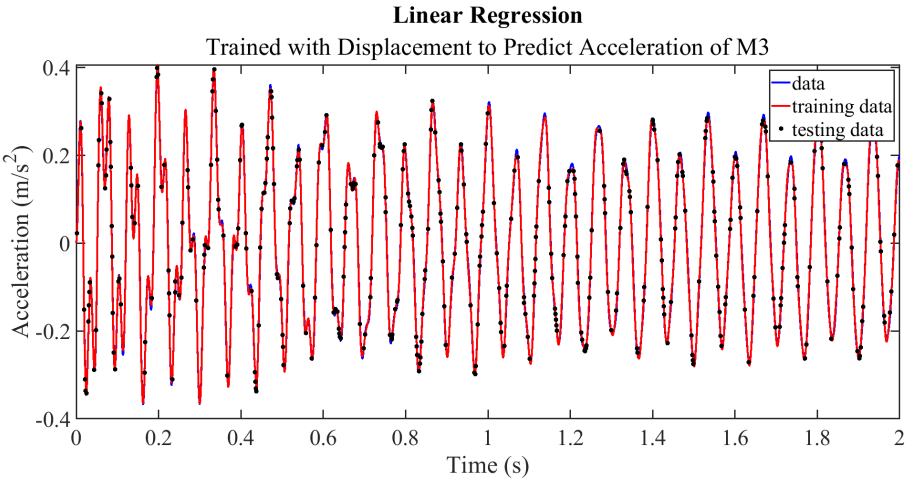

Figure 3.15 – Linear regression model trained with displacement to predict acceleration of M3.

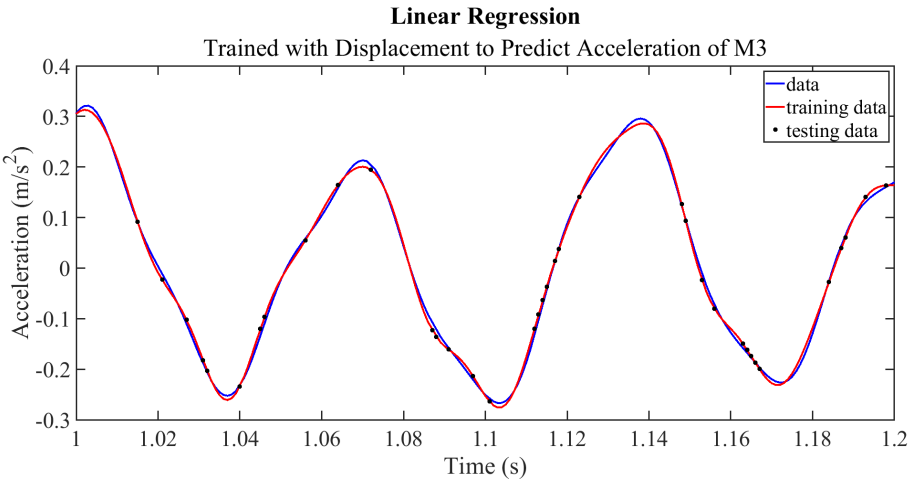

Figure 3.16 – Enlarged view of linear regression model trained with displacement to predict acceleration of M3.

3.5 Decision Tree Modeling

3.5.1 Training Model

The decision tree models were trained in MATLAB using the "fitrtree" function, with all name-value arguments left as default. For standard classification and regression trees, the node splitting process follows a set of steps by default. The algorithm first computes with weighted MSE (Eq. 3.19) of the response for a given node t [27]. The weight of observation j (w_j) is by default $1/n$, where n is the sample size.

$$\varepsilon_t = \sum_{j \in t} w_j (y_j - \bar{y}_t)^2 \tag{3.19}$$

The algorithm then computes the probability that an observation is in each node (Eq. (3.20)) [27].

$$P(T) = \sum_{j \in t} w_j \tag{3.20}$$

Each predictor is a splitting candidate, and the elements of each predictor are sorted in ascending order. The best split node is determined by maximizing the reduction of MSE of all splitting options and choosing the node with the largest reduction [27].

By default, the tree depth is controlled using MSE to merge any leaves whose sum MSE is no larger the MSE of their parent node [27]. The maximum number of splits is by default one

less than the data sample size, and the default minimum leaf and parent size are one and ten respectively [27]. The algorithm prunes the tree if the number of splits exceeds the maximum by unsplitting the least successful branches [27].

3.5.2 Results

The same 18 cases from the previous section were trained using the decision tree method.

Table 3.3 – All combinations of predictor and response variables used for training decision tree models, along with their respective R^2 values.

Predictor	Response	Trained R^2	Tested R^2
Acceleration	Displacement of M1	0.8309	0.3736
Acceleration	Displacement of M2	0.9356	0.5999
Acceleration	Displacement of M3	0.9683	0.8335
Acceleration	Velocity of M1	0.8108	0.2365
Acceleration	Velocity of M2	0.7274	0.0594
Acceleration	Velocity of M3	0.7493	0.1105
Velocity	Displacement of M1	0.9508	0.7714
Velocity	Displacement of M2	0.8711	0.354
Velocity	Displacement of M3	0.8606	0.3889
Velocity	Acceleration of M1	0.8304	0.407
Velocity	Acceleration of M2	0.6965	0.0055
Velocity	Acceleration of M3	0.8686	0.37
Displacement	Velocity of M1	0.9362	0.6556
Displacement	Velocity of M2	0.887	0.4755
Displacement	Velocity of M3	0.9844	0.9268
Displacement	Acceleration of M1	0.9433	0.7641
Displacement	Acceleration of M2	0.9323	0.7199
Displacement	Acceleration of M3	0.9955	0.9781

Table 3.4 – All combinations of predictor and response variables used for training decision tree models, sorted by ascending trained R^2 values.

Predictor	Response	Trained R^2	Tested R^2
Velocity	Acceleration of M2	0.6965	0.0055
Acceleration	Velocity of M2	0.7274	0.0594
Acceleration	Velocity of M3	0.7493	0.1105
Acceleration	Velocity of M1	0.8108	0.2365
Velocity	Acceleration of M1	0.8304	0.407
Acceleration	Displacement of M1	0.8309	0.3736
Velocity	Displacement of M3	0.8606	0.3889
Velocity	Acceleration of M3	0.8686	0.37
Velocity	Displacement of M2	0.8711	0.354
Displacement	Velocity of M2	0.887	0.4755
Displacement	Acceleration of M2	0.9323	0.7199

Acceleration	Displacement of M2	0.9356	0.5999
Displacement	Velocity of M1	0.9362	0.6556
Displacement	Acceleration of M1	0.9433	0.7641
Velocity	Displacement of M1	0.9508	0.7714
Acceleration	Displacement of M3	0.9683	0.8335
Displacement	Velocity of M3	0.9844	0.9268
Displacement	Acceleration of M3	0.9955	0.9781

The decision tree models produce significantly higher R^2 values compared to the linear regression models. All the decision tree models are considered high performance based on the previously established R^2 value of ≥0.6 for high performance models. The four highest performing cases are the same as for linear regression:

- Using displacement to predict the velocity of M3
- Using acceleration to predict the displacement of M3
- Using displacement to predict the acceleration of M3

The lowest performing cases use velocity to predict acceleration and vice versa, as observed in the linear regression models. For most cases, the tested R^2 values are significantly lower than the trained R^2 values which is a sign of overfitting. This can be seen in Fig. 3.17 for the lowest performance case, using velocity to predict the acceleration of M2, and especially in the enlarged view in Fig. 3.18.

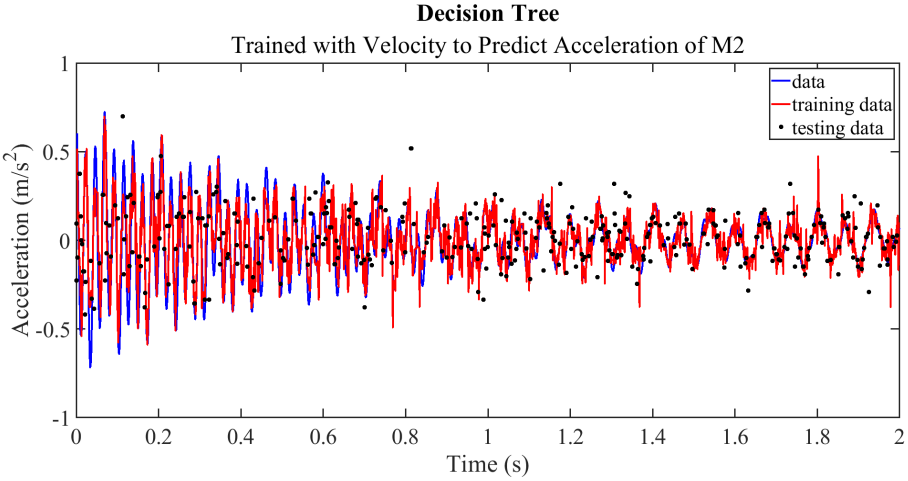

Figure 3.17 – Decision tree model trained with velocity to predict acceleration of M2.

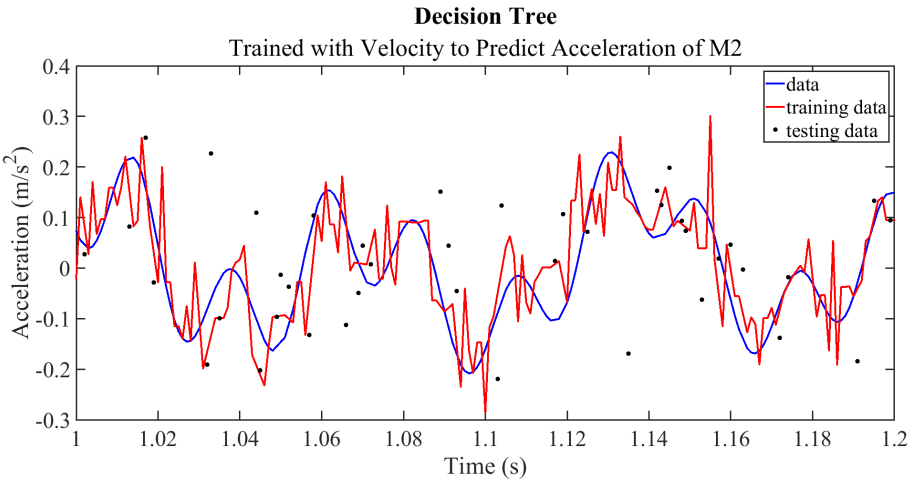

Figure 3.18 – Enlarged view of decision tree model trained with displacement to predict acceleration of M3.

The decision tree algorithm produces predictions with high noise, seen in Fig. 3.18, due to its discrete nature. The discrete nature produces a tendency of the model to overfit the data. The model overfits the data but is still able to follow it leading to a fairly strong correlation between the data and model prediction, seen in Fig. 3.19.

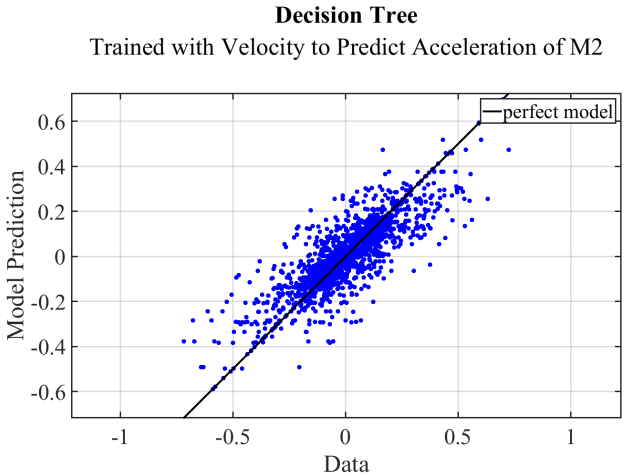

Figure 3.19 – Training data vs model prediction for linear regression model trained with velocity to predict acceleration of M2.

The highest performing model is the same as for linear regression method. Overall, the model prediction had a strong positive correlation with the data, as seen in Fig. 3.20, but is not as strong as in the linear regression model (Fig. 3.14).

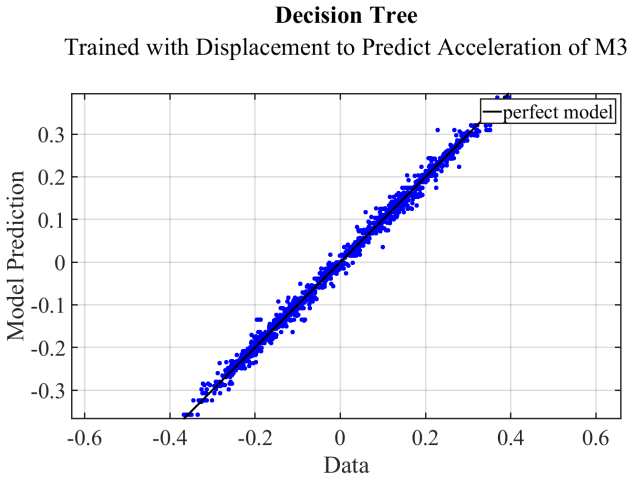

Figure 3.20 – Training data vs model prediction for linear regression model trained with displacement to predict acceleration of M3.

During the first second, the model prediction is much noisier than the linear regression model and has more discrepancies. The noise reduces with time and the discrepancies lower.

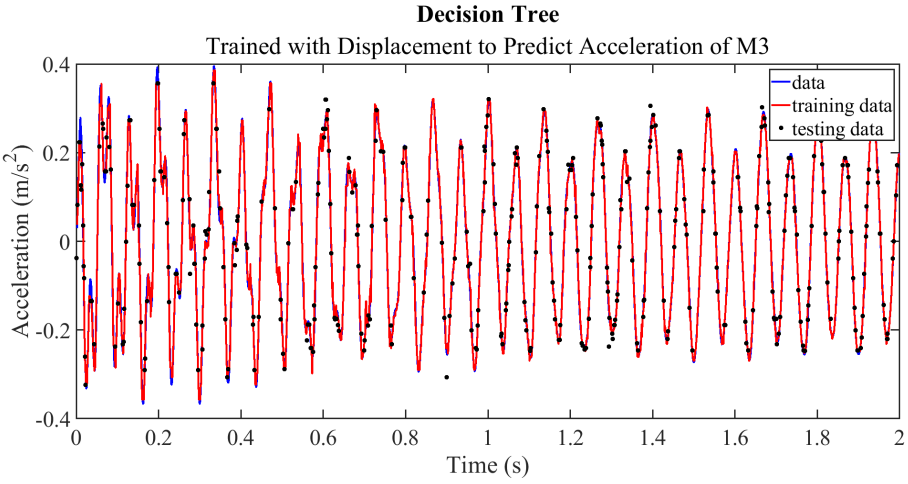

Figure 3.21 – Decision tree model trained with displacement to predict acceleration of M3.

As seen in Fig. 3.22, this decision tree model prediction is not as smooth as the prediction from the linear regression model seen in Fig. 3.16. The model is still overfitting the data, though not as severely as in Fig. 3.18 for the lowest performing model.

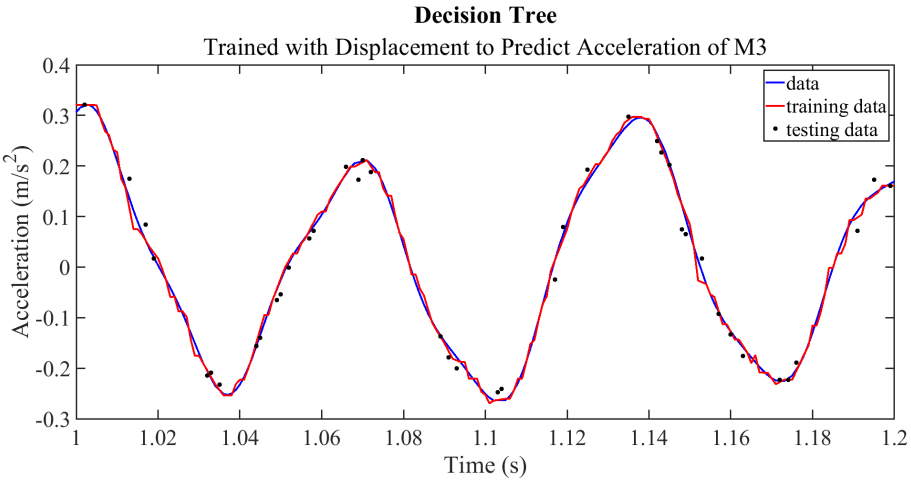

Figure 3.22 – Enlarged view of linear regression model trained with displacement to predict acceleration of M3.

3.6 Random Forest Modeling

3.6.1 Training Model

The random forest models were trained in MATLAB using the "fitrensemble" function, which returns a trained regression ensemble. Method, number of trees, and learning rate parameters were initially optimized using the automatic "OptimizeHyperparameters" option for the model using displacement to predict acceleration of M3. The optimized ensemble was determined to use the least-squares boosting method, 500 number of trees, and a learning rate of 0.55246. The least-squares boosting method builds a new tree based on the difference between the previously built trees and the response data so that the MSE is minimized [28].

The trained random forest models were expected to show an increase in performance. The initial optimization resulted in majority of the models to significantly overfit. This was later corrected with an additional optimization done on the worst performing model, using velocity for the prediction of acceleration of M2. The optimized ensemble for this model uses the bagging method with 363 trees.

3.6.2 Results

The results from both optimizations were reviewed and compared.

Table 3.5 – All combinations of predictor and response variables used for training random forest models using boosting method with 500 trees, along with their respective R^2 values.

Predictor	Response	Trained R^2	Tested R^2
Acceleration	Displacement of M1	0.9976	0.3859
Acceleration	Displacement of M2	0.9988	0.6158
Acceleration	Displacement of M3	0.9995	0.8425
Acceleration	Velocity of M1	0.9968	0.2628
Acceleration	Velocity of M2	0.9946	0.0478
Acceleration	Velocity of M3	0.9955	0.1184
Velocity	Displacement of M1	0.9993	0.7874
Velocity	Displacement of M2	0.9979	0.3574
Velocity	Displacement of M3	0.9983	0.4025
Velocity	Acceleration of M1	0.9976	0.3907
Velocity	Acceleration of M2	0.9945	0.0087
Velocity	Acceleration of M3	0.9977	0.4343
Displacement	Velocity of M1	0.9987	0.6447
Displacement	Velocity of M2	0.998	0.4764
Displacement	Velocity of M3	0.9998	0.928
Displacement	Acceleration of M1	0.9996	0.8081
Displacement	Acceleration of M2	0.9996	0.8281
Displacement	Acceleration of M3	1	0.9926

Table 3.6– All combinations of predictor and response variables used for training random forest models using boosting method with 500 trees, sorted by ascending trained R^2 values.

Predictor	Response	Trained R^2	Tested R^2
Velocity	Acceleration of M2	0.9945	0.0087
Acceleration	Velocity of M2	0.9946	0.0478
Acceleration	Velocity of M3	0.9955	0.1184
Acceleration	Velocity of M1	0.9968	0.2628
Acceleration	Displacement of M1	0.9976	0.3859
Velocity	Acceleration of M1	0.9976	0.3907
Velocity	Acceleration of M3	0.9977	0.4343
Velocity	Displacement of M2	0.9979	0.3574
Displacement	Velocity of M2	0.998	0.4764
Velocity	Displacement of M3	0.9983	0.4025
Displacement	Velocity of M1	0.9987	0.6447
Acceleration	Displacement of M2	0.9988	0.6158
Velocity	Displacement of M1	0.9993	0.7874
Acceleration	Displacement of M3	0.9995	0.8425
Displacement	Acceleration of M1	0.9996	0.8081
Displacement	Acceleration of M2	0.9996	0.8281
Displacement	Velocity of M3	0.9998	0.928
Displacement	Acceleration of M3	1	0.9926

The initial optimization created models such that all the cases resulted in a trained R^2 value of at least 0.99. Though this is ideal, the tested R^2 values show that majority of the models were overfit. The highest performing model, same as with the previous methods, used displacement to predict the acceleration of M3. This was the only model to have resulted in a R^2 value of one and had a near perfect correlation of the data and model prediction.

Random Forest
LSBoost Method
Trained with Displacement to Predict Acceleration of M3

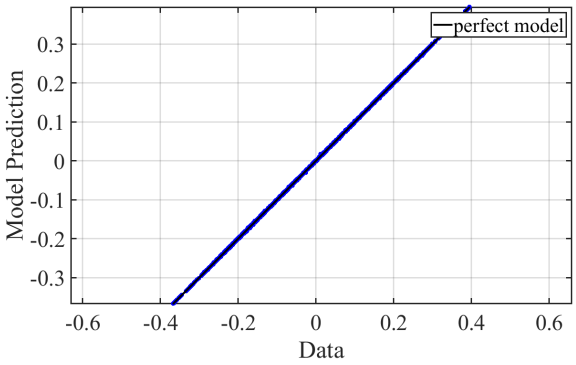

Figure 3.23 – Training data vs model prediction for linear regression model trained with displacement to predict acceleration of M3.

The discrepancies for this model are barely seen between 1.06 and 1.08 seconds in Fig. 3.24. No noticeable noise is seen in this model when compared with the decision tree model (Fig. 3.22). The prediction of this model is of higher performance than the linear regression mode in Fig. 3.16 but has more misalignment of testing data to the model prediction.

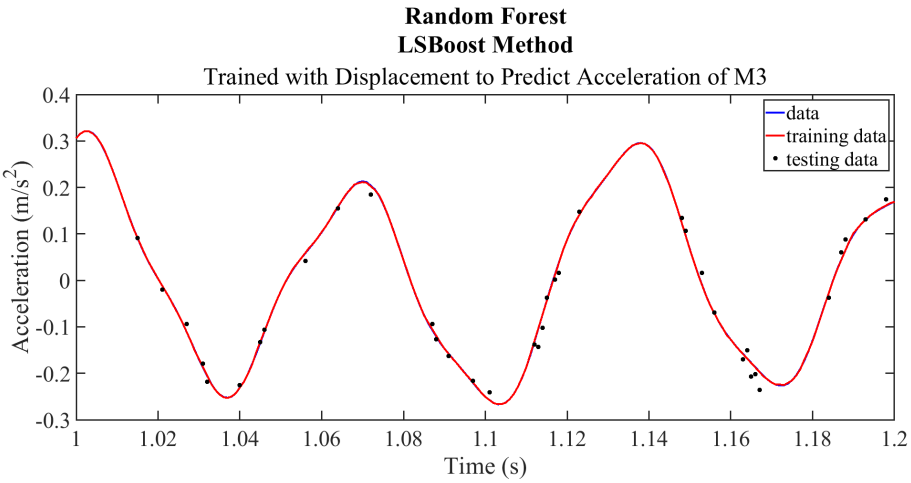

Figure 3.24 – Random forest model using the boosting method with 500 trees trained with displacement to predict acceleration of M3.

The lowest performing case used velocity to predict acceleration of M2 and had a strong positive correlation between the data and model prediction (seen in Fig. 3.25).

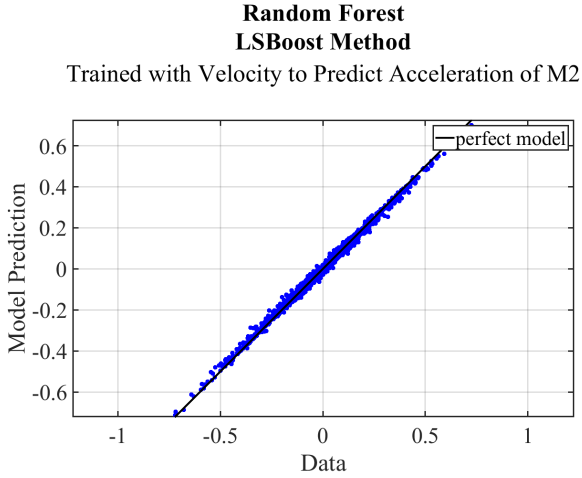

Figure 3.25 – Training data vs model prediction for boosting random forest model trained with velocity to predict acceleration of M2.

The severe overfitting in this model is seen in the plot in Fig. 3.26.

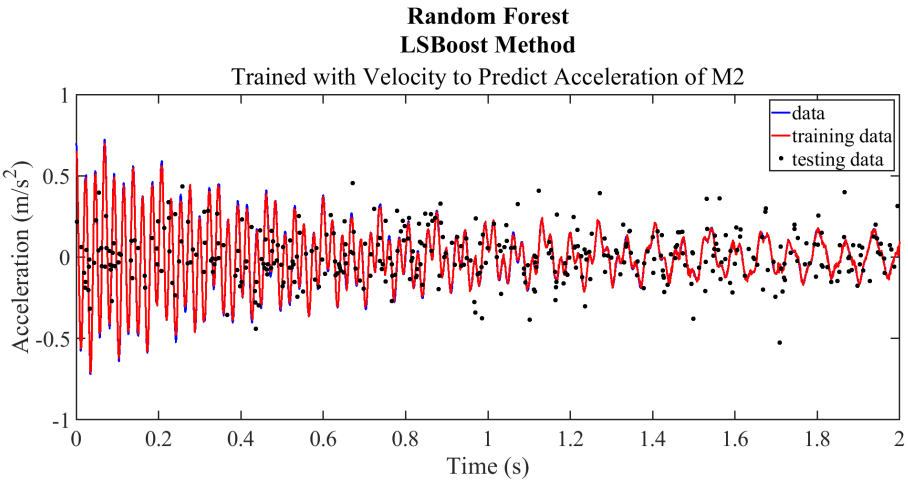

Figure 3.26 – Random forest model using the boosting method with 500 trees trained with velocity to predict acceleration of M2.

The severity of the overfitting is seen in Fig. 3.27 where most of the testing data is not aligned with the model prediction, therefore diminishing the accuracy of the model. This created the need to reoptimize the model.

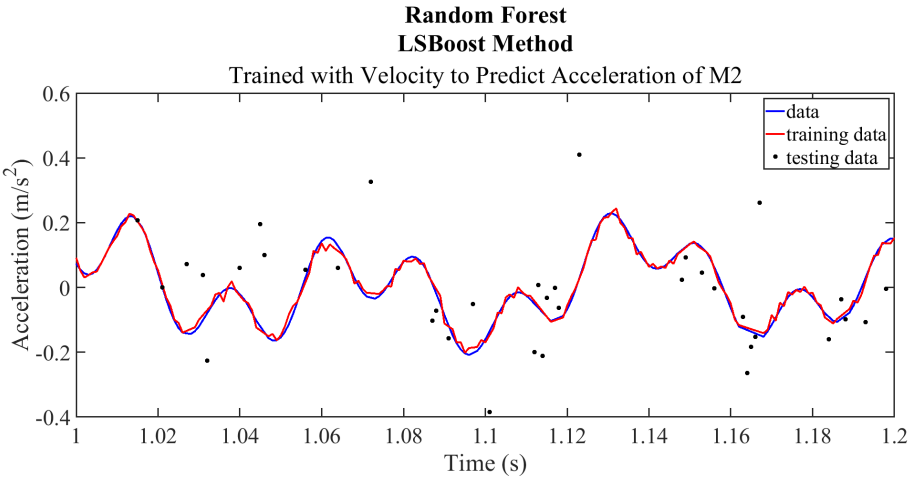

Figure 3.27 – Enlarged view of random forest model using the boosting method with 500 trees trained with velocity to predict acceleration of M2.

The worst model from Table 3.6 was optimized to reduce the overfitting. The newly optimized model was determined to have a different algorithm for optimization and number of trees; bagging with 363 trees. This decreased the trained R^2 values, while increased the tested R^2 values.

Table 3.7 – All combinations of predictor and response variables used for training random forest models using bagging method with 363 trees, along with their respective R^2 values.

Predictor	Response	Trained R^2	Tested R^2
Acceleration	Displacement of M1	0.799	0.5403
Acceleration	Displacement of M2	0.8859	0.7107
Acceleration	Displacement of M3	0.944	0.8721
Acceleration	Velocity of M1	0.7552	0.3898
Acceleration	Velocity of M2	0.7049	0.1385
Acceleration	Velocity of M3	0.7339	0.2486
Velocity	Displacement of M1	0.9229	0.8383
Velocity	Displacement of M2	0.8006	0.4778
Velocity	Displacement of M3	0.8177	0.5273
Velocity	Acceleration of M1	0.8068	0.5128
Velocity	Acceleration of M2	0.6382	0.0441
Velocity	Acceleration of M3	0.8097	0.5355
Displacement	Velocity of M1	0.8955	0.7378
Displacement	Velocity of M2	0.8315	0.6186
Displacement	Velocity of M3	0.9729	0.9329
Displacement	Acceleration of M1	0.9186	0.8357
Displacement	Acceleration of M2	0.9035	0.7564
Displacement	Acceleration of M3	0.9866	0.97

Table 3.8 – All combinations of predictor and response variables used for training random forest models using bagging method with 363 trees, sorted by ascending trained R^2 values.

Predictor	Response	Trained R^2	Tested R^2
Velocity	Acceleration of M2	0.6382	0.0441
Acceleration	Velocity of M2	0.7049	0.1385
Acceleration	Velocity of M3	0.7339	0.2486
Acceleration	Velocity of M1	0.7552	0.3898
Acceleration	Displacement of M1	0.799	0.5403
Velocity	Displacement of M2	0.8006	0.4778
Velocity	Acceleration of M1	0.8068	0.5128
Velocity	Acceleration of M3	0.8097	0.5355
Velocity	Displacement of M3	0.8177	0.5273
Displacement	Velocity of M2	0.8315	0.6186
Acceleration	Displacement of M2	0.8859	0.7107
Displacement	Velocity of M1	0.8955	0.7378
Displacement	Acceleration of M2	0.9035	0.7564

Displacement	Acceleration of M1	0.9186	0.8357
Velocity	Displacement of M1	0.9229	0.8383
Acceleration	Displacement of M3	0.944	0.8721
Displacement	Velocity of M3	0.9729	0.9329
Displacement	Acceleration of M3	0.9866	0.97

Comparing to the boosting method models in Table 3.6, the lowest and highest performing models kept their same level of performance compared to the rest. The tested R^2 values increased significantly from the boosting method. An enlarged view of the lowest performing model, using velocity to predict the acceleration of M2, showed that overfitting was improved from the boosting method (Fig. 3.26). The model prediction did however loss its accuracy, especially in the first half of the plot in Fig. 3.28.

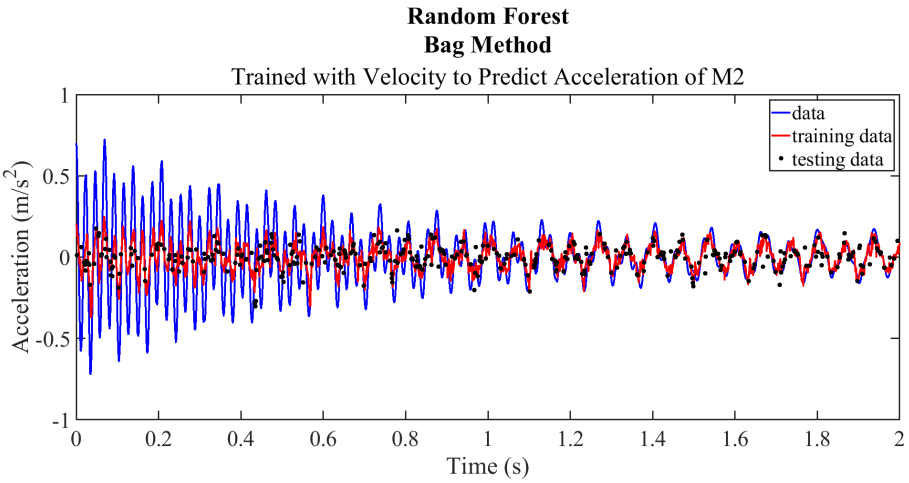

Figure 3.28 – Random forest model using the bagging method with 363 trees trained with velocity to predict acceleration of M2.

The enlarged view of the model prediction also shows the model is has more noise than the boosting method but is still able to follow the data. The testing data is also able to follow the model prediction more closely than with the boosting method.

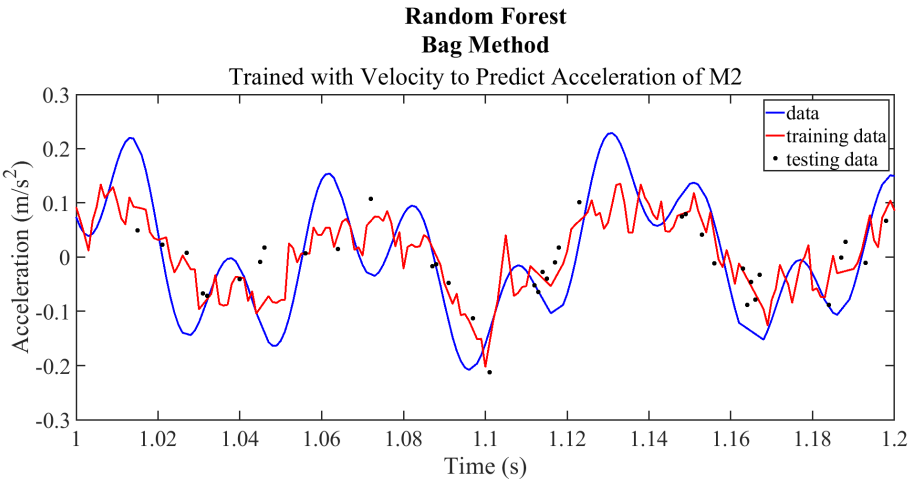

Figure 3.29 – Enlarged view of random forest model using the bagging method with 363 trees trained with velocity to predict acceleration of M2.

The R^2 values of the highest performing model were reduced with the second optimization, resulting in a cloudier correlation of data and model prediction.

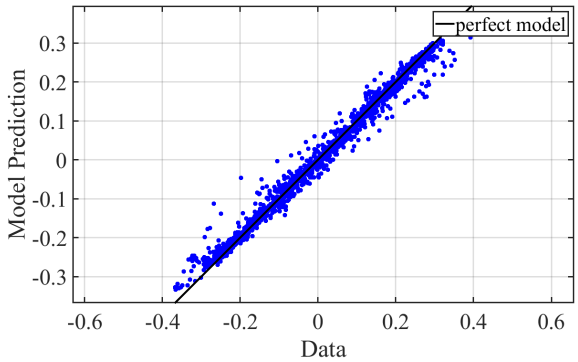

Figure 3.30 – Training data vs model prediction for bagging random forest model trained with displacement to predict acceleration of M3.

The decision tree model (Fig. 3.22) is much noisier and overfit compared to the bagging random forest model in Fig. 3.31. This model is similar to the linear regression model (Fig. 3.16) having slightly more discrepancies and overfitting, and much less overfitting when compared to the boosting method (Fig. 3.24).

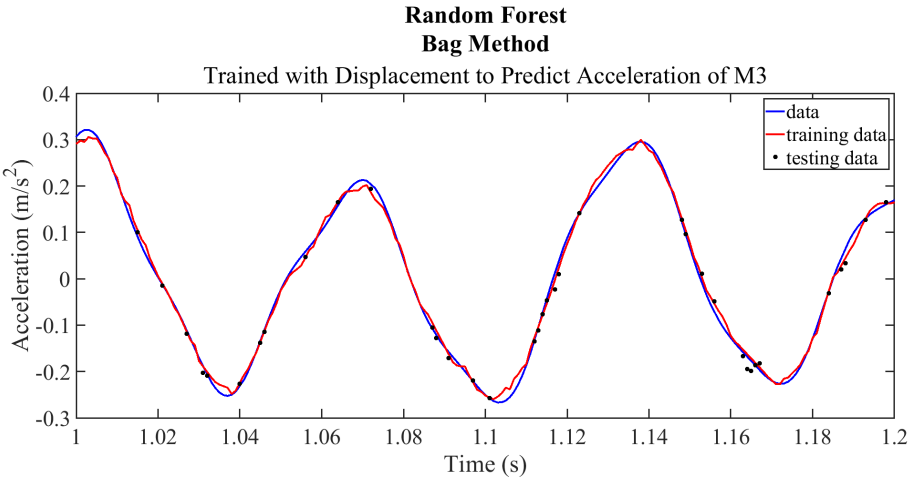

Figure 3.31 – Enlarged view of bagging random forest model trained with displacement to predict acceleration of M3.

3.7 Neural Network Modeling

3.7.1 Training Model

The neural network models were trained in MATLAB using the "fitnet" and "train" functions. Given the hidden layer size, the "fitnet" function returns a function fitting neural network that forms a generalization of the predictor response relationship [29]. Due to limited computational power, the hidden layer size was constrained to two layers with between 0-50 neurons. The optimal hidden layer size was determined, by the maximum average trained R^2 values for all 18 cases, to be one layer with 19 neurons and another with 40. The model is then trained using the "train" function until the maximum number of epochs or the performance goal is met [30].

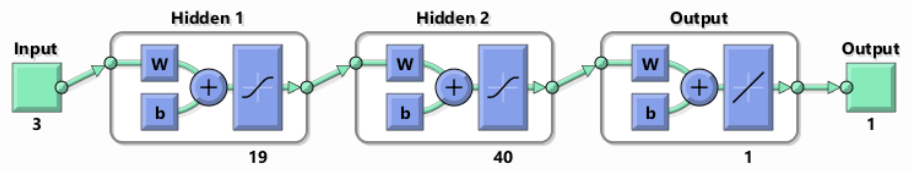

Figure 3.32 – Diagram of neural network model created in MATLAB.

3.7.2 Results

The same 18 cases from the previous section were trained using the decision tree model.

Table 3.9 – All combinations of predictor and response variables used for training neural network models, along with their respective R^2 values.

Predictor	Response	Trained R^2	Tested R^2
Acceleration	Displacement of M1	0.0014	0.004
Acceleration	Displacement of M2	0.4582	0.4355
Acceleration	Displacement of M3	0.4505	0.4091
Acceleration	Velocity of M1	0.0159	0.0309
Acceleration	Velocity of M2	0.1697	0.0725
Acceleration	Velocity of M3	0.3125	0.2753
Velocity	Displacement of M1	0.2274	0.2706
Velocity	Displacement of M2	0.1208	0.1443
Velocity	Displacement of M3	0.0082	0.0058
Velocity	Acceleration of M1	0.5402	0.5172
Velocity	Acceleration of M2	0.1504	0.0376
Velocity	Acceleration of M3	0.6268	0.4552
Displacement	Velocity of M1	0.1114	0.1324
Displacement	Velocity of M2	0.6573	0.6071
Displacement	Velocity of M3	0.9571	0.9409
Displacement	Acceleration of M1	0.8705	0.8494
Displacement	Acceleration of M2	0.9669	0.9141
Displacement	Acceleration of M3	0.9995	0.9993

Table 3.10 – All combinations of predictor and response variables used for training neural network models, sorted by ascending trained R^2 values.

Predictor	Response	Trained R^2	Tested R^2
Acceleration	Displacement of M1	0.0014	0.004
Acceleration	Displacement of M2	0.0082	0.0058
Velocity	Acceleration of M1	0.0159	0.0309
Acceleration	Displacement of M3	0.1114	0.1324
Velocity	Displacement of M2	0.1208	0.1443
Acceleration	Velocity of M3	0.1504	0.0376
Acceleration	Velocity of M2	0.1697	0.0725
Velocity	Displacement of M1	0.2274	0.2706
Velocity	Displacement of M3	0.3125	0.2753
Velocity	Acceleration of M3	0.4505	0.4091
Velocity	Acceleration of M2	0.4582	0.4355
Displacement	Velocity of M1	0.5402	0.5172
Acceleration	Velocity of M1	0.6268	0.4552
Displacement	Velocity of M2	0.6573	0.6071
Displacement	Velocity of M3	0.8705	0.8494

Displacement	Acceleration of M1	0.9571	0.9409
Displacement	Acceleration of M2	0.9669	0.9141
Displacement	Acceleration of M3	0.9995	0.9993

In other methods, models using velocity to predict acceleration and vice versa performed poorly. Interestingly, the neural networks method performed poorly for models using acceleration to predict displacement. The highest performing models also differed from previous methods. This was the only method whose highest performing models all used the same predictor to predict one kinematic property of all three masses. The performance of the models is more extremely distributed when compared to linear regression model. The lowest performing neural network models had the lowest trained R^2 values compared to Table 3.2. The overall performance of neural network models does not compare to decision tree and random forest methods, whose R^2 values were all high performance.

Another interesting observation with neural networks is the correlation between data and model predictions. In previous methods, models with low R^2 values tended to have the correlation disbursed horizontally, whereas for neural networks the correlation is more vertical.

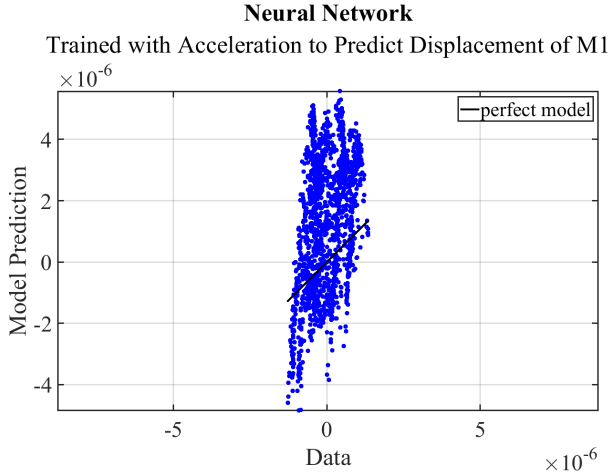

Figure 3.33 – Training data vs model prediction for neural network model trained with acceleration to predict displacement of M1.

The low performing models had issues with following the data, as well as overfitting. Other methods were at least able to follow the general trend of the data. The model using acceleration to predict displacement was the lowest performing neural network model and was only able to follow the concavity.

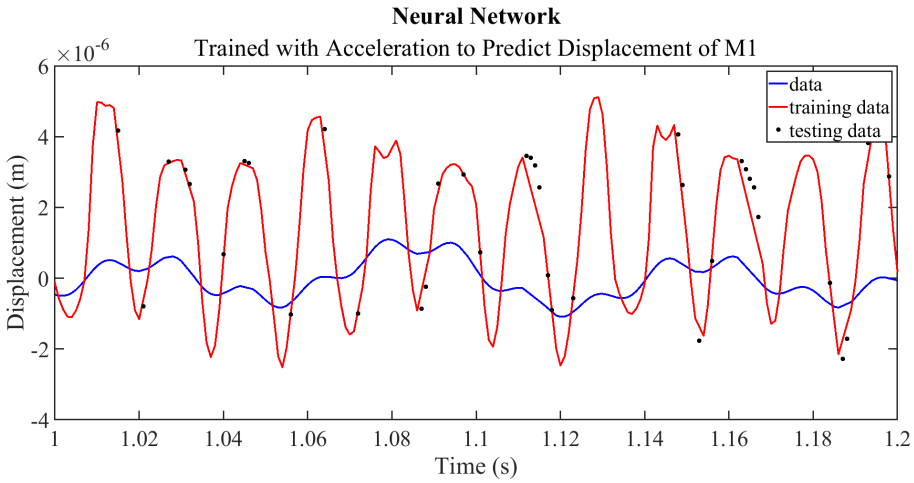

Figure 3.34 – Enlarged view of neural network model trained with acceleration to predict displacement of M1.

The difference between trained and tested R^2 values signify that the models are overfitting the data. This can be seen in Fig. 3.35 for the model using displacement to predict acceleration of M2, the second-best performing model. The model is also much noisier than decision tree and random forest models are similar values of R^2.

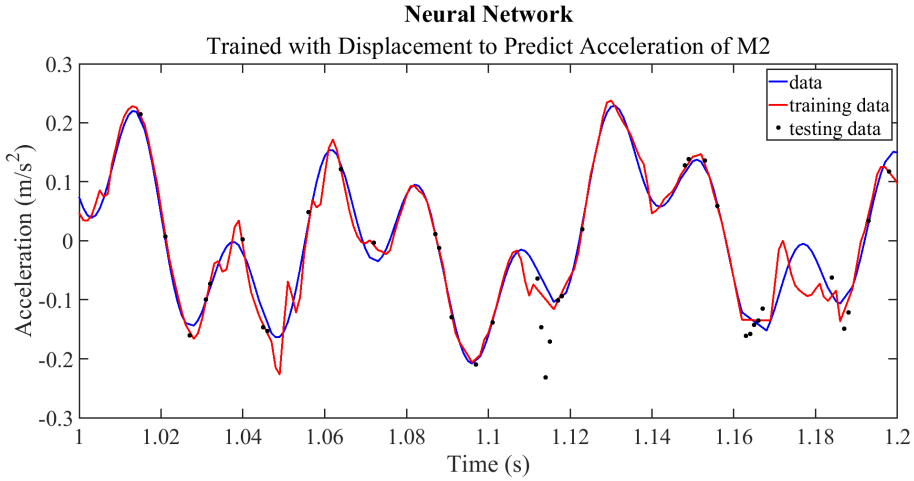

Figure 3.35 – Enlarged view of neural network model trained with displacement to predict acceleration of M1.

The model with the highest R^2 value, using displacement to predict acceleration of M3, showed little overfitting and discrepancies.

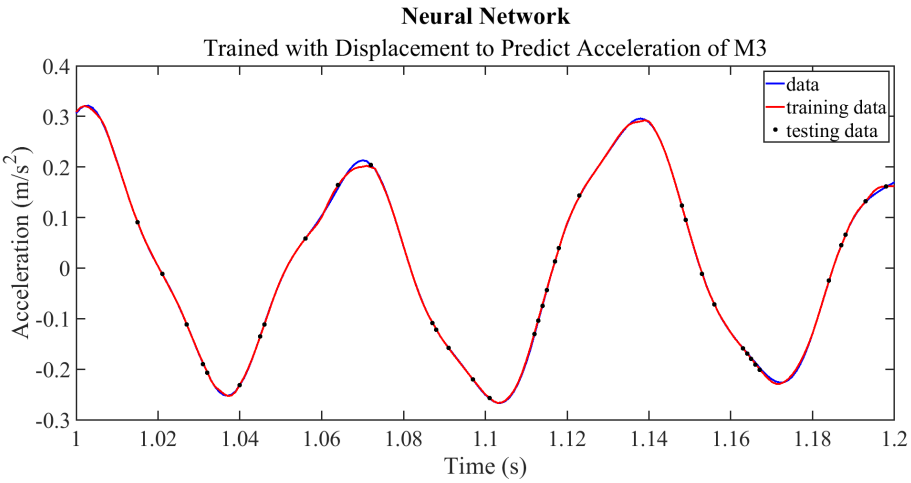

Figure 3.36 – Enlarged view of neural network model trained with displacement to predict acceleration of M3.

This model performed better when compared to the linear regression (Fig. 3.16), decision tree (Fig. 3.22), and bagging random forest (Fig. 3.31) models. The neural network model shows a few discrepancies compared to the boosting random forest model (Fig. 3.24) but has much less overfitting which may be more desirable in cases. More complex hidden layers may relieve the noise and overfitting in all other models.

4. Static Analysis of Channel Beam

Structural behavior in engineering is commonly modeled with beams due to simplicity and establishment in the field. Initial analysis of using machine learning stress predictions is conducted with channel beam geometry. Training data is collected as stress and deformation of the beam under various loading cases through Ansys simulations.

4.1 Problem Definition

Performance of machine learning to predict stress is further explored through analysis of a channel beam. The length of the beam is 1m, and all other dimensions of the cross section are seen in Fig. 4.1.

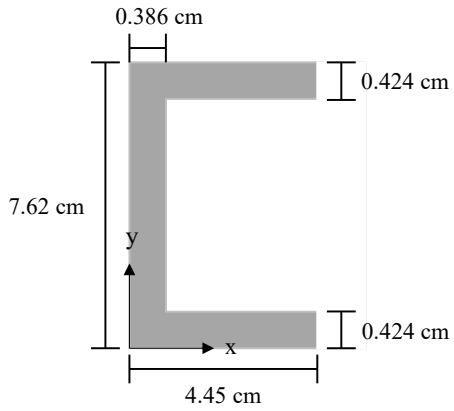

Figure 4.1 – Dimensions of channel beam.

The beam is fixed at z=0 m and will be analyzed through several cases with various loads as seen in the following figures.

Figure 4.2 – Case 1: 100 N point load in the -y direction at free end, constant along x axis.

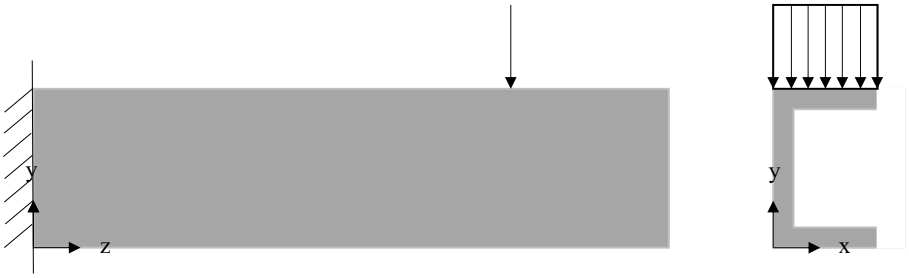

Figure 4.3 – Case 2: 100 N point load in the -y direction at three quarters length, constant along x axis.

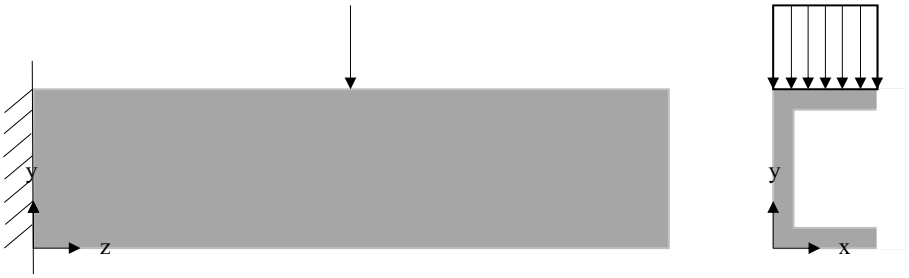

Figure 4.4 – Case 3: 100 N point load in the -y direction at half-length, constant along x axis.

Figure 4.5 – Case 4: 100 N point load in the -y direction at quarter length, constant along x axis.

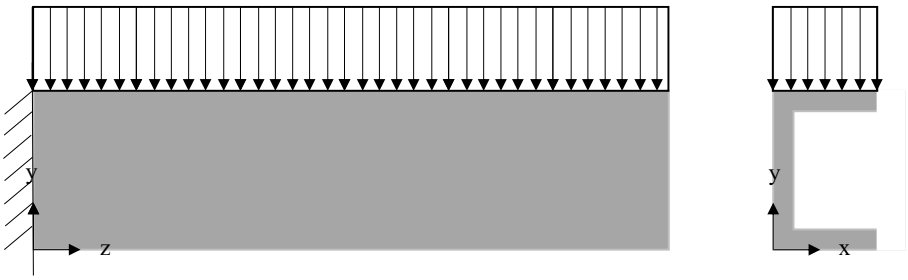

Figure 4.6 – Case 5: Constant distributed load in the -y direction of 100 N/m from fixed end to free end, constant along x axis.

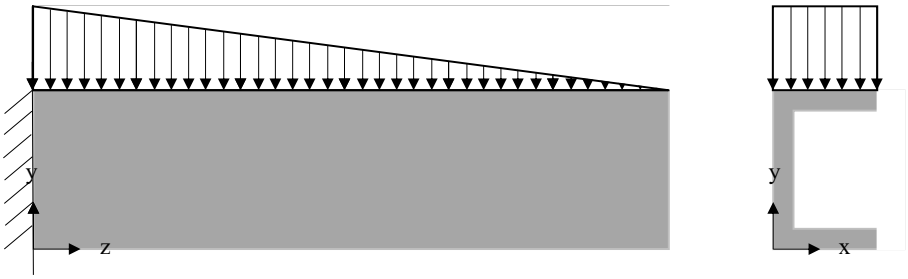

Figure 4.7 – Case 6: Linear pressure in the -y direction varying along z axis from 100 Pa at fixed end to 0 Pa at free end, constant along x axis.

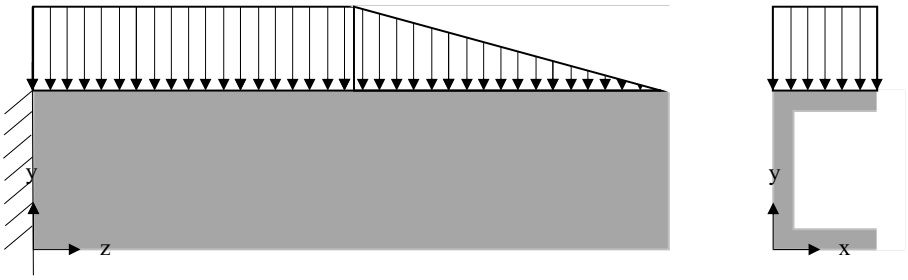

Figure 4.8 – Case 7: Constant pressure in the -y direction of 100 Pa from fixed end to half-length and linear pressure in the -y direction varying along z axis from 100 Pa at half-length to 0 Pa at free end, constant along x axis.

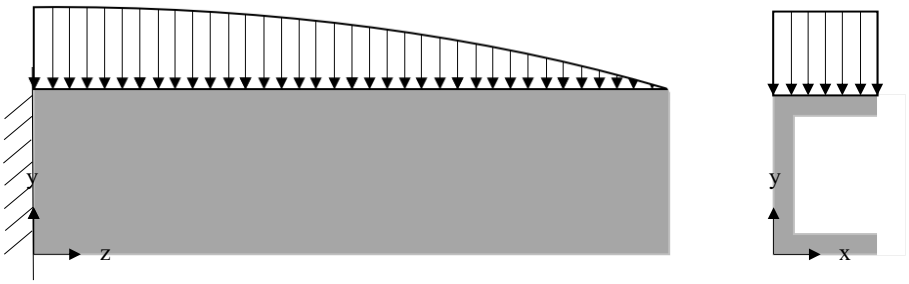

Figure 4. 9 – Case 8: Parabolic pressure varying along z axis of 100 Pa at fixed end to 0 Pa at free end, constant along x axis.

4.2 Mathematical Modeling

Simulations in Ansys are verified through analytical computations of stress. Stress is calculated with the maximum moment (M), distance from the neutral surface (y), and moment of inertia (I).

$$\sigma = -\frac{My}{I} \tag{4.1}$$

Moment of the inertia is calculated with respect to the centroidal axis that is perpendicular to the plane of the moment, in this case the x' axis. The centroid of the beam is seen in Fig. 4.2.

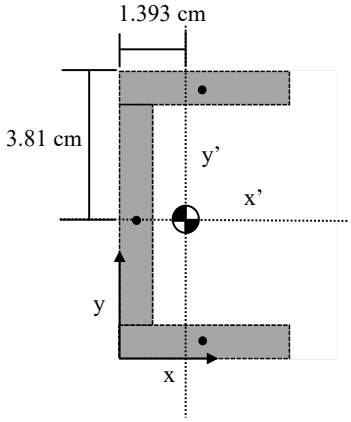

Figure 4.10 – Centroid and central axes of channel beam cross section.

Moment of inertia is calculated as the sum of moment of inertias of all three sections within the cross section. Eq. 4.2 is the equation for the moment of inertia of a rectangle along with the parallel axis theorem, where d is the perpendicular distance from the centroidal axis to the centroid of the section.

$$I_x = \frac{1}{12}bh^3 + Ad^2 \tag{4.2}$$

$$I_x = 2\left(\frac{1}{12}(4.45)(0.424)^3 + (1.886)(3.598)^2\right) + \frac{1}{12}(0.386)(6.772)^3 \tag{4.3}$$

$$I_x = 58.88\ cm^4 = 5.888 \times 10^{-7} m^4$$

For each case, the maximum stress is calculated with Eq. 1 where the distance to the neutral surface is 3.81 cm.

Table 4.1 – Maximum analytical stress for each case simulated in Ansys.

Load on Beam	Maximum Analytical Stress
Point load at full length	$6.480 \times 10^6\ Pa$
Point load at ¾ length	$4.860 \times 10^6\ Pa$
Point load at ½ length	$3.240 \times 10^6\ Pa$
Point load at ¼ length	$1.620 \times 10^6\ Pa$
Constant distributed load	$3.240 \times 10^6\ Pa$
Linear distributed pressure	$4.806 \times 10^4\ Pa$
Constant and linear distributed pressure	$8.410 \times 10^4\ Pa$
Parabolic distributed pressure	$7.205 \times 10^4\ Pa$

4.3 Ansys Simulations

The coordinate system was set at the outer bottom left corner, as shown in Fig. 4.10, with the length of the beam along the z axis. The top face was split into four sections for application of point loads along the beam. The top and bottom inner corners along the z axis were rounded with a curvature of 0.1 cm to avoid stress singularities during the meshing process.

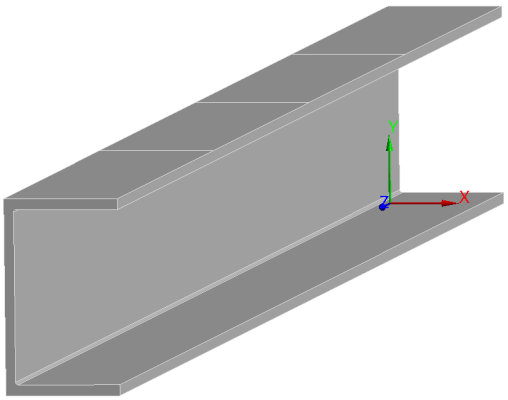

Figure 4.11 – Geometry of beam in Ansys with coordinate system.

The material was left at default as structural steel, properties provided in Table 4.2 below.

Table 4.2 – Material properties for structural steel from Ansys.

Density	$7850\ kg/m^3$
Young's Modulus	$2 \times 10^{11}\ Pa$
Thermal Conductivity	$60.5\ W/m \cdot ℃$
Specific Heat	$434\ J/kg \cdot ℃$
Tensile Yield Strength	$2.5 \times 10^8\ Pa$
Tensile Ultimate Strength	$4.6 \times 10^8\ Pa$

Each case is simulated in Ansys for deformation and stress analysis and the solution data is exported for machine learning in MATLAB. The training data for MATLAB consists of the x, y, and z coordinate of each node along with the respective deformation and stress at each node, as seen in Table 4.3. The deformation analyzed is directional deformation in the plane perpendicular to the axis on which the loading occurs on.

Commented [NG5]: Expand and explain databases used for matlab

Table 4.3 – Example of training data taken from Ansys for machine learning in MATLAB.

x Coordinate (m)	y Coordinate (m)	z Coordinate (m)	Deformation (m)	Stress (Pa)
0.0039819	0.0026226	0.057236	-2.6784e-06	7880500
…	…	…	…	…

Mesh convergence is conducted using the convergence tool for each case based on the maximum equivalent von Mises stress of the beam, with an allowable change of 1%. Final mesh produced from the convergence is compared to the analytical calculations of maximum stress to validate the refinement of the mesh. Analytically the maximum moment is located at the fixed support, therefore the analytical stresses will be compared to the stresses at the top of the beam at the fixed support in Ansys.

4.3.1 Point Load at Full Length

Table 4.4 – Convergence history for case with point load at full length.

Solution Number	Equivalent Stress (Pa)	Change (%)	Nodes	Elements
1	1.4606e+07		6333	2874
2	1.6056e+07	9.4554	12259	6233
3	1.9223e+07	17.957	43178	24524
4	2.5679e+07	29.89	97701	58949
5	2.9842e+07	13.843	216374	136252
6	3.2128e+07	7.3763	364411	234203
7	3.2027e+07	-0.31347	521234	340204

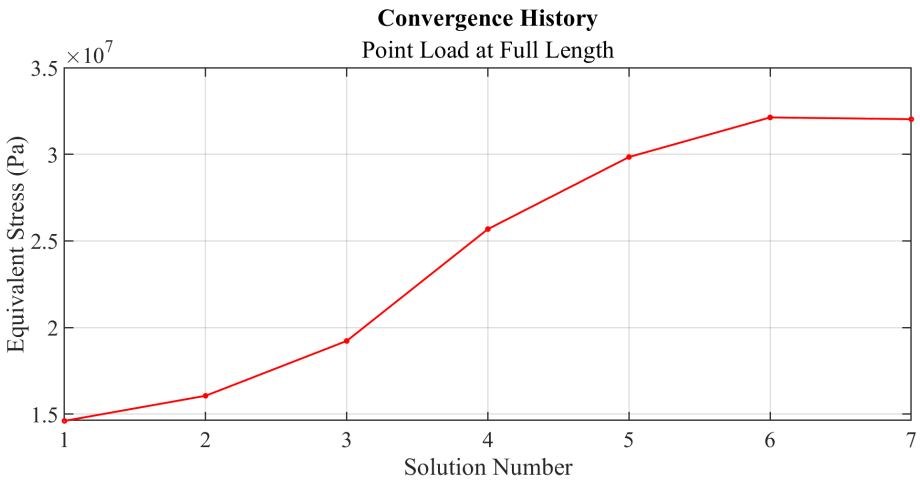

Figure 4.12 – Convergence history plot for beam with point load at full length.

The maximum analytical stress calculated for this case is about $6.480 \times 10^6\ Pa$, which is about a 8% increase to the value shown in the Ansys model below. Values within a 10% change will be considered acceptable.

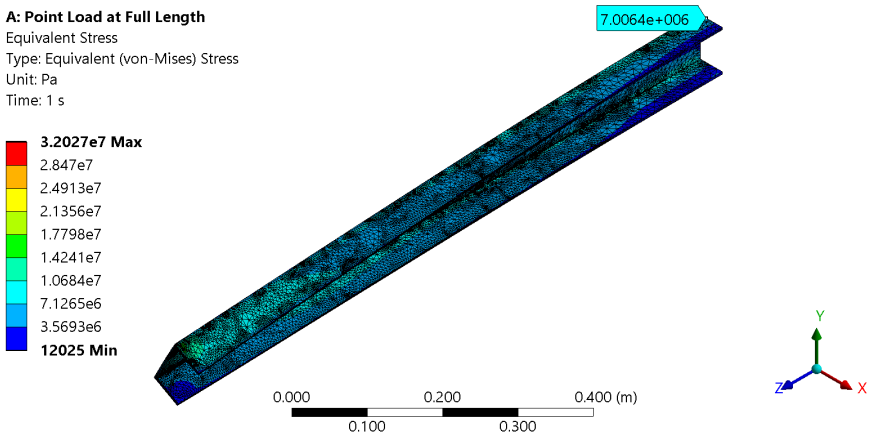

Figure 4.13 – Stress (Pa) at fixed support of beam with point load at full length in Ansys.

4.3.2 Point Load at ¾ Length

Table 4.5 – Convergence history for case with point load at ¾ length.

Solution Number	Equivalent Stress (Pa)	Change (%)	Nodes	Elements
1	1.0194e+07		6333	2874
2	1.0698e+07	4.822	32369	17721
3	1.2908e+07	18.727	105365	63532
4	1.5347e+07	17.265	148040	91425
5	1.9578e+07	24.225	293636	187569
6	2.6277e+07	29.218	442978	287304
7	3.3878e+07	25.272	862482	574720
8	4.4944 e+07	28.078	1537527	1044119
9	4.495 e+07	1.4455e-002	1642679	1117769

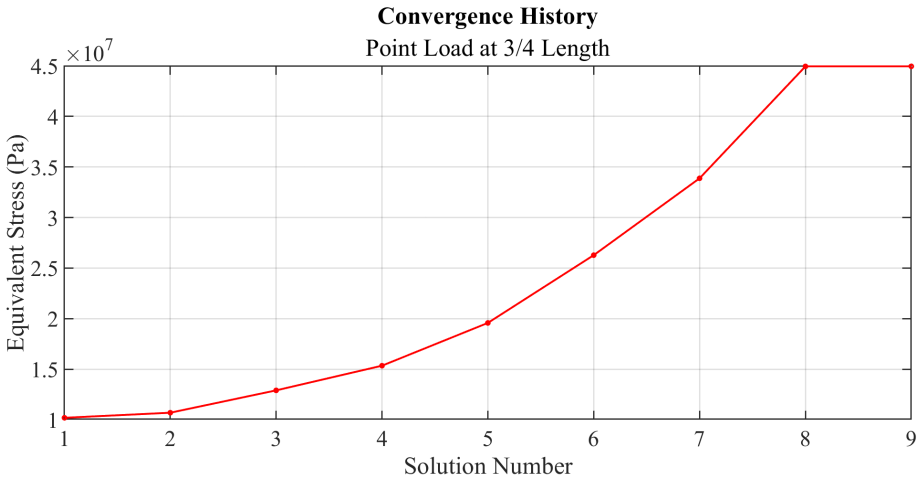

Figure 4.14 – Convergence history plot for beam with point load at ¾ length.

Maximum calculated stress is $4.860 \times 10^6\ Pa$, less than a 1% increase compared to the value from the Ansys simulation.

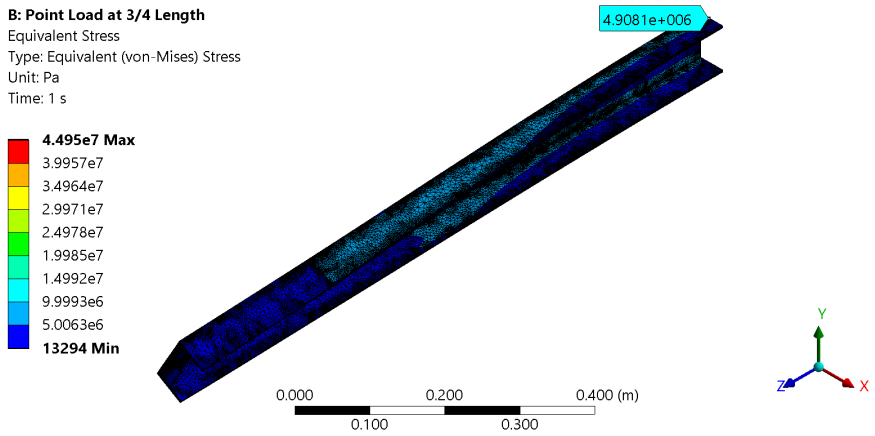

Figure 4.15 – Stress (Pa) at fixed support of beam with point load at ¾ length in Ansys.

4.3.3 Point Load at ½ Length

Table 4.6 – Convergence history for case with point load at ½ length.

Solution Number	Equivalent Stress (Pa)	Change (%)	Nodes	Elements
1	8.0161e+06		6333	2874

2	8.251e+06	3.2501	15372	8068
3	9.9848e+06	18.655	54661	31995
4	1.2981e+07	26.092	109242	67015
5	1.5368e+07	16.844	226712	144501
6	1.9562e+07	24.01	299049	192242
7	2.5042e+07	24.572	556050	366173
8	3.5341e+07	34.114	865798	578657
9	3.5359e+07	5.0679e-002	1193052	805813

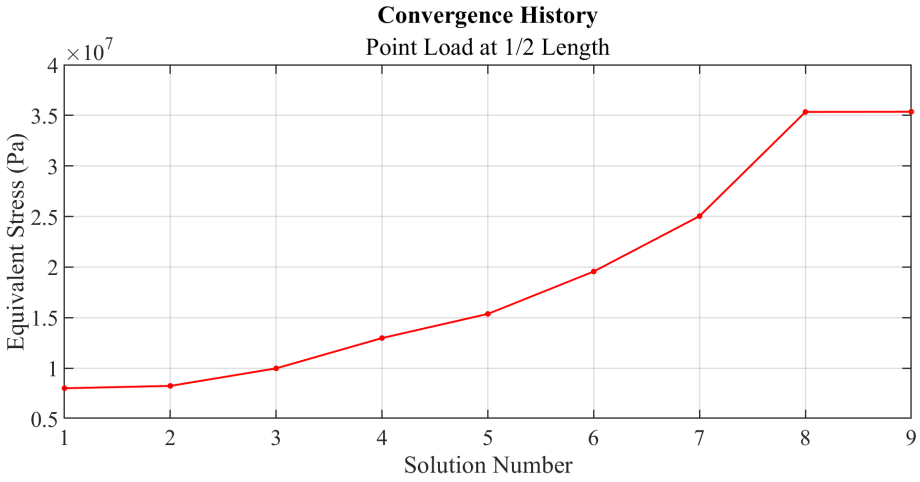

Figure 4.16 – Convergence history plot for beam with point load at ½ length.

The maximum analytical stress of $3.240 \times 10^6\ Pa$ for this case is within 1% of the value seen in the Ansys simulation below.

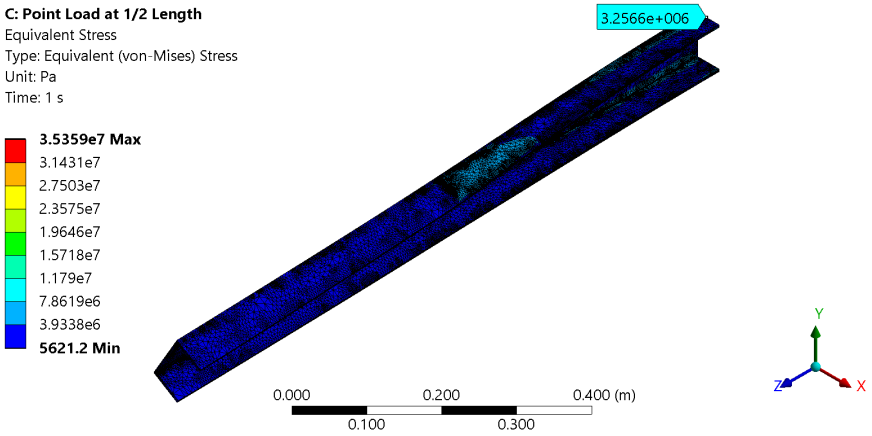

Figure 4.17 – Stress (Pa) at fixed support of beam with point load at ½ length in Ansys.

4.3.4 Point Load at ¼ Length

Table 4.7 – Convergence history for case with point load at ¼ length.

Solution Number	Equivalent Stress (Pa)	Change (%)	Nodes	Elements
1	6.359e+06		6333	2874
2	6.4602e+06	1.579	12207	6255
3	9.4802e+06	37.891	29239	16896
4	1.2041e+07	23.799	64733	39514
5	1.2954e+07	7.3023	153989	97698
6	1.3297e+07	2.6162	314370	205296
7	1.3172e+07	-0.94815	658551	441613

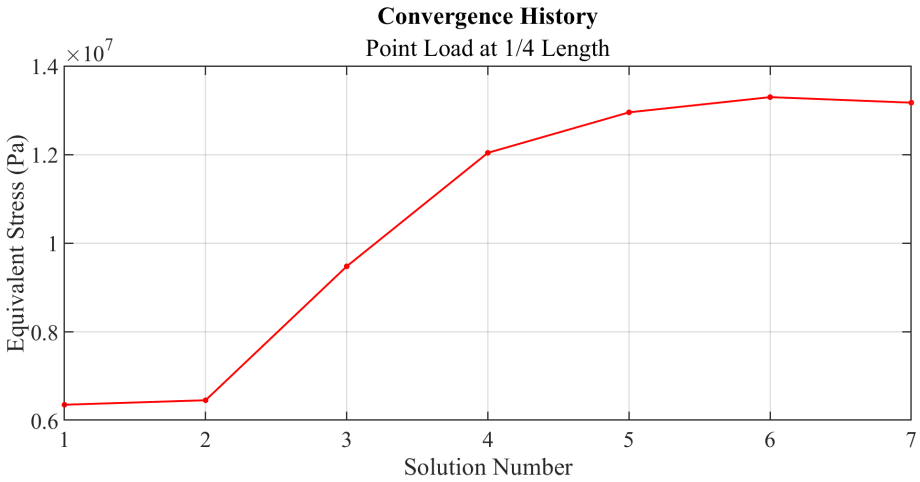

Figure 4.18 – Convergence history plot for beam with point load at ¼ length.

The stress from the Ansys simulation has about a 5% increase from the calculated analytical values of $1.620 \times 10^6\ Pa$.

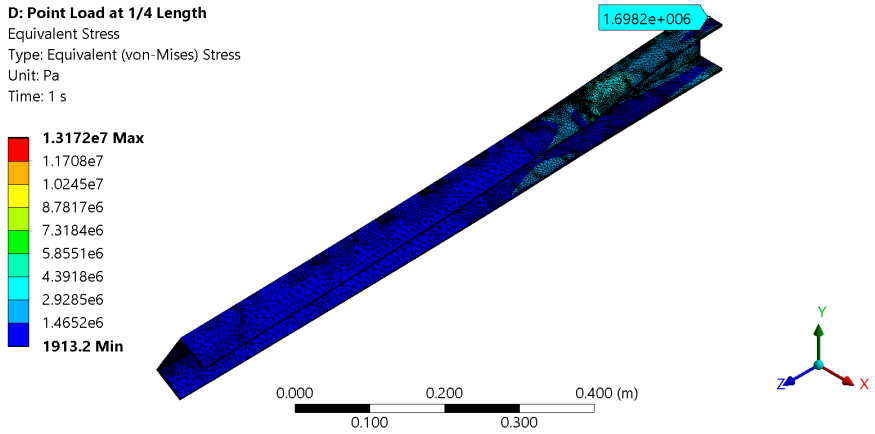

Figure 4.19 – Stress (Pa) at fixed support of beam with point load at ¼ length in Ansys.

4.3.5 Constant Distributed Load

Table 4.8 – Convergence history for case with constant distributed load.

Solution Number	Equivalent Stress (Pa)	Change (%)	Nodes	Elements
1	7.2348e+06		6333	2874

2	7.6097e+06	5.051	18709	9904
3	9.363e+06	20.66	52317	30019
4	1.1777e+07	22.84	143415	88419
5	1.4702e+07	22.091	267338	170176
6	1.8885e+07	24.91	424649	274344
7	2.6933e+07	35.128	761828	505071
8	2.6977e+07	0.16576	1429300	968170

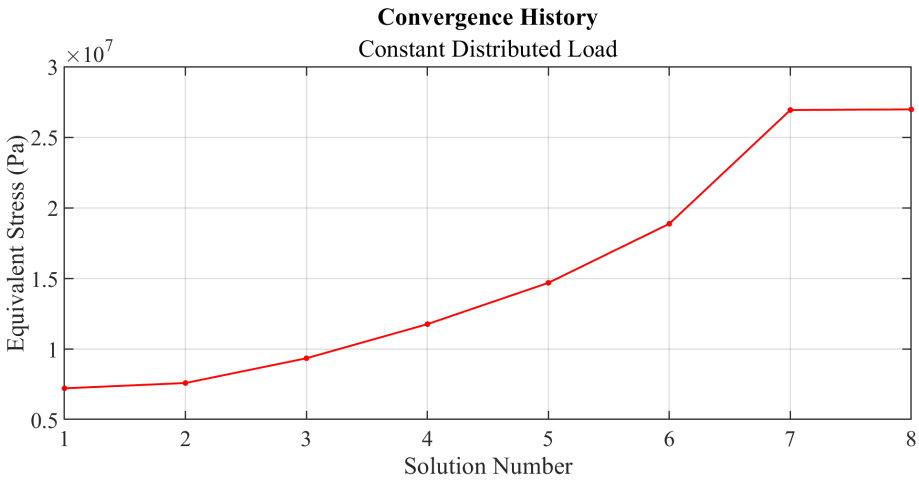

Figure 4.20 – Convergence history plot for beam with with constant distributed load.

The computed analytical stress of $3.240 \times 10^6\ Pa$ for this case has less than a 1% increase to the value observed in Ansys.

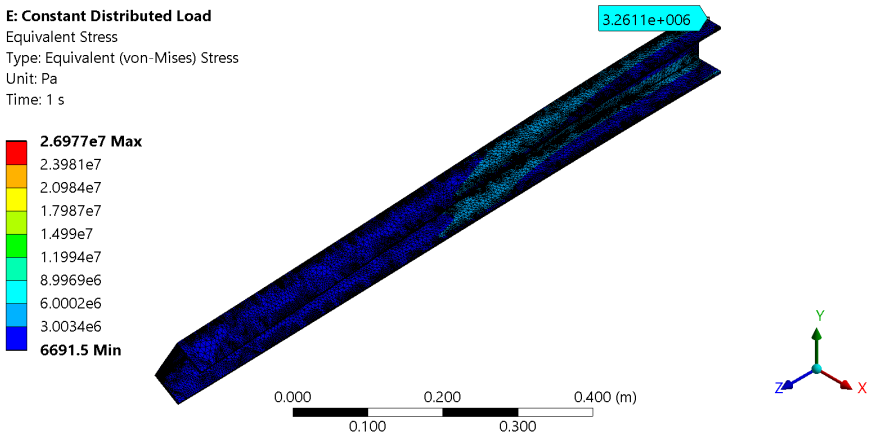

Figure 4.21 – Stress (Pa) at fixed support of beam with constant distributed load in Ansys.

4.3.6 Linear Distributed Pressure

Table 4.9 – Convergence history for case with linear distributed pressure.

Solution Number	Equivalent Stress (Pa)	Change (%)	Nodes	Elements
1	1.1699e+05		6333	2874
2	1.2392e+05	5.7558	9279	4522
3	1.616e+05	26.288	37755	21124
4	2.0608e+05	24.199	98029	58831
5	2.64e+05	24.639	192373	120319
6	3.3886e+05	24.835	318304	20307
7	4.4019e+05	26.015	670888	442683
8	4.4049e+05	6.796e-002	905717	603205

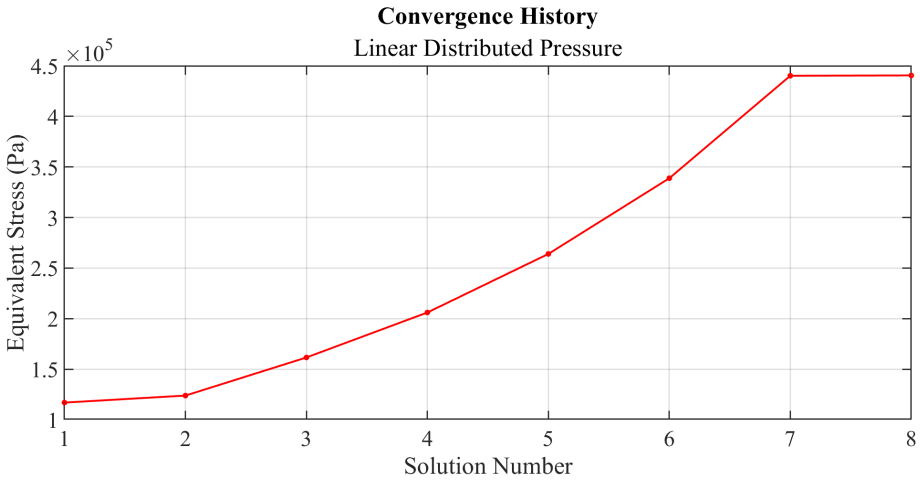

Figure 4.22 – Convergence history plot for beam with linear distributed pressure.

The observed stress in Ansys has less than a 2% increase from the analytical stress for this case of $4.806 \times 10^4\ Pa$.

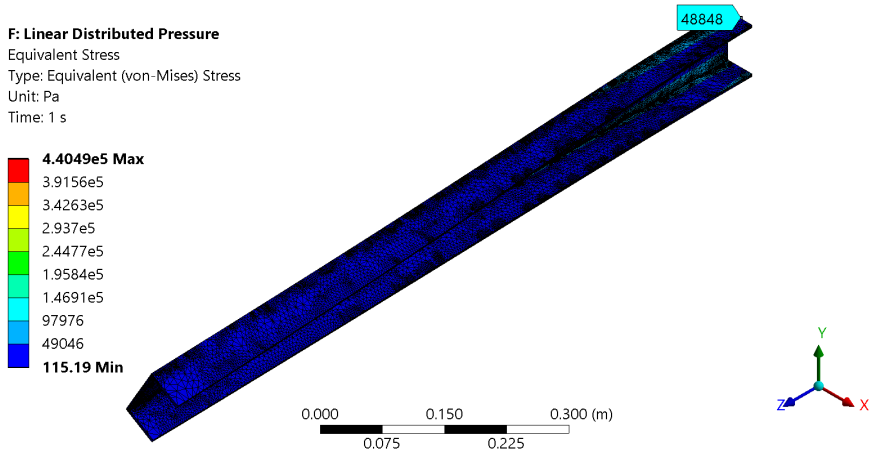

Figure 4.23 – Stress (Pa) at fixed support of beam with linear distributed pressure in Ansys.

4.3.7 Constant and Linear Distributed Pressure

Table 4.10 – Convergence history for case with constant and linear distributed pressure.

Solution Number	Equivalent Stress (Pa)	Change (%)	Nodes	Elements
1	2.0111e+05		6333	2874

2	2.1234e+05	5.4357	10762	5357
3	2.7192e+05	24.606	43254	24361
4	3.5369e+05	26.139	122080	74187
5	4.4466e+05	22.79	252942	160065
6	5.668e+05	24.151	438898	283482
7	7.3293e+05	25.564	607594	398201
8	9.5329e+05	26.137	1020141	681358
9	1.2357e+06	25.6	1723286	1172367
10	1.6034e+06	25.905	2831432	1956309
11	2.0778e+06	25.777	3432829	2382161
12	2.0778e+06	-7.92e-005	3441851	2388617

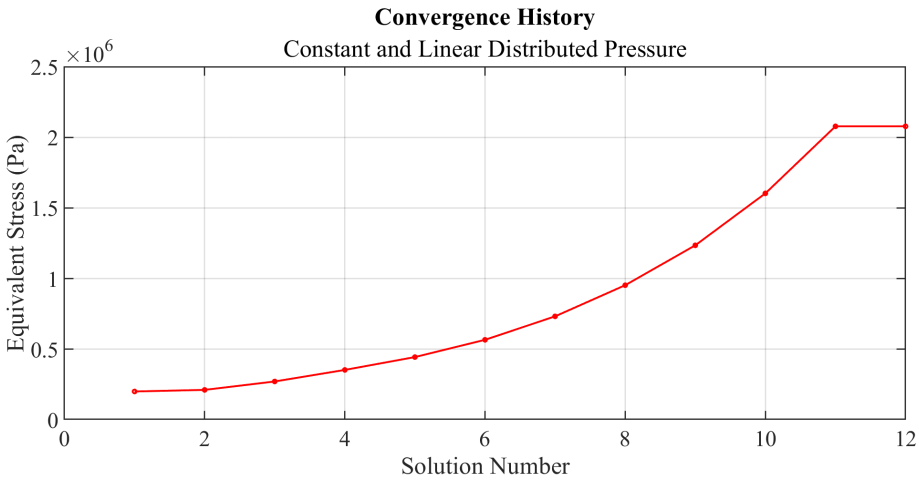

Figure 4.24 – Convergence history plot for beam with constant and linear distributed pressure.

The analytical stress of $8.410 \times 10^4\ Pa$ has less than 1% increase of the value overserved in the Ansys simulation.

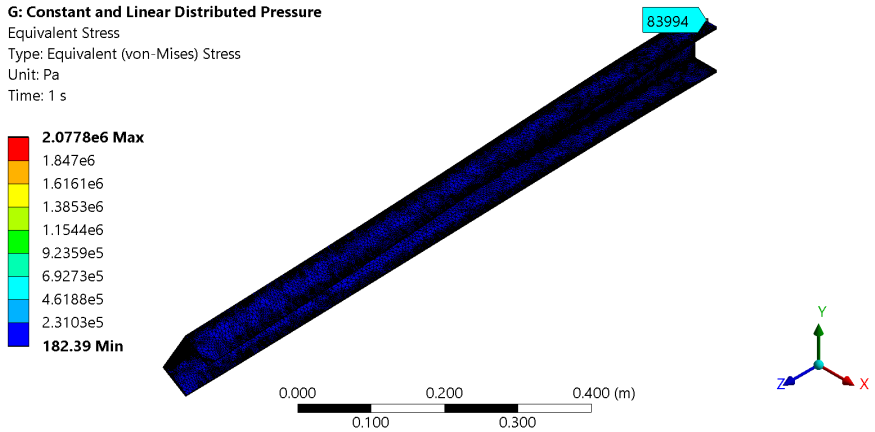

Figure 4.25 – Stress (Pa) at fixed support of beam with constant and linear distributed pressure in Ansys.

4.3.8 Parabolic Distributed Pressure

Table 4.11 – Convergence history for case with parabolic distributed pressure.

Solution Number	Equivalent Stress (Pa)	Change (%)	Nodes	Elements
1	1.7257e+05		6333	2874
2	1.8236e+05	5.5161	10762	5357
3	2.345e+05	25.015	43785	24679
4	3.0651e+05	26.619	121374	73724
5	3.8939e+05	23.82	237627	149771
6	4.9354e+05	23.593	450737	290928
7	6.4e+05	25.841	858779	570575
8	6.4029e+05	4.5861e-002	1098320	734479

Commented [NG6]: Redo plot

Load -> pressure

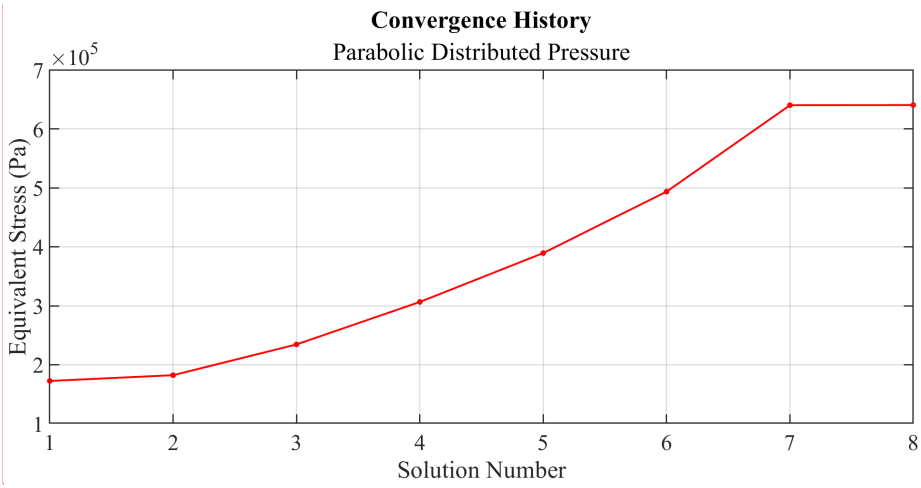

Figure 4.26 – Convergence history plot for beam with parabolic distributed pressure.

The analytical stress for this case of $7.205 \times 10^4\ Pa$ compared to the value observed in Ansys is about a 1% increase.

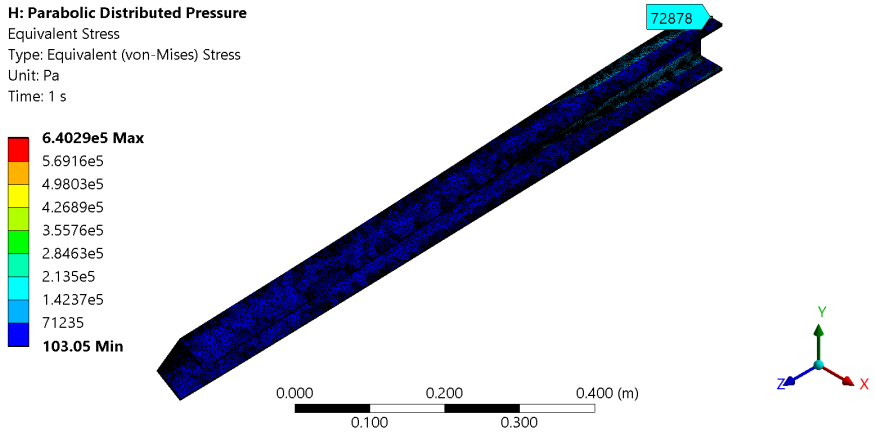

Figure 4.27 – Stress (Pa) at fixed support of beam with parabolic distributed pressure in Ansys.

4.2 Linear Regression Modeling

4.2.1 Training Model

The linear regression model was trained in MATLAB, as discussed in the previous chapter, using the “fitlm” function.

Models are trained with directional deformation and the respective node locations to predict equivalent stress. The performance of the model is plotted along each axis from the coordinate system. Due to the small dimensions of the cross-section geometry, the x and y axis plots have significantly less data than the z axis plots. The main plots used for analysis are for the stress along z axes.

4.2.2 Results

The R^2 values are calculated for each model and the model prediction is plotted against the Ansys data to evaluate the model's performance for each case.

Table 4.12 – All cases simulated in Ansys for a beam geometry using linear regression models, along with their respective R^2 values.

Load on Beam	Trained R^2	Tested R^2
Point load at full length	0.125633	0.126600
Point load at ¾ length	0.474600	0.474918
Point load at ½ length	0.463041	0.462181
Point load at ¼ length	0.247969	0.248528
Constant distributed load	0.519044	0.518328
Linear pressure	0.566604	0.568387
Constant and linear pressure	0.530843	0.534847
Parabolic pressure	0.561950	0.562508

Table 4.13 – All cases simulated in Ansys for a beam geometry using linear regression models, along with their respective R^2 values, sorted by ascending trained R^2 values.

Load on Beam	Trained R^2	Tested R^2
Point load at full length	0.125633	0.126600
Point load at ¼ length	0.247969	0.248528
Point load at ½ length	0.463041	0.462181
Point load at ¾ length	0.474600	0.474918
Constant distributed load	0.519044	0.518328
Constant and linear pressure	0.530843	0.534847
Parabolic pressure	0.561950	0.562508
Linear pressure	0.566604	0.568387

Based on the previously established high performance threshold of a R^2 value of 0.6, the linear regression model does not show high performance for the prediction of stress for a beam geometry. This is expected based on the observations of the performance of linear regression models from the previous chapter. The point load cases show the lowest performance with low R^2 values, while the more complex distributed loads and pressure have higher performance.

The lowest performing case with a point load at the full length fails to predict the stress. Higher data values are underestimated through the model prediction resulting in the low performance. Fig. 4.28 shows some accuracy in predicting values of between 0.4 and 0.6, which

corresponds with the lower residuals for those values in Fig. 4.29 as compared to the outer residuals. The model does not overfit as the test data coincides with the training data.

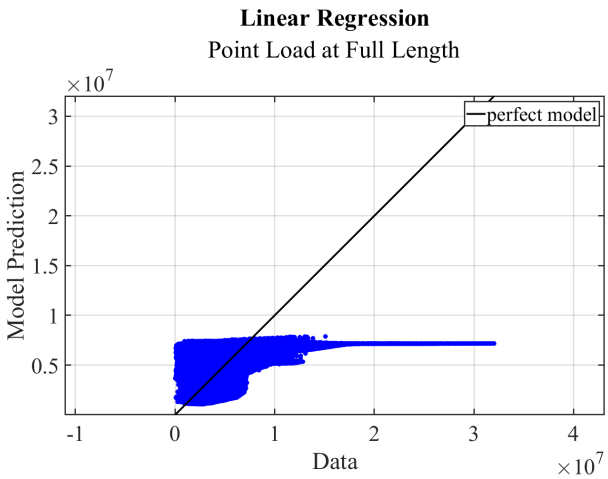

Figure 4.28 – Training data vs model prediction for linear regression model trained with directional deformation to predict equivalent stress for beam with point load at full length.

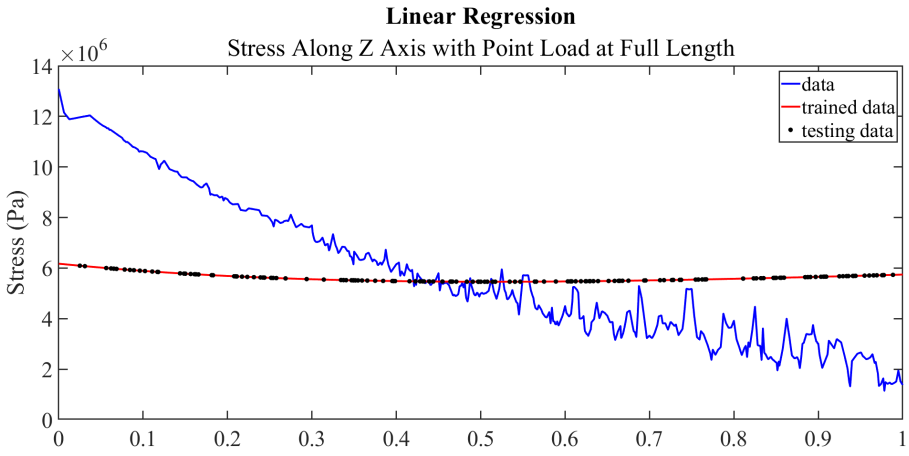

Figure 4.29 – Linear regression model trained with directional deformation to predict equivalent stress for beam with point load at full length.

Linear pressure loading shares similar difficulty in predicting stress as the point load at full length, though having the highest R^2 value of all loading cases. Fig. 4.30 exhibits

comparable behavior as seen in Fig. 4.28 in the underestimation of higher value data points. Outliers in this plot are much scarcer and the model follows the general trend of the stress.

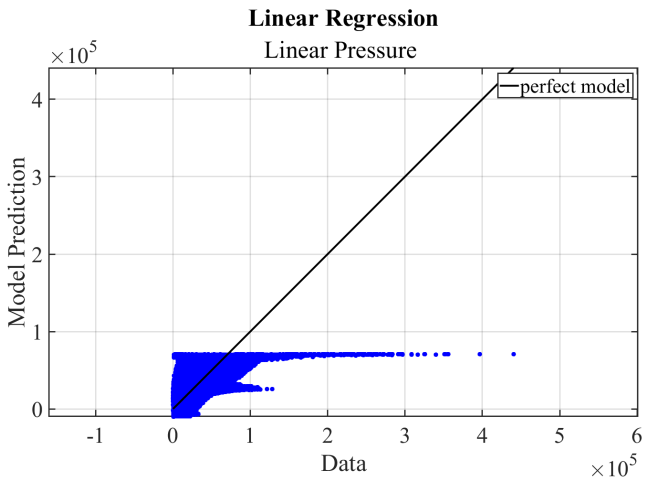

Figure 4. 30– Training data vs model prediction for linear regression model trained with directional deformation to predict equivalent stress for beam with linear pressure.

As seen in the worst performing loading case, the linear regression model in general has limited ability to follow the complexity of these cases due to its linear nature. Prediction for the linear pressure loading displays lower residuals throughout the entirety of the plot as opposed to the high residuals at the outer ends of the plot seen in the lowest performing case. The prediction tracks the negative slope of the stress but lacks the ability to predict more than that.

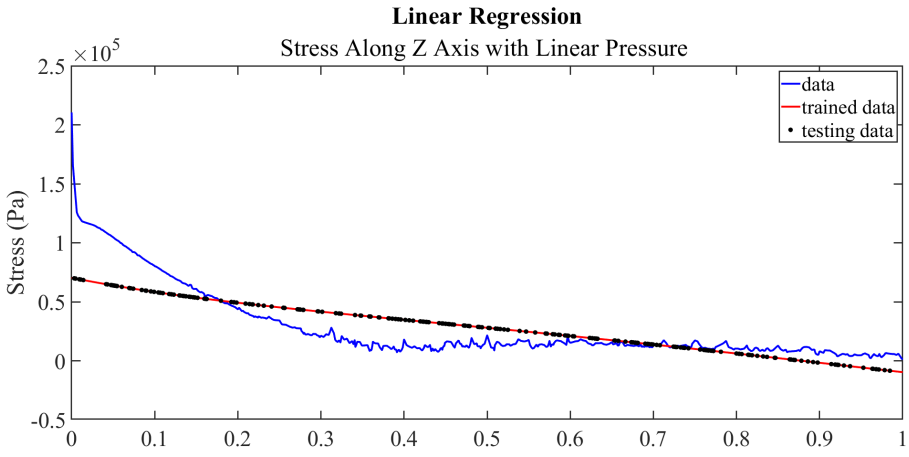

Figure 4.31 – Linear regression model trained with directional deformation to predict equivalent stress for beam with constant and linear pressure.

Stress in the beam exhibits more noise and the linear regression model is unable to properly predict the stress. The complexity of the problem is not well represented in these models as they tend to underfit the data. It should be noted that all linear regression models, even those that show low performance, do not overfit as the testing data aligns with the trained data.

4.3 Decision Tree Modeling

4.3.1 Training Model

The decision tree model was trained in MATLAB using the "fitrtree" function. All specifications of the function are discussed in the previous chapter.

4.3.2 Results

All decision tree models have a R^2 value over 0.99, therefore all models are of high performance.

Table 4.14– All cases simulated in Ansys for a beam geometry using decision tree models, along with their respective R^2 values.

Load on Beam	Trained R^2	Tested R^2
Point load at full length	0.996104	0.993641
Point load at ¾ length	0.994382	0.992902
Point load at ½ length	0.994931	0.993112
Point load at ¼ length	0.997460	0.996582
Constant distributed load	0.995256	0.993770
Linear pressure	0.995905	0.994308

Constant and linear pressure	0.995597	0.994776
Parabolic pressure	0.995748	0.994190

Table 4.15 – All cases simulated in Ansys for a beam geometry using decision tree models, along with their respective R^2 values, sorted by ascending trained R^2 values.

Load on Beam	Trained R^2	Tested R^2
Point load at ¾ length	0.994382	0.992902
Point load at ½ length	0.994931	0.993112
Constant distributed load	0.995256	0.993770
Constant and linear pressure	0.995597	0.994776
Parabolic pressure	0.995748	0.994190
Linear pressure	0.995905	0.994308
Point load at full length	0.996104	0.993641
Point load at ¼ length	0.997460	0.996582

Compared to the linear regression models, the order of lowest to highest performing models varies. Correlation between the data and model prediction seen in Fig. 4.32 for the highest performing case with a point load at quarter length. The data set for this geometry is much larger that from the previous section therefore the plot below has more outliers for such a high R^2 value. The distribution of the data points along the perfect model is constant and shows no correlation to data value.

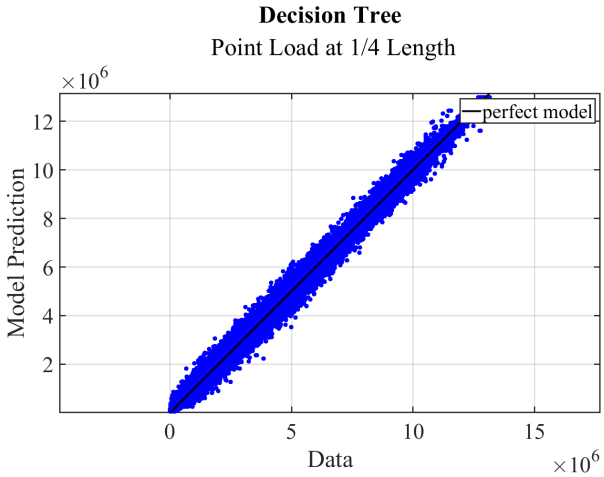

Figure 4.32 – Training data vs model prediction for decision tree model trained with directional deformation to predict equivalent stress for beam with a point load at ¼ length.

As expected, with the high R^2 values for the decision tree models, the model can predict the data with significantly higher accuracy than linear regression models. Though, in a magnified

view of the plot in Fig. 4.34, the piecewise nature of the model is seen to be unsuitable for the scattered data. Overfitting is not seen in the model as the testing data falls in line with the trained data.

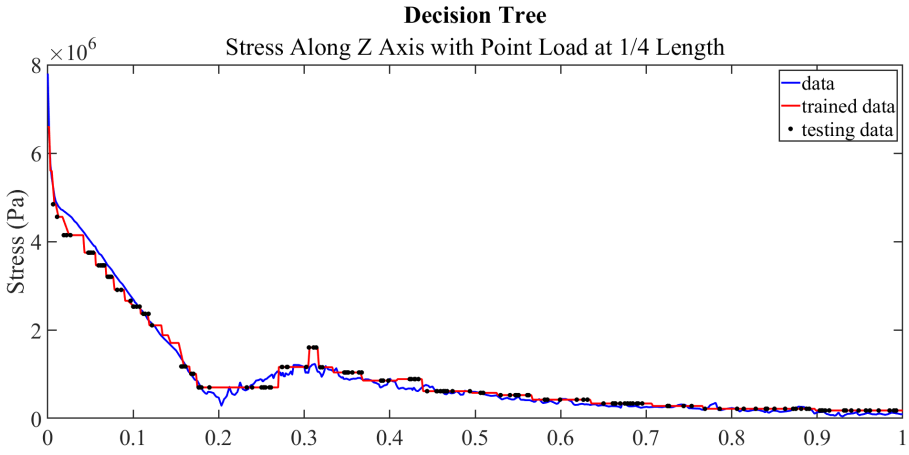

Figure 4.33 – Decision tree model trained with directional deformation to predict equivalent stress for beam with a point load at ¼ length.

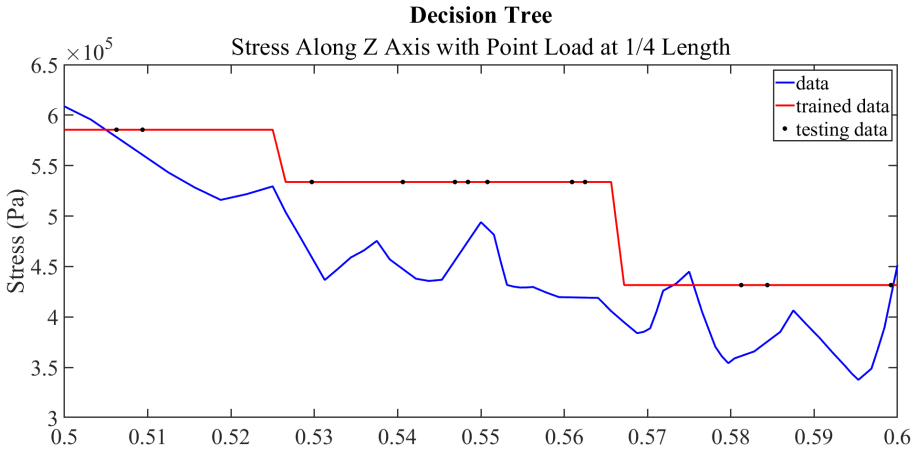

Figure 4.34 – Enlarged view of decision tree model trained with directional deformation to predict equivalent stress for beam with a point load at ¼ length.

The case with the lowest performance has a high R^2 value of 0.994, which is seen through the adequate correlation between data and prediction seen in Fig. 4.35 below. As with the previously discussed case, the inaccuracy increases as the data values increases.

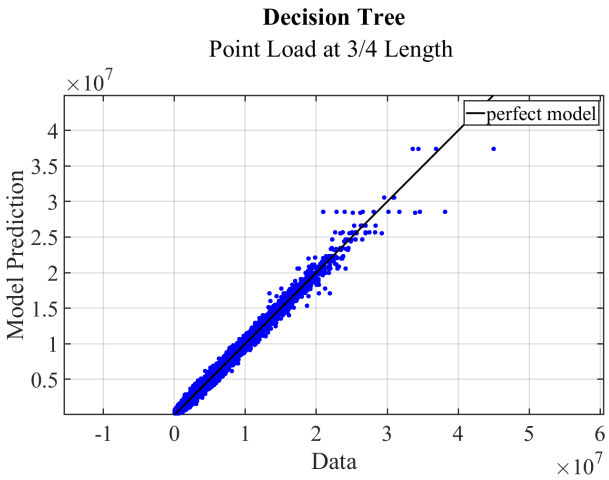

Figure 4.35 – Training data vs model prediction for decision tree model trained with directional deformation to predict equivalent stress for beam with point load at ¾ length.

The lower performance for this case, compared to others, is likely due to the higher noise in the data. A closer view shows some overfitting in the regions where the models prediction branches, but all other testing data follows the model.

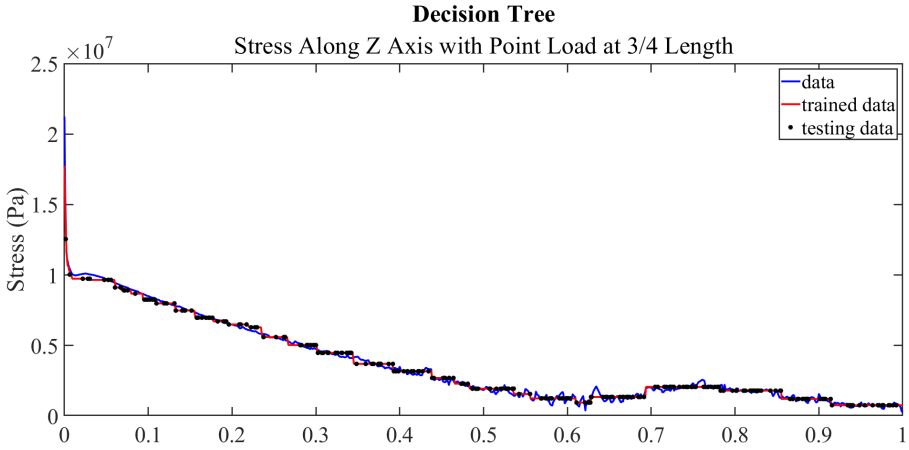

Figure 4.36 – Decision tree model trained with directional deformation to predict equivalent stress for beam with point load at ¾ length.

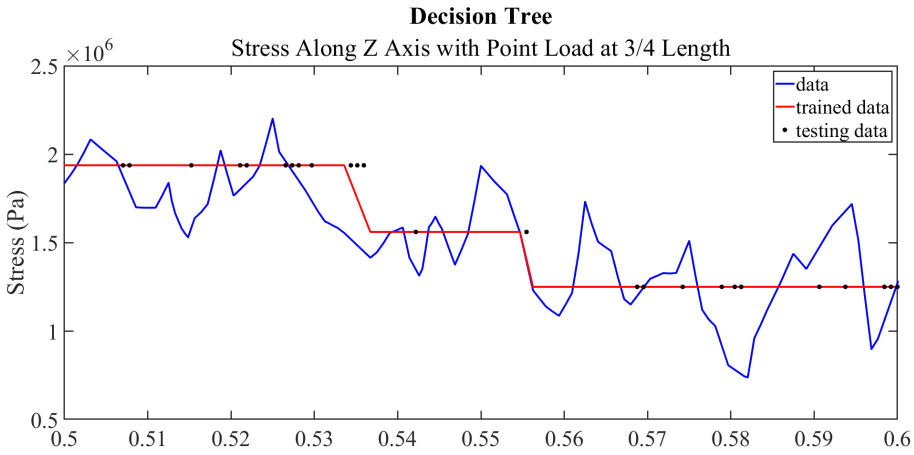

Figure 4.37 – Enlarged view of decision tree model trained with directional deformation to predict equivalent stress for beam with point load at ¾ length.

While the decision tree models produce higher R^2 values, the binary nature of the model does not fit best with the data used for training as it is not able to predict the intricacy of the data.

4.4 Random Forest Modeling

4.4.1 Training Model

The random forest models are trained in MATLAB using the "fitrensemble" function. Due to sizable computational time and limited computational power, optimization was conducted only on the first case. The model is optimized using the automatic "OptimizeHyperparameters" option for the case with a point load at full length. The resulting optimized ensemble used the bag method with 493 trees.

4.4.2 Results

Random forest models perform significantly well comparable to the decision tree models, as expected based on the results from the previous chapter. The highest and lowest performing models are identical to those in the decision tree models.

Table 4.16 – All cases simulated in Ansys for a beam geometry using random forest models, along with their respective R^2 values.

Load on Beam	Trained R^2	Tested R^2
Point load at full length	0.998012	0.996754
Point load at ¾ length	0.997757	0.996963

Point load at ½ length	0.998049	0.997052
Point load at ¼ length	0.999201	0.998882
Constant distributed load	0.997891	0.997053
Linear pressure	0.998092	0.997291
Constant and linear pressure	0.998128	0.997762
Parabolic pressure	0.998077	0.997234

Table 4.17 – All cases simulated in Ansys for a beam geometry using random forest models, along with their respective R^2 values, sorted by ascending trained R^2 values.

Load on Beam	Trained R^2	Tested R^2
Point load at ¾ length	0.997757	0.996963
Constant distributed load	0.997891	0.997053
Point load at full length	0.998012	0.996754
Point load at ½ length	0.998049	0.997052
Parabolic pressure	0.998077	0.997234
Linear pressure	0.998092	0.997291
Constant and linear pressure	0.998128	0.997762
Point load at ¼ length	0.999201	0.998882

As with decision tree models, the lowest performing models show sufficient correlation between data and prediction. Major outliers in the plot below hold less significance than those seen in the previous chapter where the data size was much smaller. This model loses accuracy as the data value increases.

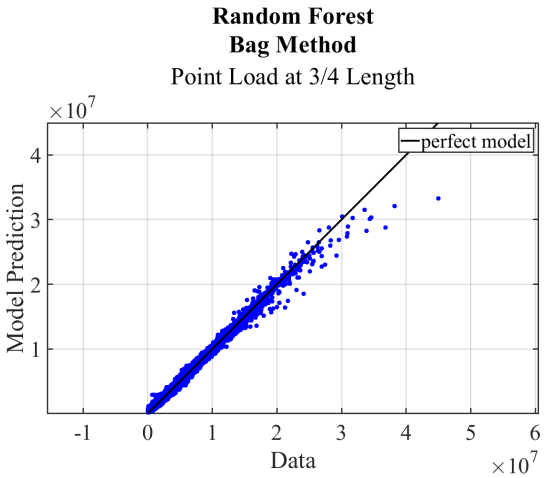

Figure 4.38 – Training data vs model prediction for random forest model trained with directional deformation to predict equivalent stress for beam with point load at ¾ length.

Of all machine learning models for the static beam, the highest R^2 value was found for the random forest model for a beam with a constant and linear distributed pressure. The model prediction falls closely along the perfect model line with a standard deviation, seen in Fig. 4.39.

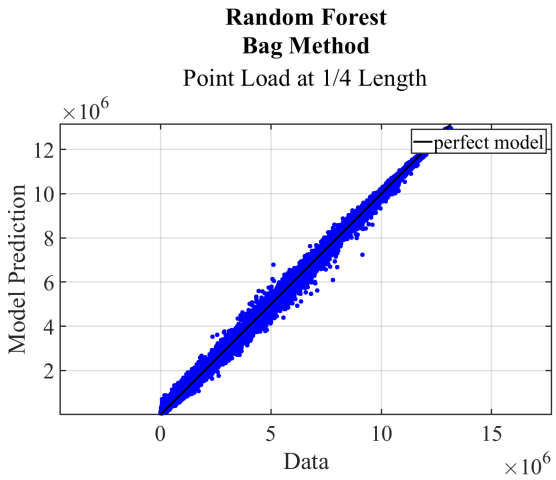

Figure 4.39 – Training data vs model prediction for random forest model trained with directional deformation to predict equivalent stress for beam with a point load at ¼ length.

Compared to previous machine learning models, random forest is most suitable for the data set used for training. Additional optimization, as in the previous chapter, was not necessary as the initial optimization for this geometry did not overfit the data. Inaccuracies are seen in the first half in the plot, but the model follows the data more competently than previous models. Closer view of the plot shows the model does not overfit and the error in the model prediction.

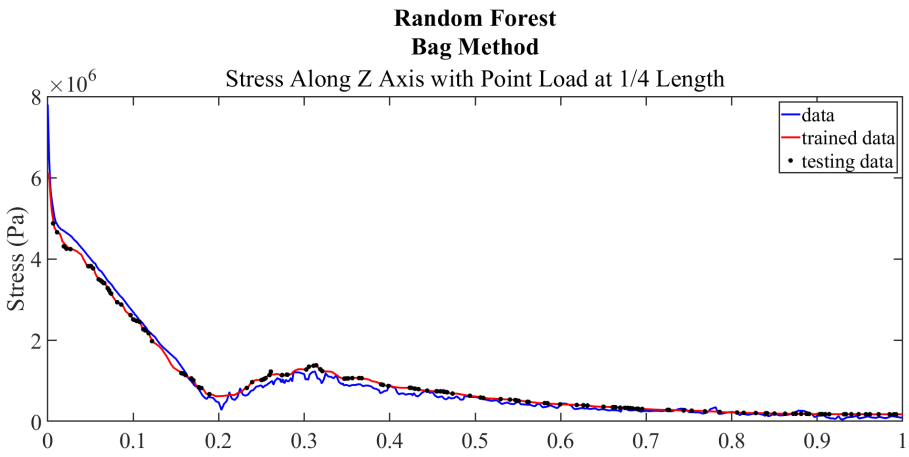

Figure 4.40 – Decision tree model trained with directional deformation to predict equivalent stress for beam with a point load at ¼ length.

Commented [NG7]: Typo in figure

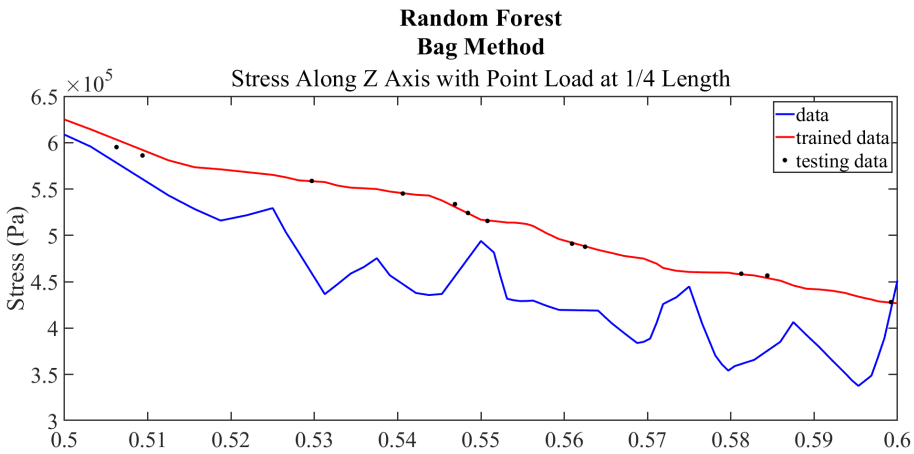

Figure 4.41 – Enlarged view of decision tree model trained with directional deformation to predict equivalent stress for beam with point load at ¼ length.

Commented [NG8]: Typo in figure

4.5 Neural Network Modeling

4.5.1 Training Model

The neural network model was trained in MATLAB using the “fitnet” and “train” functions. Due to limited computational power, the hidden layer size was constrained to two

layers with between 0-50 neurons and optimization was conducted only on the first case. The optimal hidden layer size was determined to be 50 neurons in the first layer and the other with 12. The "train" function trains the model until the maximum number of epochs or the performance goal is met [30].

4.5.2 Results

The performance of the neural network model more closely represents a normal distribution among the cases when compared to previous models. Neural network modeling produced the lowest performing model out of all previous cases, while the highest performing case was no more accurate than any of the random forest models.

Table 4.18 – All cases simulated in Ansys for a beam geometry using neural network models, along with their respective R^2 values.

Load on Beam	Trained R^2	Tested R^2
Point load at full length	0.074201	0.075263
Point load at ¾ length	0.659651	0.658877
Point load at ½ length	0.676182	0.675880
Point load at ¼ length	0.092502	0.094163
Constant distributed load	0.594866	0.594193
Linear pressure	0.978398	0.978428
Constant and linear pressure	0.891488	0.894147
Parabolic pressure	0.996953	0.996810

Table 4.19 – All cases simulated in Ansys for a beam geometry using neural network models, along with their respective R^2 values, sorted by ascending trained R^2 values.

Load on Beam	Trained R^2	Tested R^2
Point load at full length	0.074201	0.075263
Point load at ¼ length	0.092502	0.094163
Constant distributed load	0.594866	0.594193
Point load at ¾ length	0.659651	0.658877
Point load at ½ length	0.676182	0.675880
Constant and linear pressure	0.891488	0.894147
Linear pressure	0.978398	0.978428
Parabolic pressure	0.996953	0.996810

No clear pattern is apparent for the performance of the cases, with majority being considered high performance. Two models for point load cases produced the lowest R^2 values, less than 0.01, out of all models for this geometry. Fig. 4.42 shows zero correlation in the distribution of the model prediction and data as smaller values are mostly overestimated and larger values are underestimated. The model is not able to create any relation to the data as seen in Figure 4.43 but does share a general negative slope with the data. As previously seen, the testing data falls well along the neural network model and does not overfit.

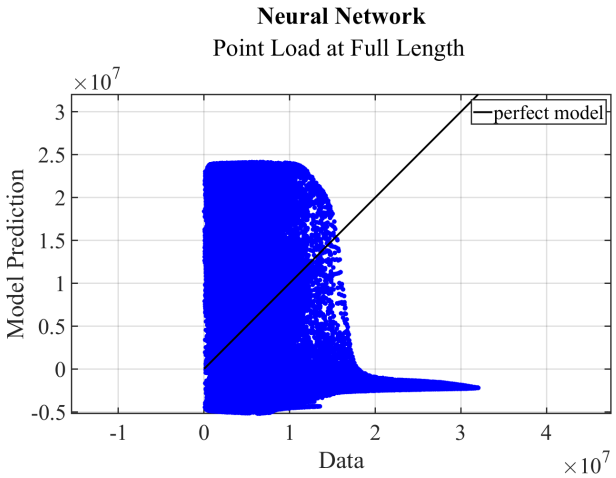

Figure 4.42 – Training data vs model prediction for neural network model trained with directional deformation to predict equivalent stress for beam with point load at full length.

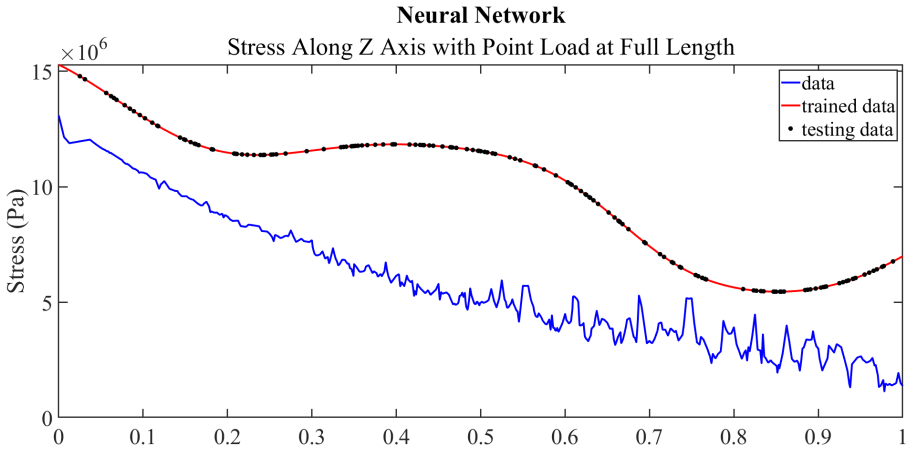

Figure 4.43 – Neural network model trained with directional deformation to predict equivalent stress for beam with point load at full length.

The second lowest performing case with a point load at quarter length shared similar characteristics in the relationship between the training data and model prediction. Smaller data values are seen to be overpredicted, while the larger data values are underpredicted as seen in Fig. 4.44 below. The prediction, as in the previous case does not follow the data in any capacity and is therefore inadequate.

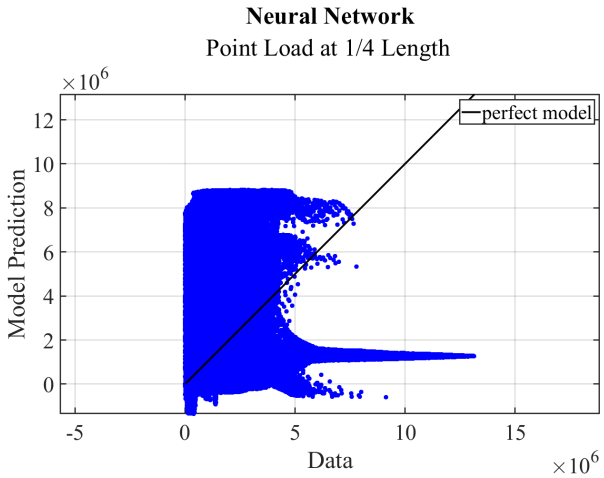

Figure 4.44 – Training data vs model prediction for neural network model trained with directional deformation to predict equivalent stress for beam with point load at ¼ length.

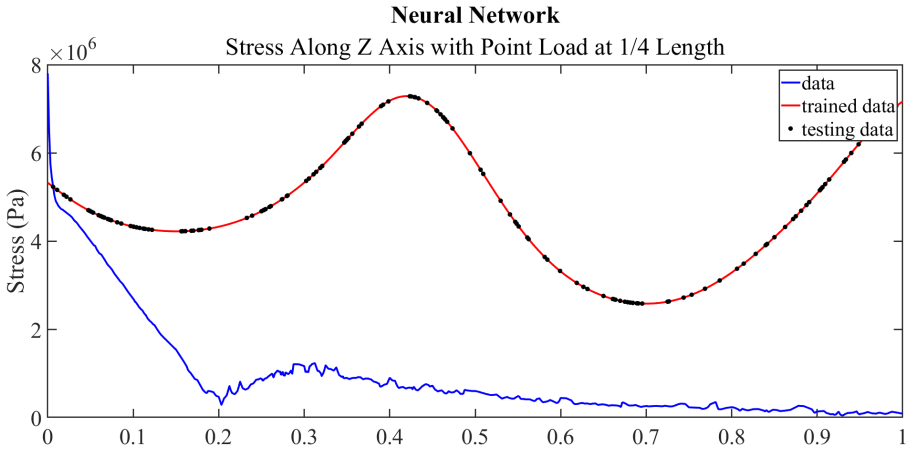

Figure 4.45 – Neural network model trained with directional deformation to predict equivalent stress for beam with point load at ¼ length.

Neural network best predicted stress from a parabolic pressure, though with less accuracy than any of the random forest models. The relationship between training data and model prediction shows direct correlation between increase of data values and increase in prediction error. The model tracks the training data and does not overfit, as seen in the enlarged view of the plot in Fig. 4.48 where the model fits well between the peaks in the data training data.

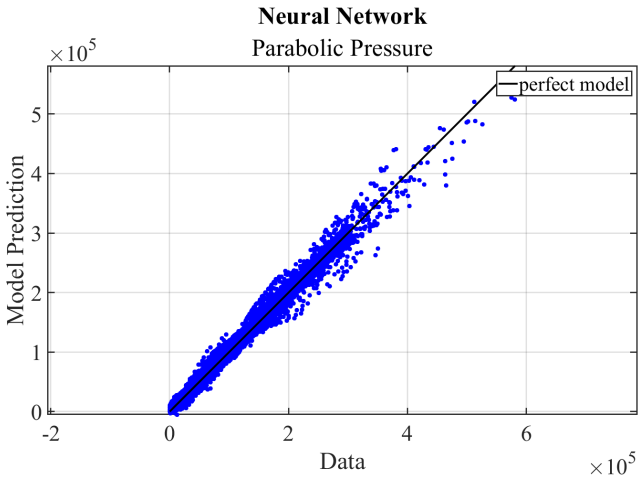

Figure 4.46 – Training data vs model prediction for neural network model trained with directional deformation to predict equivalent stress for beam with parabolic pressure.

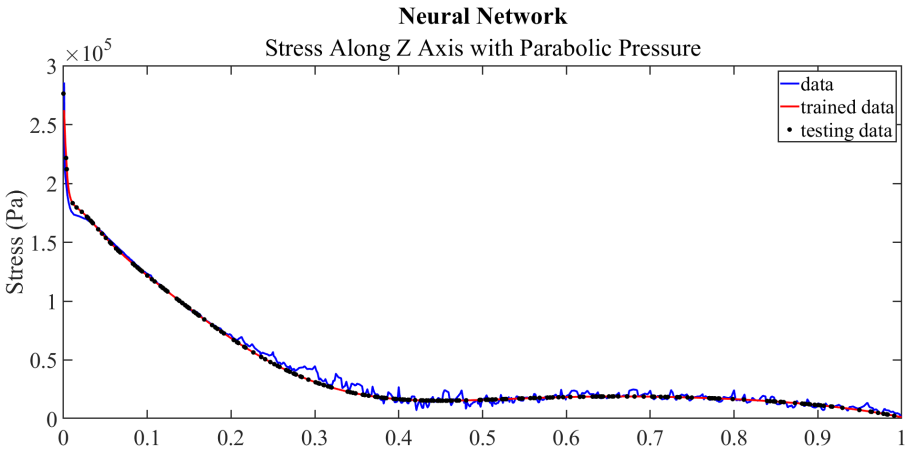

Figure 4.47 – Neural network model trained with directional deformation to predict equivalent stress for beam with parabolic pressure.

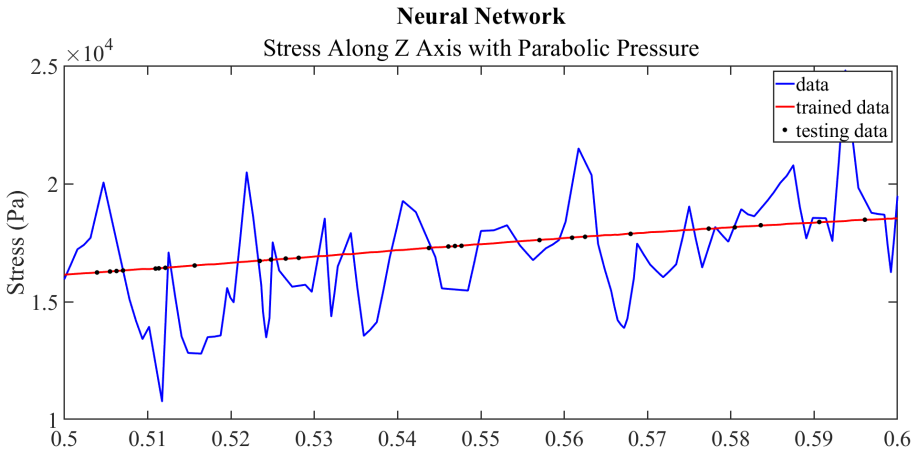

Figure 4.48 – Enlarged view of random forest model trained with directional deformation to predict equivalent stress for beam with parabolic pressure.

The model for the second-best performing case with linear pressure is less accurate but mimics the trend of the training data.

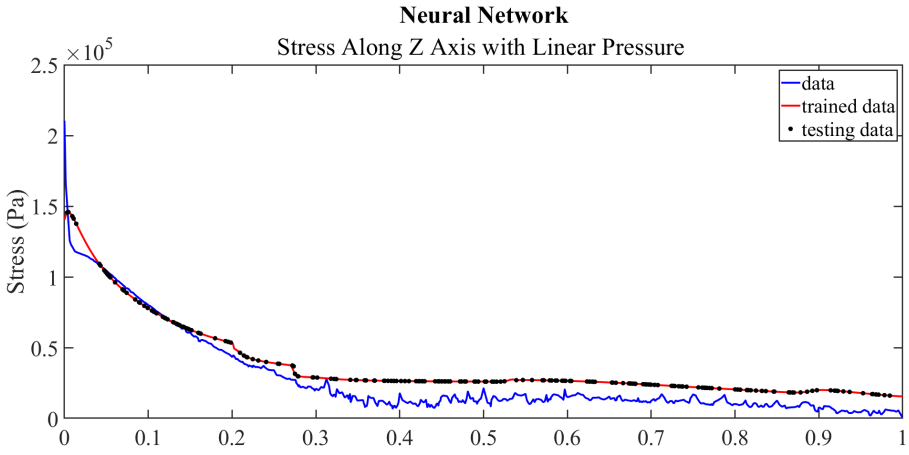

Figure 4.49 – Neural network model trained with directional deformation to predict equivalent stress for beam with linear pressure.

Though the neural network models are smoother and show fewer testing data outliners within the model prediction, it is the most unreliable due to the polarity of performance among

the cases. Significantly more computational time was taken for the training of neural network models, which devalues this model as well.

5. Static Analysis of Wing Structure

Understanding structural behavior of wings is crucial for maintenance during an aircraft's lifetime. Applicability of machine learning for aerospace applications to predict stress can be analyzed through wing geometries. Single wing geometry is analyzed through Ansys to collect training data of stress and deformation for machine learning models in MATLAB.

5.1 Problem Definition

Geometry is constructed in SOLIDWORKS to include an internal wing structure and airfoil skin based on the NACA 2412 airfoil. The internal wing structure consists of ribs, I beam spar, and two circular spars. The coordinate system is set at the leading edge of the root rib with the z axis along the length of the wing. The rib chord at the root is linearly decreased along the ribs till the tip rib is 30% of the initial chord.

Table 5.1 – Specifications of ribs in wing structure geometry.

Total number of ribs	15
Distance between ribs	0.08 m
Thickness of ribs	0.005 m
Total length of wing	1.195 m
Chord at root rib	0.2 m
Chord at tip rib	0.06 m

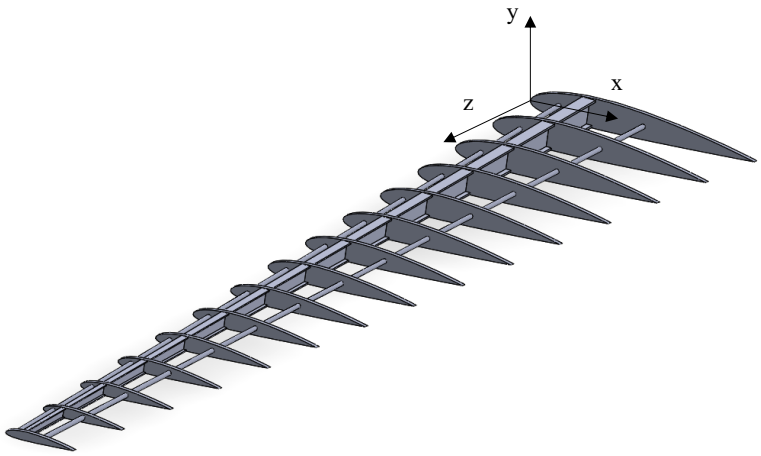

Figure 5.1 – Inner wing structure with two circular spars and a central I beam spar.

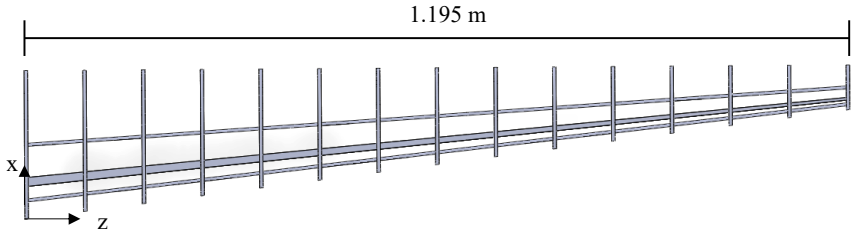

Figure 5.2 – Internal wing structure in xz plane.

The airfoil skin is created by lofting two splines that are 105% of the root and tip ribs. Outer dimensions of the chord are seen in Fig. 5.2. The airfoil is combined with the internal wing structure in SOLIDWORKS to create a single part.

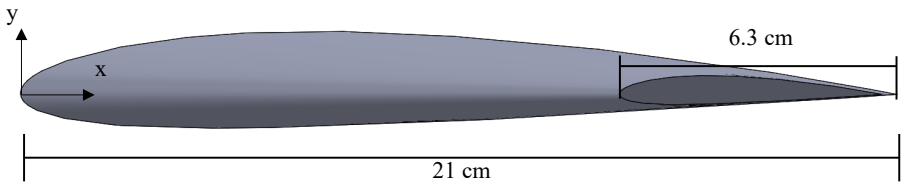

Figure 5.3 – Wing structure geometry in xy plane with root and tip chords.

Placement of the spars are determined by percent chord of the root and tip rib, seen in Fig. 5.4 below. Spars are created using the boundary feature in SOLIDWORKS with the cross-sectional sketches of the spars on the root and tip ribs, merging with all other ribs in between. Both circular spars are uniform diameter of 0.004 m for all ribs.

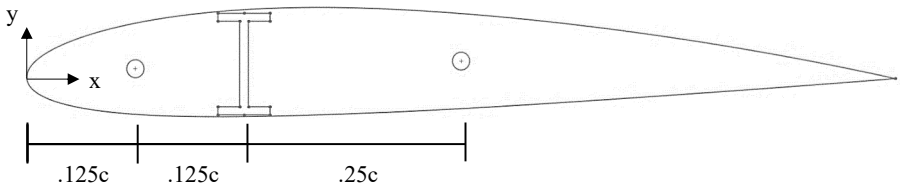

Figure 5.4 – Positions of spars along the ribs with respect to chord, shown on root rib.

Dimensions for the I-beam are shown in Fig. 5.5 for the root rib and are scaled with the ribs, where the dimensions at the tip are 30% of those on the root rib.

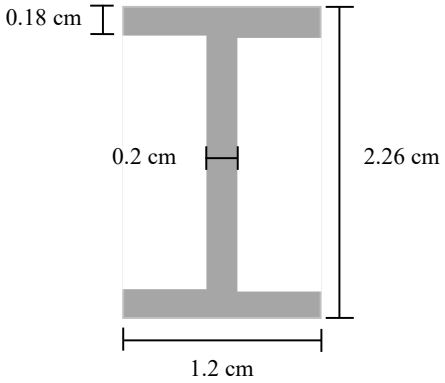

Figure 5.5 – Dimensions of I beam on the root rib.

The wing is fixed at the root (z=0 m) and will be analyzed with various loads as seen in the following figures.

Figure 5.6 – Case 1: 10 N point load in the -y direction at tip (z=1.195 m), constant along x axis.

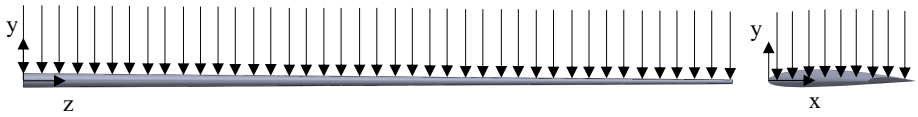

Figure 5.7 – Case 2: Constant distributed load in the -y direction of 10 N/m from root (z=0 m) to tip (z=1.195 m), constant along x axis.

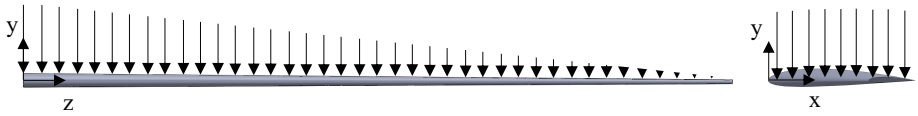

Figure 5.8 – Case 3: Linear pressure in the -y direction varying along z axis from 10 Pa at root (z=0 m) to 0 Pa at tip (z=1.195 m), constant along x axis.

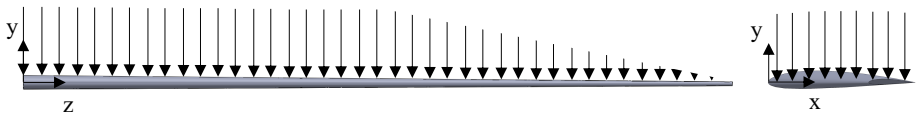

Figure 5.9 – Case 4: Constant pressure in the -y direction of 10 Pa from root (z=0 m) to half-length and linear pressure in the -y direction varying along z axis from 10 Pa at half-length to 0 Pa at tip (z=1.195 m), constant along x axis.

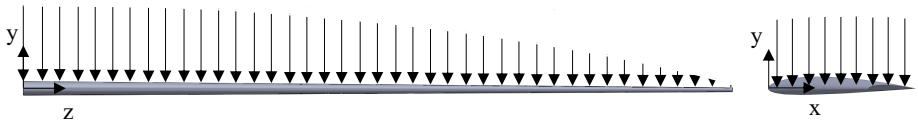

Figure 5.10 – Case 5: Parabolic pressure varying along z axis of 10 Pa at root (z=0 m) to 0 Pa at tip (z=1.195 m), constant along x axis.

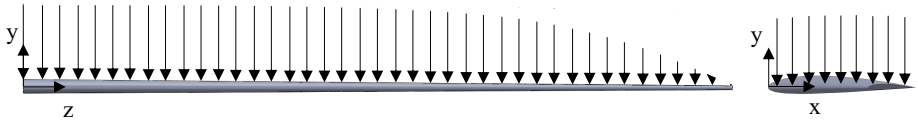

Figure 5.11 – Case 6: Elliptical pressure varying along z axis of 10 Pa at root (z=0 m) to 0 Pa at tip (z= 1.195 m), constant along x axis.

5.2 Ansys Simulations

Simulations for each case in Ansys are identical to those done in the previous chapter, but with the new wing geometry. Material is kept as structural steel; properties can be found in Table 4.2. Training data is collected from Ansys as outlined in Table 4.3 including x, y, and z coordinates along with the corresponding deformation and stress. Additionally, as the geometry is more complex and the data size is significantly larger, a path is added on the geometry for the plots produced to visualize the performance of the machine learning model. In Ansys, a path is inserted in the construction geometry along the upper chamber at x=0.25c from the root rib to the tip rib (seen in Fig. 5.12 below). Linearized deformation and stress data along the path is combined with the general data collected for machine learning in MATLAB. Data is plotted against the z coordinates along the path.

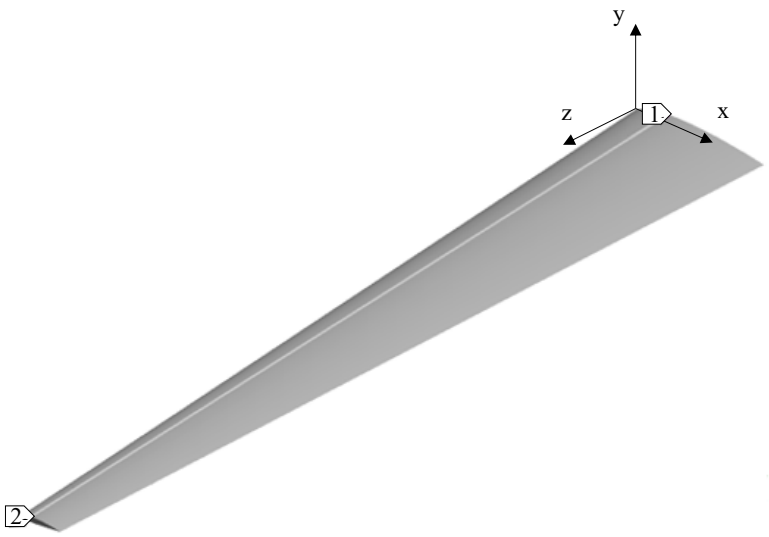

Figure 5.12 – Path on construction geometry from quarter chord on x axis of root rib (1) to quarter chord on x axis of tip rib (2) along upper chamber on airfoil skin.

To save computational time, mesh convergence is conducted for once case with a point load at full load. Due to the complexity of the geometry, convergence was manually done through altering element size, resolution, and span angle center. The final mesh is used for all cases simulated in Ansys.

Table 5.2 – Convergence history for wing with point load at full length.

Solution Number	Equivalent Stress (Pa)	Change (%)	Nodes	Elements
1	1.24707E+07		76,681	38,609
2	1.31202E+07	5.21	86,515	44,295
3	1.34239E+07	2.31	95,411	49,646
4	1.38056E+07	2.84	107,471	56,923
5	1.42516E+07	3.23	1,642,218	842,074
6	1.45611E+07	2.17	2,091,375	1,083,733
7	1.47407E+07	1.23	2,517,086	1,318,629
8	1.48408E+07	0.68	3,296,724	1,712,350

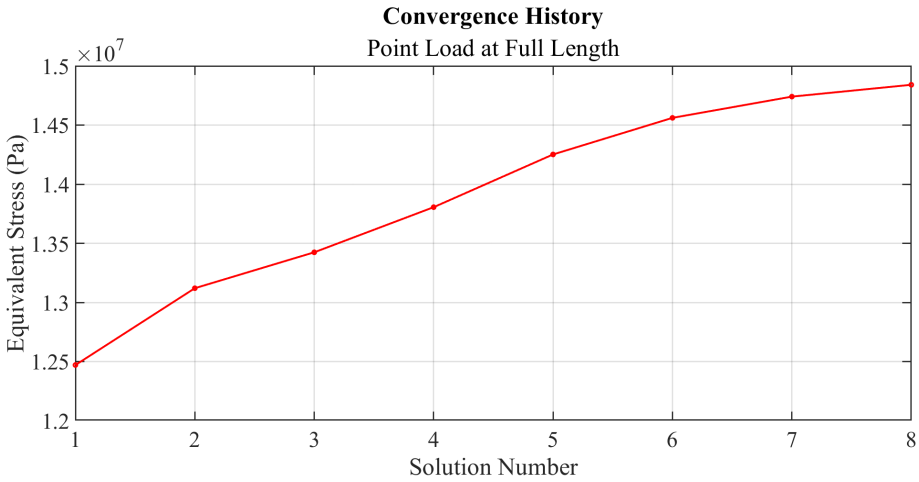

Figure 5.13 – Convergence history plot for wing with point load at full length.

5.3 Linear Regression Modeling

5.3.1 Training Model

The "fitlm" function in MATLAB is used for training linear regression models with directional deformation to predict equivalent stress along the wing. The MATLAB function is described in section 3.4.1.

5.3.2 Results

Performance of linear regression models is similar to that seen with the beam geometry, ranging in lower R^2 values, and not meeting the high performance criteria. Model for point load at full length and for linear pressure shows the lowest and highest R^2 values respectively, as in the previous chapter.

Table 5.3 – All cases simulated in Ansys for a wing geometry using linear regression models, along with their respective R^2 values.

Load on Wing	Trained R^2	Tested R^2
Point load at full length	0.228413	0.228095
Constant distributed load	0.323227	0.322534
Linear pressure	0.576809	0.579178
Constant and linear pressure	0.557836	0.558601
Parabolic pressure	0.530406	0.530207
Elliptical pressure	0.420232	0.422938

Table 5.4 – All cases simulated in Ansys for a wing geometry using linear regression models, along with their respective R^2 values, sorted by ascending trained R^2 values.

Load on Wing	Trained R^2	Tested R^2
Point load at full length	0.228413	0.228095
Constant distributed load	0.323227	0.322534
Elliptical pressure	0.420232	0.422938
Parabolic pressure	0.530406	0.530207
Constant and linear pressure	0.557836	0.558601
Linear pressure	0.576809	0.579178

As seen in the previous chapter, the linear regression models have difficulty with correctly estimating the data. Fig. 5.14 shows the lack of correlation between the training data and the model prediction, as well as the overestimation of smaller data values and underestimation of higher data values.

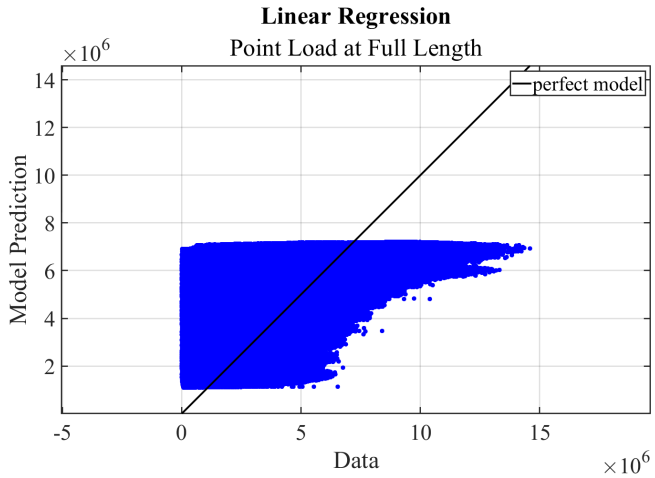

Figure 5.14 – Training data vs model prediction for linear regression model trained with directional deformation to predict equivalent stress for wing with point load at full length.

This lack of correlation is translated to a poor model that is only able to follow the trend of the data. Fig. 5.15 shows the model peaks at about the same point on the wing but is overall underestimating the stress in the wing. The ribs in the wing are seen in the plot where the stress suddenly decreases, which the model is not able to predict. The model is trained well enough to not overfit, as the testing data falls along the model prediction.

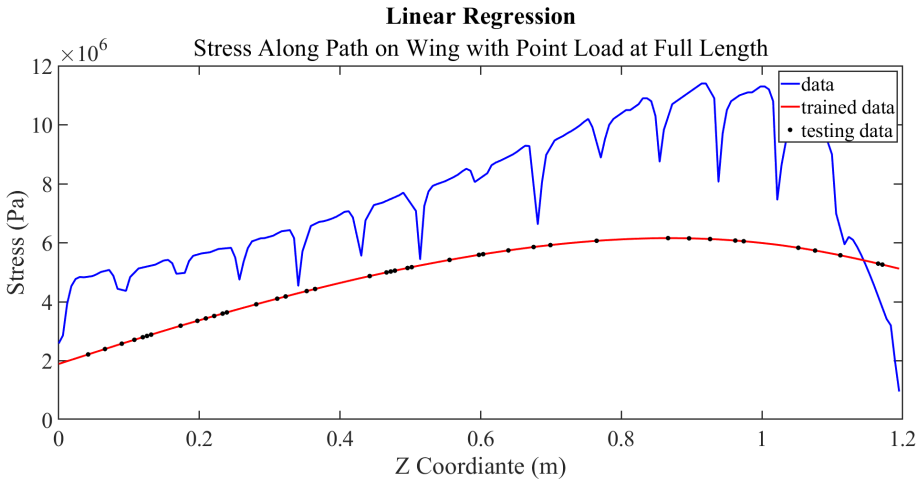

Figure 5.15 – Linear regression model trained with directional deformation to predict equivalent stress for wing with point load at full length.

As performance increases, the underestimation of data values decreases as seen in Fig. 5.16 below. The highest performing case, for a linear pressure on the wing, shows a smaller error in the amplitude of the model prediction. The nature of linear regression is not fit for the complexity of the data as it is not able to follow the variation of stress along the wing.

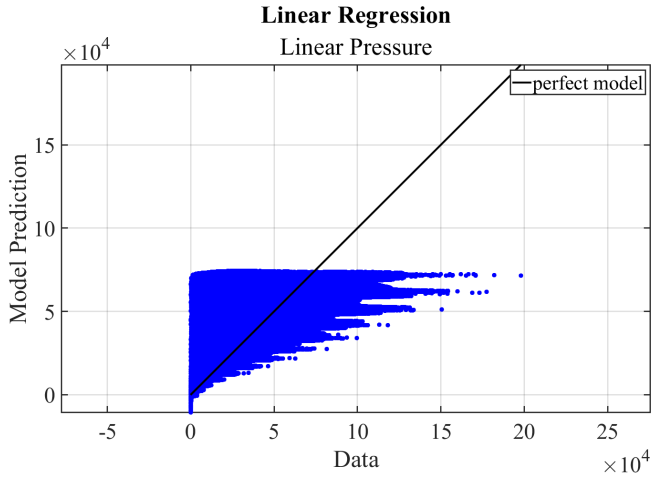

Figure 5.16 – Training data vs model prediction for linear regression model trained with directional deformation to predict equivalent stress for wing with linear pressure.

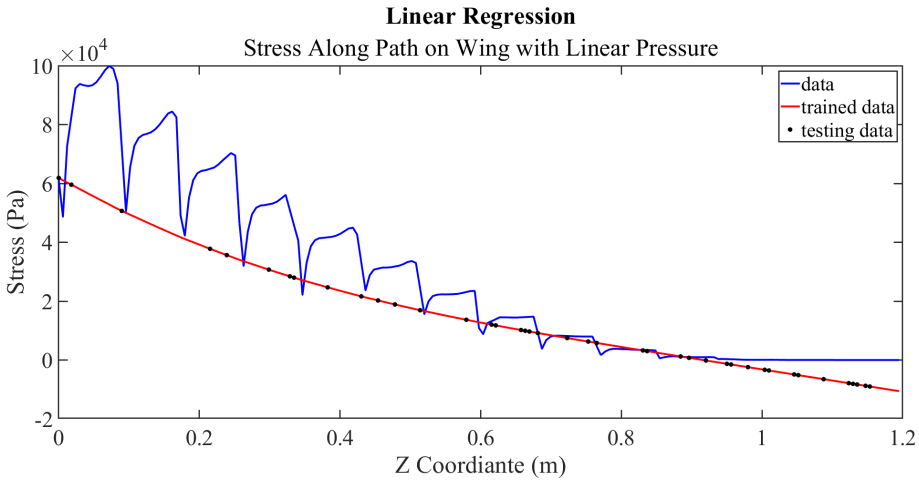

Figure 5.17 – Linear regression model trained with directional deformation to predict equivalent stress for wing with linear pressure.

5.4 Decision Tree Modeling

5.4.1 Training Model

The function "fitrtree" is used in MATLAB for training decision tree models and is discussed in section 3.5.1.

5.4.2 Results

Decision tree models for the wing geometry are all high performance models as all R^2 values are greater than 0.99, as observed with the beam geometry as well.

Table 5.5 – All cases simulated in Ansys for a wing geometry using decision tree models, along with their respective R^2 values.

Load on Wing	Trained R^2	Tested R^2
Point load at full length	0.993965	0.991747
Constant distributed load	0.993717	0.991372
Linear pressure	0.995208	0.993246
Constant and linear pressure	0.995249	0.993561
Parabolic pressure	0.994954	0.993083
Elliptical pressure	0.994243	0.992218

Table 5.6 – All cases simulated in Ansys for a wing geometry using decision tree models, along with their respective R^2 values, sorted by ascending trained R^2 values.

Load on Wing	Trained R^2	Tested R^2
Constant distributed load	0.993717	0.991372
Point load at full length	0.993965	0.991747
Elliptical pressure	0.994243	0.992218
Parabolic pressure	0.994954	0.993083
Linear pressure	0.995208	0.993246
Constant and linear pressure	0.995249	0.993561

Compared to the linear regression models, the order of performance is comparable considering the difference between the two lowest trained R^2 values and two highest R^2 values. The case with the lowest trained R^2 value is with a constant distributed load and is considered high performance. Correlation between data and model prediction is much more consistent than seen in linear regression models. The large size of the data allows for the outliers seen in Fig. 5.18 while also having a R^2 value higher than 0.99.

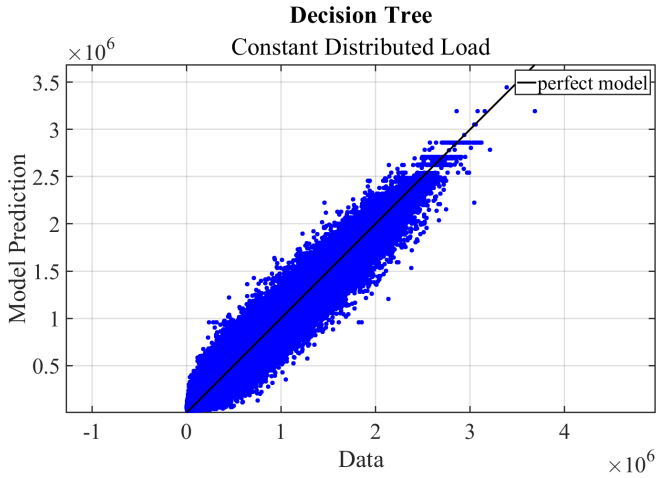

Figure 5.18 – Training data vs model prediction for decision tree model trained with directional deformation to predict equivalent stress for wing with constant distributed load.

Decision tree models predict stress along the wing with significantly higher accuracy than linear regression models, especially stress at the location of the ribs. The model is overall less smooth than linear regression models, but accurately predicts the behavior of the data. Discrepancies are more visibly present at the root of the wing, which is also where testing data tends to fall out of the model prediction. Testing data outliers throughout the model prediction suggest overfitting within the model. In an enlarged view of the plot, the rough behavior of the prediction is seen as well as the testing data outliers.

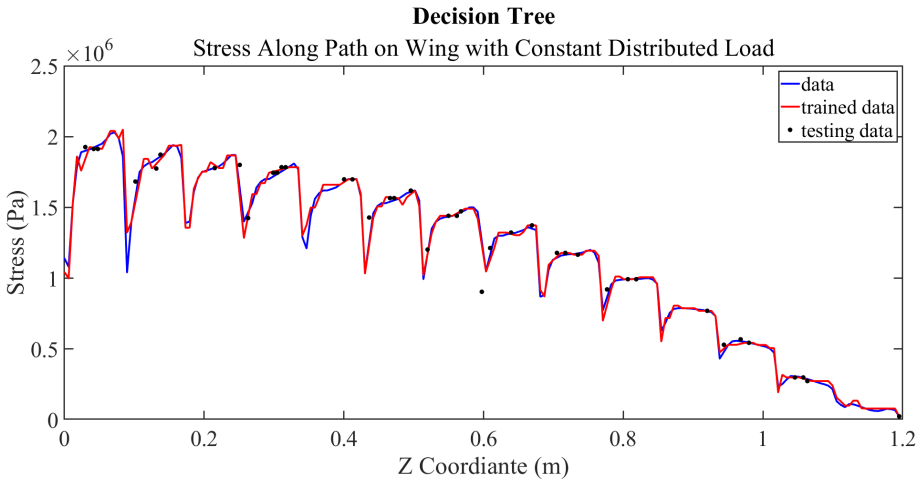

Figure 5.19 – Decision tree model trained with directional deformation to predict equivalent stress for wing with constant distributed load.

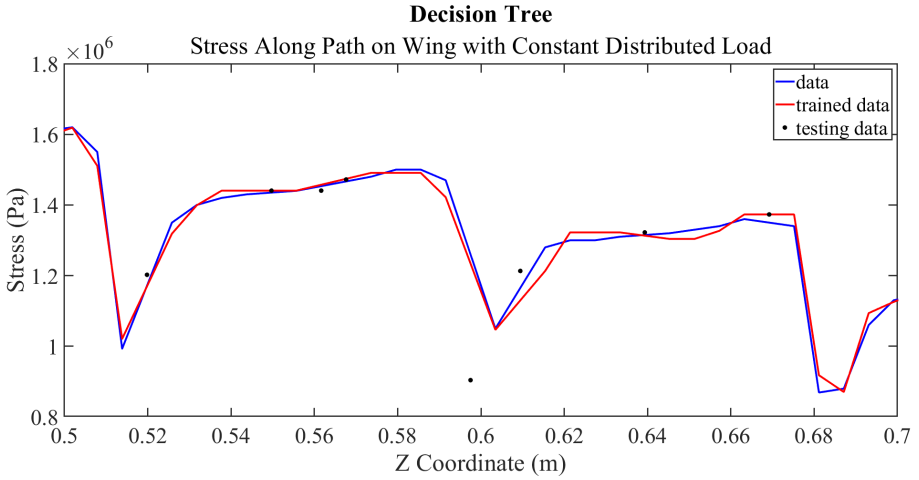

Figure 5.20 – Enlarged view of decision tree model trained with directional deformation to predict equivalent stress for wing with constant distributed load.

Since the difference between the decision tree model R^2 values has higher significant figures, the difference between correlation of data and model prediction is minute. The correlation for the highest performing model with constant and linear pressure in Fig. 5.21 compared to Fig. 5.18 is much denser but has similar outliers.

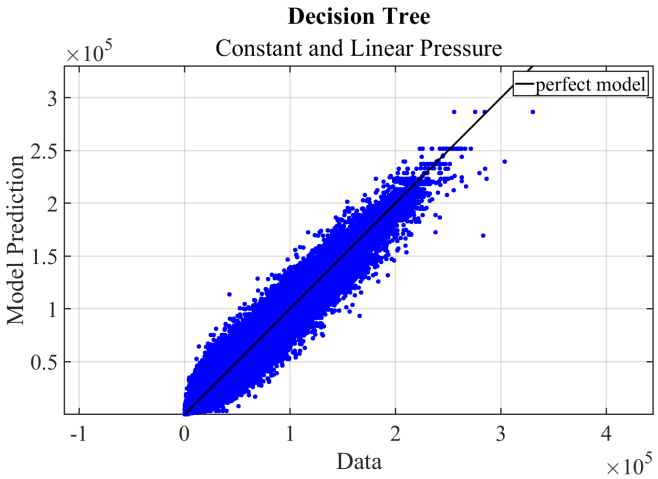

Figure 5.21– Training data vs model prediction for decision tree model trained with directional deformation to predict equivalent stress for wing with constant and linear pressure.

The highest performing model is smoother and is less overfit than the lowest performing model, as the testing data falls more in line with the model prediction. The roughness of the model is fairly similar to the data itself as seen in Fig. 5.23.

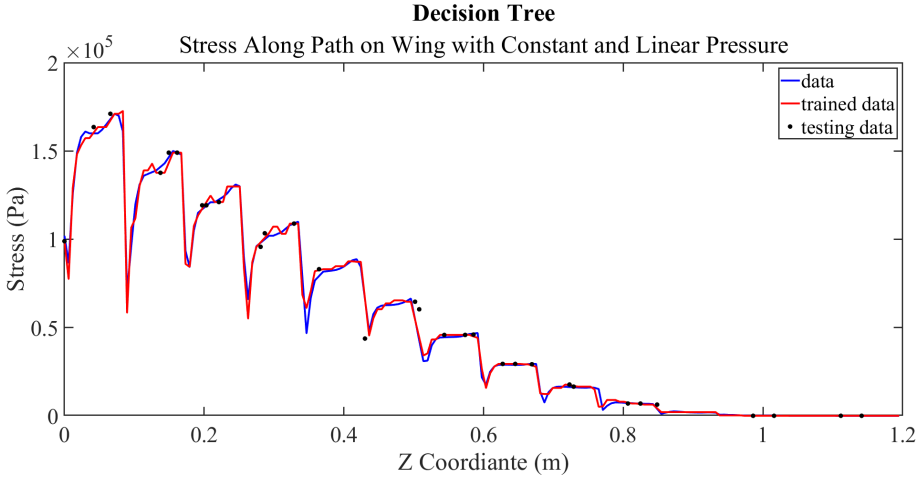

Figure 5.22 – Decision tree model trained with directional deformation to predict equivalent stress for wing with constant and linear pressure.

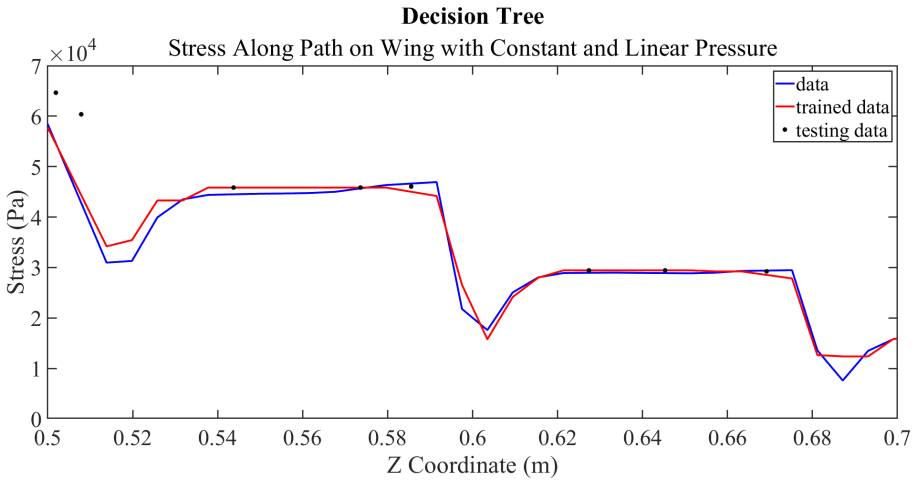

Figure 5.23 – Enlarged view of decision tree model trained with directional deformation to predict equivalent stress for wing with constant and linear pressure.

The nature of the model is seen in the plots as the model prediction is rough. Compared to linear regression models, the decision tree method is more prone to overfit. Decision tree models are overall well equipt as observed with the high R^2 values for all cases and the ability to predict the complex nature of the data.

5.5 Random Forest Modeling

5.5.1 Training Model

The function "fitrensemble" is used in MATLAB to train random forest models, along with the automatic optimization of hyperparameters. Optimization is done with the point load at full length of the wing case to reduction computational time, resulting in an ensemble using the bag method with 309 trees.

5.5.2 Results

Random forest performance is observed to be the highest out of all other machine learning methods explored. Compared to the decision tree method, random forest models have larger R^2 values and are more accurate.

Table 5.7 – All cases simulated in Ansys for a wing geometry using random forest models, along with their respective R^2 values.

Load on Wing	Trained R^2	Tested R^2
Point load at full length	0.996682	0.995510
Constant distributed load	0.996605	0.995452
Linear pressure	0.997404	0.996351

Constant and linear pressure	0.997412	0.996505
Parabolic pressure	0.997290	0.996350
Elliptical pressure	0.996901	0.995857

Table 5.8 – All cases simulated in Ansys for a wing geometry using random forest models, along with their respective R^2 values, sorted by ascending trained R^2 values.

Load on Wing	Trained R^2	Tested R^2
Constant distributed load	0.996605	0.995452
Point load at full length	0.996682	0.995510
Elliptical pressure	0.996901	0.995857
Parabolic pressure	0.997290	0.996350
Linear pressure	0.997404	0.996351
Constant and linear pressure	0.997412	0.996505

Considering the small difference between the R^2 values for random forest cases, negligible differences are seen in the correlations of data and model predictions between cases. The correlation is denser than decision tree models and shows fewer major outliers. The correlation for the lowest performing case with a constant distributed load (Fig. 5.24) shows similarity to the correlation for the highest performing case with a constant and linear pressure (Fig. 5.25).

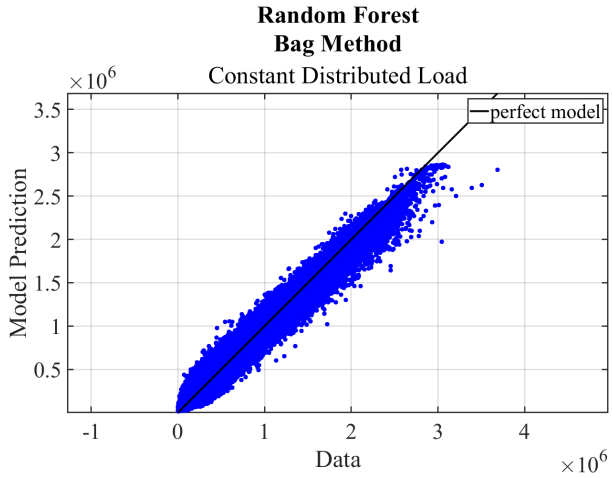

Figure 5.24– Training data vs model prediction for random forest model trained with directional deformation to predict equivalent stress for wing with constant distributed load.

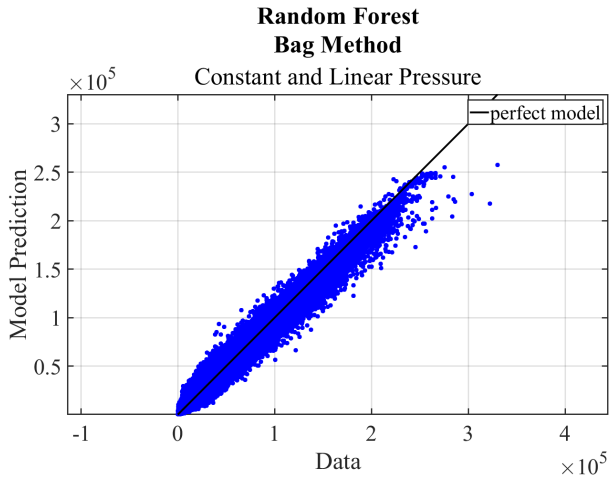

Figure 5.25 – Training data vs model prediction for decision tree model trained with directional deformation to predict equivalent stress for wing with constant and linear pressure.

The lowest performing model prediction for constant distributed load is seen in Fig. 5.26 below. Random forest models are overall smoother and follows the data with higher accuracy. An enlarged view of the prediction shows the minor difference between the data and prediction.

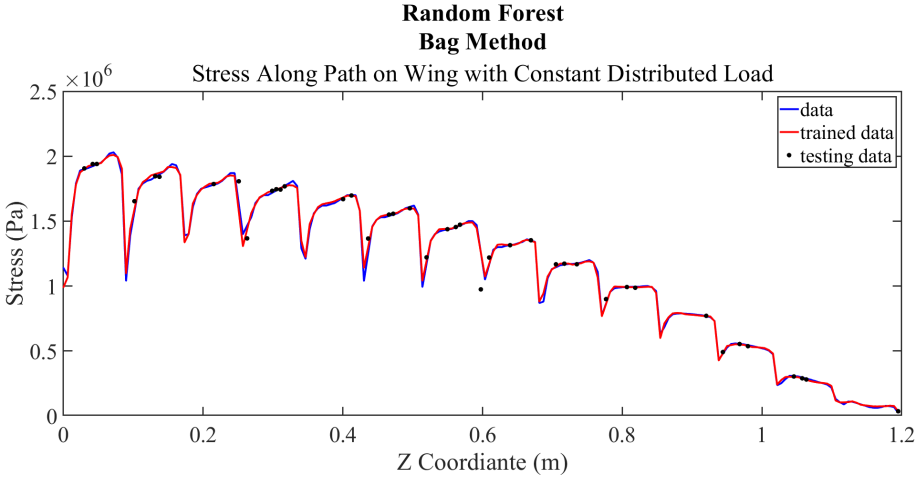

Figure 5.26 – Random forest model trained with directional deformation to predict equivalent stress for wing with constant distributed load.

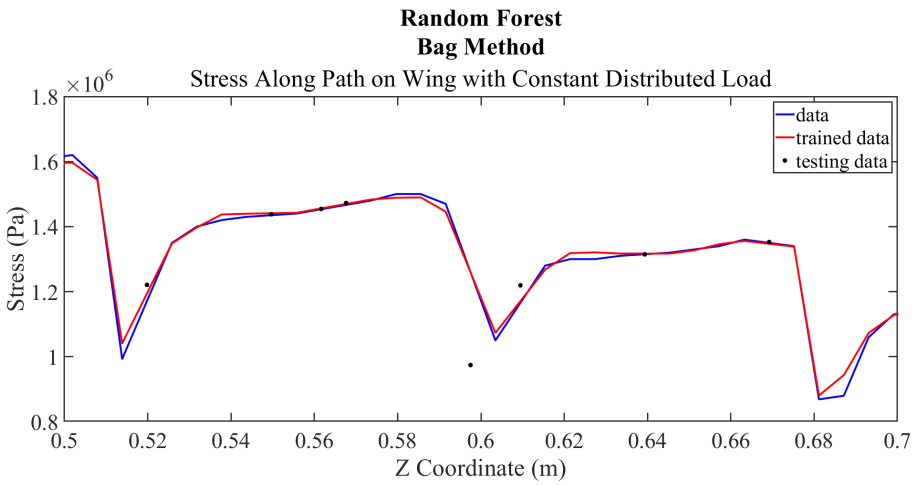

Figure 5.27 – Enlarged view of random forest model trained with directional deformation to predict equivalent stress for wing with constant distributed load.

Testing data aligns more closely with the model prediction as the R^2 value of the model increases. The highest performing model with a constant and linear pressure is less overfit, as trained and testing data discrepancies are only seen around z=0.5m in Fig. 5.28 below.

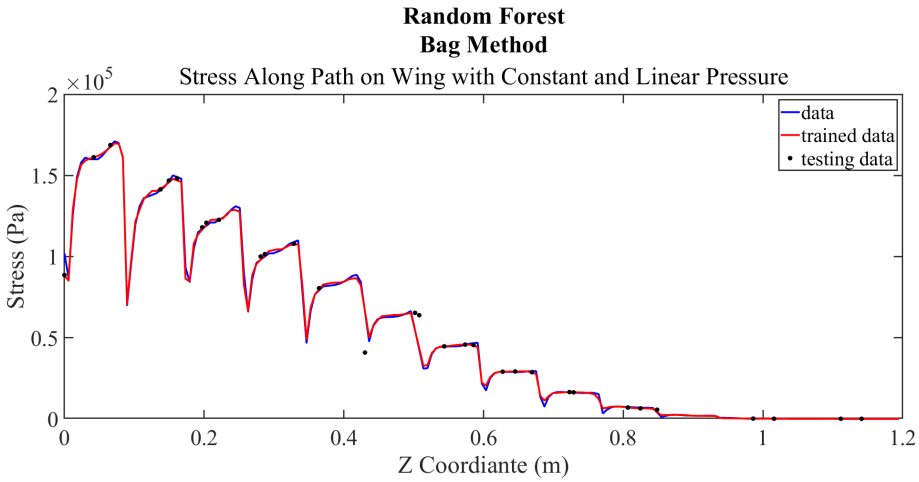

Figure 5.28 – Random forest model trained with directional deformation to predict equivalent stress for wing with constant and linear pressure.

In an enlarged view of the model with constant and linear pressure, the model prediction and testing data are seen to be more closely aligned with the data.

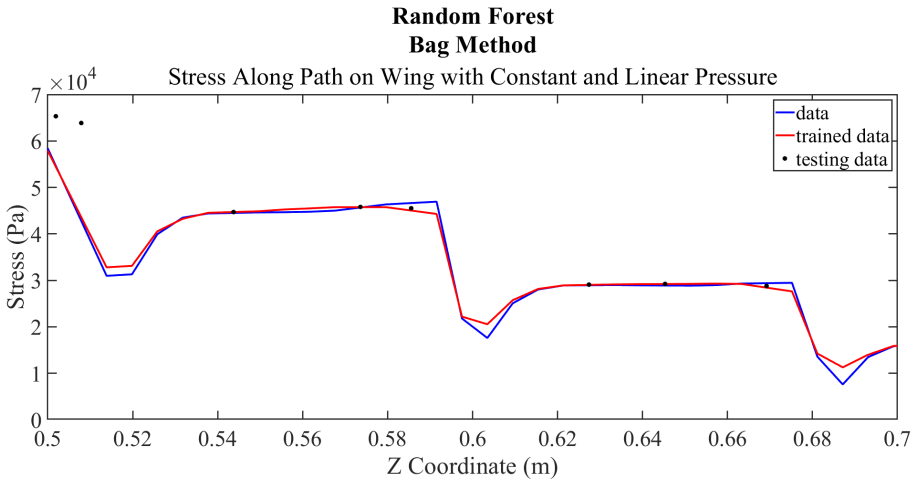

Figure 5.29 – Enlarged view of random forest model trained with directional deformation to predict equivalent stress for wing with constant and linear pressure.

Random forest models are most consistent with performance, being able to accurately model any data presented so far. Some overfitting is observed, but no more than seen in decision tree models.

5.6 Neural Network Modeling

5.6.1 Training Model

Functions “fitnet” and “train” are used in MATLAB to train the neural network models, as in previous chapters. First case with point load at full length was used for optimization of hidden layer size of two layers with 0-25 neurons due to the large size of training data. Optimal hidden layer size was determined to be 23 neurons in the first layer and 22 in the second layer

5.6.2 Results

In previous chapters, neural networks models vary in performance among the cases whereas for the wing geometry the performance is much more consistent. All R^2 values are greater than 0.9, allowing neural network models to be more comparable to decision tree models than previously observed.

Table 5.9 – All cases simulated in Ansys for a wing geometry using neural network models, along with their respective R^2 values.

Load on Wing	Trained R^2	Tested R^2
Point load at full length	0.905259	0.904510

Constant distributed load	0.962638	0.962641
Linear pressure	0.977113	0.977310
Constant and linear pressure	0.973369	0.973347
Parabolic pressure	0.969541	0.969912
Elliptical pressure	0.959893	0.959767

Table 5.10 – All cases simulated in Ansys for a wing geometry using neural network models, along with their respective R^2 values, sorted by ascending trained R^2 values.

Load on Wing	Trained R^2	Tested R^2
Point load at full length	0.905259	0.904510
Elliptical pressure	0.959893	0.959767
Constant distributed load	0.962638	0.962641
Parabolic pressure	0.969541	0.969912
Constant and linear pressure	0.973369	0.973347
Linear pressure	0.977113	0.977310

The order of performance is similar to the performance of linear regression models, difference being the order of elliptical pressure and constant distributed load. Neural network models tend to overestimate data values more than any other model type, as seen in Fig. 5.30 for the lowest performing case, though still has a large R^2 value.

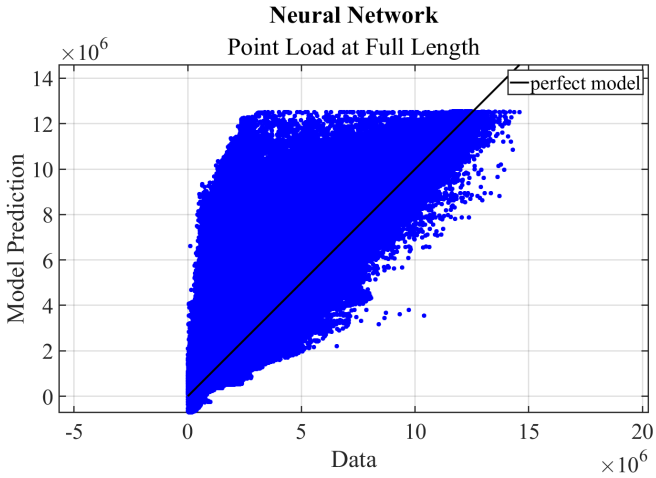

Figure 5.30 – Training data vs model prediction for neural network model trained with directional deformation to predict equivalent stress for wing with point load at full length.

The model prediction for stress in the wing with a point load at full length shows resemblance to linear regression models. Though the accuracy of the model following the amplitude of the stress is higher, it is still unable to estimate the intricacies of the data at the rib locations. The model shows little to no overfitting, as the testing data aligns with the trained data.

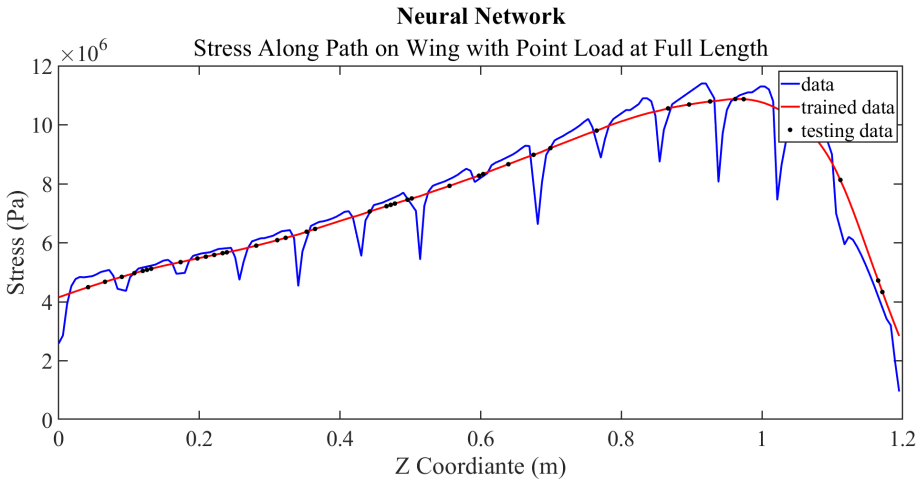

Figure 5.31 – Neural network model trained with directional deformation to predict equivalent stress for wing with point load at full length.

Correlation of data and model prediction is still irregular for the highest performing case with linear pressure but is much denser along the perfect model line. Much less overestimation is seen in Fig. 5.32, but outliers show no clear pattern. Outliers are more underestimated for larger data values when compared to Fig. 5.30.

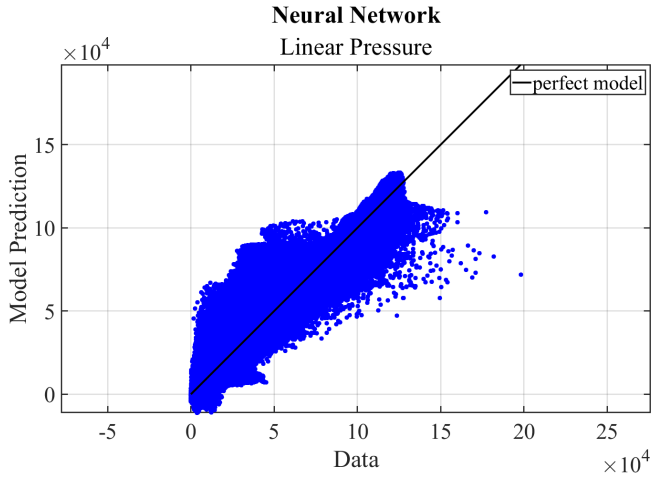

Figure 5.32 – Training data vs model prediction for neural network model trained with directional deformation to predict equivalent stress for wing with linear pressure.

Though with a higher R^2 value the model is able to estimate the changes of stress where the ribs are located, the accuracy does not compare to decision tree or random forest models. An error in the model prediction is also evident for the stress values between the ribs, where the behavior of the data is incorrectly predicted. The higher R^2 value also increases the error in the alignment of trained and testing data, as seen in an enlarged view of the plot in Fig. 5.34.

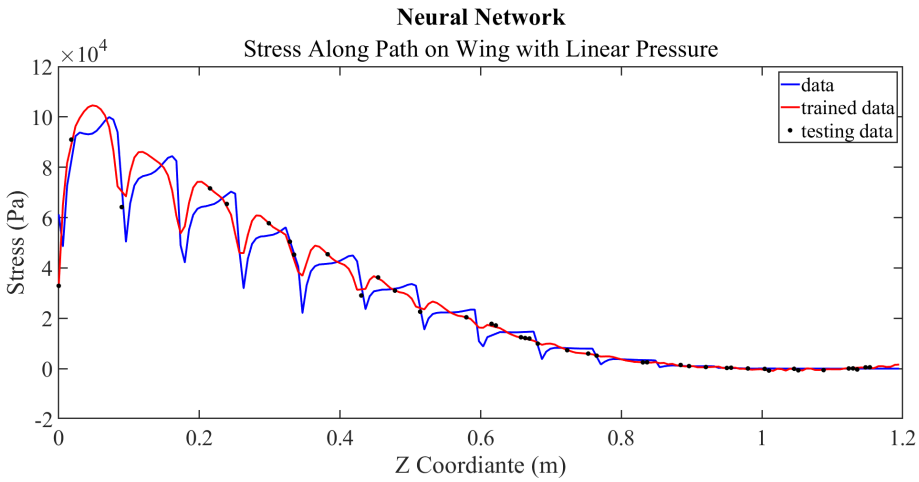

Figure 5.33 – Neural network model trained with directional deformation to predict equivalent stress for wing with linear pressure.

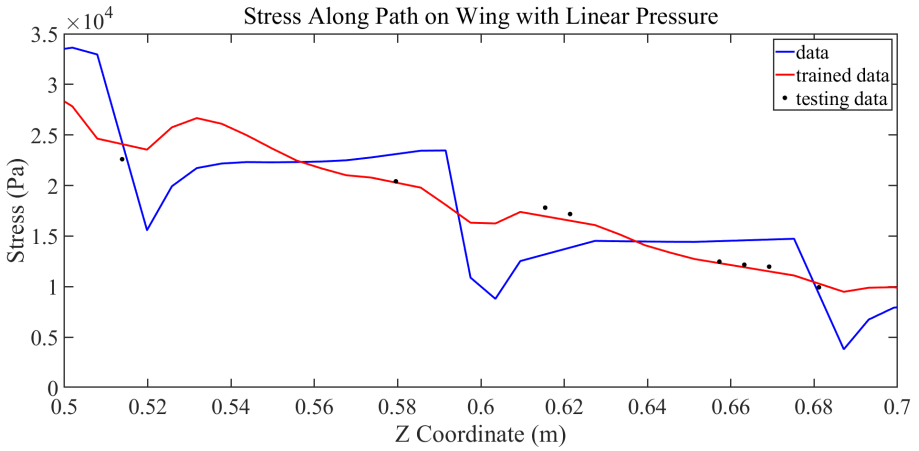

Figure 5. 34 – Enlarged view of neural network model trained with directional deformation to predict equivalent stress for wing with linear pressure.

All neural network models show the same prediction error of stress between ribs. The case with R^2 value closest to the average R^2 of the cases for the wing geometry is that with elliptical pressure. The model for this case highlights the positive correlation between an increase in R^2 and inaccuracy of predicting the specific behavior of the data.

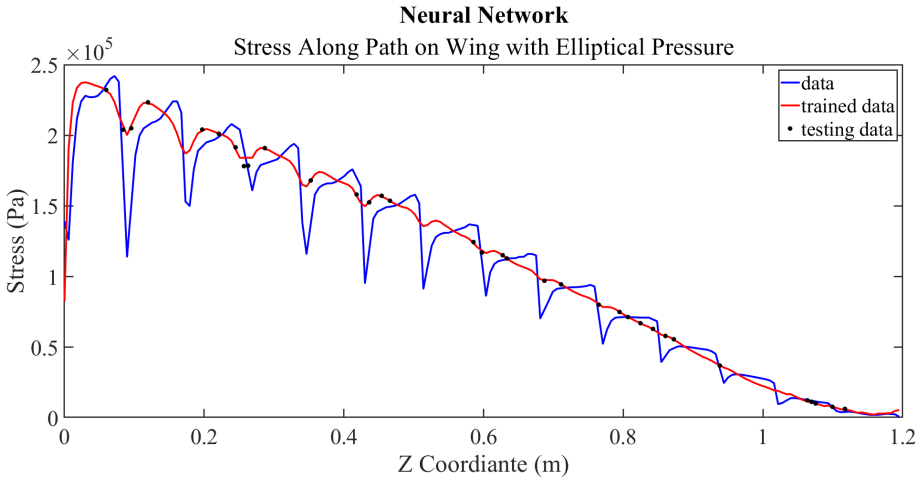

Figure 5.35 – Neural network model trained with directional deformation to predict equivalent stress for wing with elliptical pressure.

Accuracy of neural network models bear a resemblance to linear regression models, though having consistent R^2 values closer to decision tree and random forest models. Though the prediction shows close similarity to the data, the error within the prediction greatly diminishes the suitability of the model for this type of data.

6. Conclusion

Machine learning applications explored within this project show fairly consistent results regarding performance of linear regression, decision tree, random forest, and neural network algorithms. Data is collected from discretized equations of motion, for a three degree of freedom system, and FEA in Ansys, for a beam and wing geometry, for supervised regression learning. Decision tree and random forest models prove to be adequate for the training data collected, while linear regression and neural network models show poor performance.

The linear regression algorithm produced the most trained R^2 values below the high performance threshold of 0.6. In rare cases was the algorithm able to properly predict the data in the spring mass damper system models. The data vs. model prediction plots for linear regression show a tendency to overestimate and underestimate data values. Though the model does not track the training data, the testing data is seen to perfectly align with the model prediction, as expected based on the nature of the linear regression algorithm. All decision tree models fit within the high performance threshold and was able to adequately track training data. The lowest performing decision tree model showcase the noisy nature of the algorithm, as well as the tendency to overfit as the testing data did not align with the model prediction. Cases for the beam and wing geometry all have trained R^2 values larger than 0.99 and show little overfitting in the model plots. Random forest models are very similar to decision tree models, though have higher performance and less noise in the model prediction. Overfitting is seen to decrease as R^2 increases in the spring mass damper models, whereas little overfitting is seen in the beam and wing geometry models. Of all algorithms, random forest is optimal for prediction of engineering data presented in this project. Performance of neural networks models is comparable to linear regression, as well as the behavior of the prediction. The tendency to overestimate and underestimate data is overserved in neural network models and the testing data rarely misaligns with the model prediction. The algorithm was not able to properly predict the full complexity of stress in a wing.

Capabilities of random forest and decision tree models to predict engineering data is promising for further applications of machine learning in the aerospace industry. Neural network models likely did not perform to the best extent due to the high computational power and time necessary for training models. Further optimization and exploration of neural network models may demonstrate similar capabilities as random forest models.

References

[1] International Air Transport Association. *Airline Maintenance Cost Executive Commentary*. 2019.

[2] Piotrowski, D., Roach, D., Melton, A., Bohler, J., Rice, T., Neidigk, S., and Linn, J. "Implementation of Structural Health Monitoring (SHM) into an Airline Maintenance Program." *Structural Health Monitoring*, 2015, pp. 2727–2733. https://doi.org/10.12783/shm2015/338.

[3] Liang, L., Liu, M., Martin, C., and Sun, W. "A Deep Learning Approach to Estimate Stress Distribution: A Fast and Accurate Surrogate of Finite-Element Analysis." *Journal of the Royal Society Interface*, Vol. 15, No. 138, 2018. https://doi.org/10.1098/rsif.2017.0844.

[4] Krishnakumar, K., "Intelligent Systems for Aerospace Engineering: An Overview," Research and Technology Organisation Educational Notes 22 on Intelligent Systems for Aeronautics, Rhode-Saint-Genèse, Belgium, May 2002.

[5] Huang, Z., Wang, C., Chen, J., and Tian, H. "Optimal Design of Aeroengine Turbine Disc Based on Kriging Surrogate Models." *Computers & Structures*, Vol. 89, No. 1–2, 2011, pp. 27–37. https://doi.org/10.1016/j.compstruc.2010.07.010.

[6] Naranjo-Pérez, J., Infantes, M., Fernando Jiménez-Alonso, J., and Sáez, A. "A Collaborative Machine Learning-Optimization Algorithm to Improve the Finite Element Model Updating of Civil Engineering Structures." *Engineering Structures*, Vol. 225, 2020. https://doi.org/10.1016/j.engstruct.2020.111327.

[7] Yang, F., and Ren, J. "Reliability Analysis Based on Optimization Random Forest Model and MCMC." *CMES - Computer Modeling in Engineering and Sciences*, Vol. 125, No. 2, 2020, pp. 801–814. https://doi.org/10.32604/cmes.2020.08889.

[8] Otsu, K., Ono, M., Fuchs, T. J., Baldwin, I., and Kubota, T. "Autonomous Terrain Classification with Co-and Self-Training Approach." *IEEE Robotics and Automation Letters*, Vol. 1, No. 2, 2016, pp. 814–819. https://doi.org/10.1109/LRA.2016.2525040.

[9] Bertoni, A., Hallstedt, S. I., Dasari, S. K., and Andersson, P. "Integration of Value and Sustainability Assessment in Design Space Exploration by Machine Learning: An Aerospace Application." *Design Science*, Vol. 6, 2020. https://doi.org/10.1017/dsj.2019.29.

[10] Shin, D., and Kim, Y. Y. "Data-Driven Approach for a One-Dimensional Thin-Walled Beam Analysis." *Computers and Structures*, Vol. 231, 2020. https://doi.org/10.1016/j.compstruc.2020.106207.

[11] Jacobs, E. W., Yang, C., Demir, K. G., and Gu, G. X. "Vibrational Detection of Delamination in Composites Using a Combined Finite Element Analysis and Machine

Learning Approach." *Journal of Applied Physics*, Vol. 128, No. 12, 2020. https://doi.org/10.1063/5.0015648.

[12] Chow, W. T. Supervised Learning for Finite Element Analysis of Holes Under Tensile Load. In *Mechanisms and Machine Science*, Springer, 2020, pp. 1329–1339.

[13] Yuan, F.-G., Zargar, S. A., Chen, Q., and Wang, S. Machine Learning for Structural Health Monitoring: Challenges and Opportunities. No. v 11379, 2020, pp. 2.

[14] O'Higgins, E., Graham, K., Daverschot, D., and Baris, J. Machine Learning Application on Aircraft Fatigue Stress Predictions. 2020.

[15] James, G., Witten, D., Hastie, T., and Tibshirani, R. *An Introduction to Statistical Learning*. Springer Science+Business Media, New York, 2013.

[16] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J. "Phoneme Recognition Using Time-Delay Neural Networks." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 37, No. 3, 1989, pp. 328–339. https://doi.org/10.1109/29.21701.

[17] Pomerleau, D. A. Alvinn: An Autonomous Land Vehicle in a Neural Network. https://apps.dtic.mil/sti/citations/ADA218975. Accessed Oct. 4, 2021.

[18] Anderson, J. R. "The Adaptive Nature of Human Categorization." *Psychological Review*, Vol. 98, No. 3, 1991.

[19] Tesauro, G. "Temporal Difference Learning of Backgammon Strategy." *Machine Learning Proceedings 1992*, 1992, pp. 451–457. https://doi.org/10.1016/B978-1-55860-247-2.50063-2.

[20] Samuel, A. L. "Some Studies in Machine Learning Using the Game of Checkers." *IBM Journal of Research and Development*, Vol. 3, No. 3, 1959, pp. 210–229. https://doi.org/10.1147/RD.33.0210.

[21] Mitchell, T. *Machine Learning*. McGraw Hill, 1997.

[22] Fit Binary Decision Tree for Regression - MATLAB Fitrtree. https://www.mathworks.com/help/stats/fitrtree.html. Accessed Nov. 15, 2021.

[23] Oshiro, T. M., Perez, P. S., and Baranauskas, J. A. "How Many Trees in a Random Forest?" *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 7376 LNAI, 2012, pp. 154–168. https://doi.org/10.1007/978-3-642-31537-4_13.

[24] Russell, S., and Norvig, P. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, New Jersey, 2010.

[25] Haykin, S. *Neural Networks and Learning Machines*. Pearson Education, Upper Saddle River, New Jersey, 2009.

[26] Fit Linear Regression Model - MATLAB Fitlm. https://www.mathworks.com/help/stats/fitlm.html#mw_1721ca5a-8938-44ad-8366-f59bec7b00a9. Accessed Aug. 19, 2021.

[27] Fit Binary Decision Tree for Regression - MATLAB Fitrtree. https://www.mathworks.com/help/stats/fitrtree.html#bunrdhe-1. Accessed Aug. 19, 2021.

[28] Ensemble Algorithms - MATLAB & Simulink. https://www.mathworks.com/help/stats/ensemble-algorithms.html#bsw8av_. Accessed Sep. 14, 2021.

[29] Function Fitting Neural Network - MATLAB Fitnet. https://www.mathworks.com/help/deeplearning/ref/fitnet.html#bu2xeyw-2. Accessed Oct. 31, 2021.

[30] Train Shallow Neural Network - MATLAB Train. https://www.mathworks.com/help/deeplearning/ref/network.train.html#d123e185291. Accessed Oct. 31, 2021.

Appendix A – MATLAB Code for Three Degree Spring Mass Damper System

3DOF Mass Spring Damper System

```
% masses (kg)
m1 = 5; m2 = 1; m3 = 3;

% dampers (Ns/m)
c1 = 1; c2 = 1; c3 = 2;

% springs (N/m)
k1 = 700000; k2 = 10000; k3 = 50000;

% F(1) = sine @15 Hz, amplitude of 0.5
f1_a = 0.5; f1_hz = 15;

% F(2) = cosin @15 Hz, amplitude of 0.7
f2_a = 0.7; f2_hz = 15;

% F(3) = sine @15 Hz, amplitude of 0.1
f3_a = 0.1; f3_hz = 15;


% Matrices of ODE
M = diag([m1,m2,m3]);

C = ([c1+c2  -c2     0;
      -c2   c2+c3  -c3;
       0     -c3    c3]);

K = ([k1+k2  -k2     0;
      -k2   k2+k3  -k3;
       0     -k3    k3]);

% ode45 setup
tspan = 0:0.001:2;
x0 = [0; 0; 0; 0; 0; 0];

[t,x] = ode45(@(t,x) integration(t,x, M, C, K, f1_a, f2_a, f3_a, f1_hz, f2_hz, f3_hz), tspan,
x0);

for i=1:length(tspan)
    dxdt_tr(:,i) = integration(t(i),x(i,:).', M, C, K, f1_a, f2_a, f3_a, f1_hz, f2_hz, f3_hz);
end

dxdt = dxdt_tr.';

% data processing
p_m1 = x(:,1); p_m2 = x(:,2); p_m3 = x(:,3);
v_m1 = x(:,4); v_m2 = x(:,5); v_m3 = x(:,6);
a_m1 = dxdt(:,4); a_m2 = dxdt(:,5); a_m3 = dxdt(:,6);

data = [p_m1, p_m2, p_m3, v_m1, v_m2, v_m3, a_m1, a_m2, a_m3,];
```

```
% percent of data used for training
p = 0.8;
[xtrain, xval, ttrain, tval] = partition(p, t, data);

figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(t, p_m1, 'Linewidth', 3)
hold on
plot(t,p_m2, 'g', 'Linewidth', 3)
hold on
plot(t,p_m3,'r', 'Linewidth', 3)
title('Displacement')
xlabel('Time (s)')
legend('m1','m2','m3')
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))

figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(t,v_m1, 'Linewidth', 3)
hold on
plot(t,v_m2, 'g', 'Linewidth', 3)
hold on
plot(t,v_m3,'r', 'Linewidth', 3)
title('Velocity')
xlabel('Time (s)')
legend('m1','m2','m3')
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))

figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(t,a_m1, 'Linewidth', 3)
hold on
plot(t,a_m2, 'g', 'Linewidth', 3)
hold on
plot(t,a_m3,'r', 'Linewidth', 3)
title('Acceleration')
xlabel('Time (s)')
legend('m1','m2','m3')
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))
```

Decision Tree

```
%Acceleration to Solve for Displacement
%using Acceleration to solve for Displacement of M1
dectre(xtrain(:,7:9), xtrain(:,1), xval(:,7:9), xval(:,1), ttrain, tval,  'Acceleration',
```

```
'Displacement', 'M1')
%using Acceleration to solve for Displacement of M2
dectre(xtrain(:,7:9), xtrain(:,2), xval(:,7:9), xval(:,2), ttrain, tval,  'Acceleration',
'Displacement', 'M2')
%using Acceleration to solve for Displacement of M3
dectre(xtrain(:,7:9), xtrain(:,3), xval(:,7:9), xval(:,3), ttrain, tval,  'Acceleration',
'Displacement', 'M3')

%Acceleration to Solve for Velocity
%using Acceleration to solve for Velocity of M1
dectre(xtrain(:,7:9), xtrain(:,4), xval(:,7:9), xval(:,4), ttrain, tval,  'Acceleration',
'Velocity', 'M1')
%using Acceleration to solve for Velocity of M2
dectre(xtrain(:,7:9), xtrain(:,5), xval(:,7:9), xval(:,5), ttrain, tval,  'Acceleration',
'Velocity', 'M2')
%using Acceleration to solve for Velocity of M3
dectre(xtrain(:,7:9), xtrain(:,6), xval(:,7:9), xval(:,6), ttrain, tval,  'Acceleration',
'Velocity', 'M3')

%Velocity to Solve for Displacement
%using Velocity to solve for Displacement of M1
dectre(xtrain(:,4:6), xtrain(:,1), xval(:,4:6), xval(:,1), ttrain, tval,  'Velocity',
'Displacement', 'M1')
%using Velocity to solve for Displacement of M2
dectre(xtrain(:,4:6), xtrain(:,2), xval(:,4:6), xval(:,2), ttrain, tval,  'Velocity',
'Displacement', 'M2')
%using Velocity to solve for Displacement of M3
dectre(xtrain(:,4:6), xtrain(:,3), xval(:,4:6), xval(:,3), ttrain, tval,  'Velocity',
'Displacement', 'M3')

%Velocity to Solve for Acceleration
%using Velocity to solve for Acceleration of M1
dectre(xtrain(:,4:6), xtrain(:,7), xval(:,4:6), xval(:,7), ttrain, tval,  'Velocity',
'Acceleration', 'M1')
%using Velocity to solve for Acceleration of M2
dectre(xtrain(:,4:6), xtrain(:,8), xval(:,4:6), xval(:,8), ttrain, tval,  'Velocity',
'Acceleration', 'M2')
%using Velocity to solve for Acceleration of M3
dectre(xtrain(:,4:6), xtrain(:,9), xval(:,4:6), xval(:,9), ttrain, tval,  'Velocity',
'Acceleration', 'M3')

%Displacement to Solve for Velocity
%using Displacement to solve for Velocity of M1
dectre(xtrain(:,1:3), xtrain(:,4), xval(:,1:3), xval(:,4), ttrain, tval,  'Acceleration',
'Velocity', 'M1')
%using Displacement to solve for Velocity of M2
dectre(xtrain(:,1:3), xtrain(:,5), xval(:,1:3), xval(:,5), ttrain, tval,  'Acceleration',
'Velocity', 'M2')
%using Displacement to solve for Velocity of M3
dectre(xtrain(:,1:3), xtrain(:,6), xval(:,1:3), xval(:,6), ttrain, tval,  'Acceleration',
'Velocity', 'M3')

%Displacement to Solve for Acceleration
%using Displacement to solve for Acceleration of M1
```

```
dectre(xtrain(:,1:3), xtrain(:,7), xval(:,1:3), xval(:,7), ttrain, tval,  'Displacement',
'Acceleration', 'M1')
%using Displacement to solve for Acceleration of M2
dectre(xtrain(:,1:3), xtrain(:,8), xval(:,1:3), xval(:,8), ttrain, tval,  'Displacement',
'Acceleration', 'M2')
%using Displacement to solve for Acceleration of M3
dectre(xtrain(:,1:3), xtrain(:,9), xval(:,1:3), xval(:,9), ttrain, tval,  'Displacement',
'Acceleration', 'M3')
```

Random Forest (LSBoost)

```
%Acceleration to Solve for Displacement
%using Acceleration to solve for Displacement of M1
ranforlsb(xtrain(:,7:9), xtrain(:,1), xval(:,7:9), xval(:,1), ttrain, tval,  'Acceleration',
'Displacement', 'M1')
%using Acceleration to solve for Displacement of M2
ranforlsb(xtrain(:,7:9), xtrain(:,2), xval(:,7:9), xval(:,2), ttrain, tval,  'Acceleration',
'Displacement', 'M2')
%using Acceleration to solve for Displacement of M3
ranforlsb(xtrain(:,7:9), xtrain(:,3), xval(:,7:9), xval(:,3),ttrain, tval,  'Acceleration',
'Displacement', 'M3')

%Acceleration to Solve for Velocity
%using Acceleration to solve for Velocity of M1
ranforlsb(xtrain(:,7:9), xtrain(:,4), xval(:,7:9), xval(:,4), ttrain, tval,  'Acceleration',
'Velocity', 'M1')
%using Acceleration to solve for Velocity of M2
ranforlsb(xtrain(:,7:9), xtrain(:,5), xval(:,7:9), xval(:,5), ttrain, tval,  'Acceleration',
'Velocity', 'M2')
%using Acceleration to solve for Velocity of M3
ranforlsb(xtrain(:,7:9), xtrain(:,6), xval(:,7:9), xval(:,6), ttrain, tval,  'Acceleration',
'Velocity', 'M3')

%Velocity to Solve for Displacement
%using Velocity to solve for Displacement of M1
ranforlsb(xtrain(:,4:6), xtrain(:,1), xval(:,4:6), xval(:,1), ttrain, tval,  'Velocity',
'Displacement', 'M1')
%using Velocity to solve for Displacement of M2
ranforlsb(xtrain(:,4:6), xtrain(:,2), xval(:,4:6), xval(:,2), ttrain, tval,  'Velocity',
'Displacement', 'M2')
%using Velocity to solve for Displacement of M3
ranforlsb(xtrain(:,4:6), xtrain(:,3), xval(:,4:6), xval(:,3), ttrain, tval,  'Velocity',
'Displacement', 'M3')

%Velocity to Solve for Acceleration
%using Velocity to solve for Acceleration of M1
ranforlsb(xtrain(:,4:6), xtrain(:,7), xval(:,4:6), xval(:,7), ttrain, tval,  'Velocity',
'Acceleration', 'M1')
%using Velocity to solve for Acceleration of M2
ranforlsb(xtrain(:,4:6), xtrain(:,8), xval(:,4:6), xval(:,8), ttrain, tval,  'Velocity',
'Acceleration', 'M2')
%using Velocity to solve for Acceleration of M3
```

```
ranforlsb(xtrain(:,4:6), xtrain(:,9), xval(:,4:6), xval(:,9), ttrain, tval,  'Velocity',
'Acceleration', 'M3')

%Displacement to Solve for Velocity
%using Displacement to solve for Velocity of M1
ranforlsb(xtrain(:,1:3), xtrain(:,4), xval(:,1:3), xval(:,4), ttrain, tval,  'Acceleration',
'Velocity', 'M1')
%using Displacement to solve for Velocity of M2
ranforlsb(xtrain(:,1:3), xtrain(:,5), xval(:,1:3), xval(:,5), ttrain, tval,  'Acceleration',
'Velocity', 'M2')
%using Displacement to solve for Velocity of M3
ranforlsb(xtrain(:,1:3), xtrain(:,6), xval(:,1:3), xval(:,6), ttrain, tval,  'Acceleration',
'Velocity', 'M3')

%Displacement to Solve for Acceleration
%using Displacement to solve for Acceleration of M1
ranforlsb(xtrain(:,1:3), xtrain(:,7), xval(:,1:3), xval(:,7), ttrain, tval,  'Displacement',
'Acceleration', 'M1')
%using Displacement to solve for Acceleration of M2
ranforlsb(xtrain(:,1:3), xtrain(:,8), xval(:,1:3), xval(:,8), ttrain, tval,  'Displacement',
'Acceleration', 'M2')
%using Displacement to solve for Acceleration of M3
ranforlsb(xtrain(:,1:3), xtrain(:,9), xval(:,1:3), xval(:,9), ttrain, tval,  'Displacement',
'Acceleration', 'M3')
```

Random Forest (Bag Method)

```
%Acceleration to Solve for Displacement
%using Acceleration to solve for Displacement of M1
ranforbag(xtrain(:,7:9), xtrain(:,1), xval(:,7:9), xval(:,1), ttrain, tval,  'Acceleration',
'Displacement', 'M1')
%using Acceleration to solve for Displacement of M2
ranforbag(xtrain(:,7:9), xtrain(:,2), xval(:,7:9), xval(:,2), ttrain, tval,  'Acceleration',
'Displacement', 'M2')
%using Acceleration to solve for Displacement of M3
ranforbag(xtrain(:,7:9), xtrain(:,3), xval(:,7:9), xval(:,3),ttrain, tval,  'Acceleration',
'Displacement', 'M3')

%Acceleration to Solve for Velocity
%using Acceleration to solve for Velocity of M1
ranforbag(xtrain(:,7:9), xtrain(:,4), xval(:,7:9), xval(:,4), ttrain, tval,  'Acceleration',
'Velocity', 'M1')
%using Acceleration to solve for Velocity of M2
ranforbag(xtrain(:,7:9), xtrain(:,5), xval(:,7:9), xval(:,5), ttrain, tval,  'Acceleration',
'Velocity', 'M2')
%using Acceleration to solve for Velocity of M3
ranforbag(xtrain(:,7:9), xtrain(:,6), xval(:,7:9), xval(:,6), ttrain, tval,  'Acceleration',
'Velocity', 'M3')

%Velocity to Solve for Displacement
%using Velocity to solve for Displacement of M1
ranforbag(xtrain(:,4:6), xtrain(:,1), xval(:,4:6), xval(:,1), ttrain, tval,  'Velocity',
```

```
'Displacement', 'M1')
%using Velocity to solve for Displacement of M2
ranforbag(xtrain(:,4:6), xtrain(:,2), xval(:,4:6), xval(:,2), ttrain, tval,  'Velocity',
'Displacement', 'M2')
%using Velocity to solve for Displacement of M3
ranforbag(xtrain(:,4:6), xtrain(:,3), xval(:,4:6), xval(:,3), ttrain, tval,  'Velocity',
'Displacement', 'M3')

%Velocity to Solve for Acceleration
%using Velocity to solve for Acceleration of M1
ranforbag(xtrain(:,4:6), xtrain(:,7), xval(:,4:6), xval(:,7), ttrain, tval,  'Velocity',
'Acceleration', 'M1')
%using Velocity to solve for Acceleration of M2
ranforbag(xtrain(:,4:6), xtrain(:,8), xval(:,4:6), xval(:,8), ttrain, tval,  'Velocity',
'Acceleration', 'M2')
%using Velocity to solve for Acceleration of M3
ranforbag(xtrain(:,4:6), xtrain(:,9), xval(:,4:6), xval(:,9), ttrain, tval,  'Velocity',
'Acceleration', 'M3')

%Displacement to Solve for Velocity
%using Displacement to solve for Velocity of M1
ranforbag(xtrain(:,1:3), xtrain(:,4), xval(:,1:3), xval(:,4), ttrain, tval,  'Acceleration',
'Velocity', 'M1')
%using Displacement to solve for Velocity of M2
ranforbag(xtrain(:,1:3), xtrain(:,5), xval(:,1:3), xval(:,5), ttrain, tval,  'Acceleration',
'Velocity', 'M2')
%using Displacement to solve for Velocity of M3
ranforbag(xtrain(:,1:3), xtrain(:,6), xval(:,1:3), xval(:,6), ttrain, tval,  'Acceleration',
'Velocity', 'M3')

%Displacement to Solve for Acceleration
%using Displacement to solve for Acceleration of M1
ranforbag(xtrain(:,1:3), xtrain(:,7), xval(:,1:3), xval(:,7), ttrain, tval,  'Displacement',
'Acceleration', 'M1')
%using Displacement to solve for Acceleration of M2
ranforbag(xtrain(:,1:3), xtrain(:,8), xval(:,1:3), xval(:,8), ttrain, tval,  'Displacement',
'Acceleration', 'M2')
%using Displacement to solve for Acceleration of M3
ranforbag(xtrain(:,1:3), xtrain(:,9), xval(:,1:3), xval(:,9), ttrain, tval,  'Displacement',
'Acceleration', 'M3')
```

Neural Network

```
% hidden layers
hl = [19 40];

%Acceleration to Solve for Displacement
%using Acceleration to solve for Displacement of M1
neunet(xtrain(:,7:9), xtrain(:,1), xval(:,7:9), xval(:,1), hl, ttrain, tval,  'Acceleration',
'Displacement', 'M1')
%using Acceleration to solve for Displacement of M2
neunet(xtrain(:,7:9), xtrain(:,2), xval(:,7:9), xval(:,2), hl, ttrain, tval,  'Acceleration',
```

```
'Displacement', 'M2')
%using Acceleration to solve for Displacement of M3
neunet(xtrain(:,7:9), xtrain(:,3), xval(:,7:9), xval(:,3), hl, ttrain, tval,  'Acceleration',
'Displacement', 'M3')

%Acceleration to Solve for Velocity
%using Acceleration to solve for Velocity of M1
neunet(xtrain(:,7:9), xtrain(:,4), xval(:,7:9), xval(:,4), hl, ttrain, tval,  'Acceleration',
'Velocity', 'M1')
%using Acceleration to solve for Velocity of M2
neunet(xtrain(:,7:9), xtrain(:,5), xval(:,7:9), xval(:,5), hl, ttrain, tval,  'Acceleration',
'Velocity', 'M2')
%using Acceleration to solve for Velocity of M3
neunet(xtrain(:,7:9), xtrain(:,6), xval(:,7:9), xval(:,6), hl, ttrain, tval,  'Acceleration',
'Velocity', 'M3')

%Velocity to Solve for Displacement
%using Velocity to solve for Displacement of M1
neunet(xtrain(:,4:6), xtrain(:,1), xval(:,4:6), xval(:,1), hl, ttrain, tval,  'Velocity',
'Displacement', 'M1')
%using Velocity to solve for Displacement of M2
neunet(xtrain(:,4:6), xtrain(:,2), xval(:,4:6), xval(:,2), hl, ttrain, tval,  'Velocity',
'Displacement', 'M2')
%using Velocity to solve for Displacement of M3
neunet(xtrain(:,4:6), xtrain(:,3), xval(:,4:6), xval(:,3), hl, ttrain, tval,  'Velocity',
'Displacement', 'M3')

%Velocity to Solve for Acceleration
%using Velocity to solve for Acceleration of M1
neunet(xtrain(:,4:6), xtrain(:,7), xval(:,4:6), xval(:,7), hl, ttrain, tval,  'Velocity',
'Acceleration', 'M1')
%using Velocity to solve for Acceleration of M2
neunet(xtrain(:,4:6), xtrain(:,8), xval(:,4:6), xval(:,8), hl, ttrain, tval,  'Velocity',
'Acceleration', 'M2')
%using Velocity to solve for Acceleration of M3
neunet(xtrain(:,4:6), xtrain(:,9), xval(:,4:6), xval(:,9), hl, ttrain, tval,  'Velocity',
'Acceleration', 'M3')

%Displacement to Solve for Velocity
%using Displacement to solve for Velocity of M1
neunet(xtrain(:,1:3), xtrain(:,4), xval(:,1:3), xval(:,4), hl, ttrain, tval,  'Acceleration',
'Velocity', 'M1')
%using Displacement to solve for Velocity of M2
neunet(xtrain(:,1:3), xtrain(:,5), xval(:,1:3), xval(:,5), hl, ttrain, tval,  'Acceleration',
'Velocity', 'M2')
%using Displacement to solve for Velocity of M3
neunet(xtrain(:,1:3), xtrain(:,6), xval(:,1:3), xval(:,6), hl, ttrain, tval,  'Acceleration',
'Velocity', 'M3')

%Displacement to Solve for Acceleration
%using Displacement to solve for Acceleration of M1
neunet(xtrain(:,1:3), xtrain(:,7), xval(:,1:3), xval(:,7), hl, ttrain, tval,  'Displacement',
'Acceleration', 'M1')
%using Displacement to solve for Acceleration of M2
```

```
neunet(xtrain(:,1:3), xtrain(:,8), xval(:,1:3), xval(:,8), hl, ttrain, tval,  'Displacement',
'Acceleration', 'M2')
%using Displacement to solve for Acceleration of M3
neunet(xtrain(:,1:3), xtrain(:,9), xval(:,1:3), xval(:,9), hl, ttrain, tval,  'Displacement',
'Acceleration', 'M3')
```

```
function [xtrain, xval, ttrain, tval] = partition(p, t, x)
k = length(t);
idx = randperm(k);
ttrain = t(idx(1:round(p*k)),:);
tval = t(idx(round(p*k)+1:end),:);
xtrain = x(idx(1:round(p*k)),:);
xval = x(idx(round(p*k)+1:end),:);
end

function solvelr = linreg(a, d, b, c, ttrain, tval, training, predicting, mass)
% linear regression model
% a - train data | d - predictor variable | b - test data

data = sortrows([ttrain,d]);

mdl = fitlm(a, d);
Y = feval(mdl,a);
trained = sortrows([ttrain, Y]);

yfit = feval(mdl,b);
tested = sortrows([tval, yfit]);

RMSE = sqrt(mean((Y - d).^2));

R2_trained = corr(Y, d).^2;
R2_tested = corr(yfit, c).^2;

figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(data(:,2), trained(:,2),'.b', 'MarkerSize',25)
hold on
plot(data(:,2), data(:,2), 'k', 'Linewidth', 3)
axis equal
daspect([1 1 1])
grid on
set(gca,'FontSize',22)
legend('','perfect model')
xlabel('Data')
ylabel('Model Prediction')
title('Linear Regression')
subtitle(["Trained with "+training+" to Predict "+predicting+" of "+mass+"",""])
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))
```

```
figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(data(:,1), data(:,2),'b', 'Linewidth', 3)
hold on
plot(trained(:,1), trained(:,2), 'r', 'Linewidth', 3)
hold on
plot(tested(:,1), tested(:,2),  'k.', 'MarkerSize',25, 'Linewidth', 3)
set(gca,'FontSize',22)
legend('data', 'training data', 'testing data');
xlabel('Time (s)')
if strcmp(predicting,'Displacement')
    y = 'Displacement (m)';
elseif strcmp(predicting,'Velocity')
    y = 'Velocity (m/s)';
elseif strcmp(predicting, 'Acceleration')
    y = 'Acceleration (m/s^2)';
end
ylabel(""+y+"")
title('Linear Regression')
subtitle(['Trained with ',training,' to Predict ',predicting,' of ',mass,''])
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))

% PLOT FOR ENLARGED VIEW
figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(data(:,1), data(:,2),'b', 'Linewidth', 3)
hold on
plot(trained(:,1), trained(:,2), 'r', 'Linewidth', 3)
hold on
plot(tested(:,1), tested(:,2),  'k.', 'MarkerSize',25, 'Linewidth', 3)
set(gca,'FontSize',22)
legend('data', 'training data', 'testing data')
xlabel('Time (s)')
xlim([1 1.2])
if strcmp(predicting,'Displacement')
    y = 'Displacement (m)';
elseif strcmp(predicting,'Velocity')
    y = 'Velocity (m/s)';
elseif strcmp(predicting, 'Acceleration')
    y = 'Acceleration (m/s^2)';
end
ylabel(""+y+"")
title('Linear Regression')
subtitle(['Trained with ',training,' to Predict ',predicting,' of ',mass,''])
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))
snapnow
close all
end

function solvedt = dectre(a, d, b, c, ttrain, tval, training, predicting, mass)
```

```
% decision tree model
% a - train data | d - predictor variable | b - test data

data = sortrows([ttrain,d]);

mdl = fitrtree(a, d);

Y = predict(mdl,a);
trained = sortrows([ttrain, Y]);

yfit = predict(mdl,b);
tested = sortrows([tval, yfit]);

RMSE = sqrt(mean((Y - d).^2));

R2_trained = corr(Y, d).^2;
R2_tested = corr(yfit, c).^2;

figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(data(:,2), trained(:,2),'.b', 'MarkerSize',25)
hold on
plot(data(:,2), data(:,2), 'k', 'Linewidth', 3)
axis equal
daspect([1 1 1])
grid on
set(gca,'FontSize',22)
legend('','perfect model')
xlabel('Data')
ylabel('Model Prediction')
title('Decision Tree')
subtitle(["Trained with "+training+" to Predict "+predicting+" of "+mass+"",""])
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))

figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(data(:,1), data(:,2),'b', 'Linewidth', 3)
hold on
plot(trained(:,1), trained(:,2), 'r', 'Linewidth', 3)
hold on
plot(tested(:,1), tested(:,2),  'k.', 'MarkerSize',25, 'Linewidth', 3)
set(gca,'FontSize',22)
legend('data', 'training data', 'testing data');
xlabel('Time (s)')
if strcmp(predicting,'Displacement')
    y = 'Displacement (m)';
elseif strcmp(predicting,'Velocity')
    y = 'Velocity (m/s)';
elseif strcmp(predicting, 'Acceleration')
    y = 'Acceleration (m/s^2)';
end
ylabel(""+y+"")
title('Decision Tree')
```

```
subtitle(['Trained with ',training,' to Predict ',predicting,' of ',mass,''])
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))

% PLOT FOR ENLARGED VIEW
figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(data(:,1), data(:,2),'b', 'Linewidth', 3)
hold on
plot(trained(:,1), trained(:,2), 'r', 'Linewidth', 3)
hold on
plot(tested(:,1), tested(:,2),  'k.', 'MarkerSize',25, 'Linewidth', 3)
set(gca,'FontSize',22)
legend('data', 'training data', 'testing data')
xlabel('Time (s)')
xlim([1 1.2])
if strcmp(predicting,'Displacement')
    y = 'Displacement (m)';
elseif strcmp(predicting,'Velocity')
    y = 'Velocity (m/s)';
elseif strcmp(predicting, 'Acceleration')
    y = 'Acceleration (m/s^2)';
end
ylabel(""+y+"")
title('Decision Tree')
subtitle(['Trained with ',training,' to Predict ',predicting,' of ',mass,''])
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))
snapnow
close all
end

function solverflsb = ranforlsb(a, d, b, c, ttrain, tval, training, predicting, mass)
% random forest model (lsboost method)
% a - train data | d - predictor variable | b - test data

data = sortrows([ttrain,d]);

mdl = fitrensemble(a, d, 'NumLearningCycles', 500, 'LearnRate',0.55246);

Y = predict(mdl,a);
trained = sortrows([ttrain, Y]);

yfit = predict(mdl,b);
tested = sortrows([tval, yfit]);

RMSE = sqrt(mean((Y - d).^2));

R2_trained = corr(Y, d).^2;
R2_tested = corr(yfit, c).^2;
```

```
figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(data(:,2), trained(:,2),'.b', 'MarkerSize',25)
hold on
plot(data(:,2), data(:,2), 'k', 'Linewidth', 3)
axis equal
daspect([1 1 1])
grid on
set(gca,'FontSize',22)
legend('','perfect model')
xlabel('Data')
ylabel('Model Prediction')
title({'Random Forest','LSBoost Method'})
subtitle(["Trained with "+training+" to Predict "+predicting+" of "+mass+"",""])
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))

figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(data(:,1), data(:,2),'b', 'Linewidth', 3)
hold on
plot(trained(:,1), trained(:,2), 'r', 'Linewidth', 3)
hold on
plot(tested(:,1), tested(:,2),  'k.', 'MarkerSize',25, 'Linewidth', 3)
set(gca,'FontSize',22)
legend('data', 'training data', 'testing data');
xlabel('Time (s)')
if strcmp(predicting,'Displacement')
    y = 'Displacement (m)';
elseif strcmp(predicting,'Velocity')
    y = 'Velocity (m/s)';
elseif strcmp(predicting, 'Acceleration')
    y = 'Acceleration (m/s^2)';
end
ylabel(""+y+"")
title({'Random Forest','LSBoost Method'})
subtitle(['Trained with ',training,' to Predict ',predicting,' of ',mass,''])
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))

% PLOT FOR ENLARGED VIEW
figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(data(:,1), data(:,2),'b', 'Linewidth', 3)
hold on
plot(trained(:,1), trained(:,2), 'r', 'Linewidth', 3)
hold on
plot(tested(:,1), tested(:,2),  "k.", 'MarkerSize',25, 'Linewidth', 3)
set(gca,'FontSize',22)
legend('data', 'training data', 'testing data')
xlabel('Time (s)')
xlim([1 1.2])
if strcmp(predicting,'Displacement')
```

```
    y = 'Displacement (m)';
elseif strcmp(predicting,'Velocity')
    y = 'Velocity (m/s)';
elseif strcmp(predicting, 'Acceleration')
    y = 'Acceleration (m/s^2)';
end
ylabel(""+y+"")
title({'Random Forest','LSBoost Method'})
subtitle(['Trained with ',training,' to Predict ',predicting,' of ',mass,''])
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))
snapnow
close all
end

function solverfbag = ranforbag(a, d, b, c, ttrain, tval, training, predicting, mass)
% random forest model (bag method)
% a - train data | d - predictor variable | b - test data

data = sortrows([ttrain,d]);

mdl = fitrensemble(a, d,'Method', 'Bag', 'NumLearningCycles', 363);

Y = predict(mdl,a);
trained = sortrows([ttrain, Y]);

yfit = predict(mdl,b);
tested = sortrows([tval, yfit]);

RMSE = sqrt(mean((Y - d).^2));

R2_trained = corr(Y, d).^2;
R2_tested = corr(yfit, c).^2;

figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(data(:,2), trained(:,2),'.b', 'MarkerSize',25)
hold on
plot(data(:,2), data(:,2), 'k', 'Linewidth', 3)
axis equal
daspect([1 1 1])
grid on
set(gca,'FontSize',22)
legend('','perfect model')
xlabel('Data')
ylabel('Model Prediction')
title({'Random Forest','Bag Method'})
subtitle(["Trained with "+training+" to Predict "+predicting+" of "+mass+"","",""])
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))
```

```
figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(data(:,1), data(:,2),'b', 'Linewidth', 3)
hold on
plot(trained(:,1), trained(:,2), 'r', 'Linewidth', 3)
hold on
plot(tested(:,1), tested(:,2),  'k.', 'MarkerSize',25, 'Linewidth', 3)
set(gca,'FontSize',22)
legend('data', 'training data', 'testing data');
xlabel('Time (s)')
if strcmp(predicting,'Displacement')
    y = 'Displacement (m)';
elseif strcmp(predicting,'Velocity')
    y = 'Velocity (m/s)';
elseif strcmp(predicting, 'Acceleration')
    y = 'Acceleration (m/s^2)';
end
ylabel(""+y+"")
title({'Random Forest','Bag Method'})
subtitle(['Trained with ',training,' to Predict ',predicting,' of ',mass,''])
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))

% PLOT FOR ENLARGED VIEW
figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(data(:,1), data(:,2),'b', 'Linewidth', 3)
hold on
plot(trained(:,1), trained(:,2), 'r', 'Linewidth', 3)
hold on
plot(tested(:,1), tested(:,2),  "k.", 'MarkerSize',25, 'Linewidth', 3)
set(gca,'FontSize',22)
legend('data', 'training data', 'testing data')
xlabel('Time (s)')
xlim([1 1.2])
if strcmp(predicting,'Displacement')
    y = 'Displacement (m)';
elseif strcmp(predicting,'Velocity')
    y = 'Velocity (m/s)';
elseif strcmp(predicting, 'Acceleration')
    y = 'Acceleration (m/s^2)';
end
ylabel(""+y+"")
title({'Random Forest','Bag Method'})
subtitle(['Trained with ',training,' to Predict ',predicting,' of ',mass,''])
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))
snapnow
close all
end

function solvenn = neunet(a, d, b, c, hl, ttrain, tval, training, predicting, mass)
```

```
% neural network model
% a - training data | d - predictor variable | b - test data

data = sortrows([ttrain,d]);

net = fitnet(hl);
net = train(net, a', d');

Y = net(a');
trained = sortrows([ttrain, Y']);

yfit = net(b');
tested = sortrows([tval, yfit']);

RMSE = sqrt(mean((Y' - d).^2));

R2_trained = corr(Y', d).^2;
R2_tested = corr(yfit', c).^2;

figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(data(:,2), trained(:,2),'.b', 'MarkerSize',25)
hold on
plot(data(:,2), data(:,2), 'k', 'Linewidth', 3)
axis equal
daspect([1 1 1])
grid on
set(gca,'FontSize',22)
legend('','perfect model')
xlabel('Data')
ylabel('Model Prediction')
title('Neural Network')
subtitle(["Trained with "+training+" to Predict "+predicting+" of "+mass+"",""])
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))

figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(data(:,1), data(:,2),'b', 'Linewidth', 3)
hold on
plot(trained(:,1), trained(:,2), 'r', 'Linewidth', 3)
hold on
plot(tested(:,1), tested(:,2),  'k.', 'MarkerSize',25, 'Linewidth', 3)
set(gca,'FontSize',22)
legend('data', 'training data', 'testing data');
xlabel('Time (s)')
if strcmp(predicting,'Displacement')
    y = 'Displacement (m)';
elseif strcmp(predicting,'Velocity')
    y = 'Velocity (m/s)';
elseif strcmp(predicting, 'Acceleration')
    y = 'Acceleration (m/s^2)';
end
ylabel(""+y+"")
```

```
title('Neural Network')
subtitle(['Trained with ',training,' to Predict ',predicting,' of ',mass,''])
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))

% PLOT FOR ENLARGED VIEW
figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(data(:,1), data(:,2),'b', 'Linewidth', 3)
hold on
plot(trained(:,1), trained(:,2), 'r', 'Linewidth', 3)
hold on
plot(tested(:,1), tested(:,2),  'k.', 'MarkerSize',25, 'Linewidth', 3)
set(gca,'FontSize',22)
legend('data', 'training data', 'testing data')
xlabel('Time (s)')
xlim([1 1.2])
if strcmp(predicting,'Displacement')
    y = 'Displacement (m)';
elseif strcmp(predicting,'Velocity')
    y = 'Velocity (m/s)';
elseif strcmp(predicting, 'Acceleration')
    y = 'Acceleration (m/s^2)';
end
ylabel(""+y+"")
title('Neural Network')
subtitle(['Trained with ',training,' to Predict ',predicting,' of ',mass,''])
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))
snapnow
close all
end

function dxdt = integration(t,x, M, C, K, f1_a, f2_a, f3_a, f1_hz, f2_hz, f3_hz)
F = [f1_a*sin(f1_hz*2*pi*t);
     f2_a*cos(f2_hz*2*pi*t);
     f3_a*sin(f3_hz*2*pi*t)];

dxdt = zeros(size(x));
dxdt(1:3) = x(4:6);
dxdt(4:6) = M\( F - C*x(4:6) - K*x(1:3));
end
```

Appendix B – MATLAB Code for Static Channel Beam

```
% percent of data used for training
p = 0.8;
% hidden layers for neural network
hl = [50 12];

% point load of 100 N at 1 m
ml('SBpl100DD.txt', 'SBpl100ES.txt', p, 'Point Load at Full Length', hl)

% point load of 100 N at 0.75 m
ml('SBpl75DD.txt', 'SBpl75ES.txt', p, 'Point Load at 3/4 Length', hl)

% point load of 100 N at 0.5 m
ml('SBpl50DD.txt', 'SBpl50ES.txt', p, 'Point Load at 1/2 Length', hl)

% point load of 100 N at 0.25 m
ml('SBpl25DD.txt', 'SBpl25ES.txt', p, 'Point Load at 1/4 Length', hl)

% constant distributed load of 100 N/m
ml('SBcdlDD.txt', 'SBcdlES.txt', p, 'Constant Distributed Load', hl)

% linear distributed load of 100 Pa at 0 m to 0 Pa at 1 m
ml('SBldlDD.txt', 'SBldlES.txt', p, 'Linear Distributed Load ', hl)

% constant distributed load of 100 Pa at 0-0.5 m + linear distributed load of
% 100 Pa at 0.5 m to 0 Pa at 1 m
ml('SBcldlDD.txt', 'SBcldlES.txt', p, 'Constant and Linear Distributed Load', hl)

% parabolic distributed load 100 Pa at 0 m to 0 Pa at 1 m
ml('SBpdlDD.txt', 'SBpdlES.txt', p, 'Parabolic Distributed Load', hl)


function [dtrain, dval] = partition(p, x)
k = length(x);
idx = randperm(k);
dtrain = x(idx(1:round(p*k)),:);
dval = x(idx(round(p*k)+1:end),:);
end

function solvelr = linreg(a, d, b, c, load)
% linear regression model
% a - train data | d - predictor variable | b - test data

size(a,1)
mdl = fitlm(a, d);
Y = feval(mdl,a);

yfit = feval(mdl,b);

R2_trained = corr(Y, d).^2;
R2_tested = corr(yfit, c).^2;
fprintf('LR TRAINED %s for : %f\n', load, R2_trained)
```

```
fprintf('LR TESTED %s for : %f\n', load, R2_tested)

pltm(d, Y, 'Linear Regression', 'none', load)
plt(a, d, Y, b, c, yfit, 'X', 'Linear Regression', 'none', load)
plt(a, d, Y, b, c, yfit, 'Y', 'Linear Regression', 'none', load)
plt(a, d, Y, b, c, yfit, 'Z', 'Linear Regression', 'none', load)

end

function solvedt = dectre(a, d, b, c, load)
% decision tree model
% a - train data | d - predictor variable | b - test data

mdl = fitrtree(a, d);

Y = predict(mdl,a);
yfit = predict(mdl,b);

R2_trained = corr(Y, d).^2;
R2_tested = corr(yfit, c).^2;
fprintf('DT TRAINED %s for : %f\n', load, R2_trained)
fprintf('DT TESTED %s for : %f\n', load, R2_tested)

pltm(d, Y, 'Decision Tree', 'none', load)
plt(a, d, Y, b, c, yfit, 'X', 'Decision Tree', 'none', load)
plt(a, d, Y, b, c, yfit, 'Y', 'Decision Tree', 'none', load)
plt(a, d, Y, b, c, yfit, 'Z', 'Decision Tree', 'none', load)

end

function solverf = ranfor(a, d, b, c, load)
% random forest model (bag method)
% a - train data | d - predictor variable | b - test data

mdl = fitrensemble(a, d,'Method', 'Bag', 'NumLearningCycles', 493);

Y = predict(mdl,a);

yfit = predict(mdl,b);

R2_trained = corr(Y, d).^2;
R2_tested = corr(yfit, c).^2;
fprintf('RF TRAINED %s for : %f\n', load, R2_trained)
fprintf('RF TESTED %s for : %f\n', load, R2_tested)

pltm(d, Y, 'Random Forest', 'Bag Method', load)
plt(a, d, Y, b, c, yfit, 'X', 'Random Forest', 'Bag Method', load)
plt(a, d, Y, b, c, yfit, 'Y', 'Random Forest', 'Bag Method', load)
plt(a, d, Y, b, c, yfit, 'Z', 'Random Forest', 'Bag Method', load)

end

function solvenn = neunet(a, d, b, c, load, hl)
% neural network model
```

```
% a - training data | d - predictor variable | b - test data

net = fitnet(hl);
net = train(net, a', d');

Y = net(a');

yfit = net(b');

R2_trained = corr(Y', d).^2;
R2_tested = corr(yfit', c).^2;
fprintf('NN TRAINED %s for : %f\n', load, R2_trained)
fprintf('NN TESTED %s for : %f\n', load, R2_tested)

pltm(d, Y', 'Neural Network', 'none', load)
plt(a, d, Y', b, c, yfit', 'X', 'Neural Network', 'none', load)
plt(a, d, Y', b, c, yfit', 'Y', 'Neural Network', 'none', load)
plt(a, d, Y', b, c, yfit', 'Z', 'Neural Network', 'none', load)

end

function pltm = pltm(d, Y, model, method, load)
figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(d, Y,'.b', 'MarkerSize',25)
hold on
plot(d, d, 'k', 'Linewidth', 3)
axis equal
daspect([1 1 1])
grid on
set(gca,'FontSize',22)
legend('','perfect model')
xlabel('Data')
ylabel('Model Prediction')
if strcmp(method,'none')
    title([model])
else
    title({model,method})
end
subtitle([load])
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))

snapnow
close all
end

function plt = plt(a, d, Y, b, c, yfit, saxis, model, method, load)

if strcmp(saxis,'X')
    i = 1;
    j = 2;
    k = 3;
```

```
elseif strcmp(saxis,'Y')
    i = 2;
    j = 1;
    k = 3;
elseif strcmp(saxis,'Z')
    i = 3;
    j = 1;
    k = 2;
end

pltdata = [a,d,Y];

plta = pltdata(pltdata(:,j)==0,:);
pltb = plta(plta(:,k)==0,:);
pltfinal = sortrows(pltb,i);


testeddata = [b,c,yfit];

tplta = testeddata(testeddata(:,j)==0,:);
tpltb = tplta(tplta(:,k)==0,:);
tpltfinal = sortrows(tpltb,i);

figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(pltfinal(:,i), pltfinal(:,5), 'b', 'LineWidth', 3)
hold on
plot(pltfinal(:,i), pltfinal(:,6), 'r', 'LineWidth', 3)
hold on
plot(tpltfinal(:,i), tpltfinal(:,6),  'k.', 'MarkerSize',25, 'Linewidth', 3)
set(gca,'FontSize',22)
if strcmp(method,'none')
    title([model])
else
    title({model,method})
end
subtitle(['Stress Along ',saxis,' Axis with ',load,''])
xlabel(['',axis,' Axis (m)'])
ylabel('Stress (Pa)')
legend('data','trained data', 'testing data')
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))

if strcmp(saxis,'Z')
    figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
    plot(pltfinal(:,i), pltfinal(:,5), 'b', 'LineWidth', 3)
    hold on
    plot(pltfinal(:,i), pltfinal(:,6), 'r', 'LineWidth', 3)
    hold on
    plot(tpltfinal(:,i), tpltfinal(:,6),  'k.', 'MarkerSize',25, 'Linewidth', 3)
    set(gca,'FontSize',22)
    if strcmp(method,'none')
        title([model])
```

```
    else
        title({model,method})
    end
    subtitle(['Stress Along ',saxis,' Axis with ',load,''])
    xlim([0.5 0.6])
    xlabel(['',axis,' Axis (m)'])
    ylabel('Stress (Pa)')
    legend('data','trained data', 'testing data')
    hold on
    set(gca,'FontSize',35)
    set(gca,'fontname','times')
    set(gca,'linewidth',2)
    set(gca,'LooseInset',get(gca,'TightInset'))
end

snapnow
close all
end

function learn = ml(deformation, stress, p, load, hl)
A = importdata(deformation);
B = importdata(stress);

%   x | y | z | deformation | stress
data = [A.data(:,2:5), B.data(:,5)];
assignin('base','data',data)

% percentage of data to be used for training
[dtrain, dval] = partition(p, data);

a = dtrain(:,1:4);
d = dtrain(:,5);
b = dval(:,1:4);
c = dval(:,5);

linreg(a, d, b, c, load)
dectre(a, d, b, c, load)
ranfor(a, d, b, c, load)
neunet(a, d, b, c, load, hl)
end
```

Appendix C – MATLAB Code for Static Wing

```
% percent of data used for training
p = 0.8;
% hidden layers for neural network
hl = [23 22];

% point load of 10 N at Full Length
ml('Wpl10DD.txt','Wpl10ES.txt','Wpl10DDpath.txt','Wpl10ESpath.txt', p, 'Point Load at Full
Length', hl)

% constant distributed load of 10 N/m
ml('WcdlDD.txt','WcdlES.txt','WcdlDDpath.txt','WcdlESpath.txt', p, 'Constant Distributed Load',
hl)

% linear pressure of 10 Pa at 0 m to 0 Pa at full length
ml('WlpDD.txt','WlpES.txt','WlpDDpath.txt','WlpESpath.txt', p, 'Linear Pressure', hl)

% constant pressure of 10 Pa at 0-0.5 m + linear pressure of
% 10 Pa at 0.5 m to 0 Pa at full length
ml('WclpDD.txt','WclpES.txt','WclpDDpath.txt','WclpESpath.txt', p, 'Constant and Linear
Pressure', hl)

% parabolic pressure 10 Pa at 0 m to 0 Pa at full length
ml('WppDD.txt','WppES.txt','WppDDpath.txt','WppESpath.txt', p, 'Parabolic Pressure', hl)

% elliptical pressure 10 Pa at 0 m to 0 Pa at full length
ml('WepDD.txt','WepES.txt','WepDDpath.txt','WepESpath.txt', p, 'Elliptical Pressure', hl)

function [dtrain, dval] = partition(p, x)
k = length(x);
idx = randperm(k);
dtrain = x(idx(1:round(p*k)),:);
dval = x(idx(round(p*k)+1:end),:);
end

function solvelr = linreg(a, d, b, c, pdata, load)
% linear regression model
% a - train data | d - predictor variable | b - test data

mdl = fitlm(a, d);
Y = feval(mdl,a);

yfit = feval(mdl,b);

R2_trained = corr(Y, d).^2;
R2_tested = corr(yfit, c).^2;
fprintf('LR TRAINED %s for : %f\n', load, R2_trained)
fprintf('LR TESTED %s for : %f\n', load, R2_tested)

pltm(d, Y, 'Linear Regression', 'none', load)
plt(a, d, Y, b, c, pdata, yfit, 'Linear Regression', 'none', load)
```

```
end

function solvedt = dectre(a, d, b, c, pdata, load)
% decision tree model
% a - train data | d - predictor variable | b - test data

mdl = fitrtree(a, d);

Y = predict(mdl,a);
yfit = predict(mdl,b);

R2_trained = corr(Y, d).^2;
R2_tested = corr(yfit, c).^2;
fprintf('DT TRAINED %s for : %f\n', load, R2_trained)
fprintf('DT TESTED %s for : %f\n', load, R2_tested)

pltm(d, Y, 'Decision Tree', 'none', load)
plt(a, d, Y, b, c, pdata, yfit, 'Decision Tree', 'none', load)

end

function solverf = ranfor(a, d, b, c, pdata, load)
% random forest model (bag method)
% a - train data | d - predictor variable | b - test data

mdl = fitrensemble(a, d,'Method', 'Bag', 'NumLearningCycles', 309);

Y = predict(mdl,a);

yfit = predict(mdl,b);

R2_trained = corr(Y, d).^2;
R2_tested = corr(yfit, c).^2;
fprintf('RF TRAINED %s for : %f\n', load, R2_trained)
fprintf('RF TESTED %s for : %f\n', load, R2_tested)

pltm(d, Y, 'Random Forest', 'Bag Method', load)
plt(a, d, Y, b, c, pdata, yfit, 'Random Forest', 'Bag Method', load)

end

function solvenn = neunet(a, d, b, c, pdata, load, hl)
% neural network model
% a - training data | d - predictor variable | b - test data

net = fitnet(hl);
net = train(net, a', d');

Y = net(a');

yfit = net(b');

R2_trained = corr(Y', d).^2;
R2_tested = corr(yfit', c).^2;
```

```
fprintf('NN TRAINED %s for : %f\n', load, R2_trained)
fprintf('NN TESTED %s for : %f\n', load, R2_tested)

pltm(d, Y', 'Neural Network', 'none', load)
plt(a, d, Y', b, c, pdata, yfit', 'Neural Network', 'none', load)
end

function pltm = pltm(d, Y, model, method, load)
figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(d, Y,'.b', 'MarkerSize',25)
hold on
plot(d, d, 'k', 'Linewidth', 3)
axis equal
daspect([1 1 1])
grid on
set(gca,'FontSize',22)
legend('','perfect model')
xlabel('Data')
ylabel('Model Prediction')
if strcmp(method,'none')
    title([model])
else
    title({model,method})
end
subtitle([load])
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))

snapnow
close all
end

function plt = plt(a, d, Y, b, c, pdata, yfit, model, method, load)

ploted = [a,d,Y];
idx = find(ismember(ploted(:,1:3),pdata(:,1:3),'rows'));
pltdata = sortrows(ploted(idx,:),3);

tested = [b,c,yfit];
idx = find(ismember(tested(:,1:3),pdata(:,1:3),'rows'));
testeddata = sortrows(tested(idx,:),3);

figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(pltdata(:,3), pltdata(:,5), 'b', 'LineWidth', 3)
hold on
plot(pltdata(:,3), pltdata(:,6), 'r', 'LineWidth', 3)
hold on
plot(testeddata(:,3), testeddata(:,6),  'k.', 'MarkerSize',25, 'Linewidth', 3)
set(gca,'FontSize',22)
if strcmp(method,'none')
    title([model])
else
```

```
    title({model,method})
end
subtitle(['Stress Along Path on Wing with ',load,''])
xlabel(['Z Coordiante (m)'])
ylabel('Stress (Pa)')
legend('data','trained data', 'testing data')
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))

% enlarged view
figure('Units', 'pixels', 'Position', [0 0 1920 1080]);
plot(pltdata(:,3), pltdata(:,5), 'b', 'LineWidth', 3)
hold on
plot(pltdata(:,3), pltdata(:,6), 'r', 'LineWidth', 3)
hold on
plot(testeddata(:,3), testeddata(:,6),  'k.', 'MarkerSize',25, 'Linewidth', 3)
set(gca,'FontSize',22)
if strcmp(method,'none')
    title([model])
else
    title({model,method})
end
subtitle(['Stress Along Path on Wing with ',load,''])
xlim([0.5 0.7])
xlabel(['Z Coordinate (m)'])
ylabel('Stress (Pa)')
legend('data','trained data', 'testing data')
set(gca,'FontSize',35)
set(gca,'fontname','times')
set(gca,'linewidth',2)
set(gca,'LooseInset',get(gca,'TightInset'))

snapnow
close all
end

function learn = ml(deformation, stress, pd, ps, p, load, hl)

fA = importdata(deformation);
fB = importdata(stress);

pA = importdata(pd);
pB = importdata(ps);

%   x | y | z | deformation | stress
fdata = [fA.data(:,2:5), fB.data(:,5)];
pdata = [pA.data(:,2:5), pB.data(:,5)];
data = [fdata;pdata];

% percentage of data to be used for training
[dtrain, dval] = partition(p, data);
```

```
a = dtrain(:,1:4);
d = dtrain(:,5);
b = dval(:,1:4);
c = dval(:,5);

linreg(a, d, b, c, pdata, load)
dectre(a, d, b, c, pdata, load)
ranfor(a, d, b, c, pdata, load)
neunet(a, d, b, c, pdata, load, hl)
end
```