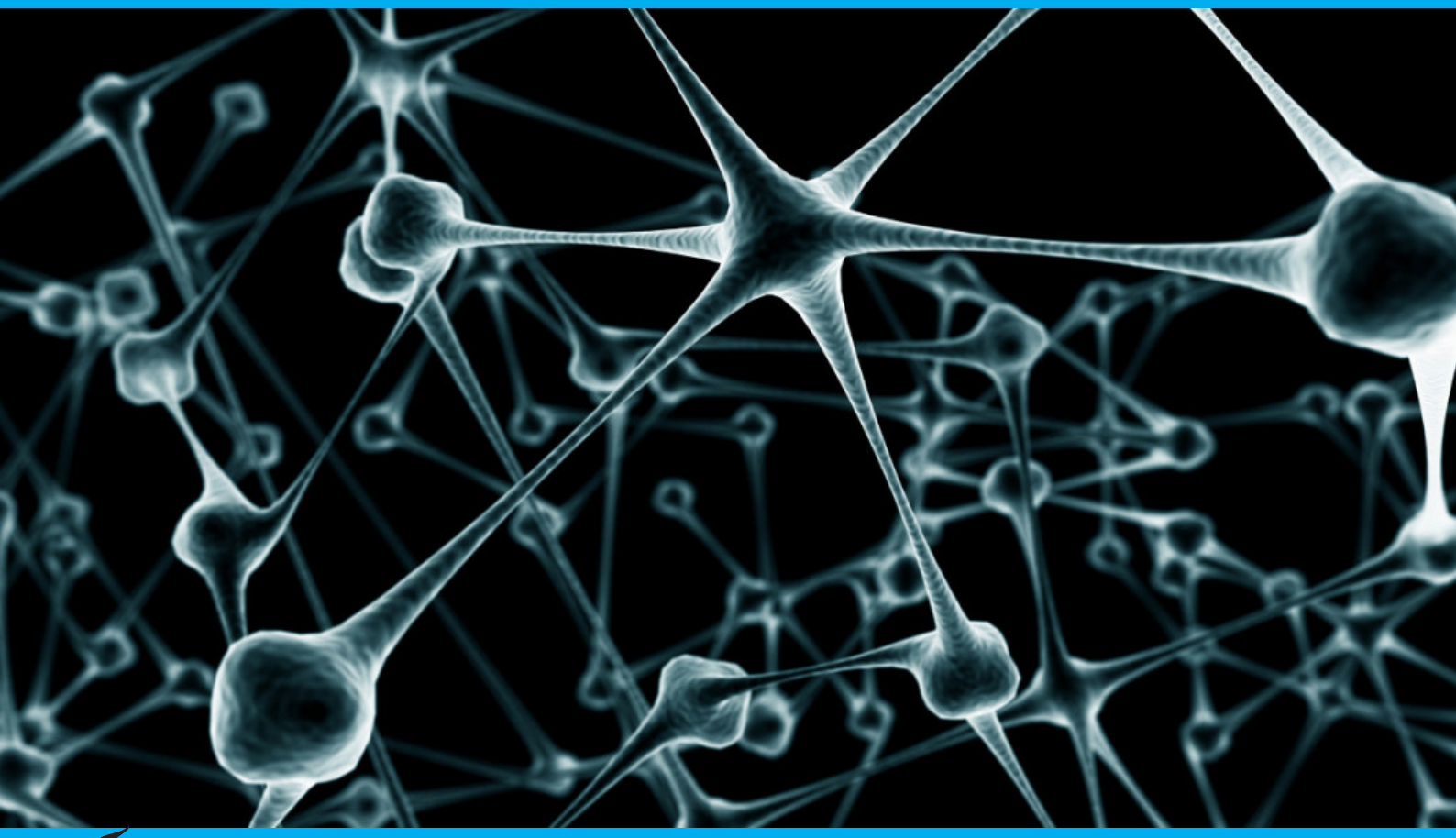
Predicting results of geometrical nonlinear FE analyses using Artificial Neural Networks

Applied to stiffened steel plated structures in sea lock gates

T. Verhoog

Master thesis

ISBN 000-00-0000-000-0



TU Delft

Predicting results of geometrical nonlinear FE analyses using Artificial Neural Networks

Applied to stiffened steel plated structures in sea lock gates

by

T. Verhoog

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Friday November 22, 2019 at 13:00.

Student number:	4248082	
Project duration:	March 1, 2019 – November 22, 2019	
Thesis committee:	Prof. dr. ir. J.G. Rots,	TU Delft, chair committee
	Dr. ir. P.C.J. Hoogenboom,	TU Delft, supervisor
	Dr. ir. R. Abspoel,	TU Delft
	Ir. P.J.C. van Lierop,	Iv-Infra B.V.

TU Delft

Preface

In this report I present the work I have carried out for obtaining the MSc degree Civil Engineering at Delft University of Technology. The past period of working out my MSc project has sometimes been a rough, but always been a rewarding experience. Diving deep into new topics like machine learning was challenging, yet very interesting. Apart from gaining experience in finite element modelling and structural mechanics, I also learned ton of new things about machine learning and programming in general, which I really enjoyed.

So first of all I would like to thank Pieter van Lierop, for having the innovative and open mindset and giving me the trust and the freedom to explore my own ideas and path. Special thanks for Pierre Hoogenboom, for being a helpful supervisor who could help me to steer into the right direction. Thank you Jan Rots and Roland Abspoel for taking part in my graduation committee and thereby showing your curiosity and confidence in my project.

Also many thanks to Yirui Yao, my personal ANSYS expert to whom I could ask all my technical questions. Dennis Alsemgeest, thank you for lending your ear and inspiring me when I needed it.

Of course an infinite amount of gratitude to my mother Heleen, who has always supported me in all the decisions I took in life and for giving me the opportunity to go to college. And finally, above all I want to thank my wonderful girlfriend Freya, for being the best personal mental coach I can imagine.

Oh, and not to forget my friend Dave, who always challenged me to explain my thesis topic to other people in three sentences.

T. Verhoog
Delft, October 2019

Contents

List of Figures

List of Tables

Summary

In this research project, an attempt is made to fuse the fields of structural mechanics and machine learning. The goal is to find out if models can be created that are capable of predicting the outcomes of (nonlinear) finite element analyses. These models are created by means of Artificial Neural Networks, which is a powerful method in the domain of machine learning. The focus will be on stiffened steel plated structures that are part of a sea lock gate.

The power of a trained neural network is that it is able to compute the output for a given set of input parameters within a fraction of a second. Running a complete finite element analysis on the other hand can take a significant amount of time, especially in case of geometrically and/or physically nonlinear analyses. When relying on nonlinear finite element analyses for performing a structural design optimization, a trained network can therefore save a huge amount of time. It also allows to evaluate many more design options, possibly finding a more optimal design than what would be possible with a manual design optimization.

An automated procedure has been created to generate datasets by running FE analyses in batch mode. Parametric models are set up in ANSYS FE software, for which random sets of input parameters are generated. After running the analyses, the output is collected and organised in datasets that can be used for training. The sizes of the datasets and the dimensionality of the design spaces are varied in order to study the influence of these quantities on the accuracies of the predictions produced by the neural networks.

Genetic algorithms, which is another machine learning technique, are deployed for the optimization of the *hyperparameters* of the neural networks, which are basically the *settings* of the network which determine the learning behaviour. Three standard interpolation techniques (Kriging and polynomial interpolation) are also fitted to the same datasets in order to compare the performance of the neural networks to these interpolation techniques.

The final result is an overview of the accuracies of the predictions made with the neural networks on validation datasets. It was found that the neural networks produced accurate predictions on the maximum deflection of a simply supported, stiffened steel plate loaded by a uniform pressure. Most of the relative errors were within a range of 5% error for design problems with 4 dimensions. Predictions of the linear buckling load of stiffened steel plates were found to be mostly within the range of 10% error for design problems with 4 dimensions. When increasing the number of free design variables from 4 to 8, the errors were found to be mostly within the range of 20% error.

The predictions of maximum equivalent stresses in stiffened steel plated sections obtained by geometrically nonlinear FE analyses were found to be in the range of ± 20%. These models had 13 free design variables. Improvements were made on the available options of hyperparameters for neural networks. With these improvements, new predictions were made on the maximum equivalent stresses and the accuracies were found to be slightly better, with errors ranging between ± 15%. For implementation of the predictive model in a design optimization algorithm, these errors are considered to be too high. It is expected that the dimensionality of this problem (13 design parameters) combined with very irregular results due to the presence of peak stresses and different buckling shapes, resulted in these deviations.

Additional datasets are generated with results of FE analyses of simplified, unstiffened steel plates. Both geometrically and physically nonlinear analyses are performed, with uni-directional compression applied to the plate edge directly as a displacement. The number of free design variables was set equal to either 1 or 4 free variables. It was found that even with small datasets (with 32 training samples), the neural networks produced very accurate results on predicting maximum equivalent stresses, maximum equivalent mechanical strains (in case of physically nonlinear analyses) and total reaction force. A neural network was found to be capable of producing a nonlinear load-displacement curve of a compressed rectangular plate with elastic-plastic material model.

1 Introduction

1.1. Background and motivation

In recent years, Iv-Infra B.V. has been responsible for the structural design of several new sea lock gates like the ones used for the new Panama Canal. One important design aspect for this type of steel structures is the buckling stability of the stiffened skin plates under the combined action of out of plane water pressure and in plane compressive stress. The buckling stability of these stiffened skin plates is usually assessed by means of geometrically non-linear finite element analyses (GNLA).

In the early design stages, quick decisions on the dimensioning of the structure have to be made. Choices made in this initial phase have a significant influence on the following design phases. The flexibility of making major changes in the design decreases with time. In order to come to an optimal design in terms of e.g. material use, it is very important to find optimal dimensions early in the design process.

An approach to finding an optimal design would require evaluation of many design options from which the best design can be selected. As mentioned, these designs are normally evaluated using GNLA. While such analyses are accurate, it is also a time consuming process, which is preferably avoided in early design stages. Repeating this process for multiple design options therefore obviously is not very efficient.

Predictive modelling using Machine Learning The time needed for a design optimization would be dramatically decreased if there would exist an explicit formula for obtaining the buckling resistance of the steel structure, given a set of input parameters. Unfortunately, the complex behaviour of such a specific structure cannot be captured in an simple analytical formula.

This is where Machine Learning can come into play. The main idea of Machine Learning (ML) is to create a predictive model that maps a set of output parameters to a provided set of input parameters *without* explicitly programming how the model must manipulate these data. Instead, the ML model *learns* the relation between input and output by experience. The theory of ML is extensively discussed in Chapter 2.

The main advantage of a predictive model, trained by means of a Machine Learning algorithm, is that it is able to compute the desired output in a fraction of a second, rather than running a computationally expensive FE analysis. In the case of the computation of the buckling resistance of the stiffened skin plate, the user would not need to set up and run a complete FE analysis. Instead, the desired output can be directly obtained by providing the input values to the trained model.

This reduction in computational cost could provide great benefits for use in a design optimization algorithm. Since the required amount of computational time of the trained predictive model is very low, many design options can be evaluated in a relatively short period of time compared to the full FEA. In this way the predictive model can play a key role in finding an optimal design quickly.

The drawback is that the generation of the training data still requires running a number of FE analyses. One should aim to generate a dataset large enough to ensure accurate predictions of the predictive model, but also as small as possible to limit the computational cost.

1.2. Objectives

The main goal of this research is to find out to what extend it is possible to create a model that is able to accurately predict the output results of FE analyses. The focus will be on stiffened steel plated structures as a submodel of a steel lock gate. These predictive models will be created by means of Artificial Neural Networks (ANN's). The performance of these ANN's are evaluated and compared with several well-known interpolation techniques, in order to draw conclusions about whether ANN's can properly predict the outcomes of the FE analyses. This predictive model can serve as a building block for use in a design optimization algorithm.

Research questions

1. Can the results of linear and/or nonlinear finite element analyses be accurately predicted by means of Artificial Neural Networks?
2. How does the performance of these neural networks compare to the performance of several well known interpolation techniques
3. Is it worth the investment of time and computational resources to create such a predictive model?
4. How does the accuracy of the predictions relate to the number of training samples and the complexity of the model?

1.3. Scope

In this section the most important limitations of the research are listed. Notes about further limitations of the research are added in the relevant sections in this paper where necessary.

Machine learning methods There are many different Machine Learning techniques available. Each technique has its strengths and weaknesses. In this research, only Artificial Neural Networks will be used for creating a predictive model. Artificial Neural Networks are capable of capturing highly non-linear behaviour of a system, when provided sufficient training data (see theory section in Chapter 2).

In order to find the optimal 'settings' for these neural networks, Genetic Algorithms are deployed for 'hyperparameter optimization'. A Genetic Algorithm (GA) is also a machine learning algorithm. However, this algorithm is *not* used as a predictive model, instead it will *only* be used for finding the optimal hyperparameters for an ANN.

Design optimization As mentioned in Section 1.1, the predictive model can be used for quick design optimizations. In this context, the predictive model serves as the 'explicit' formula for finding the resistance of the structure. The creation of such a design optimization algorithm however is *not* included within the scope of this research. Recommendations considering this point are added to the recommendations in Chapter 7.

1.4. Report outline

The report starts with a general introduction to machine learning theory in Chapter 2. In this chapter, the theories of Artificial Neural Networks and Genetic Algorithms are explained. Chapter 3 discusses the properties of the main mechanical model which is analysed. All steps and considerations for creating the FE model are included in this chapter. In Chapter 4, the complete process from start to finish for the creation and validation of predictive models will be treated. It starts with the creation of parametric FE models which are used to generate training data. Designs with random sets of input parameters are generated and analyzed in the FE program. The results are used to train the ANN's. The process of the optimization of these ANN's with Genetic Algorithms is also discussed. Chapter 5 summarizes the results of the predictions made with the neural networks on the data. The accuracies of these predictions are compared with some well known, standard interpolation techniques. The conclusions of the project can be found in Chapter 6, after which in Chapter 7, an overview is given of the recommendations by the author for future work.

2

Machine Learning theory

This chapter will cover the basic theory behind the machine learning methods that are used in this project.

Section 2.1 will give a very brief introduction to the general concept of machine learning. In Section 2.2 the theory behind Artificial Neural Networks will be discussed. The theory behind Genetic Algorithms is discussed in Section 2.3.

This chapter only discusses the theoretical backgrounds of these algorithms. The actual application of the algorithms in this project are described in the relevant sections of Chapter 4.

2.1. Introduction to Machine Learning

Machine learning (ML) is a subdomain of the broader field of Artificial Intelligence. ML is concerned with the creation of algorithms that allow a computer to learn certain tasks from experience. Rather than explicitly programming those tasks in a set of 'concrete' rules, the computer learns to perform the tasks by itself. This is incredibly useful in cases where the underlying relations of a (physical) process are unknown, or too complex to capture in a mathematical model.

Hand-written digit recognition A well-known example of a machine learning application is hand written digit recognition. See Figure 2.1 for some examples of hand written digits that were scanned and saved to a computer. Each digit is represented by a 28 x 28 pixel image, each pixel having a grayscale value between 0 and 1.

Now imagine that a programmer would have to write a program that takes an image of a hand-written digit as shown in Figure 2.1 as input, and the program must tell which digit it truly is. The input is represented by $28 \times 28 = 784$ grayscale values between 0 and 1. A set of rules would have to be written that takes these input values and tell whether the provided number is e.g. a 6. This would be an impossible task, especially considering that each hand-written digit is unique. Imagine writing a rule based program if the scale of complexity would be increased even further to e.g. object recognition in coloured pictures. This would not be feasible.

That is where machine learning comes into play. This is a classic example of a *classification* task. Two major types of tasks are generally performed by trained ML models

- Classification is the task of assigning a discrete label or category to the input.
- Regression is the task of mapping one or more continuous value(s) to the given input.

Classification Computer programs can be trained to perform classification tasks in numerous fields of practice. One famous example is the recognition of handwritten digits. Given a sufficient number of training examples with handwritten digits, labelled with the correct integer value corresponding to that digit, the computer can be trained to recognize hand written digits. The goal is to enable the program to recognize hand written digits it has never seen before during the training process. The accuracy of such a program will

never be 100%. It is up to the ML practitioner to decide on the acceptable accuracy and to set up the ML algorithm such that this accuracy can be achieved.

Regression Regression tasks aim at approximating a continuous output value, given a set of input values. One well known educational example for regression tasks is the prediction of house prices in Boston. Based on the dataset containing several features (input variables) like e.g. crime rate, number of rooms and nitric oxide concentration, combined with the actual house price of that property, the ML algorithm must be able to predict house prices as accurately as possible.

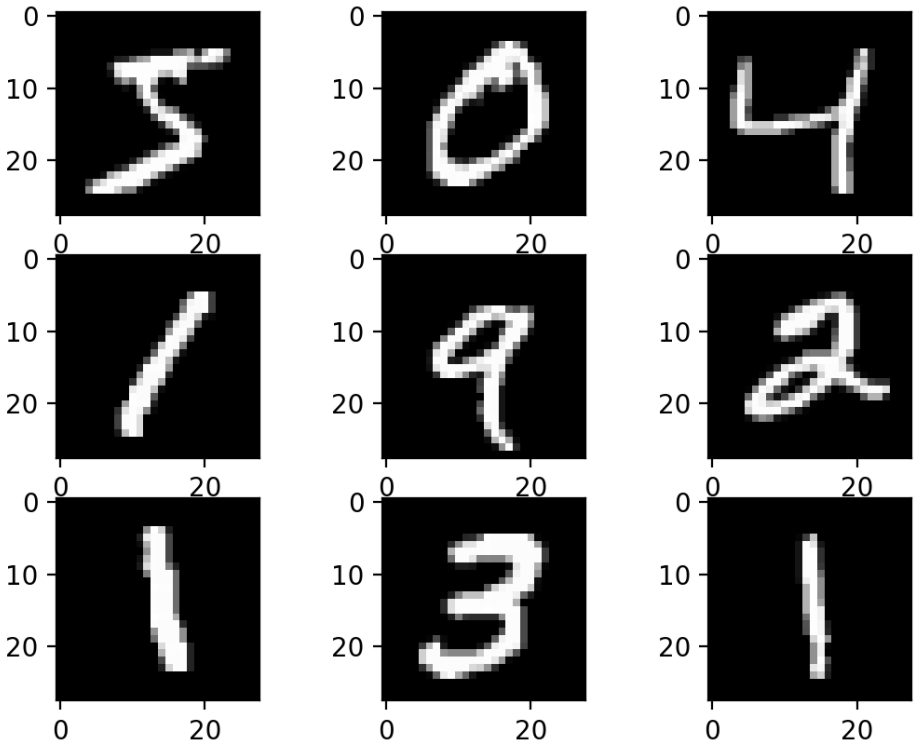


Figure 2.1: Images of hand written digits from the MNIST dataset [14]

2.2. Artificial Neural Networks

Artificial Neural Networks (ANN's) are a powerful tool within the domain of Machine Learning. The Artificial Neural Network is inspired by the (human) brain. The brain as found in nature consists of billions interconnected neurons which are capable of learning and performing very complex tasks. A human being is able to recognize a face of another person within a fraction of a second. Tasks like these feel very natural and easy to perform.

In this section, the theory of these artificial neural networks are breifly discussed. Many varieties of neural networks exist, but only the classical *feedforward neural network* model will be treated here.

The content in this section is mostly a synthesis from [19], combined with various internet sources.

Introduction The main goal of an ANN is to predict output values for a set of given input values. The power lies in the fact that the neural network is capable to predict on data which it has never seen during the training process. To verify this capability, the performance of the network must be evaluated on a set of test samples of the dataset that were not used for training. The dataset must be split in a set with training samples and

a set with testing samples. The training samples are used for training the network as described in Section 2.2.3. The testing set is used for evaluating the performance on unseen data. A network performs well if it *generalizes* well to unseen data. A longer training process does not necessarily mean that the network predicts better. *Overfitting* to data is a common problem in all machine learning algorithms, and must be avoided. Overfitting is the phenomenon where the artificial neural predicts very accurately on the training data, but at the cost of predicting very poorly on the testing data. In this way the trained network is useless since it is not able to generalize on unseen data.

2.2.1. Network architecture: The building blocks of the ANN

The Artificial Neural Network is a network constructed from a set of interconnected artificial *neurons*. The neurons are grouped in so-called *layers* of neurons.

The artificial neuron The artificial neuron, also called a 'node', is a so-called processing unit. See Figure 2.2 for a schematic representation of an artificial neuron. The neuron receives information from *each* neuron in the preceding layer of i neurons. Then, the neuron processes the data coming in and produces an output value. which is sent to all neurons in the next layer.

The output value a_i out of each preceding neuron is multiplied with a corresponding weight factor w_i. Inside the neuron, all values are summed up and a bias term b_i is added to this sum. Finally, the value of the sum $z = \sum_{i=1}^{n} a_i w_i + b$ is passed through an activation function $f(z)$. The output value $f(z)$ is sent to the next layer of neurons, following the same procedure again.

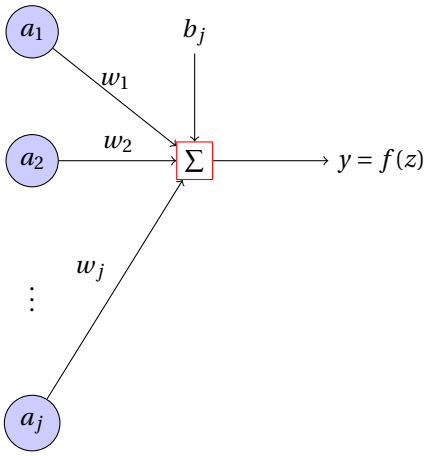


Figure 2.2: Schematic representation of a single artificial neuron processing information from its preceding neurons. The weighted output values of the previous neurons are summed up and a bias term is added. The sum is finally passed through a (non-linear) activation function.

Layers of neurons The Artificial Neural Network is composed of so-called *layers* of neurons. The basic structure of a feedforward artificial neural network consists of three types of layers: The input layer, a set of hidden layers and an output layer. See Figure 2.3 for an example of a *single-layer* neural network. The word single in the term single-layer network refers to the fact that the network consists of only one single *hidden* layer. This can be confusing since the network actually contains three layers. The input layer and output layer are always included in a neural network. Figure 2.4 shows an example of a multi-layer neural network containing two hidden layers.

Input values are entered into the neurons in the input layer. The number of neurons in the input layer is equal to the number of input variables, or *features* of the problem. In the input layer, the input values are *not* modified by an activation function. Therefore the output value of an input neuron is equal to the value of the actual data input value $a_i = x_i$. The values a_i are sent from the input layer to the first *hidden layer*.

The neurons in the hidden layers *do* modify the incoming data by means of an activation function. This activation function is usually a certain non-linear function which enables the network to predict the non-linear behaviour of the system.

The output layer finally produces the values to be predicted by the network. The number of neurons in the output layer is equal to the amount of output parameters to be predicted. The values passed through the output neurons can be modified by an activation function, but the motivation for this depends on the situation. In general, for regression tasks, the output values are passed through a linear activation function while for classification tasks, non-linear activation functions like the sigmoid function may be a good option.

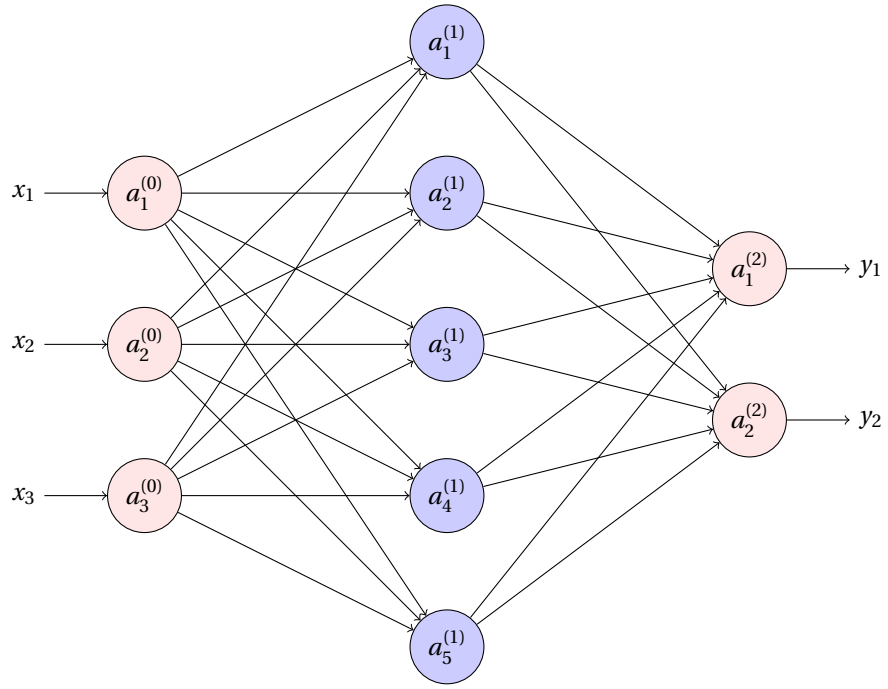


Figure 2.3: Example of single-layer network

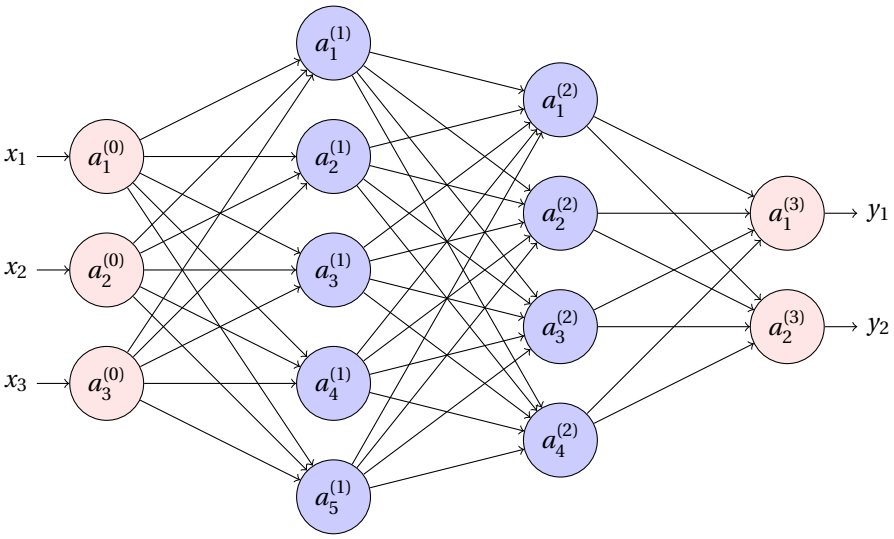


Figure 2.4: Example of multi-layer network

Weights and biases See Figure 2.4. Each individual arrow that forms a connection between two neurons, has a so-called *weight* associated with it. This weight is a continuous number, usually ranging between the values 0 and 1. The set of weights between each layer is represented as a *weight matrix*. The exact formulation of such a matrix is presented in Section 2.2.2. The neural network shown in Figure 2.4 thus has three weight matrices in total. Next to that, each neuron (except neurons in the input layer) has a *bias* term associated

with it. The bias term b is a scalar value that is added to the sum of the weighted input. The bias term enables the output of the activation function to be shifted horizontally, and thereby includes a certain threshold value influencing the strength of the output signal.

Upon construction of the neural network, the values of the weights and biases are initialized at random. At this point, the network is untrained and therefore the prediction accuracy of the network with these initial weights and biases will be very poor. The aim of the training process is to update all these weights and biases until the prediction accuracy is satisfactory. This iterative procedure is called the *training process* of the neural network and will be described in section Section 2.2.3.

2.2.2. Matrix notation

This section will provide the mathematical formulation of the process of mapping output values to a set of input parameters by a neural network.

Vector notation single neuron Refer to Figure 2.2 for the schematic representation of a single neuron. The vector notation corresponding to a single neuron is as follows. The input values a_j coming from a previous layer are represented in a row vector

$$a_j = \begin{bmatrix} a_1 & a_2 & \cdots & a_j \end{bmatrix} \tag{2.1}$$

The weight vector describing the weights between the actual layer and the previous layer are represented as a column vector.

$$w_j = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_j \end{bmatrix} \tag{2.2}$$

The z value of the neuron is the sum of the weighted input values and the added bias term b.

$$z = \begin{bmatrix} a_1 & a_2 & \cdots & a_j \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_j \end{bmatrix} + b = x_1 w_1 + x_2 w_2 + \cdots + x_j w_j + b \tag{2.3}$$

Weight matrix When the layer does not consist of a single neuron like in the example above, but multiple neurons, the weights are represented in a 2-dimensional matrix $W_{j,k}$ where j is the number of neurons in the previous layer $(L-1)$ and k is the number of neurons in the actual layer (L). See Figure 2.5 and Figure 2.6 for the visualization of the weights $w_{j,k}$ in the actual network. Not all connections are displayed for demonstration purposes. The resulting weight matrix between two layers has the shape as shown in (2.4).

$$W_{j,k} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,k} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ w_{j,1} & w_{j,2} & \cdots & w_{j,k} \end{bmatrix} \tag{2.4}$$

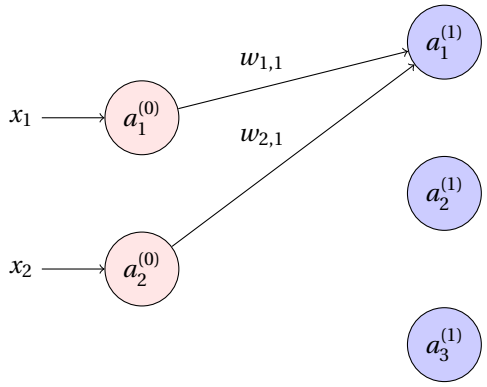


Figure 2.5: The indices of the weights corresponding to the connections of input neurons to the first neuron in the hidden layer. The first index corresponds to the node number in the preceding layer $(L-1)$.

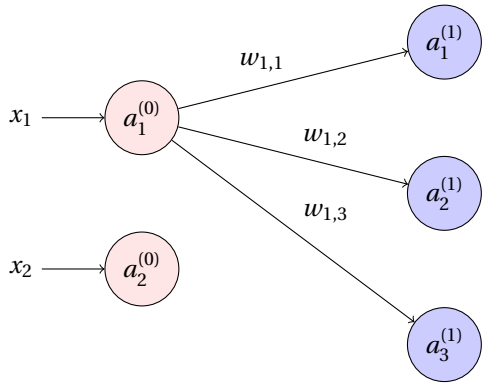


Figure 2.6: The indices of the weights corresponding to the connections of one input neuron to all neurons in the hidden layer. The second index corresponds to the node number of the actual layer (L)

Input data Until now, the input data was represented as a vector only, corresponding to a single data sample. In reality, the neural network is trained by a dataset consisting of many data samples. The actual dataset is an $i \times j$ matrix where i is the number of data samples and j is the number of features, or input variables, of a sample. The dataset is denoted as $X_{i,j}$

$$X_{i,j} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,j} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ x_{i,1} & x_{i,2} & \cdots & x_{i,j} \end{bmatrix} \tag{2.5}$$

Matrix multiplication The intermediate output matrix at layer (L) is computed defined as follows

$$A_{i,k}^{(L)} = f\left(A_{i,j}^{(L-1)} W_{j,k}^{(L)} + B_{i,k}^{(L)}\right) \tag{2.6}$$

The resulting activation matrix $A_{i,k}^{(L)}$ will serve as input again for the next layer $(L+1)$. This process is repeated for each layer until finally the output from the output layer is produced. The output matrix is a matrix $Y_{i,k}$ where i is the number of data samples and k is the number of neurons in the output layer, which is equal to the number of output parameters.

The notation for the different elements is summarized below

$A_{i,k}^{(L)}$: Activation matrix of layer (L)

$A_{i,j}^{(L)}$: Activation matrix of preceding layer $(L-1)$

$W_{j,k}^{(L)}$: Weight matrix for connections between layer $(L-1)$ and layer (L)

$b_i^{(L)}$: Bias vector for layer (L)

$f(z)$: Activation function

2.2.3. Training process of the network

The training process of a neural network is an iterative procedure with the aim of *minimizing the error* of the produced output values. This error of the prediction is described as the *loss* function or the *cost* function. The loss can be defined in different ways. Training process of a neural network is the minimization of this loss function in an iterative manner. The complete process is presented visually in Figure 2.7.

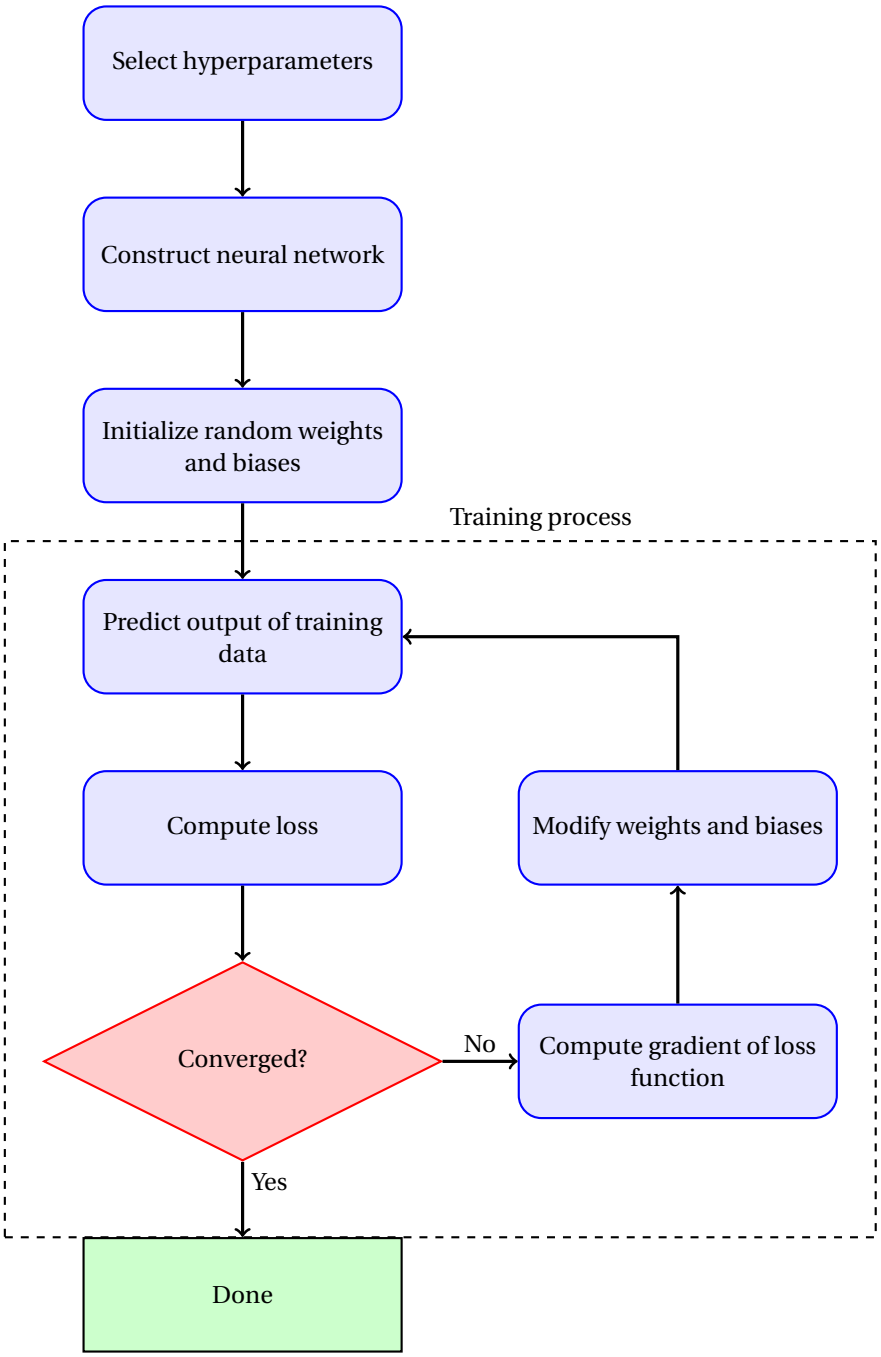


Figure 2.7: Flowchart describing the general training process of an Artificial Neural Network.

Initialize weights and biases Initially, the values of the weights and biases are chosen randomly from a particular probability distribution. This initial set of weights and biases make for a poor performance of the neural network. The neural network is untrained and therefore its error will be very high. By means of gradient descent, this error will be decreased in each iteration.

The probability distribution from which the initial weights and biases are drawn is an important parameter of the network. More information on initializers is provided in Section 2.2.4.

Define the loss function The loss function is a function that gives a certain error metric of the predicted values compared to the true data values. A common loss function for regression problems is the Mean Squared Error (MSE) which is defined in (2.7). As presented in Section 2.2.2, upon presentation of a dataset $X_{i,j}$, the network produces a matrix with output values $Y_{i,k}$. Let $\hat{y}_i$ be a single column vector with predicted values for all training samples for one particular ouput variable and $y_i$be the true output values which are known from the dataset. The MSE over all training samples is defined as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{2.7}$$

This function returns a scalar value which is always positive, regardless of whether the individual errors are positive or negative. The unit of the MSE is equal to the unit squared of the output parameter to be predicted. In order to get a more intuitive feeling for the magnitude of the error, one can take the square root of the MSE, resulting in the Root Mean Squared Error.

Miniziming the loss by gradient descent The loss function is a function of the predicted values $\hat{y}_i$ which, in turn, is a function of *all* the weights and biases in the neural network. The goal is to modify all the weights and biases in such a way that the loss becomes as small as possible. The weights and biases of the network are updated in an iterative manner. This minimization takes place by a method called *gradient descent*

In each iteration, the gradient of the loss function is computed. Let C be the loss as a function of all weights and biases. Let v_i be a vector containing *all* weights and biases. Then the loss vector is a function of the weight vector $C_i = f(v_i)$. The gradient of the loss function is a vector containing all partial derivatives $\frac{\delta C_i}{\delta v_i}$.

The gradient vector of the loss function is defined as

$$\nabla \mathbf{C} = \begin{bmatrix} \frac{\delta C}{\delta v_1} & \frac{\delta C}{\delta v_2} & \cdots & \frac{\delta C}{\delta v_n} \end{bmatrix}^T \tag{2.8}$$

The gradient vector represents the direction in n-dimensional space in which the slope of the loss function is maximal. See Figure 2.8 for a visualization in two dimensions. It describes the change of C relative to the changes in v_i. This gives information about the required changes Δv_i in order to decrease the loss. The values of all weights and biases are updated by an amount $\Delta v_i = -\eta \nabla C_i$. The minus sign ensures a *decrease* of the loss. The factor η represents the *learning rate*, which is an important parameter determining the step size of each update. The effects of the learning rate are described in Section 2.2.4.

It is not guaranteed that the gradient descent algorithm ends up in the global minimum. See Figure 2.9 for a visualization of this problem. Whether the optimization yields a global or a local minimum not only depends on the optimizer, but also the initialized values of the weights and biases play an important role.

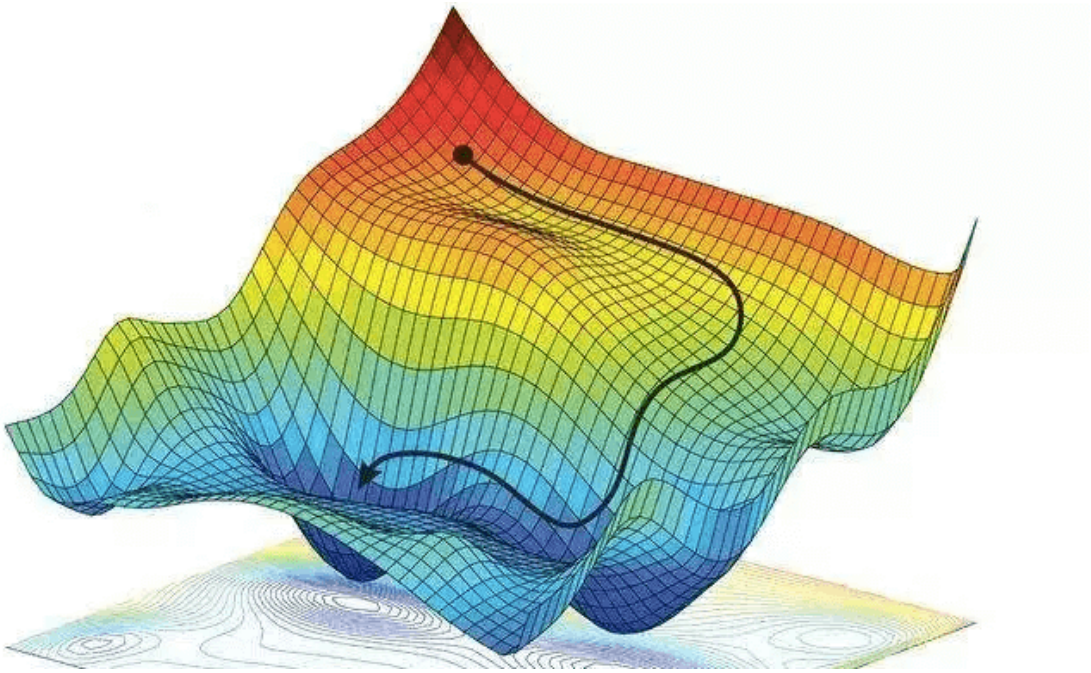


Figure 2.8: Visual impression of gradient descent in 2 dimensions [3]

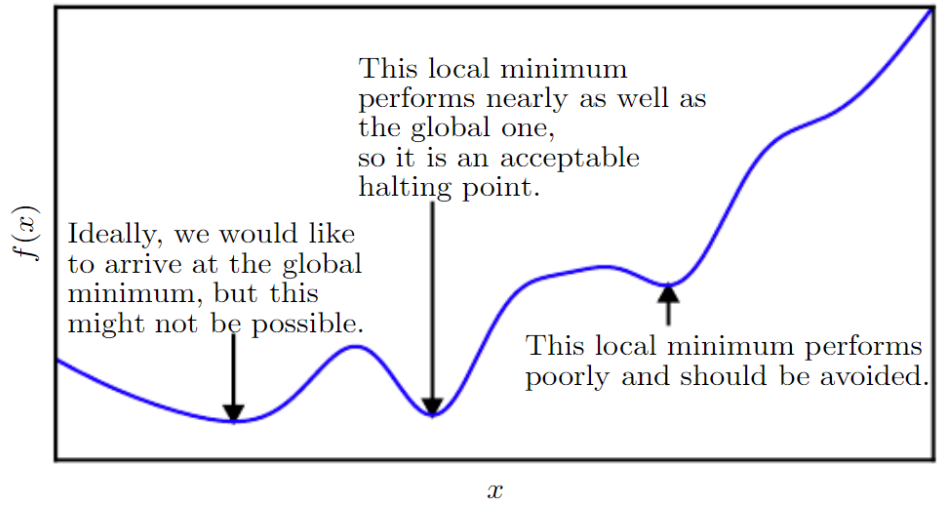


Figure 2.9: Schematisation figure demonstrating finding either a global or a local minimum [26]

2.2.4. Hyperparameters

The predicting performance of the ANN is highly dependent on the chosen set of *hyperparameters*. These hyperparameters are to be distinguished from the model parameters, i.e. the values of the weights and biases. Hyperparameters can be described as the global *settings* of the ANN, like the number of hidden layers.

In practice it is not possible to tell [33] which set of hyperparameters will give a well performing neural network. The optimal set of hyperparameters depends on the type of problem and the nature of the dataset. Although some rules of thumb exist, the optimal hyperparameters can only be found by trial and error. Different sets of hyperparameters will yield different values of accuracy after training, which are to be compared.

A non exhaustive list of some hyperparameters for Artificial Neural Networks is presented below. In the following paragraphs, the meaning of some of these hyperparameters and their influence on the model are briefly explained.

- Number of hidden layers
- Number of neurons per layer
- Activation function
- Optimizer
- Learning rate
- Kernel initializer and bias initializer
- Batch size
- Number of training epochs

Number of hidden layers The number of hidden layers is the number of layers between the input and the output layer. According to the Universal Approximation Theorem, *any* function can be approximated by a single-layer network with a finite number of neurons in the hidden layer. However, this may mean that the number of neurons in the hidden layer is so large that it is infeasible to train the neural network. In order to capture complex behaviour of the system while limiting the number of neurons needed, multiple hidden layers must used in the network. A neural network consisting if many hidden layers is what is referred to as a *deep neural network*, hence the term *deep learning* which is often used to indicate machine learning using ANN's.

Number of neurons per layer Increasing the number of neurons in a layer may improve the capability of the model to capture more complex behaviour of the system, but this is not necessarily the case. Note that it can be beneficial to keep the amount of neurons as small as possible. When increasing the number of neurons, the number of connections drastically increases with it. This means that the weight matrices become larger, thus increasing the computational resources needed to compute the gradient of the loss function and optimizing the values of the weights.

Activation function The z value is passed through an activation function. The activation function can have multiple purposes, depending on the type of activation function and the problem at hand. A simple step function like the Heaviside function can determine whether a neuron should 'fire' and send information to the following layer of neurons or not.

$$H(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{2.9}$$

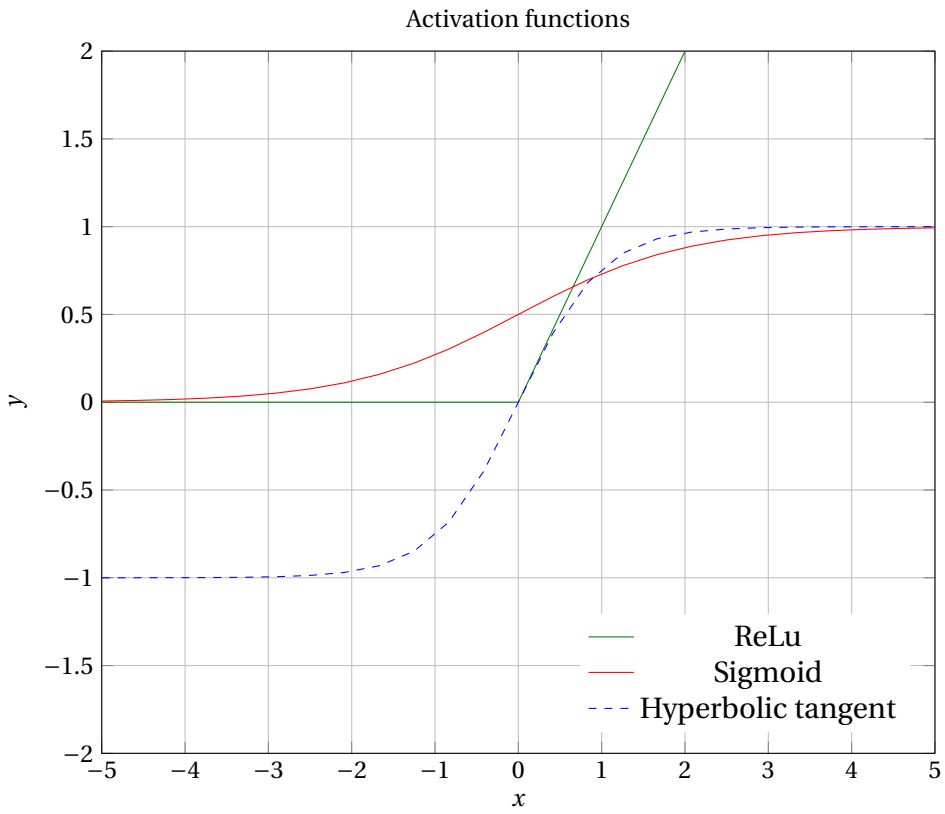


Figure 2.10: Some examples of many available activation functions

Non-linear activation functions enable the neural network to capture complex non-linear behavior of the predicted system. If only linear activation functions would be used in the ANN, the final result would just be a linear combination of the input values. Adding more hidden layers does not improve the ANN since all hidden layers can be replaced by a single hidden layer [1].

Learning Rate As described in Section 2.2.3, the learning rate η represents a proportion of the gradient vector to be used for updating the weights and biases of the neural network. The higher the learning rate, the greater the step size of the update. While a higher learning rate may result in quicker convergence, one may also risk divergence, resulting in *increasing* values of the training loss. The goal is to find an optimal learning rate which allows for quick convergence. Figure 2.11 shows different examples of learning rates and the behaviour of the training process.

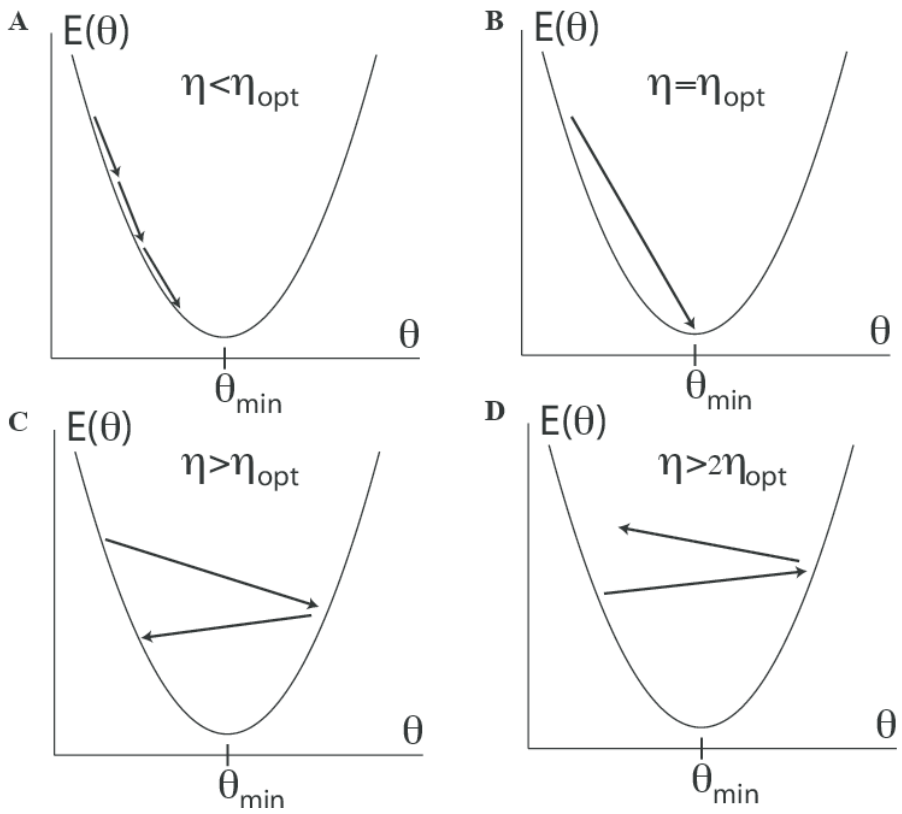


Figure 2.11: Four plots showing different scenarios caused by different learning rates. In plot A, the learning rate is set a little lower than the optimum value. The solution converges. In plot B, the learning rate is exactly equal to the optimum value. The solution converges in one iteration. In plot C the learning rate is set larger than the optimum value. It will converge after some oscillations around the final solution. In plot D the learning rate is set too large. The solution will diverge.

Kernel initializer and bias initializer The kernel initializer and bias initializer determine from which statistical distribution the initial weights and biases are drawn from. These can be e.g. random uniform, standard normal, but it is also possible to initialize all weights and/or biases with zero values.

2.3. Genetic algorithms

This section will treat the theory on genetic algorithms. In Section 2.3.2, the main process will be explained step by step, along with some terminology. In the subsequent sections, some steps from the main process will be elaborated further.

2.3.1. Why Genetic Algorithms

A genetic algorithm (also called evolutionary algorithm) is a heuristic method based on the theory of evolution by Charles Darwin. The aim of the algorithm is to find an optimum for problems for which no explicit formula or relation is known.

In basic mathematics it can be easy to find a minimum or maximum of a given function by setting the derivative of the function equal to 0. If the relation of the problem at hand is not differentiable, or there is no explicit mathematical relation known in the first place, this approach will not work. One could try to evaluate the response of the system just by trying out many different sets of input variables, after which the lowest or highest response values can be selected as the optimum. Although this may work to some extend, it is not a very efficient approach and the probability to truly find a global optimum is very small.

A Genetic Algorithm is a very powerful method to find optima of a certain system. Instead of trying to evaluate the response of a system at many points with brute force only, it is an iterative procedure in which the algorithm zooms in to promising regions where the system tends to be optimal. In a next iteration, the system will be evaluated mostly in this promising region after which the probability of finding an optimum is higher.

This procedure is repeated until an optimum is found, or until the maximum number of iterations is reached.

It may be clear that the genetic algorithm does not have a Regression or classification task. Its sole purpose is optimization. But this optimization algorithm can be very useful for optimizing machine learning regressors or classifiers. The performance of an artificial neural network highly depends on the selected hyperparameters. According to the 'no-free-lunch theorem [33], it is not possible to tell in advance which hyperparameters will work out well. This must be found out by trial and error. One could manually try different sets of hyperparameters to see how the ANN will perform after training on the data, but this is a very inefficient process. This is where genetic algorithms can be very useful. Since there is no known (mathematical) relation between the selected hyperparameters and the final performance of the ANN, we can try to find the optimal set of hyperparameters for the ANN by using a genetic algorithm.

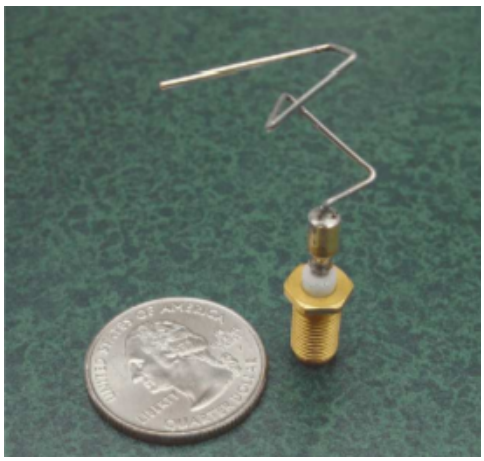


Figure 2.12: Antenna created by NASA optimized with genetic algorithms for optimal radiation patterns [22]

Example NASA antenna

2.3.2. Main process

The process of a genetic algorithm is similar to the evolutionary process in the natural world. The properties of an individual determine the fitness of the individual and thereby its chance of survival. Fit individuals will survive and pass on their genes to the next generation. After each successive generation, the overall fitness of the individuals will improve. These same principles hold for a genetic algorithm.

The general process is as follows. Some terminology used in these steps are explained in more detail in the subsequent sections.

First, an initial population is created. This first population consists of a number of randomly generated individuals. After creating the initial population, the following set of steps are repeated for each new generation.

1. Fitness evaluation: The fitness of each individual is evaluated and assigned to the individual.
2. Selection: Individuals are selected to survive for the next generation.
3. Variation: The gene pool is varied to explore new possible solutions.
4. Repeat: Repeat until stopping criterium is met.

The genetic algorithm can be stopped when one of the following conditions is met.

- The maximum number of generations is reached.
- The desired level of fitness is reached.
- The fitness of individuals in new generations stops improving (convergence).

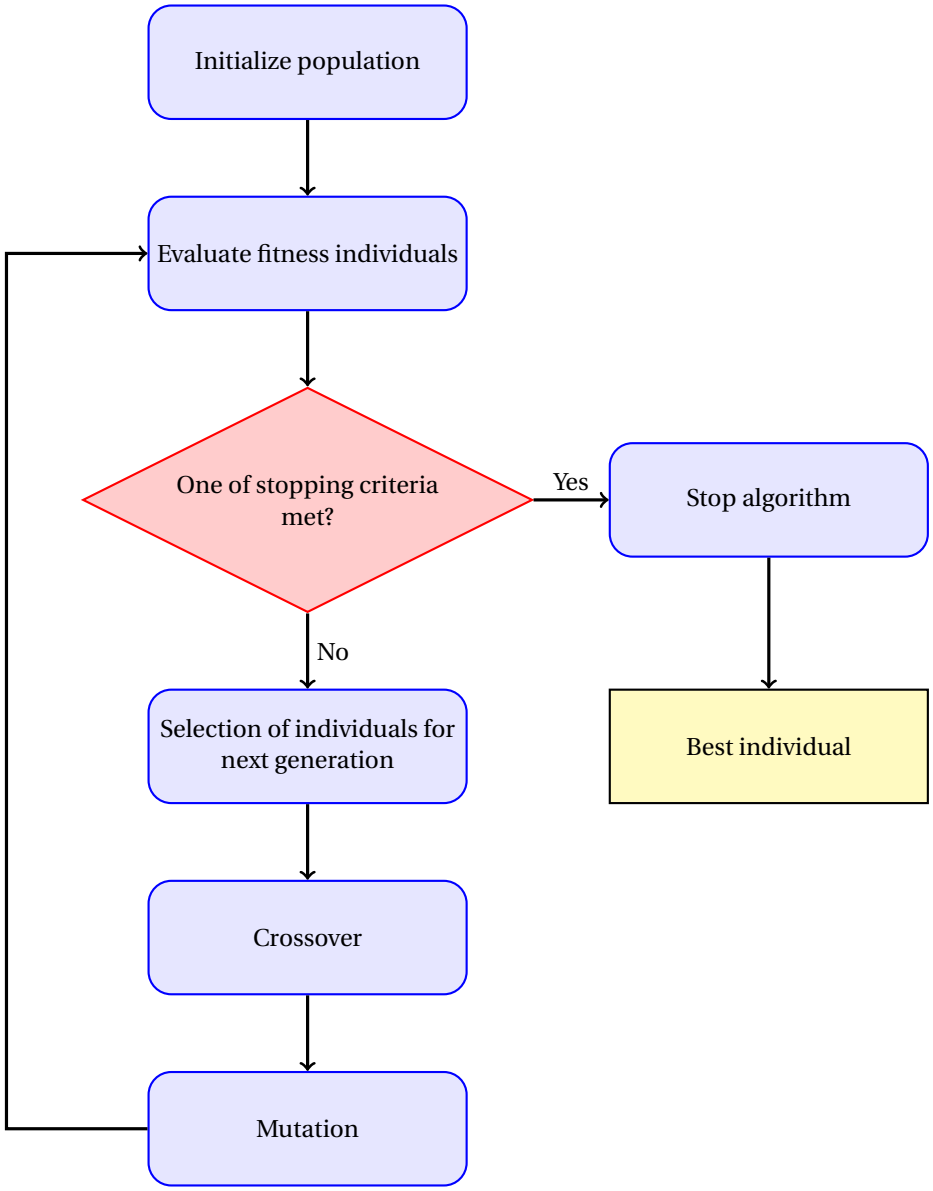


Figure 2.13: Flowchart visualizing the genetic algorithm for hyperparameter optimization for neural networks.

Some terminology

- **Population**: A population is a collection of individuals. The number of individuals in a population is called the population size.
- **Individual**: An individual is analogous to an individual in the biological sense. It represents one unique entity, like an individual animal in the natural world. Its properties are defined by a 'genotype', also called its 'chromosome', which is basically a set of genes.
- **Gene**: The gene is one element in the genotype of the individual. In Figure 2.14, each gene is represented by a binary value. In general, a gene can contain any type of information like continuous numbers, letters, mathematical operators etc. The type of genes depends on the problem to be solved.
- **Fitness**: A fitness value is assigned to each individual. This fitness value will determine its chance of survival during the evolution. The better the fitness of an individual, the higher the chance of survival and passing (some of) its genes to the next generation. The fitness of the individual is determined by a fitness function. The fitness function also depends on the problem and must be determined by the

programmer of the genetic algorithm.

Global or local optimum The evolution of the algorithm will finally lead to an optimum. There is however no guarantee that this is a *global* optimum. The genetic algorithm is a highly stochastic process. One very important fact is that the initial population is randomly generated. There will be individuals in the neighbourhood of some local optimum. But it is not guaranteed that there exist individuals close to the *global* optimum. If the parameter space is not properly covered, the algorithm will tend to converge to a local optimum. It is therefore important to ensure enough variation in the initial population such that the total parameter space is properly covered.

The behaviour of a genetic algorithm depends on the settings of its hyperparameters. Some features and hyperparameters will be discussed in the next section.

2.3.3. Fitness evaluation

The goal of the genetic algorithm is optimization. In order to find the best performing individual, its fitness value must be evaluated. Therefore a specific fitness function must be formulated in order to evaluate the fitness and make fair comparisons.

In the case of the OneMax problem, the goal is to find a set of genes which make up an individual of which the sum of the binary values are highest. In this case, the fitness function can simply be defined as the sum of the binary values. An individual with many genes being equal to one will have a higher fitness than an individual with many zeros in its genotype.

In case of hyperparameter optimization of machine learning models, the fitness of an individual could be represented by e.g. the accuracy of its predictions. The fitness function would involve multiple steps: First, the machine learning model must be compiled with its hyperparameters (the genotype of the individual). Then the model is trained on a training dataset and tested on a validation dataset. The performance of the predictive model on the validation dataset will be the fitness value of the model.

2.3.4. Selection

After the fitness of the entire population is evaluated, strong individuals will be selected to survive for the next generation. Different selection methods exist, of which two will be explained in this section

Tournament selection A common selection algorithm is *tournament selection*. This procedure is based on competition between several individuals. A fixed number of individuals are randomly drawn from the population. The individual with the best fitness score wins and is selected for the next generation. This process is repeated until a desired number of individuals is selected for the next generation.

Elitism Elitism is the concept of selecting a number of the very best performing individuals in a population. With tournament selection, there is a probability (although small) that the fittest individual in a population is never chosen for a competition. In this way it cannot compete for passing to the next generation. In order to avoid this, elitism can be implemented to ensure that the very best individuals are always selected for the next generation.

The performed selection procedures make up a new generation, called the offspring. After selection, this new generation will undergo variation of the gene pool in order to explore new possible solutions.

2.3.5. Variation

Evolution cannot take place without sufficient variation in the gene pool. Two main methods exist for variation. Crossover and mutation

Crossover Crossover operations are performed on two individuals in the offspring. This means that some genes are interchanged between two 'parents', creating two new individuals. This process is also called 'mating'.

Random pairs of individuals are created from the offspring, resulting in a list of pairs. The algorithm iterates over this list. For each pair of 'parents', some genes may be interchanged, making up new children. The

chance of two parents interchanging their genes is determined by the *crossover rate*. This is a number between 0 and 1 determining the probability of two parents sharing their genes.

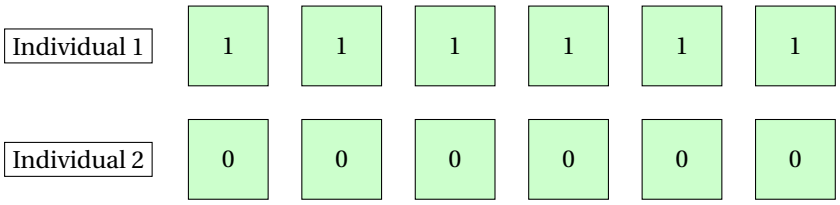


Figure 2.14: Two individuals before crossover

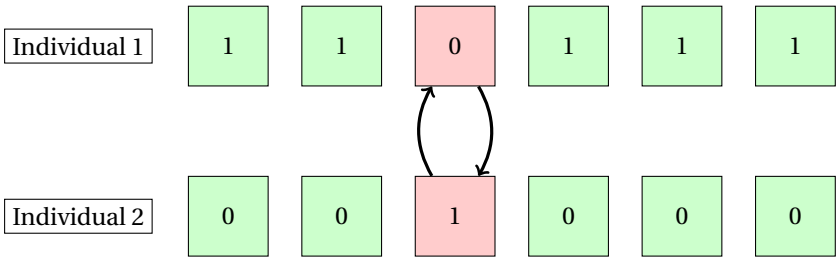


Figure 2.15: Both individuals exchange one gene during crossover operation.

Mutation Some individuals will undergo a mutation operation. When mutated, the individual will have one or more of its genes altered. The chance of an individual being mutated depends on the *mutation rate*. This is also a number between 0 and 1 setting the probability of a mutation taking place.

How an individual is being mutated depends on the way its genotype is constructed. In the case of the One-Max problem, an individual consists of a set of binary values. A logical mutation on such a genotype would be to switch the binary value of one or more genes from 0 to 1 or vice versa. If case an individual would consist of genes representing continuous numbers, it could be mutated by modifying the value of one or more continuous numbers. The value of the change can have a specified statistical distributions like e.g. a Gaussian distribution.

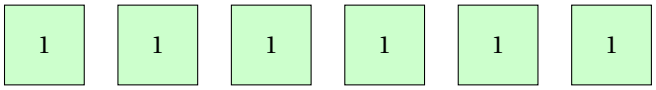


Figure 2.16: Before mutation

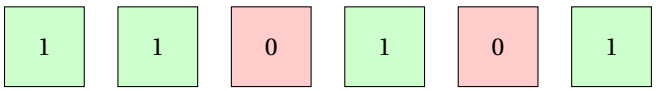


Figure 2.17: after mutation

Setting the crossover rate and mutation rate The crossover rate and the mutation rate are important hyperparameters that determine the behaviour of the genetic algorithm. The crossover rate is the probability of two parents interchanging their genes and thereby creating new offspring. The mutation rate is the probability of an individual being mutated. The ratio between these two values must be balanced in order to control the convergence behaviour of the algorithm. The goal is to find a good balance between *exploration* and *exploitation*.

Exploration is the concept of searching for possible fit individuals over the available solution space. The higher the *mutation rate* the more variation of genotypes is created and therefore the more exploration takes place. On the one hand this increases the chance of finding new regions where good solutions may exist. On the other hand, a high mutation rate can stagnate the convergence of the algorithm.

Exploitation is the concept of 'zooming in' to promising regions of the solution space. This means that the algorithm is focusing on good individuals, and searches for possibly better solutions in the vicinity of these promising individuals. In general this is accomplished by setting a higher crossover rate. The higher the crossover rate, the more the algorithm is focusing on exploitation. While a high exploitation may lead to quick convergence, there is a good chance that the algorithm converges to a *local* optimum instead of the global optimum.

3

Mechanical model and FE analysis

In this chapter, the main mechanical model and the procedures for solving the model using finite element analyses are described. All choices and considerations regarding the FE model can be found in this chapter.

Since the analyses will be done in an automated manner, critical engineering judgement is not always possible during running of the analyses. Some complications were encountered for which a simplified approach had to be chosen. Section 3.9 discusses these complications in detail. At the end of the chapter, in Section 3.10, an example of a FE analysis is given.

Important note: It is important to note that the predictive model will produce its output values with a certain (in)accuracy. Therefore it *cannot* be used as a verification tool, as this would be too unsafe. The aim of the predictive model is only to be used in a design optimization algorithm, or to get insight in the response of the output parameters as a function of certain input parameters. A complete FE analysis is needed to verify the resistance of the structure to be designed.

3.1. Introduction main mechanical problem

In recent years, Iv-Infra has been responsible for the design of a number of new sea lock gates. In the year 2015, the last of 16 steel lock gates for the new Panama canal was installed. More recently, new lock gates have been designed for the new sea locks at IJmuiden in the Netherlands, which are to be the largest sea locks in the world. These giant steel structures are so-called rolling gates, moving horizontally over a rail system at the bottom of the lock during operation. Figure 3.1 shows a picture of 4 of the sea lock gates for the new Panama canal.

The FE analyses performed in this thesis are based on the buckling analyses for the new lock gates of the harbour of IJmuiden. Project OpenIJ will be used as a reference project [34] in this thesis.

Figure 3.1: Sea lock gates for the new Panama canal [23].

The global model of the lock gate consists of a steel frame that is covered with steel skin plates. See Figure 3.2. These skin plates are stiffened with longitudinal stiffeners. The lock gate is loaded horizontally by water pressure, caused by retaining the water. This loading results in a global deformation of the lock gate, comparable to the bending of an Euler-Bernouilli beam, resulting in in-plane compressional stresses in the skin plates and the longitudinal stiffeners. See Figure 3.8 in Section 3.5. These in-plane compressional stresses, combined with the out of plane water pressure, can cause instability of the skin plates and the longitudinal stiffeners.

The global model can be split up in a number of individual sections. Such a section is shown in Figure 3.3. The aim of each FE analysis is to find the maximum occurring stresses and deflections of the skin plate and stiffeners in such a particular section.

The analyses will be performed by means of non-linear buckling analyses in ANSYS. The adopted limit state criteria for the resistance are described in Section 3.6. The rules and guidelines provided by EN 1993-1-5 annex C [9] for ultimate limit state verifications are applied in this project and will be referred to regularly.

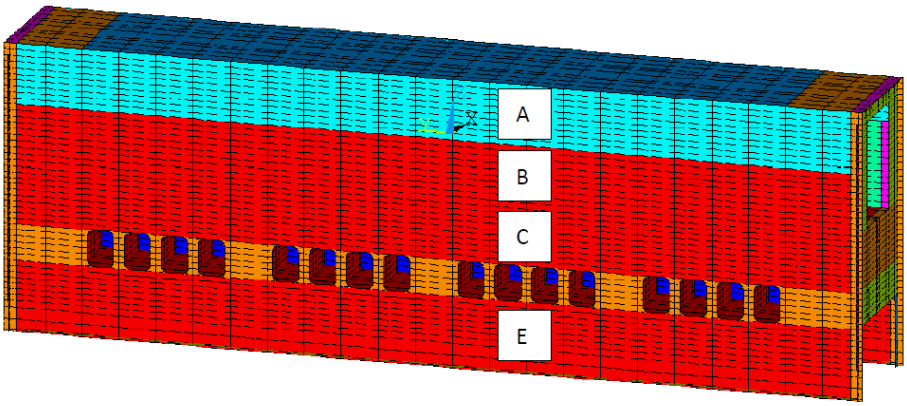


Figure 3.2: Global model of the lock gate [34]. The global model is split up in a number of separate sections to be modelled separately. See Figure 3.3 for a FEM model of such a section.

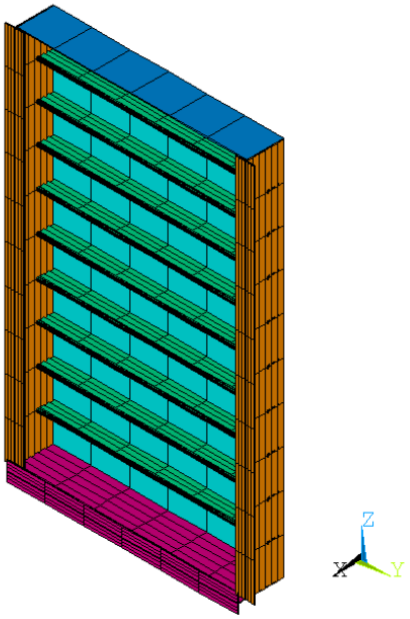


Figure 3.3: Individual section from the global model. Stiffened skin plate is attached to two supporting columns.

3.2. Geometry

The geometry of the sub-model consists of three main elements: The skin plate, the longitudinal T-stiffeners and the columns, each element having its own set of dimensions. See Figure 3.4 for the 3D representation of the sub-model. The geometry of the sub-model can be completely described by the set of geometrical input parameters as shown in Table 3.1. The longitudinal stiffeners are equally distributed lanong the plate height h_{plate}.

Note that buckling of the columns is not considered in the FE analyses. However, the columns are included in the model since these will have an influence on the rotational stiffness along the line of attachment between the skin plate and the column web.

The ranges of the geometry parameter values are listed in Table 4.8 in Section 4.4.2.

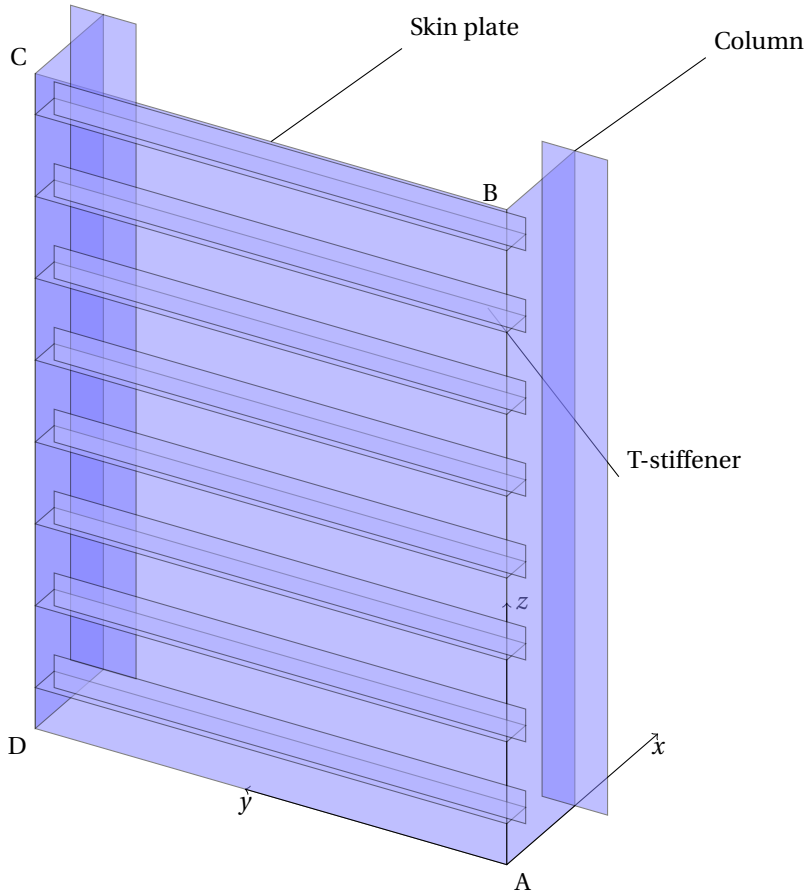


Figure 3.4: 3D representation of sub-model with configuration number 1: Longitudinal stiffeners are fixed to the column webs. The ranges of the geometry parameter values are listed in Table 4.8 in Section 4.4.2.

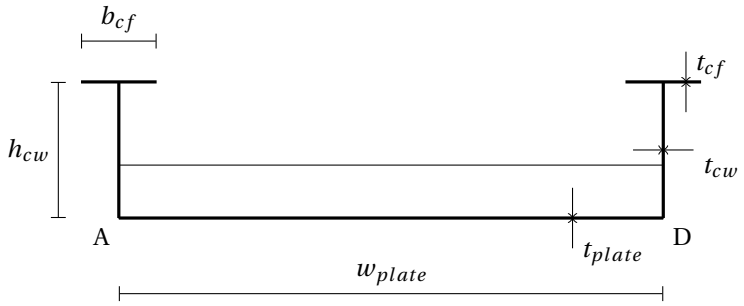


Figure 3.5: Cross sectional drawing through the xy-plane indicating the dimensions of the columns and the plate

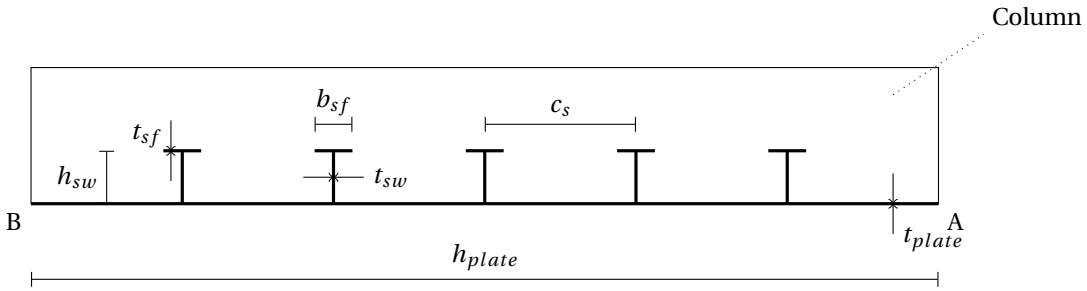


Figure 3.6: Cross sectional drawing through the xz-plane indicating the dimensions of the longitudinal T-stiffeners and the plate

Structural Element	Symbol	Dimension description
Skin plate	t_{plate}	Skin plate thickness
	h_{plate}	Skin plate height
	w_{plate}	Skin plate width (= c.t.c. distance columns)
T-stiffeners	n_s	Number of T-stiffeners
	c_s	Center-to-center distance
	h_{sw}	T-stiffener web height
	t_{sw}	T-stiffener web thickness
	b_{sf}	T-stiffener flange width
	t_{sf}	T-stiffener flange thickness
Columns	h_{cw}	Column web height
	t_{cw}	Column web thickness
	b_{cf}	Column flange width
	t_{cf}	Column flange thickness

Table 3.1: Overview of geometric parameters of the sub-model. The symbols are indicated in the cross sectional drawings in figures 3.5 and 4.3

3.3. Material

The material properties are adopted from the reference project [34]. Structural steel S355 is used of which the material properties are as follows.

Material property	Symbol	Value
Yield stress	σ_y	355 MPa
Elastic modulus	E	$2.1 \cdot 10^5$ MPa
Poisson ratio	ρ	0.3 [-]

Table 3.2: Overview of material properties as used throughout the project

These parameters are fixed parameters for *all* FE samples for the training data. The material properties could be included as free variables, expanding the possible design space, but this is outside the scope of this project.

The limit state criteria in this project are based on the material properties. These criteria are defined in Section 3.6.

For the final evaluation of the structure, a non-linear material model can be defined. These material models are explained in Section 3.8.2.

3.4. Boundary conditions

This section provides an overview of the boundary conditions applied in the model. The boundary conditions are expressed as displacements and rotations in the global coordinate system. See Table 3.3 and Figure 3.4 for the indications of the coordinate system.

Axis	Direction
x-axis	Normal to the skin plate
y-axis	Parallel to longitudinal stiffeners
z-axis	Perpendicular to longitudinal stiffener

Table 3.3: Directions of the axes in the global coordinate system

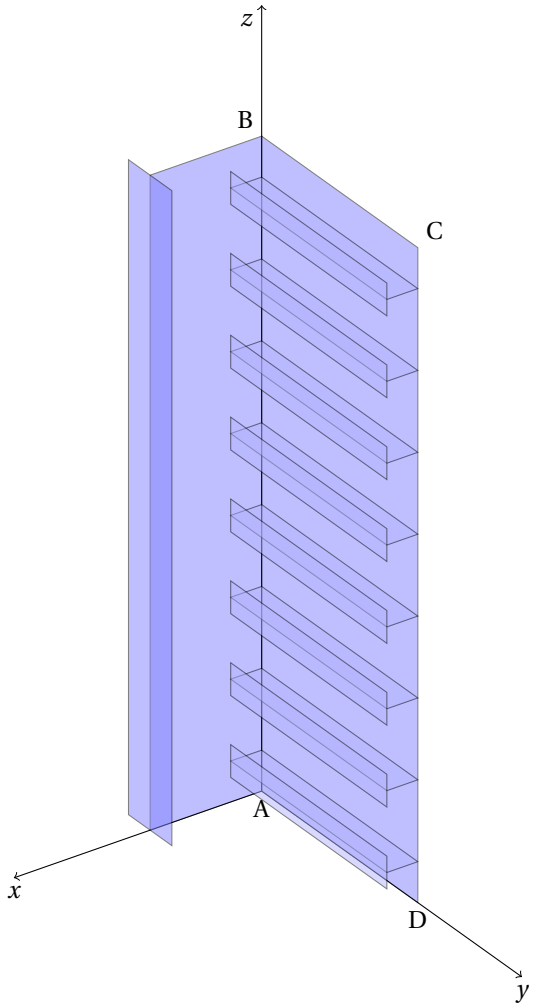


Figure 3.7: 3D representation of the model. The symmetrical geometry and loading allows for modelling only half of the structure. Symmetric boundary conditions are applied along the section C-D.

Table 3.4 gives an overview of the applied boundary conditions. Note that all boundary conditions defined in the table represent the boundary conditions as applied directly to the nodes. A dash means that no constraint is applied explicitly to the corresponding degree of freedom.

BC	**Edge**	**Location [mm]**	$\mathbf{u_x}$	$\mathbf{u_y}$	$\mathbf{u_z}$	$\theta_\mathbf{x}$	$\theta_\mathbf{y}$	$\theta_\mathbf{z}$
1	B-C: Plate edge	$z = h_{plate}$, $x = 0$	0	-	$-u_{zz}$	-	-	-
2	B-C: Column sections	$z = h_{plate}$, $x > 0$	0	u_{yy}	$-u_{zz}$	-	-	-
3	A-D: Plate edge	$z = 0$, $x = 0$	0	-	u_{zz}	-	-	-
4	A-D: Column sections	$z = 0$, $x > 0$	0	u_{yy}	u_{zz}	-	-	-
5	A-B: Plate edge + stiffener sections	$y = 0$	-	u_{yy}	-	-	-	-
6	C-D: Plate edge + stiffener sections	$y = 0.5 \cdot w_{plate}$	-	0	-	0	-	0

Table 3.4: Boundary conditions applied to the nodes of the FE model.

Some remarks about the boundary conditions

- **BC1, BC3**: The plate edges are free to rotate around the y-axis, while in the real, complete model, a steel plate might be attached perpendicular to the skin plate. In order to simplify the model by avoiding extra design parameters for these steel plates, these plate elements are not included in the model. An unconstrained rotation around the y-axis is a conservative approach for this simplification.

- **BC2, BC4**: The cross sections of the columns have a displacement equal to the displacements along the right plate imposed by the in-plane loading. This is in correspondence with the complete model where both these elements exhibit the same displacements. These fixed displacements of the column webs and flanges in y-direction reduces lateral instability of the column, which is not considered in this analysis.
- **BC1-5**: The in-plane compression stresses σ_{yy} and σ_{zz} are both applied as displacements u_{yy} and u_{zz}. The motivations for this choice is explained in Section 3.5.2
- **BC 6**: Symmetric boundary conditions are applied along the center line between two columns. Constrained u_y, θ_x and θ_z are derived automatically by ANSYS upon application of this symmetric boundary condition.

3.5. Loading conditions

The loading components applied to the sub-model are derived directly from the global model. The global model, i.e. the complete lock gate, is loaded horizontally by a combination of hydrostatic pressure and wave impact forces. These horizontal pressures make the lock gate deform in horizontal x-direction, analogous to a simply supported bending beam, loaded by a uniform pressure. These deformations result in in-plane stresses in the skin plate and the longitudinal stiffeners. Figure 3.8 shows the y-component of the bending stresses in the skin plate from the reference project [34].

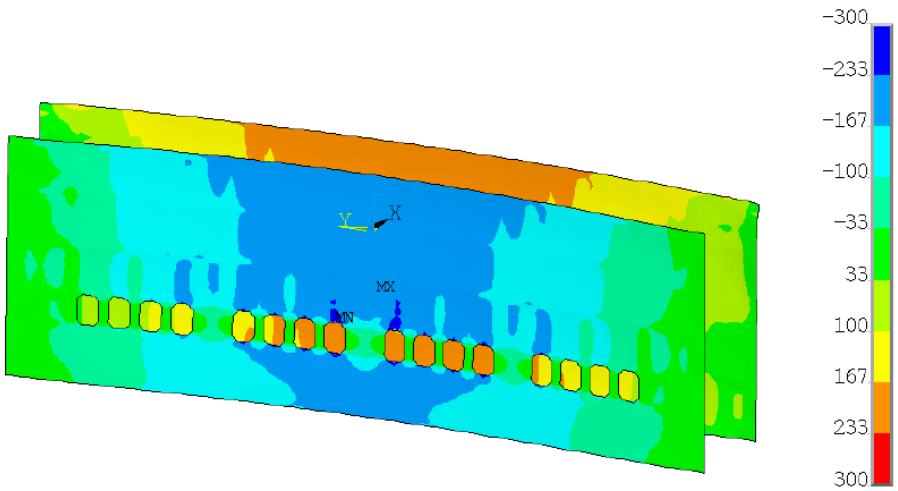


Figure 3.8: Bending stresses σ_y in skin plates in the global model

3.5.1. Out of plane loading

The out of plane loading on the sub-model is defined as a surface pressure in x-direction. It is defined as a linearly varying pressure, varying along the height of the model (z-axis) and is defined by two parameters. See Figure 3.9 for a schematic view. For simplification of the parameter space of the predictive model, only one load combination per analysis is possible.

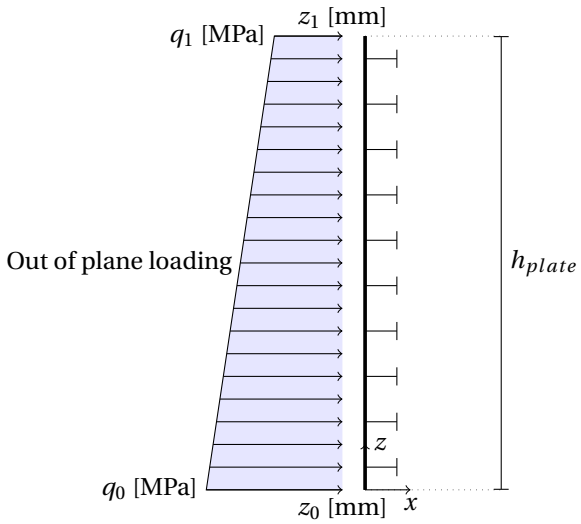


Figure 3.9: Schematic of the out of plane loading acting on the structure

Location	Symbol	Unit
$z = 0$	q_0	MPa
$z = h_{plate}$	q_1	MPa

Table 3.5: Out of plane loading parameters

3.5.2. In-plane loading

See Figure 3.10. In the actual FE model, the in-plane loading σ_{yy} will be applied as displacements, even though the results of the global analysis are expressed as stresses. The in plane loadings are applied as displacements because the thickness of the elements (skin plate, webs and flanges of the longitudinal stiffeners) will most likely not be constant along the entire length of the lock gate. Therefore, at the interfaces between adjacent sub-models, stress discontinuities will be present. Displacements, on the other hand, are continuous along the entire length of the model. Therefore, the displacements are derived from the stress and the stiffness of the sub-model as shown in equations (3.1) and (3.2).

Also the vertical stress σ_{zz} is applied as a displacement u_{zz}. While it is not strictly necessary since the plate thickness is usually constant along the total height of the lock gate, this is chosen in order to avoid divergence of the non-linear analysis. It does have some drawbacks regarding the Poisson effect. These points are discussed in more detail in Section 3.9.1

$$u_{yy} = \frac{\sigma_{yy}}{E} \cdot \frac{1}{2} w_{plate} \tag{3.1}$$

$$u_{zz} = \frac{\sigma_{zz}}{E} \cdot \frac{1}{2} h_{plate} \tag{3.2}$$

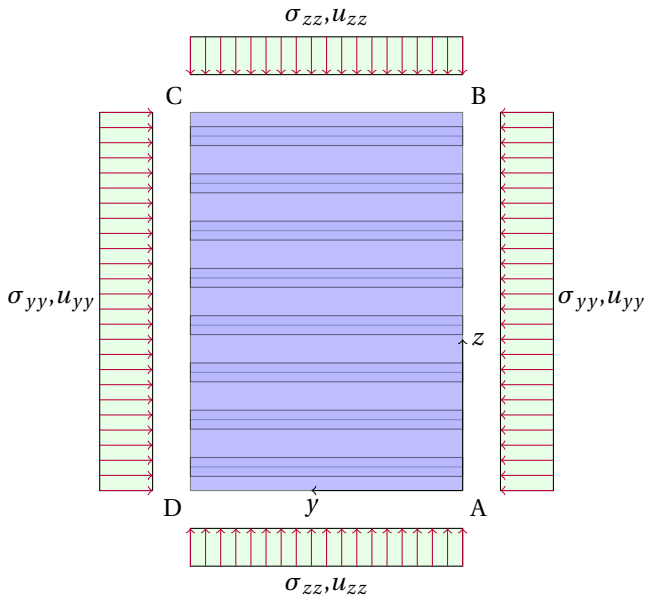


Figure 3.10: Schematic view of the in-plane compression loads applied to the model.

Loading component	Symbol	Description
Out of plane	q_0	Pressure at $z = 0$
	q_1	Pressure at $z = h_{plate}$
In plane	σ_{yy}	Compression stress in y-direction
	σ_{zz}	Compression stress in z-direction

Table 3.6: Summary of loading parameters for the parametric model. In plane stresses are still defined as actual stresses. The transformation to displacements is done within the actual parametric FE model.

Remarks In this project, no load factors are included. For both the input for the training data as well as for input parameters in the final model, raw values for loading are used. It is up to the user to include these load factors.

Shear stresses are not considered in the model. Since the critical sections are in the center of the lock gate, shear stresses can be neglected. For other sections where shear stresses are significant, separate analyses are to be performed where shear buckling phenomena are included.

3.6. Limit state criteria

For the evaluation of the resistance of the structure, two possible limit state criteria can be considered following the guidelines in EN 1993-1-5 C.8 [9] Note 1 and 2.

1. Attainment of the (Von Mises) yielding criterion
2. Maximum principle strain reaches $\epsilon = 5\%$

Other criteria are also possible. An example is setting a limit to the size of the yield zone.

In case criterion 1 is used as the limit state criterion, computations can be performed with just linear material properties. In case criterion 2 is used as limit state criterion, the nonlinear stress strain behaviour of the material needs to be defined beyond the elastic limit of the material. Possible nonlinear material models as prescribed in EN and are discussed in Section 3.8.2.

Limit state criterion for training and testing data For the generation of training data, the maximum load is attained when the maximum equivalent stress reaches the *yield* criterion at some point in the structure. This choice is based on the following motivations.

- The final predictive model will be used in the context of the preliminary design stages, where quick design choices have to be made. In this stage, the client will prefer a more conservative design. Therefore it is not preferable to allow for plasticity. This leaves some margin for changes in (loading) requirements without the need for drastic design modifications later in the design process.
- The training data for training the predictive model are generated by a series of automated FE analyses. When nonlinear material behaviour is included in the model, divergence in the non-linear analyses might cause problems. A method would be needed to determine whether the buckling load is reached or divergence has occurred. This is outside the scope of this project. A recommendation for further researched is added in Chapter 7.

3.7. Mesh

3.7.1. Element type

The structure will be modelled with four-node quadrilateral shell elements. In ANSYS, these are referred to as SHELL181 elements. Figure 3.11 shows the geometry of this shell element as demonstrated in the ANSYS Element Reference [11]. Each node has six degrees of freedom: three translational and three rotational degrees of freedom. According to the element reference [11] the SHELL181 element *"is well-suited for linear, large rotation, and/or large strain nonlinear applications."* It also states that the element allows for follower effects of distributed pressures. This is particularly important for modelling the out of plane water pressure in combination with a geometrical nonlinear analysis.

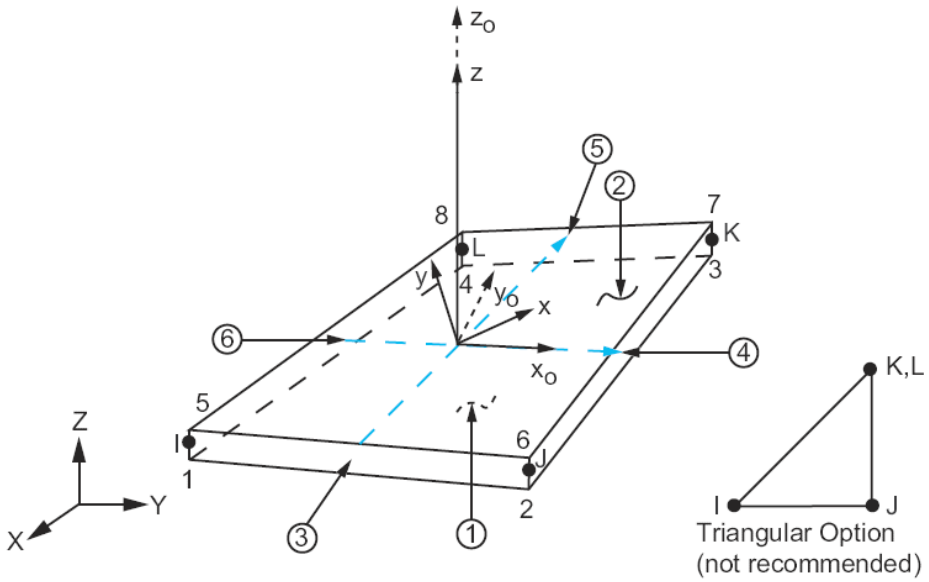


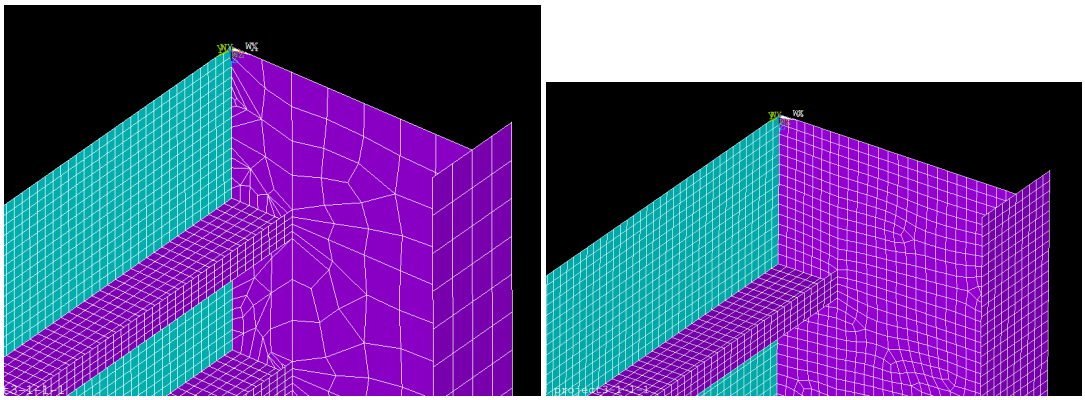
Figure 3.11: Schematic representation of the SHELL181 quadrilateral element from the ANSYS element reference [11]

3.7.2. Mesh quality

During the automated running of FE analyses, there is no possibility to make a judgement about the mesh quality by eye. The generation of the mesh must be performed as robust as possible. Therefore, some precautions are taken to ensure proper mesh quality. A comparison of the resulting mesh qualities with and without these precautions is shown in Figure 3.16.

- All structural components in the model will be assigned the same mesh size. This avoids sharp discontinuities in mesh size between any two adjacent meshed areas, which might result in shape errors. See Figure 3.12a for an example with different mesh size for the columns.
- Settings for meshing are set such that mapped meshing is applied where possible. Where mapped meshing is inappropriate, free meshing will be used. This setting is set with the `MSHKEY` command (see ANSYS command reference [10]).

- Furthermore, quadrilateral elements that violate either shape error or warning limits are split into triangular elements, even though triangular elements are not recommended as shown in Figure 3.11. This setting is set with the `MOPT` command.



(a) A sharp discontinuity between mesh sizes of the column elements and the other elements of the model. A warning is produced by ANSYS which states that shape testing revealed that a number of elements violate shape warning limits.

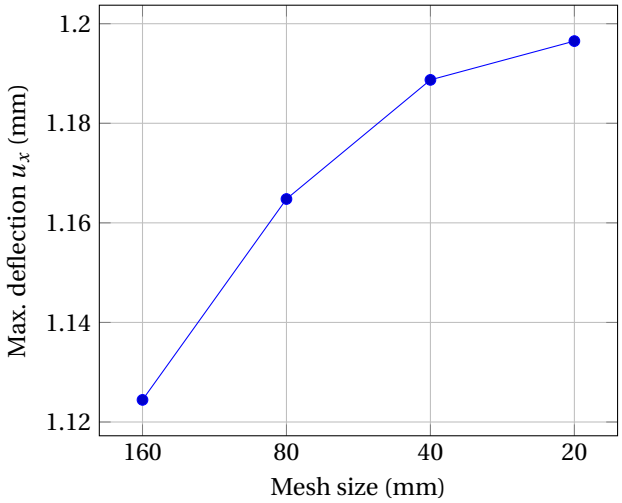
(b) Improved mesh quality. Some free meshing is visible in the column web, but none of the elements violate shape warning limits.

Figure 3.12: Comparison of mesh quality when using different mesh settings

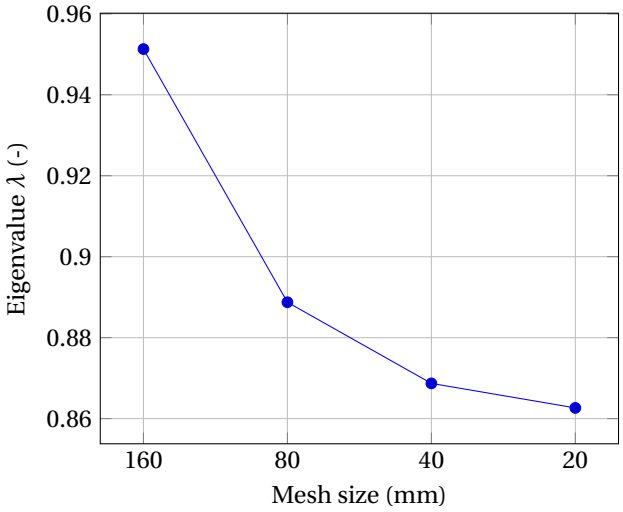
3.7.3. Mesh size

As noted in Section 3.9.2, stress singularities form a problem in the automated FE analyses. To reduce the stress peaks, the element size is not set too small. A balance must be found between mesh convergence and avoiding peak stresses.

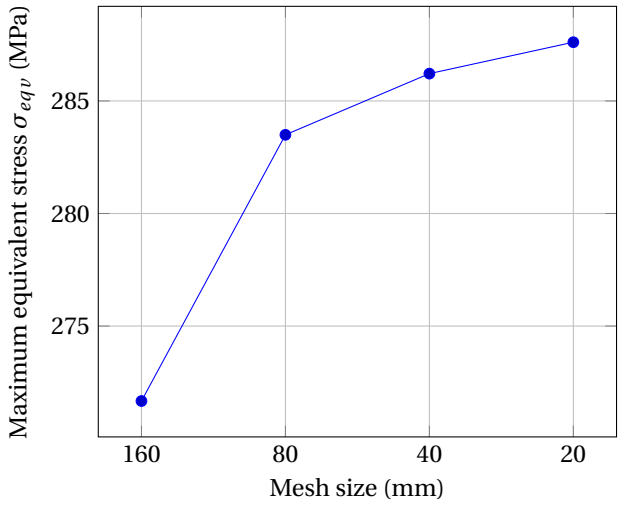
Mesh convergence analysis The element size will be based on a mesh convergence analysis. A linear static FE analysis with arbitrary, yet realistic, input parameters is run for a series of varying mesh sizes. The results are plotted in Figure 3.13. The maximum deflection, the eigenvalue and the maximum equivalent stresses are plotted against the mesh size in mm. A mesh size of 10 mm resulted in a memory overload.



(a) Mesh convergence plot of the maximum deflection [mm]



(b) Mesh convergence plot of the first eigenvalue [-]



(c) Mesh convergence plot of the maximum stress [MPa]

Figure 3.13: Mesh convergence plots

In order to reduce the influence of stress singularities, it is avoided to set the element size too small. The

mesh size is set equal to a value of 40 mm for all analyses, even though the output values have not yet fully converged. The center-to-center distance between the longitudinal stiffeners has a lower bound of 240 mm. In this way it is guaranteed that there are always at least 240/40 = 6 elements between any two longitudinal stiffeners.

3.8. Non-linear FE analysis

The main objective of the FEA is to find the maximum stresses in a nonlinear buckling analysis of the plated structure. EN 1993-1-5 [9] annex C provides guidelines for the use of FE analysis for finding the nonlinear buckling resistance of steel plated structures. The rules and guidelines in the document are used as a guidance for the FE analyses in this project. However, some complications were encountered regarding the automation of the nonlinear analyses. Therefore some simplifications have been made. The encountered complications and the work-arounds are described in their respective sections 3.8.1 and 3.8.2.

3.8.1. Geometrical nonlinearities

Initial geometric imperfections must be applied to the structure in order to initiate the buckling of the structure in a nonlinear analysis. This initial imperfection will be amplified upon loading, changing the stiffness of the system. At each load step, the initial imperfection will grow until finally, instability occurs.

The selection of an appropriate imperfection is a difficult task and requires the critical judgement of the engineer. The applied imperfection and its scale have a big influence on the resulting resistance of the structure. Generally, multiple analyses will have to be performed with varying types of initial imperfections, of which the lowest resistance is governing.

Application of imperfections for the automated FE analyses For the automated FE analyses, a simplified approach has been adopted for applying the geometrical imperfections. Initially, the aim was to apply geometric imperfections following the guidelines described in EN 1993-1-5 [9] annex C.5. This approach is described in the next paragraph.

However, difficulties were encountered with determining the scale factors for the imperfections based on the buckled shape. A visual inspection of the buckled shape is needed to determine the type of buckling mode and the accompanying scaling factor. Such an inspection is obviously not possible during an automated series of FE analyses. An attempt was made to determine the character of the buckled shape by reading the displacements of the nodes of the deformed shape. Due to limited time it was decided to take a different approach. A recommendation for further research is added in Chapter 7.

Therefore, instead of determining the scale factor based on displacement data of the first eigenmode, the scale factor will be input as a free parameter. First, a linear eigenvalue analysis is performed. The displacements resulting from the eigenvalue analysis are then scaled to the input scaling factor using the `UPGEOM` command in ANSYS APDL.

Guidelines imperfection in Eurocode EN 1993-1-5 [9] C.5 (2) Note 1 states: *"Geometric imperfections may be based on the shape of the critical plate buckling modes with amplitudes given in the National Annex."* These amplitudes are given in Figure 3.14 and are based on the deformed shape. The corresponding types of imperfections are shown in Figure 3.15. Some examples are given to clarify the approach.

See Figure 3.16a and Figure 3.16b. These global buckling shapes correspond to imperfection number 2 *global bow of longitudinal stiffener*. The corresponding scaling factor is based on either the length of the longitudinal stiffener or the wavelength of the buckle.

Figure 3.16c shows local twist of the longitudinal stiffeners, corresponding to imperfection number 4 of the table in Figure 3.15. The amplitude for this imperfection is equal to $\frac{1}{50} h_{sw}$ where h_{sw} is the height of the longitudinal stiffener. Figure 3.16d shows a combination of local buckling of subpanels (imperfection number 3) and twist of the longitudinal stiffeners. For both types of imperfections a scaling factor can be determined, of which the largest value will be chosen to create the most unfavourable imperfection.

Type of imperfection	Component	Shape	Magnitude
global	member with length ℓ	bow	see EN 1993-1-1, Table 5.1
global	longitudinal stiffener with length a	bow	min (a/400, b/400)
local	panel or subpanel with short span a or b	buckling shape	min (a/200, b/200)
local	stiffener or flange subject to twist	bow twist	1 / 50

Figure 3.14: Guidelines for setting the amplitudes of imperfections. Table C.2 in [9]

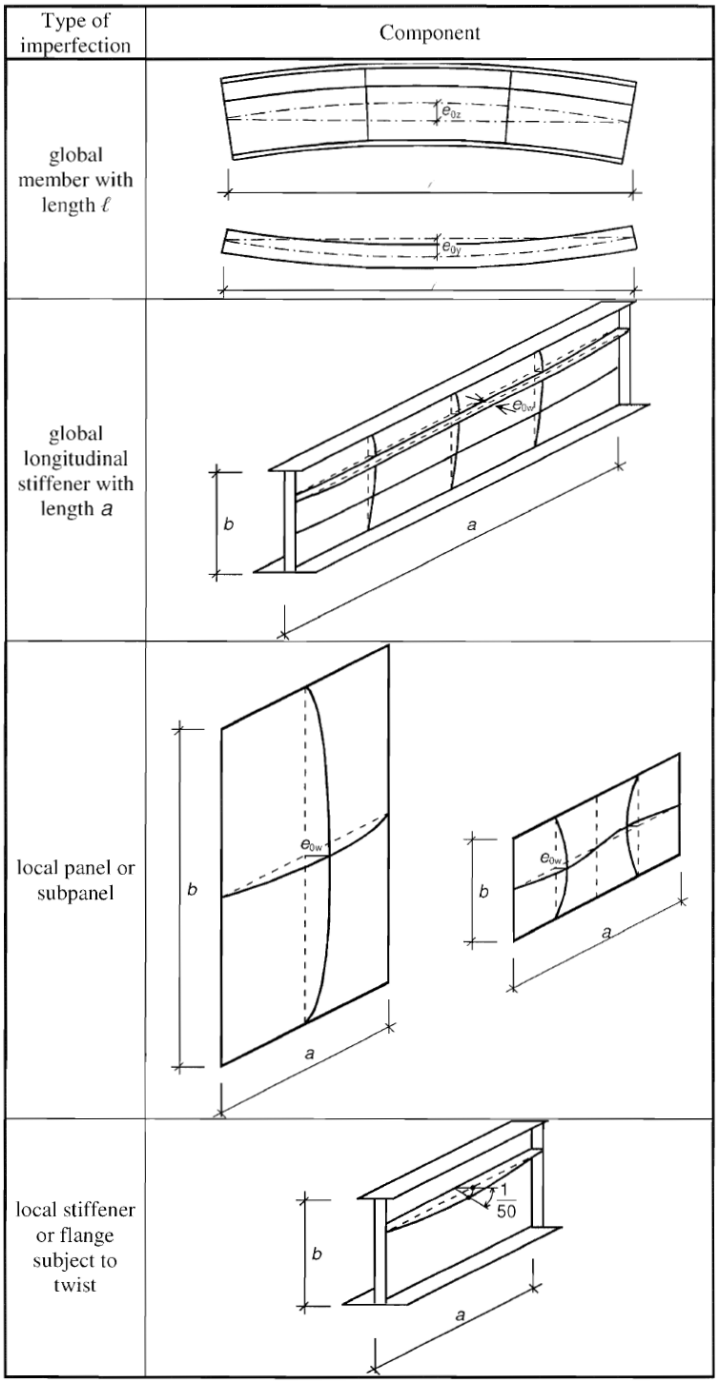
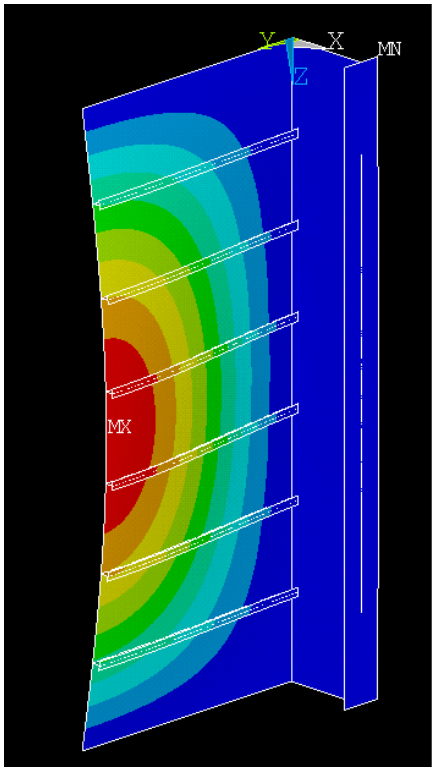
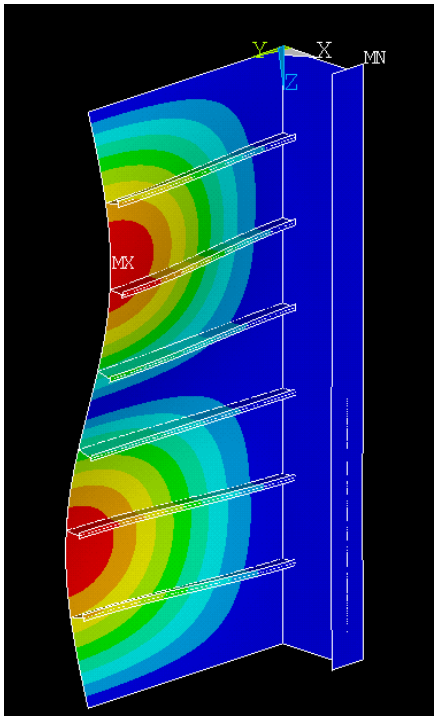


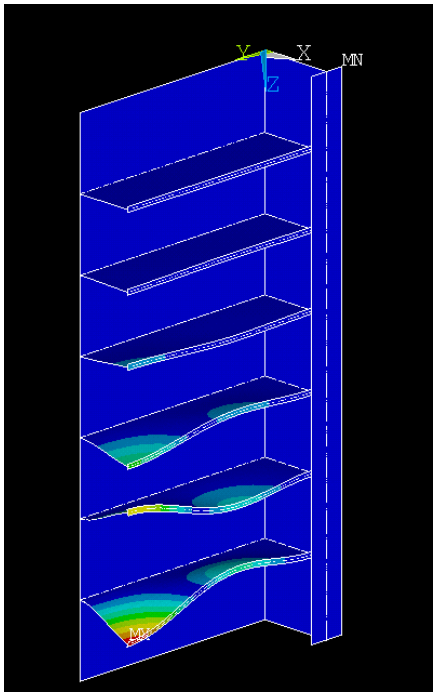
Figure 3.15: Table showing the geometrical characteristics of imperfections. Figure C.1 in [9]



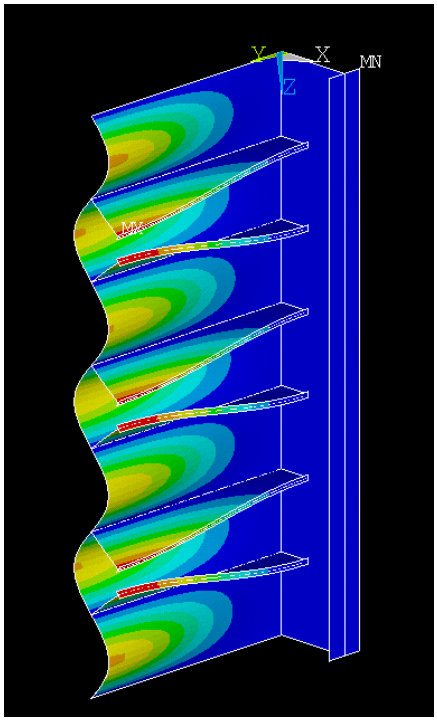
(a) Global buckling of the skin plate. One half wavelength fits over the height of the plate.



(b) Global buckling of the skin plate. One full wavelength fits over the height of the plate.



(c) Local buckling of the stiffeners. The scale factor is based on the height of the longitudinal stiffeners.



(d) A combination of buckling of local subpanels and twisting of the longitudinal stiffeners.

Figure 3.16: Comparison

3.8.2. Material non-linearities

Eurocode EN 1993-1-5 C.6 provides four different material models that can be used for nonlinear buckling analysis of steel plated structures. The accompanying figures are shown in Figure 3.17.

As already pointed out in Section 3.6, a linear elastic material model will be used for the generation of training data. Plasticity may result in an unconverged solution. The handling of unconverged solutions during automated FE analyses is outside the scope of this project. Recommendations regarding divergence and material nonlinearity are included in Chapter 7.

1. Elastic-plastic without strain hardening
2. Elastic-plastic with a nominal plateau slope
3. Elastic-plastic with linear strain hardening
4. True stress-strain curve modified from the test results

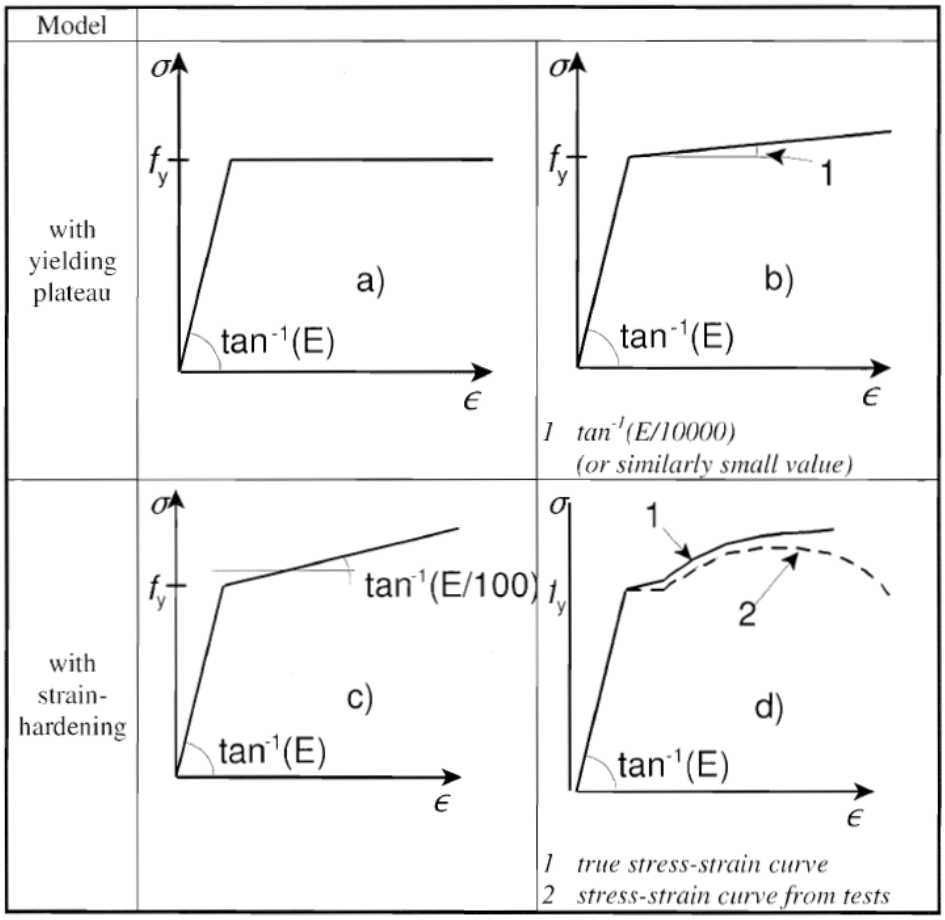


Figure 3.17: Nonlinear material models. Figure C.2 from EN-1993-1-5 [9]

3.9. Encountered difficulties

3.9.1. Divergence

Initially, the load σ_{yy} was applied as a displacement and the vertical load σ_{zz} was applied as a stress directly. The reason to apply the load in z-direction as a stress instead of a displacement was to avoid reaction forces higher than the input stress value, due to the Poisson effect. During generation of automated FE analyses, divergence was encountered. Therefore, a workaround will have to be made to handle the divergence issues.

Some settings were changed to see if convergence of this analysis could be made possible

1. Instead of automatic time stepping, the arc-length method was used. The default values for the minimum and maximum arc-length were maintained as provided by ANSYS. The solution did not result in a converged solution.

2. The vertical stress σ_{zz} was applied as a displacement u_{zz} instead of as a stress directly. (See equation (3.2) on page 28). This made the analysis a displacement controlled analysis, resulting in convergence. Both automatic time stepping and the arc-length method led to a converged solution with this adjustment.

For this project it is therefore decided to apply both compressive stresses along the edges as displacements instead of stresses. The possibility of unconverged solutions when deploying a load-controlled analysis is the main motivation for this. Filtering out unconverged solutions is outside the scope of this project. These incorrect data may negatively affect the prediction performance of the Artificial Neural Networks. Automatic time stepping is used instead of the arc-length method since automatic time stepping showed to reduce the computation time.

Implications of displacement controlled analysis The application of the vertical stress in the form of a displacement has the following implications.

- During pre stressing of the structure, the Poisson ratio of the material results in higher compressional stresses along the edges than defined. This leads to an underestimation of the eigenvalue.
- The nodes belonging to the cross sections of the ends of the columns are uniformly displaced. This results in a fully constrained cross section of the column ends, making for a stiffer behaviour of the column.

3.9.2. Stress singularities

Stress singularities regularly occurred in several analyses. These stress peaks were often found in different locations of the model, usually either in the corner of the plate or at the attachment of the stiffeners flanges to the column webs (Figure 3.18). The Artificial Neural Network are to be trained to predict the maximum stress values in the model, based on a set of geometrical and loading input parameters. These peak stresses however also depend on the mesh size. See Figure 3.19 where the maximum stress does not converge with refining the mesh. Trying to train the neural network on these values will negatively influence the accuracy of the predictive model.

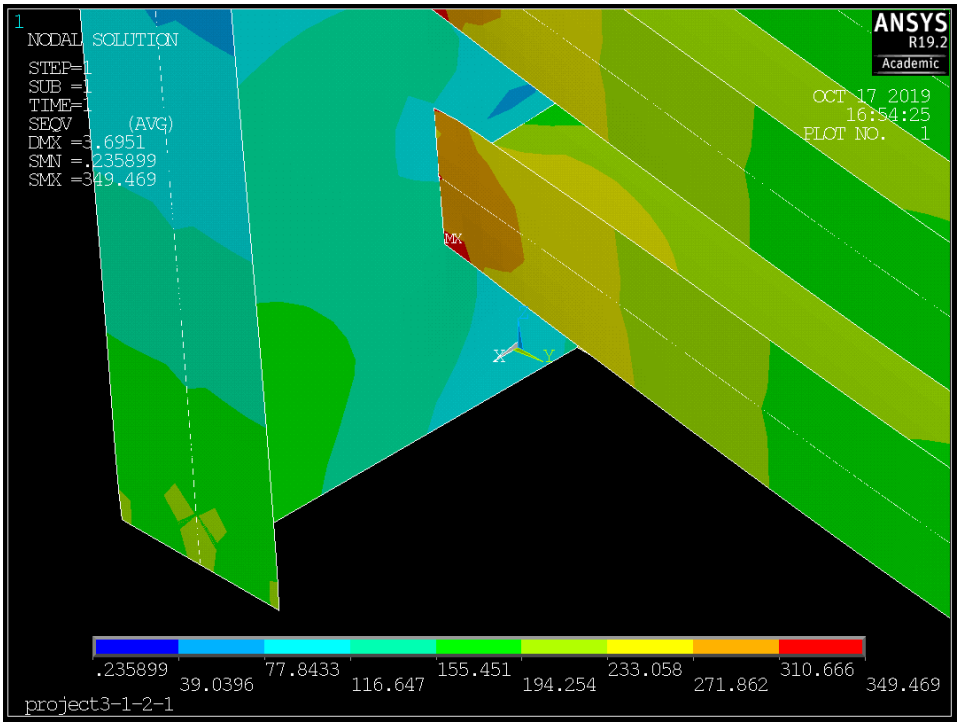


Figure 3.18: Peak stress at the interface between the stiffener flange and the column web. The maximum equivalent stress equals σ_{equ} = 349.5 MPa at a mesh size of 40 mm. Figure 3.13 shows how this peak stress increases when refining the mesh.

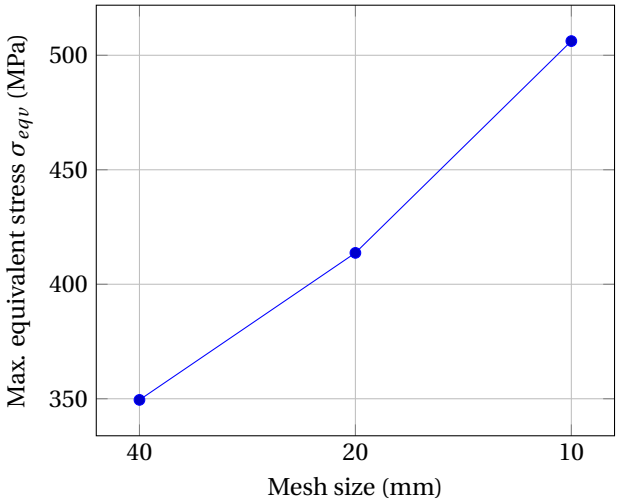


Figure 3.19: Plot maximum equivalent stress against the mesh size. The maximum equivalent stress increases upon refining the mesh, indicating a stress peak.

Since no personal judgement can be made during automated FE analyses about whether or not the maximum stress is a stress singularity or not, a work-around is necessary to cope with these singularities. Below are listed some options to work around the issue of stress singularities, together with their implications. Recommendations for further research regarding these points are included in Chapter 7.

1. Include a nonlinear material model. Allowing the material to yield upon reaching the yield stress will result in redistribution of stresses around the stress peak. The maximum occurring plastic strain in the model could be used as an output parameter. Prediction of the plastic strain is also in line with the limit state criterion of the maximum strain $\epsilon < 0.05$ as defined in the rules and guidelines of EN-1993-1-5 [9]. The drawback of this approach is the possibility of divergence in a nonlinear analysis.
2. Use larger finite elements. The ratio of nodal force over the element size will be lower, thus decreasing the peak stress values. This unfortunately can negatively influence the accuracy of the rest of the model.
3. Ignore the regions where stress peaks occur. The effect of the singularity diminishes the further away from it. The drawback is that all stresses in this region will be ignored, even if a governing stress value other than the peak stress is present here.

A combination of methods number 2 and 3 will be applied in this project. The settings for the mesh size are discussed in Section 3.7.3. Furthermore, the values of the maximum stresses of 3 different regions will be used as output parameter. One stress value will be the maximum stress in the entire model, including peak stresses. The two other output stress values will be the maximum stresses in either the plate or the stiffeners, ignoring the regions close to the column webs. A region with the size 10% of the skin plate width will be ignored here.

Predictive models will be trained on both output stress values. It is assumed that the predictions of the stresses ignoring the singularities will be more accurately.

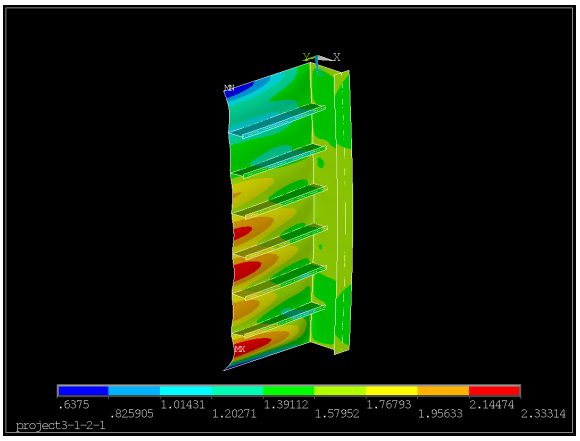
3.10. Summary and example FE analysis

This section shows a short example of the FE as will be performed for generation of the data. A design with arbitrary design parameters is selected. These are listed in Table 3.7. The result plots are presented in Figure 3.20 and Figure 3.21.

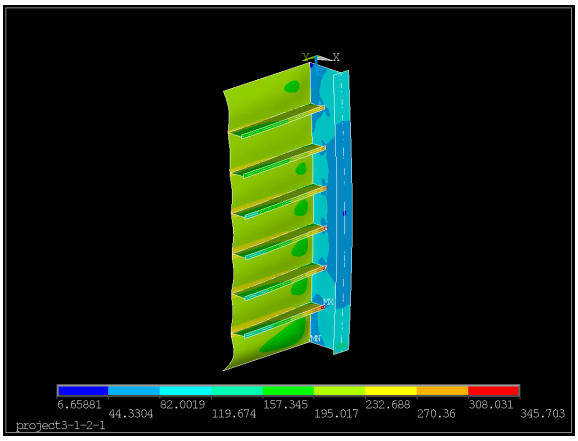
Figure 3.21 shows the first buckling mode. This buckling mode is used as initial imperfection scaled with an arbitrary scale factor equal to 10. The nonlinear analysis is performed using automated time stepping.

Element	Value
Skin plate	$t_{plate} = 12$ mm
	$h_{plate} = 3570$ mm
	$w_{plate} = 3100$ mm
T-stiffeners	$n_s = 6$
	$h_{sw} = 220$ mm
	$t_{sw} = 9$ mm
	$b_{sf} = 50$ mm
	$t_{sf} = 20$ mm
Columns	$h_{cw} = 515$ mm
	$t_{cw} = 12$ mm
	$b_{cf} = 250$ mm
	$t_{cf} = 15$ mm
Out of plane load	$q_0 = 0.05$ MPa
	$q_1 = 0.03$ MPa
In plane load	$\sigma_y = 200$ MPa
	$\sigma_z = 75$ MPa

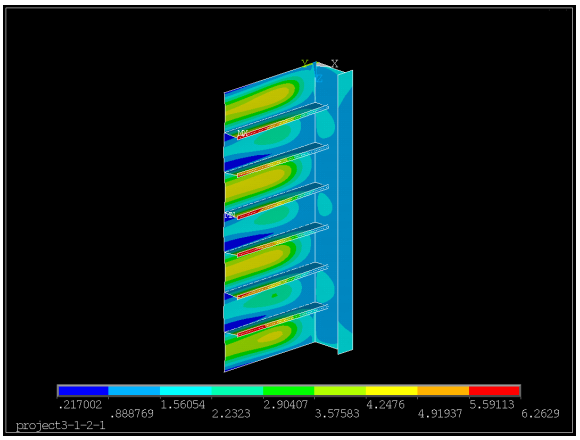
Table 3.7: Overview of geometric and loading parameters of the model example FE analsysis



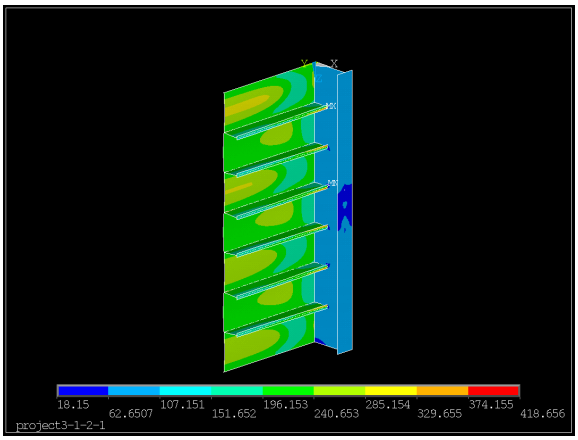
(a) Total deformation upon application of loading *before* eigenvalue analysis and nonlinear analysis. Plotted deformations are the sum of all displacement vectors in x, y and z direction



(b) Von Mises stresses upon application of loading *before* eigenvalue analysis and nonlinear analysis. Maximum stress close to yielding stress



(c) Total deformations at the end of geometrical nonlinear analysis



(d) Von Mises stresses at the end of geometrical nonlinear analysis. Maximum stress far beyond yield stress

Figure 3.20: Maximum deformations and stresses during preloading and at the end of nonlinear analysis

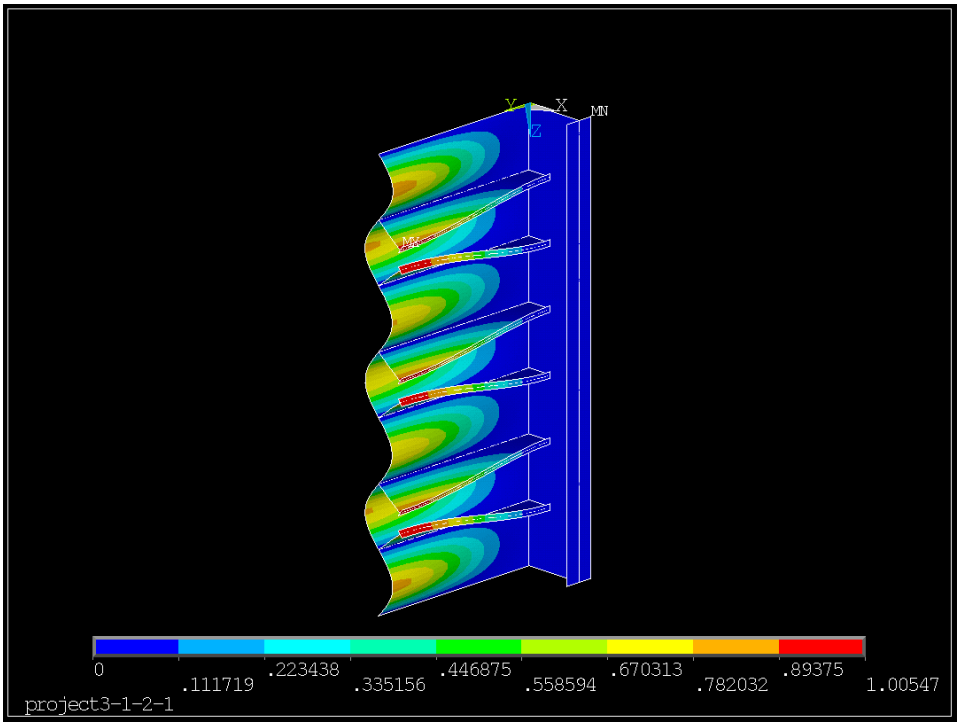


Figure 3.21: First buckling mode which is used as initial imperfection shape

4

Approach

4.1. Introduction

This chapter will discuss the complete approach for generating training data, training predictive models and analyzing the results. Figure 4.1 shows a flowchart of the global process. References to the corresponding sections are added to the flowchart where applicable.

Section 4.2 discusses the choices and considerations for the software used. In Section 4.3 a small overview of the mechanical models is given. Apart from the main mechanical model which was described in Chapter 3, also two simplified models will be treated. Section 4.4 describes the process of generating the datasets. These datasets are generated by setting up a parametric model in ANSYS FEA after which a series of analyses can be performed automatically for a number of unique sets of input values. In Section 4.5 some details are given about how these data are prepared before they can be used for training the neural networks. The process of training a neural network is discussed in Section 4.6. Then Section 4.7 explains how the hyperparameters for these neural networks are optimized using a genetic algorithm. Then finally in Section 4.8, the other interpolation techniques are briefly introduced.

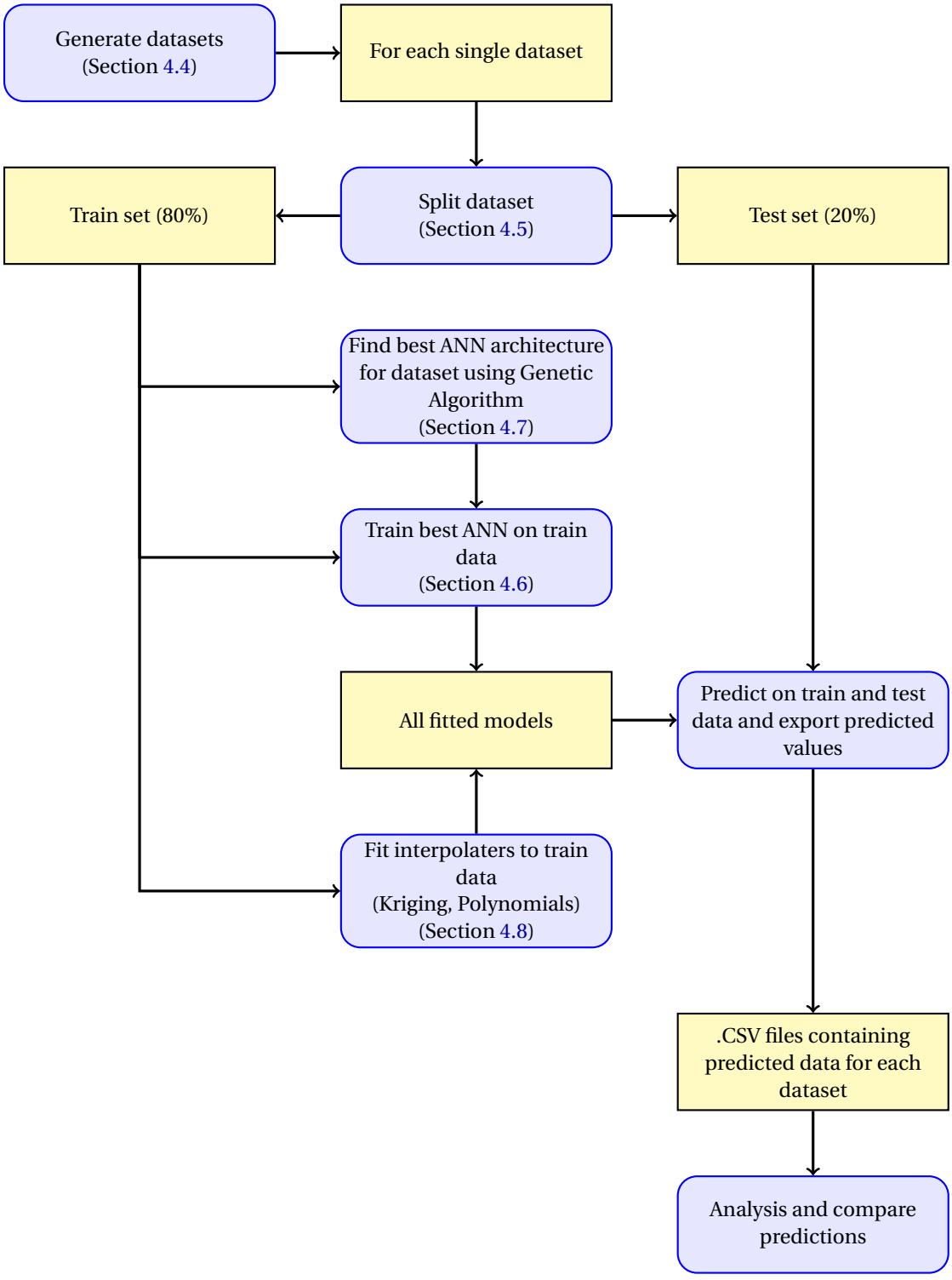


Figure 4.1: Flowchart vizualizing the global process from data generation up to training neural networks and comparing the accuracies with interpolation techniques

4.2. Software

This section will give an overview of the software packages and modules that are used in this project, including the motivations for selecting these.

4.2.1. Python

The Python programming language will be used as the central framework for all processes within this project. All tasks, ranging from creating parametric input files for FE software, deploying Machine Learning algorithms, up to collecting and comparing all results, are done using programming in Python. All the Python scripts that are written for these tasks can be found in the appendices.

Python is a high-level programming language which has a very clear syntax, meaning that it is relatively easy to read and understand the written code compared to other languages. The language is open-source and knows a wide community of developers and contributors. This enables the user to import so called packages with functionalities written for certain tasks. In this way, the Python user is not forced to reinvent the wheel for each specific problem he encounters.

Many Python packages have also been developed for Machine Learning applications. These packages enable the users to use common ML algorithms without having to code the exact mathematical formulations and algorithms themselves. In this way, ML models can be constructed relatively quickly, reducing the chance of making errors. Some commonly used packages are listed below with a short introduction and the motivations why or why not the package is chosen to use in this project.

- **TensorFlow:** TensorFlow is an open-source ML package that is very suitable for many ML algorithms. TensorFlow is developed by Google and enables the user to specify a certain *graph* of mathematical operations. This graph is essentially a series of mathematical steps to be performed. TensorFlow translates this graph, or series of mathematical operations, to high performance C++ binaries. The benefit is that the TensorFlow functions figures out how to properly perform the mathematical operations as efficiently as possible across the different processing units of the computer. The user can focus on the ML problem itself instead of worrying about e.g. matching multidimensional tensors. TensorFlow is a relatively low-level application which can be used directly, or it can be used via a more high-level application like Keras. (See next item).
- **Keras:** Keras is an open-source package developed specifically for the design of Artificial Neural Networks. It employs the functionalities of TensorFlow in the background, making it a higher level API. This enables the user to focus more on quick experimentation and a more intuitive creation of ANN's. While Keras is easier to use than TensorFlow, it is also less flexible. For the creation of more advanced ANN's or ML algorithms in general, it is advised to use a more low-level application like TensorFlow instead. In this project, the focus will be limited to very common feed-forward neural networks. It is perfectly possible to construct this type of networks using Keras. Therefore, in this project Keras will be used for constructing Artificial Neural Networks.
- **DEAP:** DEAP stands for Distributed Evolutionary Algorithms in Python. It is developed for deploying evolutionary algorithms in a convenient way. The theory of Evolutionary Algorithms, also called Genetic Algorithms, is discussed in Chapter 2. Using DEAP, the user is able to conveniently define a genotype of individuals in the desired format, create generations of random individuals and let the evolution run generation after generation. Predefined crossover functions and mutation functions can be applied to the individuals, but it is also possible to define custom crossover and mutation functions. There is also the possibility of parallel processing in which computation tasks can be distributed across multiple CPU cores to enhance the computational speed of the Genetic Algorithm. The application of the package DEAP in this project is described in Section 4.7
- **Scikit-Learn:** Scikit-Learn is a general purpose ML library containing functions for many ML algorithms. It also contains a package for Artificial Neural Networks, called the Multi-Layer Perceptron regressor. However, the options and functionalities in this package are relatively limited compared to those in the Keras package. That is the reason why Scikit-Learn will not be used for the construction of Artificial Neural Networks in this project. But apart from the ML algorithms, many additional tools are available in the library that are useful in other stages of the process. Some of these tools will be used in this project for pre-processing of the datasets (see Section 4.5). Also, the interpolation techniques as described in Section 4.8 will be implemented using the Scikit-Learn library.

4.2.2. Finite element software

The finite element software used in this project is ANSYS. There are two environments in which this software can be used: ANSYS Workbench and ANSYS APDL. In this project, the APDL framework will be used.

In the following subsections, both frameworks and their features will be briefly introduced. In Section 4.2.3, the advantages and disadvantages of both frameworks are discussed, supporting the motivation for choosing APDL in this project instead of Workbench.

APDL: Ansys Parametric Design Language
The abbreviation APDL stands for Ansys Parametric Design Language, which is the native language of the ANSYS software. In APDL, all FEM modelling and analysis can be performed by entering a sequence of APDL commands or inputting a script file. It also has a graphical user interface, although very limited compared to the Workbench environment.

Generally, the APDL environment is more commonly used in scientific context, where more advanced pre-processing and post-processing steps are required. The user has a more 'direct' connection to the actual FE solver. It is possible to perform many mathematical operations like creating matrices, vectors and perform operations on these.

In ANSYS APDL, all tasks are performed within one interface only. All steps, starting from the modelling of the CAD geometry up to the actual solving, take place in one single interface.

ANSYS Workbench ANSYS Workbench is the modern interface which has a more user friendly Graphical User Interface and more advanced graphical features. Workbench uses the very same solver as the one in APDL, but the user does not need to know or use any commands. This makes it more intuitive for the user to quickly model and solve the problem.

The Workbench environment can be considered as the main, overall interface. As opposed to the APDL environment, where all tasks are performed in one and the same interface, Workbench includes multiple 'native' software packages, each capable of its own set of tasks.

For the modelling, two native CAD modules are available: DesignModeler and SpaceClaim. The latter one has a built-in Python interpreter in which a CAD model can be generated by writing a Python script. In this way, a rule-based design can be generated based on a set of defined parameters, which is a very useful feature for parametric design tasks.

A second important native package is ANSYS Mechanical, used for the structural analysis. In this program, the boundary conditions and loading are applied, the geometry is meshed and the actual solving is initiated. In this program, scripting is also possible. However, this is done in JScript, which is similar to JavaScript.

The Workbench itself also has a built-in IronPython console in which all the general processes within the Workbench environment can be scripted. This is a very useful feature for automating tasks inside Workbench.

Surrogate modelling ANSYS Workbench has a built-in feature for creating surrogate models. This feature is called the Response Surface. and is part of the ANSYS DesignXplorer module. There are five different fitting methods available for constructing a Response Surface.

It is interesting to note that one of these fitting methods makes use of Artificial Neural Networks. This was not foreseen before starting this project. The documentation regarding the settings or deployment of these ANN's is very limited However, the only hyperparameter than can be modified manually is the number of neurons, which is limited to a maximum of 10 neurons. It is not clear which activation functions, optimizers or other hyperparameters are used for these ANN's. Considering these limitations, it is expected that the Response Surface using ANN will not be able to use the full potential of ANN's.

4.2.3. Considerations Ansys Workbench or APDL

In this section, the considerations for choosing an ANSYS environment used in this project will be discussed. At the start of the project, the plan was to use the Workbench environment for generation of the training data. Also it was planned to provide a comparison of the performance of the Response Surface with the performance of the Artificial Neural Networks created in this project. But during the course of the project, many limitations in Workbench were encountered. It was discovered that APDL would be preferred above Workbench. In the next subsections, a more in depth discussion of both interfaces will explain why eventually APDL would be preferred in this project.

Since no direct comparison can be made between the ANSYS Response Surface and the Artificial Neural Networks, two of the available fitting methods of the Response Surface, Kriging interpolation and quadratic interpolation, are deployed directly using the Scikit-Learn library in Python. The implementation of these methods is explained in Section 4.8.

Overall workflow Choosing to use APDL for the FEM computations simplifies the total workflow since in this case there are only two frameworks to deal with. Instead of using different submodules, scripting API's and even different scripting languages as required in Workbench, all processes, ranging from generating data up to applying machine learning algorithms and analyzing the results, can be controlled directly from one Python script.

License issues ANSYS is an expensive software package. Therefore, the number of available licenses within a company or university is limited. These licenses are available through a remote license server. When the maximum number of licenses is already used by the employees or students, it is not possible to use to program or parts of the program.

This lead to issues when automating tasks inside Workbench for running a series of analyses in order to create a dataset. For each new analysis, the program needs to open SpaceClaim in the background for creating the CAD geometry. Next, for solving the model, also Mechanical has to be opened in the background. Each of these native programs require their own license. When running many successive analyses, it often occurs that the maximum number of licenses is reached when trying to open such a program. This results in license failures and therefore unsolved analyses.

These issues do not occur in APDL, since all tasks are performed within in one interface only as pointed out in Section 4.2.2. Once the user has access to the license, the program can be used and all automated tasks can be performed without any issues. This is a major advantage and is one of the main motivations for using the APDL environment in this project.

Computational time As pointed out in the previous subsection, each time when an analysis is performed in Workbench, the native programs have to be opened in the background. The starting up of these programs requires some time and computational effort. This is not a problem when trying to solve a few analyses, but for larger numbers of design points these time delays can count up to significant amounts.

For a simple linear elastic analysis with less than 10.000 elements it was found that running the analysis in Workbench took approximately 47 seconds, whereas solving the same model in APDL took less than one second only. Note that the actual solving time is equal in both environments, since both use the same solver. The difference in computation time comes from the initiation of native software packages in Workbench.

Since many analyses will have to be run in order to create the datasets, this drastic time reduction is an important reason to choose APDL.

Parameter constraints When running a series of FE analyses of random designs, some of these designs are known to be infeasible beforehand. Geometrical clashed might also be possible. When generating data for a predictive model, one would want to avoid generation of data which is known to be infeasible or has an invalid geometry, since it is useless to train a model on these data. Refer to Section 4.4.3 for a more in-depth explanation of this concept.

In Workbench, it is not possible to directly set constraints on the relations between input parameters. There is a plug-in available called DxUtilities which can be downloaded from the ANSYS website and integrated in the program. Here the user can define functions which describe certain conditions which have to be fulfilled for a design to be considered feasible. In order to avoid unnecessary computations of design points that are known to be infeasible, these designs can be filtered out.

This added functionality is unfortunately only available for the generation of *design* points. For generating *verification* points in the Response Surface module, necessary for evaluating the accuracy of the response surface, it is not possible to set these parameter constraints.

A workaround could be to generate the design points inside the DesignOfExperiments module using the DxUtilities plug-in, after which these can be exported to a .CSV file and imported back as verification points.

However, it is not guaranteed that these verification points are positioned far away enough from the design points. Therefore, when the verification points are too close to the original design points, the evaluation of the performance of the model might give the false impression that the response surface performs very well on the verification points while in reality this may not be the case.

In this project, the parameter constraints will be defined within Python functions in the main Python script. Valid splitting techniques for creating training datasets and verification datasets are used with the Scikit-Learn data processing tools. This guarantees a valid verification of the accuracy of the predictive models.

4.3. Simplified mechanical models

Apart from the main mechanical problem as described in Chapter 3, two additional, simplified mechanical models are used for creation of predictive models. These simplified models serve to reduce the complexity while getting acquainted with the machine learning algorithms, as well as providing insight in performance of different predictive models for different mechanical models and output parameters. In total, three different mechanical problems will be analyzed.

1. Elastic deflection of stiffened, simply supported steel plate under out of plane loading
2. Linear eigenvalue analysis of stiffened, steel plate under in plane compressional stresses
3. Non-linear buckling analysis of stiffened steel plate (main problem as described in Chapter 3)

Model number	Nature of analysis	Output parameter(s)
1	Linear	$u_{x,max}$ [mm]
2	Linear	λ_1 [-]
3	Nonlinear, linear	$\sigma_{eqv,max}$ [MPa], λ_1, $u_{x,max}$

Table 4.1: Overview of material properties as used throughout the project

The simplified models share the following properties with the main mechanical model 3. These can be referred to in Chapter 3.

- Material properties (linear)
- Loading conditions

Note that the loading conditions in the main problem include a combination of in-plane and out of plane loading. The simplified models are only loaded by one component per model: Model 1 is only loaded by out of plane pressure in x-direction. Model 2 is only loaded by in plane compressional stresses in y and/or z-direction.

Geometry The simplified models 1 and 2 share the same geometric parameters and support conditions. The difference between the geometry of the simplified models and the main problem is that the simplified models do not contain any supporting columns. No reduced modelling is applied to the simplified models. See Figure 4.2 for the geometry of the simplified structure. Table 4.2 shows an overview of the geometrical parameters and their description.

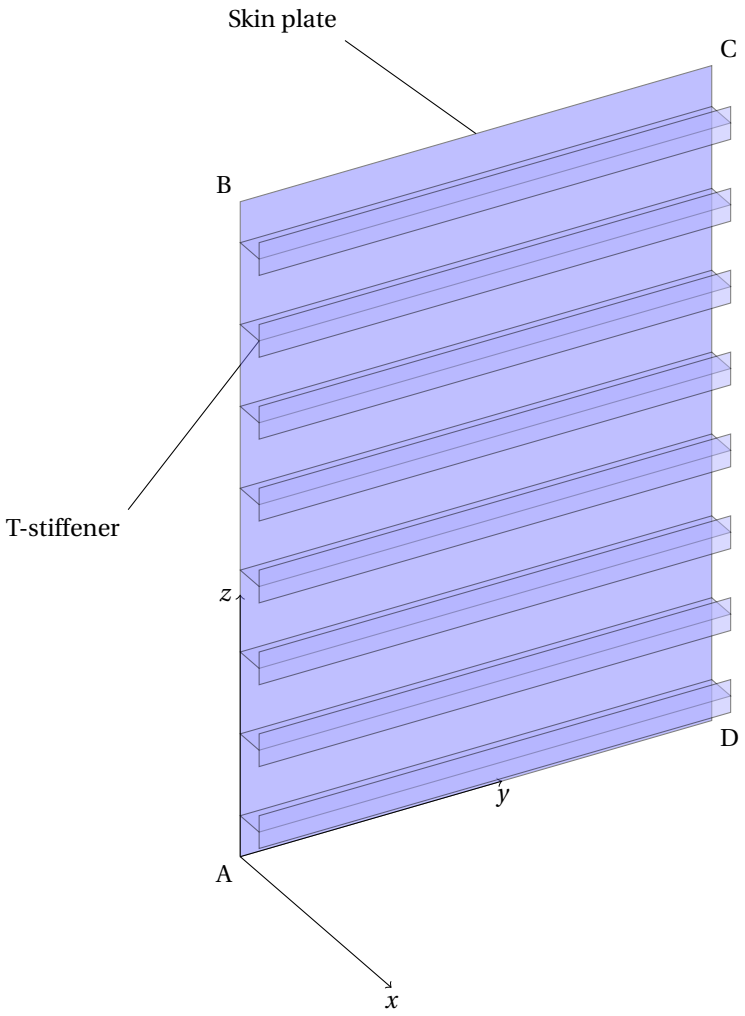


Figure 4.2: 3D geometry of the simplified models numbers 1 and 2. No columns included. All edges are supported in x-direction

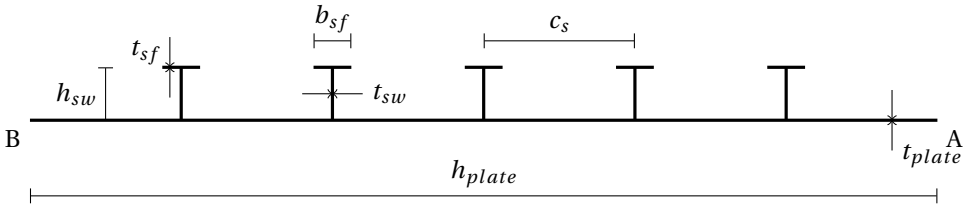


Figure 4.3: Cross sectional drawing through the xz-plane indicating the dimensions of the longitudinal T-stiffeners and the plate

Structural Element	Symbol	Description
Skin plate	t_{plate}	Skin plate thickness
	h_{plate}	Skin plate height
	w_{plate}	Skin plate width (= c.t.c. distance columns)
T-stiffeners	n_s	Number of T-stiffeners
	c_s	Center-to-center distance
	h_{sw}	T-stiffener web height
	t_{sw}	T-stiffener web thickness
	b_{sf}	T-stiffener flange width
	t_{sf}	T-stiffener flange thickness

Table 4.2: Overview of geometry parameters of the simplified models. Geometry parameters of the columns are not included.

Boundary conditions Both simplified models have different boundary conditions. See Table 4.3 and Table 4.4 for overviews of the boundary conditions for simplified problem 1 and 2 respectively.

#	Description	Location [mm]	u_x	u_y	u_z	θ_x	θ_y	θ_z
1	A-B: Plate edge	$y = 0$, $x = 0$	0	-	-	-	-	-
2	B-C: Plate edge	$z = h_{plate}$, $x = 0$	0	-	-	-	-	-
3	C-D: Plate edge	$y = w_{plate}$, $x = 0$	0	-	-	-	-	-
4	A-D: Plate edge	$z = 0$, $x = 0$	0	-	-	-	-	-

Table 4.3: Boundary conditions applied to the nodes of simplified **model 1**. Only the plate edges are supported in x-direction.

#	Description	Location [mm]	u_x	u_y	u_z	θ_x	θ_y	θ_z
1	A-B: Plate edge, stiffener sections	$y = 0$	0	u_{yy}	0	-	-	-
2	B-C: Plate edge	$z = h_{plate}$, $x = 0$	0	-	$-u_{zz}$	-	-	-
3	C-D: Plate edge, stiffener sections	$y = w_{plate}$	0	$-u_{yy}$	0	-	-	-
4	A-D: Plate edge	$z = 0$, $x = 0$	0	-	u_{zz}	-	-	-

Table 4.4: Boundary conditions applied to the nodes of simplified **model 2**.

Range of parameter values For both simplified models, multiple datasets will be generated. Each dataset will have a different number of free variables and a different number of data samples. This is done in order to study the relation between these properties and the possible accuracies of the predictors. An overview of the applied parameter value ranges for all mechanical models is provided in tables 4.8 and 4.9 in Section 4.4.2.

Note that these simplified models only serve for experimental purposes. The parameter value ranges and their relations are not limited by 'practical' values. It is not considered whether these values are reasonable or can be considered an efficient design.

4.4. Generation of data

A sufficient amount of training data is required for training the predictive models to ensure high accuracy. A dataset is generated that contains the results of a number of FE analyses. See Table 4.5 for the general shape of a dataset. Each row in this dataset is a list containing the input parameter values for a single analysis and the corresponding output values found by solving the actual FE analysis. Each column of the table represents a specific input or output parameter. The data are stored in a comma-separated values file (CSV file) which makes it convenient to be used in different frameworks like Python and Excel.

	Input 1	Input 2	⋯	Input n	Output 1	Output 2
Sample 1	.	.	.	.	.	.
Sample 2	.	.	.	.	.	.
⋮	.	.	.	.	.	.
Sample m	.	.	.	.	.	.

Table 4.5: General shape of a dataset. Each row represents a single FE analysis summarizing its input parameters and the resulting output parameters produced by the analysis.

Figure 4.4 shows a flowchart that visualizes the complete process of the generation of datasets. Hyperlinks are included inside the flowchart that redirect to the corresponding section where the process is described in detail.

4.4.1. Create parametric models

Parametric models form the basis for each FE analysis. These parametric models are set up as ordinary text files containing APDL scripts. These APDL scripts are a series of APDL commands that are read by the ANSYS APDL software, after which ANSYS performs each line of commands subsequently. All necessary steps are contained within such a command file.

1. Define input parameter values
2. Create geometry
3. Define element settings
4. Mesh geometry
5. Add boundary conditions and loading
6. Set solve settings
7. Solve
8. Post processing

For each FE analysis, a unique APDL file is created.

Template APDL files Each FE analysis has a unique set of input parameters. Also, among different mechanical models, different solver settings and post processing methods are needed. Therefore, so called APDL template files are created. Such a template file contains the most general APDL commands that are suitable for *all* projects. A part of the commands that are unique to one of the projects are left out. These template files are then modified by a Python script in order to complete it with the correct commands corresponding to the set of input parameters and analysis settings for the single FE analysis. The general workflow in the Python script for reading a template file and writing the final APDL files is as follows:

1. Read template file, corresponding to the actual project.
2. Search for the keywords in the template
3. Replace the keywords with the correct APDL commands
4. Write the final APDL input file

An example of a template input file is included in Appendix G An example of a complete APDL file is included in Appendix K

4.4.2. Define projects

In the context of this thesis, a *project* is a collection of datasets belonging the same mechanical model (model 1, 2 or 3) and having the same number of free input parameters. These projects are defined inside an Excel file, which is read by the Python script. Also the ranges of the parameter values are defined within this Excel file.

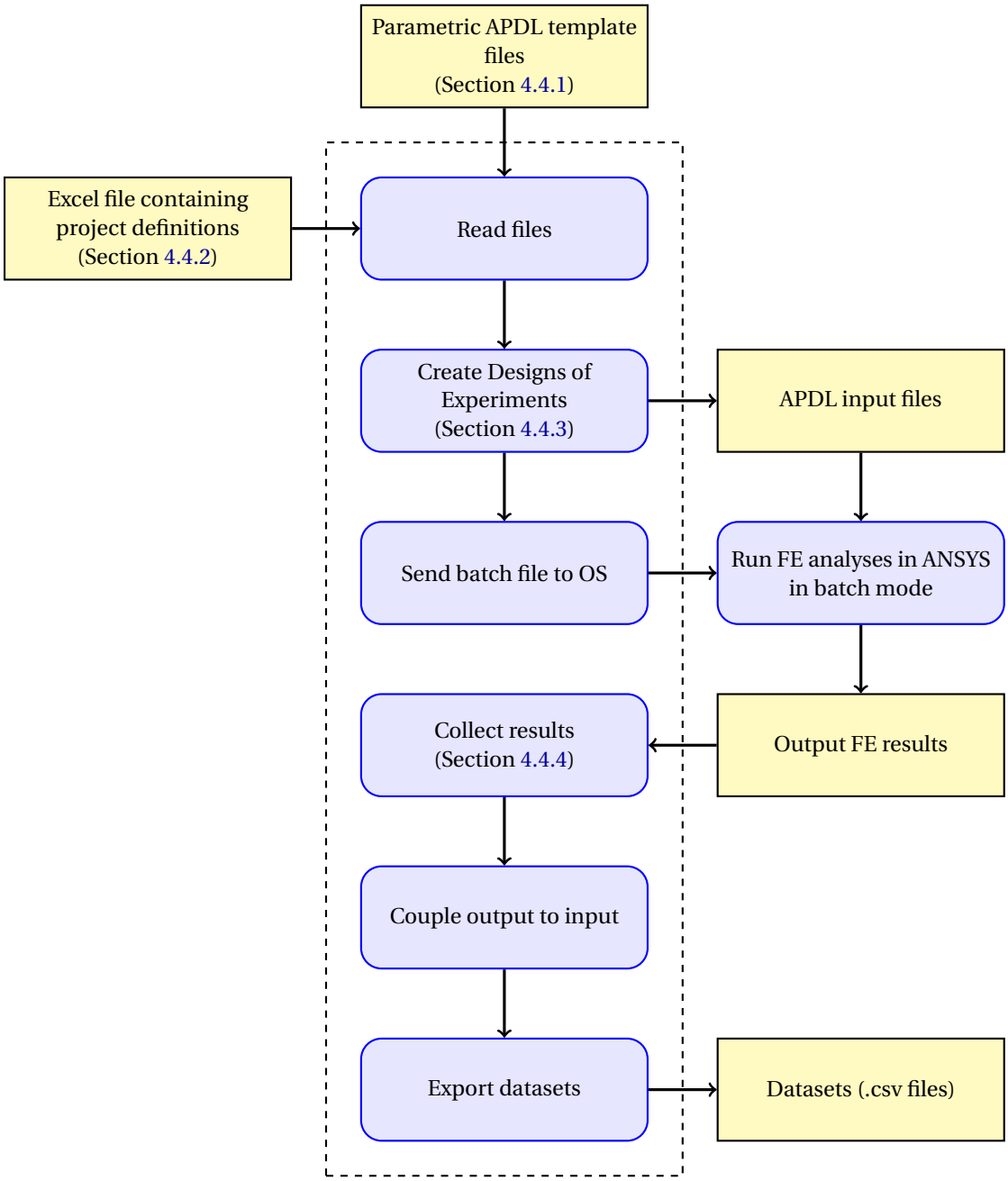


Figure 4.4: Flowchart describing the process of generating datasets that will serve as training data for the neural networks. All processes within the dashed frame are automatically performed in the Python scripts.

This is organized in such a way in order to study the relations of prediction accuracy as a function of the number of free variables and the number of data samples in a dataset. Table 4.6 visualizes the project numbers. Each 🗎-icon represents a single dataset for which an artificial neural network will be trained.

			Number of data samples		
Mechanical model	**Project number**	**Num. of variables**	**40 samples**	**80 samples**	**160 samples**
1	1-1-1-1	4	🗎	🗎	🗎
	1-2-1-1	5	🗎	🗎	🗎
	1-3-1-1	6	🗎	🗎	🗎
	1-4-1-1	7	🗎	🗎	🗎
	1-5-1-1	8	🗎	🗎	🗎
2	2-1-1-1	4	🗎	🗎	🗎
	2-2-1-1	5	🗎	🗎	🗎
	2-3-1-1	6	🗎	🗎	🗎
	2-4-1-1	7	🗎	🗎	🗎
	2-5-1-1	8	🗎	🗎	🗎
3	3-1-2-1	13	🗎	🗎	🗎
	3-1-2-2	13	🗎	🗎	🗎
	3-1-2-3	13	🗎	🗎	🗎
	3-1-2-4	13	🗎	🗎	🗎
	3-1-2-5	13	🗎	🗎	🗎

Table 4.6: Definition of projects, each belonging to a mechanical model, a number of free variables and parameters ranges. For each project, multiple datasets are created with a varying number of data samples.

Proj. num.	**Num. Vars**	**Variables**	**Outputparameter**
1-1-1-1	4	t_p,h_{sw},t_{sw},q_0,	$u_{x,max}$
1-2-1-1	5	t_p,n_s,h_{sw},t_{sw},q_0,	$u_{x,max}$
1-3-1-1	6	h_p,t_p,n_s,h_{sw},t_{sw},q_0,	$u_{x,max}$
1-4-1-1	7	t_p,n_s,h_{sw},t_{sw},w_{sf},t_{sf},q_0,	$u_{x,max}$
1-5-1-1	8	h_p,t_p,n_s,h_{sw},t_{sw},w_{sf},t_{sf},q_0,	$u_{x,max}$
2-1-1-1	4	t_p,n_s,h_{sw},t_{sw},	λ_1
2-2-1-1	5	h_p,t_p,n_s,h_{sw},t_{sw},	$\lambda 1$
2-3-1-1	6	t_p,n_s,h_{sw},t_{sw},w_{sf},t_{sf},	λ_1
2-4-1-1	7	h_p,t_p,n_s,h_{sw},t_{sw},w_{sf},t_{sf},	λ_1
2-5-1-1	8	h_p,t_p,n_s,h_{sw},t_{sw},w_{sf},t_{sf},σ_{zz},	λ_1
3-1-2-1	13	h_p,t_p,n_s,h_{sw},t_{sw},w_{sf},t_{sf},q_0,σ_{zz},	$\sigma_{max,all}$
3-1-2-2	13	h_p,w_p,t_p,n_s,h_{sw},t_{sw},w_{sf},t_{sf},q_0,σ_{yy},σ_{zz},c_{imp},	$u_{x,max}$
3-1-2-3	13	h_p,w_p,t_p,n_s,h_{sw},t_{sw},w_{sf},t_{sf},q_0,q_1,σ_{yy},σ_{zz},c_{imp},	$\sigma_{max,plate,mid}$
3-1-2-4	13	h_p,w_p,t_p,n_s,h_{sw},t_{sw},w_{sf},t_{sf},q_0,q_1,σ_{yy},σ_{zz},c_{imp},	$\sigma_{max,stiff.,mid}$
3-1-2-5	13	h_p,w_p,t_p,n_s,h_{sw},t_{sw},w_{sf},t_{sf},q_0,q_1,σ_{yy},σ_{zz},c_{imp},	λ_1

Table 4.7: Overview of project numbers including the input parameters and output parameters

Proj. num.	h_p [mm]	w_p [mm]	t_p [mm]	n_s [-]	h_{sw} [mm]	t_{sw} [mm]	b_{sf} [mm]	t_{sf} [mm]
1-1-1-1	2000	2000	8, 24	6	60, 200	6, 12	0	0
1-2-1-1	2000	2000	8, 24	4, 8	60, 200	6, 12	0	0
1-3-1-1	2000, 4000	2000	8, 24	4, 14	60, 200	6, 12	0	0
1-4-1-1	2000	2000	8, 24	4, 8	60, 200	6, 12	60, 160	6,12
1-5-1-1	2000, 4000	2000	8, 24	4, 14	60, 200	6, 12	60, 160	6,12
2-1-1-1	2000	2000	8, 24	4, 8	60, 200	6, 12	0	0
2-2-1-1	2000, 4000	2000	8, 24	4, 14	60, 200	6, 12	0	0
2-3-1-1	2000	2000	8, 24	4, 8	60, 200	6, 12	60, 160	6,12
2-4-1-1	2000, 4000	2000	8, 24	4, 14	60, 200	6, 12	60, 160	6,12
2-5-1-1	2000, 4000	2000	8, 24	4, 14	60, 200	6, 12	60, 160	6,12
3-1-2-1	3000, 6000	2000, 5000	6, 30	4, 25	100, 400	6, 24	50, 350	6, 30
3-1-2-2	3000, 6000	2000, 5000	6, 30	4, 25	100, 400	6, 24	50, 350	6, 30
3-1-2-3	3000, 6000	2000, 5000	6, 30	4, 25	100, 400	6, 24	50, 350	6, 30
3-1-2-4	3000, 6000	2000, 5000	6, 30	4, 25	100, 400	6, 24	50, 350	6, 30
3-1-2-5	3000, 6000	2000, 5000	6, 30	4, 25	100, 400	6, 24	50, 350	6, 30

Table 4.8: Overview of geometry parameter value ranges for all projects. A single value in a cell implies a fixed value for that parameter. Two values in a cell represent the minimum and maximum possible values respectively.

Proj.	q_0 [MPa]	q_1 [MPa]	σ_{yy} [MPa]	σ_{zz} [MPa]	AMP [mm]
1-1-1-1	0.05, 0.2	Q0	0	0	0
1-2-1-1	0.05, 0.2	Q0	0	0	0
1-3-1-1	0.05, 0.2	Q0	0	0	0
1-4-1-1	0.05, 0.2	Q0	0	0	0
1-5-1-1	0.05, 0.2	Q0	0	0	0
2-1-1-1	0	0	1.0	0	0
2-2-1-1	0	0	1.0	0	0
2-3-1-1	0	0	1.0	0	0
2-4-1-1	0	0	1.0	0	0
2-5-1-1	0	0	1.0	0.1, 0.5	0
3-1-2-1	0.05, 0.2	0.05, 0.2	100, 250	50, 150	1.0, 12.5
3-1-2-2	0.05, 0.2	0.05, 0.2	100, 250	50, 150	1.0, 12.5
3-1-2-3	0.05, 0.2	0.05, 0.2	100, 250	50, 150	1.0, 12.5
3-1-2-4	0.05, 0.2	0.05, 0.2	100, 250	50, 150	1.0, 12.5
3-1-2-5	0.05, 0.2	0.05, 0.2	100, 250	50, 150	1.0, 12.5

Table 4.9: Overview of loading parameter value ranges for all projects. A single value in a cell implies a fixed value for that parameter. Two values in a cell represent the minimum and maximum possible values respectively. **AMP** is the scaling factor with which the buckling shape is multiplied in order to include geometric imperfections

4.4.3. Create design of experiments

For creation of each dataset, a proper Design Of Experiments (DOE) must be created. The DOE is defined as a set of samples, each sample containing a unique set of input values for a FE analysis. The design of experiments in this case is a $m \times n$ matrix where m represents the number of samples and n being the number of input parameters for each sample.

All input parameters and their possible values define the total design space. This design space is the collection of *all* possible combinations of values for the input parameters. Each sample will be a unique combination of values for its input variables. This will be called a design point. The design point can be visualized as a single point in the n-dimensional design space, where n is equal to the number of free variables.

Two important features regarding the design points determine how well the predictive models can be trained. The first one is the number of design points used for the training process. It is expected that the predictive model will perform better for a higher number of design points. The second one is the sampling method of the design points from the design space.

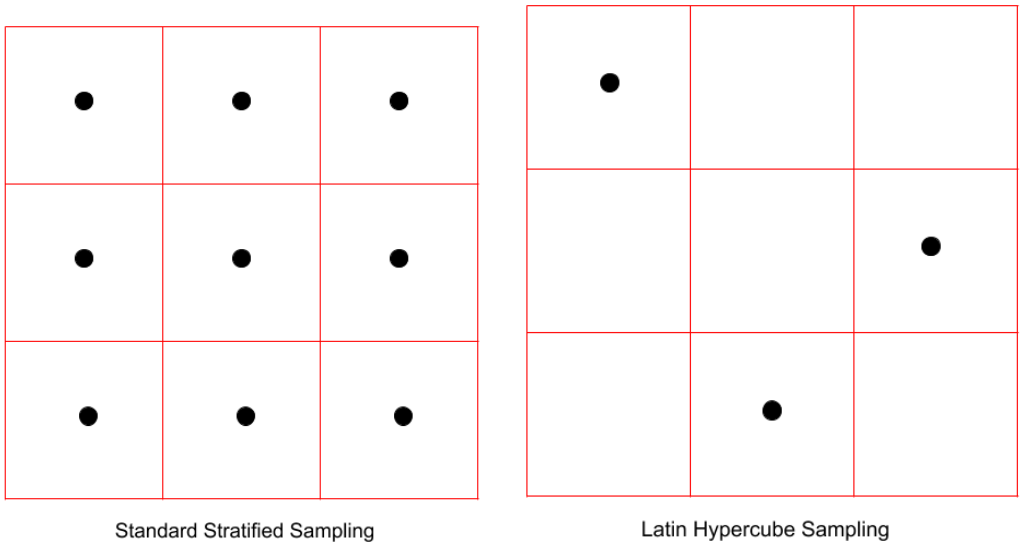
Number of samples The predictive models are expected to perform better when trained with a higher number of training samples. The drawback is that the generation of training data is computationally expensive. For each design point, a complete FE analysis will have to be run. Therefore it is important to find a good balance between limiting the amount of time needed for generating the datasets and making sure that enough samples are available for accurate prediction.

In this thesis, the number of samples will be varied in order to draw conclusions about the resulting performance of the predictive models. For the two simplified mechanical models 1 and 2, datasets with 40, 80 and 160 samples will be generated. For the main mechanical model, 80, 160 and 320 samples will be generated. Out of each dataset, 80% of the samples will be used for actually training the predictive models. The other 20% of the samples will be used for validating the performance of the predictive models. More about splitting datasets is written in Section 4.5.

Sampling method Choosing the right sampling method ensures a proper distribution of data points in the design space. For all datasets, the Latin Hypercube Sampling (LHS) method will be used. The LHS is a statistical sampling method for drawing random samples from a multidimensional design space. The main advantage of the LHS method over standard grid sampling is that the design space can be covered with less data samples since input values are not repeated. A general rule of thumb in machine learning is that at least 5 values for each input parameters are needed [25] to capture the response of the system accurately. When applying the grid like sampling method for a problem with 8 input parameters, a dataset containing $5^8 = 390625$ data samples would be needed, which is obviously infeasible in the context of this thesis.

In LHS, the range of each parameter is split up in m equally spaced partitions. The sampling method is set up such that from each partition, only *one* value is selected. In this way, each sampling point has a unique value for each input parameter.

See Figure 4.5 for a visualization of both concepts in 2 dimensions. Imagine that for both input parameters, 3 unique values have to be drawn In the grid sampling method in Figure 4.5a, 9 data points are needed to cover the design space, while for the LHS method in Figure 4.5b, the ranges of both input parameters are covered with only 3 data points.



(a) A grid-like sampling method requires m^n data samples to cover the design space, where m is the number of unique values for a single parameter and n the number of dimensions.

(b) The Latin Hypercube sampling method requires less data while still covering the range of the dimensions with the same amount of unique values.

Figure 4.5: Comparison of grid sampling method and Latin Hypercube Sampling method

For sampling with the LHS method in Python, a special package is be imported. This library is called *doepy*, and is available for free. The source code can be found on the GitHub page [2].

Filter out infeasible designs For some combinations of input values it is known that the design will be infeasible. This can be based on experience, limitations in the Eurocode or a simple mathematical proof that the structure will fail. Other design points may be infeasible due to geometrical clashes, like flanges of T-stiffeners overlapping eachother. It can also be the case that the structure cannot be fabricated at all, due to a lack of working space for the welder.

In order to avoid generation of useless data and training predictive models on these data, the infeasible designs have to be filtered out from the design space in advance. This can be done by specifying a set of conditions that each design point has to fulfil. If the design point does not fulfil the conditions, the design point will be filtered out.

Of course, each time when design points are filtered out, the number of available design points decreases. Since a specific number of design points is desired, new samples will have to be drawn. In order to make sure that all data points remain optimally distributed, a completely new dataset will have to be drawn with a higher number of samples to compensate for the samples that will be filtered out. This is an iterative procedure which is visualized in Figure 4.6. A visualization is shown in Figure 4.7

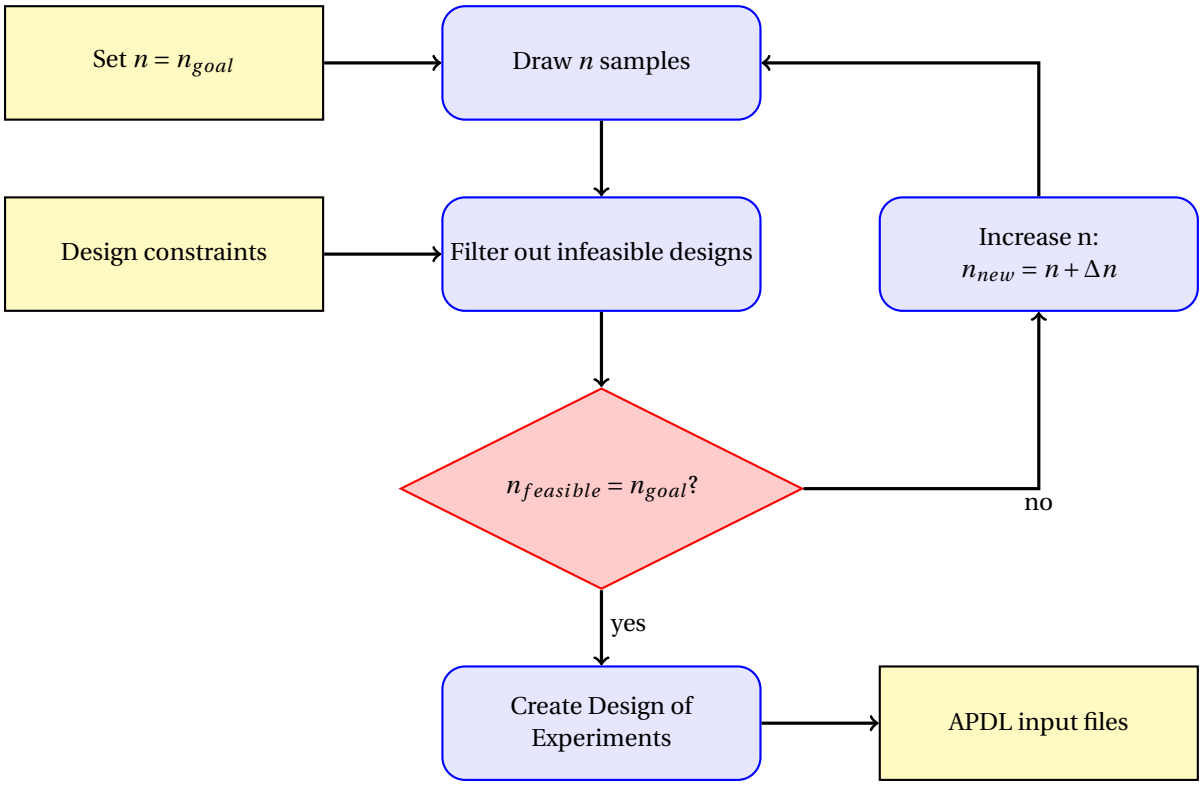


Figure 4.6: Flowchart visualizing the iterative procedure of sampling data and filtering infeasible designs

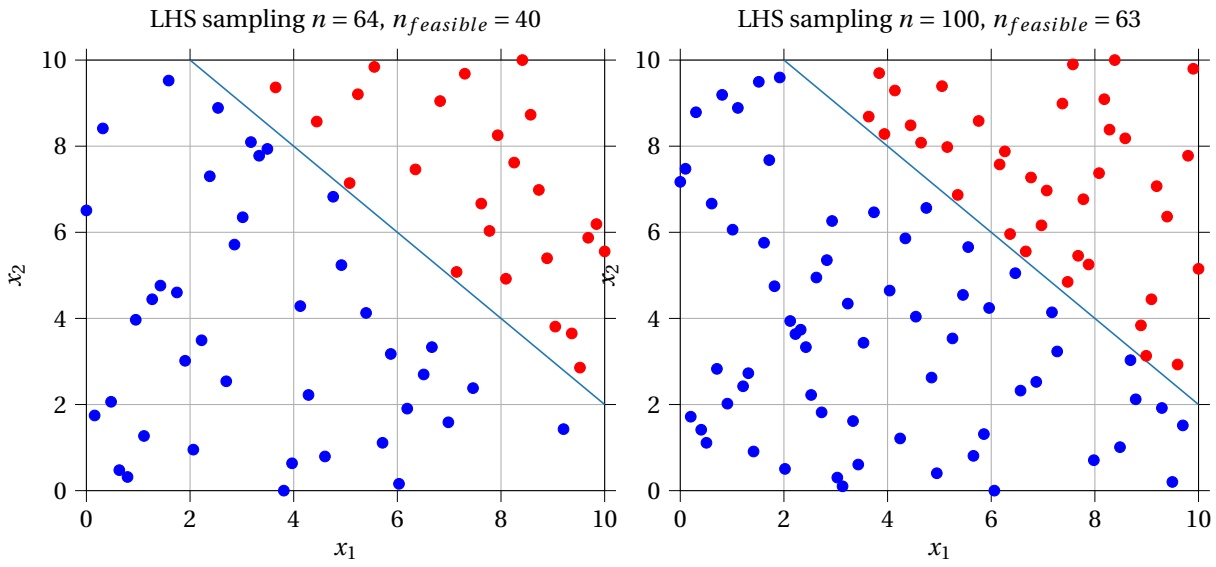


Figure 4.7: Visualization of filtering infeasible designs and drawing more samples to end up with desired number of feasible samples. The problem has two variables: x_1 and x_2. The variables have the same ranges: $0 \leq x_1 \leq 10$ and $0 \leq x_2 \leq 10$. A design where the sum of x_1 and x_2 is higher than 12 can be considered infeasible. The red dots represent the infeasible designs that are filtered out. In the right figure, the amount of samples is increased to 100 samples after which 63 designs remain. The iteration is repeated until the number feasible designs is equal to the number of desired samples.

4.4.4. Run FE analyses and collect the results

For each data sample a APDL input file is created. Also a batch file is written which contains commands that will send each APDL file to the ANSYS solver. The analyses are run in batch mode and the relevant output values of the FE analyses are collected and coupled to the corresponding input data. These data are organized in datasets as shown in Table 4.5 in Section 4.4. These tabular data are then exported to .CSV files. Some modifications will have to be performed on the data before using these data for training. These steps are explained in Section 4.5.

4.5. Preparation of data

4.5.1. Split data

In order to be able to verify the performance of the trained artificial neural network, a dataset will be split into a training dataset and a testing dataset, which is a common procedure in machine learning. The testing dataset is held out, so that the neural network is trained on *only* the training dataset. The testing dataset is used to verify the prediction accuracy of the predictive model on data it has not been trained on. It is important to verify this performance since it is the key idea of a predictive model to use it for all data points, not only for data it has already seen during the training process. The advised percentage of the dataset that is held out as a testing set is between 5% and 25%. In this project, 20% of the available data in each dataset will be held out for testing the predictive model.

The selection of test data points must be performed with care. It is not advisable to simply select 20% of the data points at either end of the dataset, as these points might be clustered in a particular region of the design space. Instead, the data points must be selected randomly. A special Python function available in the Sci-kit Learn library called *train-test-split* will be used for performing this operation.

4.5.2. Scale data

It is very important to scale the ranges of all input parameters to the same range. These ranges are preferably small, e.g. values between 0 and 1. The different input variables may have different units and different scales. The thickness of a steel plate for example is in the order of (tens of) millimetres, while the Young's modulus of this steel material may have a value of 2^{11} N/m^2. Big differences in scales along variables may result in poor performance of the neural network [15].

Two important scaling methods are

- Min-Max scaling (normalization)
- Standard scaling (standardization)

Where Min-Max scaling scales all data to values within a specified interval [5] and and Standard scaling scales all the data to a standard normal distribution [8].

It is advised to standardize the dataset if the data is normally distributed [15]. Otherwise, min-max scaling is a better option. Since the LHS method results in uniform distributions for all input parameters, all input variables will be scaled to fit within a range between 0 and 1.

Scaling output data may be necessary in case multiple output parameters are to be predicted by one neural network and when the values have a large difference in range. Since in this project only one output parameter is to be predicted at a time, scaling is not required and will thus not be applied.

4.5.3. Modify input data

There are more options available for to manipulate the data in order to improve training capabilities of the predictive models. These are both related to decreasing the dimensionality of the problem.

Feature selection Feature selection is the process of choosing certain input parameters to be included in the training dataset. Some variables might be excluded when it is expected that these parameters have a low influence on the output parameters. In this project however, the response of the system is determined by interaction of all the input variables described by physical laws. It is therefore expected that *all* input variables have an influence on the output values. In this project, no input variables are excluded.

Feature extraction Feature extraction is the process of creating new input variables that are derived from the original input variables. These derived variables may be expected to have a high influence on the output values. An example in the domain of structural mechanics might be to derive the torsional stiffness of the system from the input variables describing the dimensions and material properties of the system.

Feature extraction is not within the scope of this project. A suggestion for further research considering this point is added in Chapter 7

4.6. Predictive modeling using Artificial Neural Networks

Theory of Artificial Neural Networks is treated in detail in Chapter 2. This section will discuss the application of the method in this thesis, together with motivations for certain choices.

4.6.1. Create the ANN model

For creation and fitting ANN models, the Python library called Keras is used. In Section 4.2, an introduction to this library is given. Many different types of neural networks exist and can be created with the Keras library. However, in this project, the ANN's will be limited to single hidden layer feedforward neural networks.

For this purpose, a custom Python class is created and stored in a separate module, in which the functions for creating and fitting the ANN are written. This class and its functions can conveniently be re-used in different scripts. The script file containing this class has been added in Appendix B. The desired hyper parameters can be provided in the function arguments.

4.6.2. Choose hyper parameters

The capability of the ANN to fit to the data is highly dependent on network architecture and the chosen set of hyperparameters. Choosing the right set of hyperparameters is one of the most important and time consuming tasks when working with Artificial Neural Networks, or any machine learning method in general. Since the final performance of the network can only be evaluated after training it, the time needed for evaluating multiple combinations of hyperparameters can quickly become a limitation. Therefore it is necessary to find a way to find the optimal hyperparameters in the most efficient way.

Hyperparameter	Options
Number of neurons	1 - 50
Activation function	Linear, Hyperbolic tangent, ReLu, Elu, Selu, Exponential, Sigmoid, Softplus
Optimizer	RMSprop, Stochastic gradient descent, ADAM, ADAGRAD, ADADELTA, ADAMAX, NADAM
Kernel initializer	Random uniform, Constant, Zeros
Bias initializer	Random uniform, Constant, Zeros

Table 4.10: An overview of the hyperparameters to be optimized and the possible options

Hyperparameter optimization For finding the optimal hyperparameters for Artificial Neural Networks, different approaches are possible.

- **Manual optimization**
  This is essentially the most basic yet cumbersome way of tuning hyperparameters. This is a process of trial and error of manually adjusting the hyperparameters. The performance is evaluated after training the network. While it is useful for getting a feeling quickly for which hyperparameters work well and which do not, it is very labour intensive for analyzing multiple sets of hyperparameters.
- **Grid search**
  A grid-like sampling method is used for selecting hyperparameters, much like the example shown in Figure 4.5a. Each point in the design space represents a neural network with a unique set of hyperparameters. Each neural network is trained and the best performing network is selected. The drawback of this method is that the number of individual settings increases exponentially with the number of hyperparameters (the curse of dimensionality), making it computationally expensive to train and test each neural network.
- **Randomized search**
  Randomized search is much like a Monte-Carlo simulation, randomly selecting a fixed number of samples from the total design space of hyperparameters. Since a fixed number of samples is drawn, the curse of dimensionality does not hold and the computational time can be controlled.
- **Genetic algorithm**
  The main shortcoming of the grid search and random search methods is that for each set of hyperparameters, a neural network has to be trained *regardless* of the results of the already performed evaluations. In this way, the knowledge about previous evaluations is not used and therefore many neural networks are trained with hyperparameters which are known to yield bad performing networks A genetic algorithm takes a different approach.

  First, a set of hyperparameters is randomly selected (just like random search), after which each set is evaluated. Then the algorithm zooms in into promising subregions of the design space with well-performing neural networks. New samples are drawn with hyperparameters from these subregions, expecting to find even better performing neural networks. This process is repeated until the set of hyperparameters is found to produce sufficiently accurate neural networks. In this thesis, this method will be used for hyperparameter optimization. The implementation is described in Section 4.7. The theory of genetic algorithms can be found in Chapter 2.

Hyperparameters to be optimized For ANN's there are many different hyperparameters that can be tweaked. In this project, the focus will be on five different hyper parameters that will be varied. These are listed in Table 4.10.

Note that the learning rate is not included in the list. This is chosen to avoid complexities in the hyper parameter optimization using Genetic Algorithms. Some optimizers use a fixed learning rate, while other optimizers have an adaptive learning rate which changes during the training process. Providing a value for the learning rate will only be useful in case an optimizer is applied with a fixed learning rate. The complexity that arises with genetic algorithms concerning this point will be explained in Section 4.7.1.

Define loss function Different loss functions are suitable for different problems. For classification problems, accuracy is a common loss function which is a percentage of correct classifications. For regression problems like in this project, the mean squared error of the prediction is chosen as a loss function. See equation (2.7).

4.6.3. Training the ANN on data

After constructing the ANN, the model can be fit to the training data. Upon starting the training process, the training data must be provided, together with the number of training epochs.

In general, the longer the ANN is trained to the provided training data, the lower the loss quantity *can* become, until convergence is reached. But a lower loss quantity does not necessarily imply a better performance. It is very likely that the model performs very well at predicting the outcomes of the *training* data, but very poorly on the *testing* data.

Number of training epochs Choosing a fixed number of training epochs beforehand is tricky, since too low a number might result in underfitting, while too many epochs might result in overfitting. This problem can be solved by defining so called *callback* functions. A callback function as used in the Keras library is a function that is executed at the end of *each* training epoch with which the training process can be better controlled. Two callback functions will be used in this project.

EarlyStopping callback The EarlyStopping callback function is a function that stops the training process when the model accuracy stops improving or even starts to become worse.

Before reading on, a clear distinction must be made here between two terms.

- **Loss:** The value of the error, computed over the prediction of all *training* samples. The weights in the neural network will be updated, based on optimization of this loss value.
- **Validation loss:** The value of the error, computed over the prediction of all *testing* samples. This value is *only* used for checking the capability of the network to predict on unseen data. The testing samples are *not* used for training, i.e. updating the network weights and biases.

The EarlyStopping callback function is a function that can be added to the ANN fitting function in Keras to reduce the chance of overfitting. The callback function monitors the value of the *validation loss* at the end of each training epoch. When the validation loss does not decrease (i.e. improve) with a certain specified value, or even increases (meaning that the performance gets worse), the training process will be stopped.

The callback function requires two arguments:

- Δ_{min}: the minimum required change in validation loss.
- Patience: The number of training epochs to continue after registering an insufficient amount of improvement. Once a training epoch did not result in the specified minimum amount of improvement, the callback function is called. This patience setting makes the callback more 'forgiving' in case the validation loss reaches a short plateau. It can happen that the validation loss starts decreasing again after some epochs.

ModelCheckpoint callback The ModelCheckpoint callback is a function that saves the model of the ANN to an external file after each training epoch *provided* that the accuracy has improved. At the very end of a training process, it is guaranteed that the saved is the model with the lowest validation loss as found during the training process, even if the validation loss started increasing again at some point.

The drawback of including this callback function is that it demands more computational effort, drastically increasing the time needed for a training session. Therefore, this callback will only be included in the final refitting stage, after the hyperparameter optimization (Section 4.7) has already taken place.

Visualization of callbacks in training process See Figure 4.8 for an example of the history of a training process. The number of training epochs is fixed to 10.000 epochs and no EarlyStopping callback is provided here. At some point, approximately around training epoch 1250, the validation loss reaches its minimum

and does not improve anymore. The training process is continued until the specified number of epochs is reached.

Figure 4.9 shows an example where an EarlyStopping callback is provided. After the validation loss reaches its minimum around training epoch 200, it starts increasing again. At this point, the callback function is called with a patience value of 1000 epochs. Therefore, 1000 epochs later, the training process is terminated. No more improvement is observed after training epoch 200. After approximately epoch 400, the values of the validation loss starts oscillating, clearly indicating an unstable training process. The ANN model is saved at the point where the validation loss was lowest.

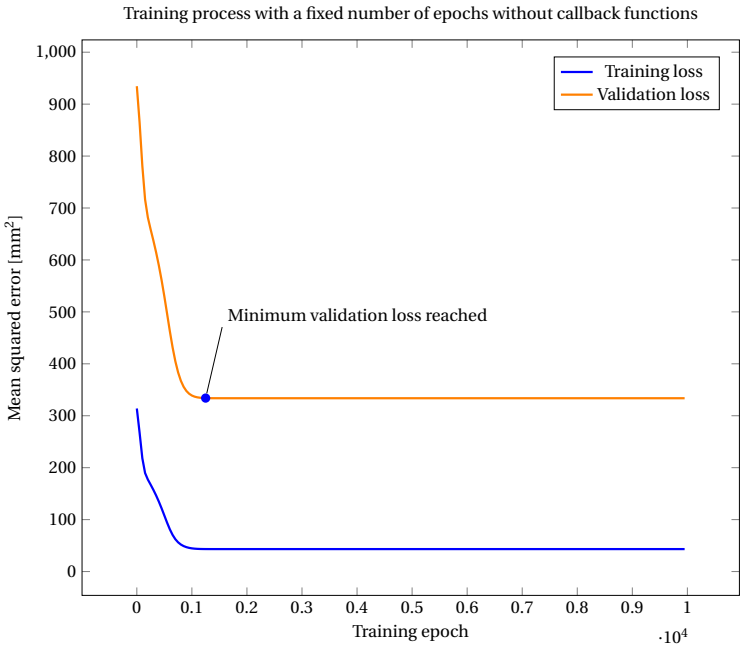


Figure 4.8: Training session with 10.000 training epochs without a defined earlystopping callback function. The validation loss reaches a plateau but the training process is continued until the defined 10.000 epochs are done. This is a waste of computational effort during a hyperparameter optimization algorithm.

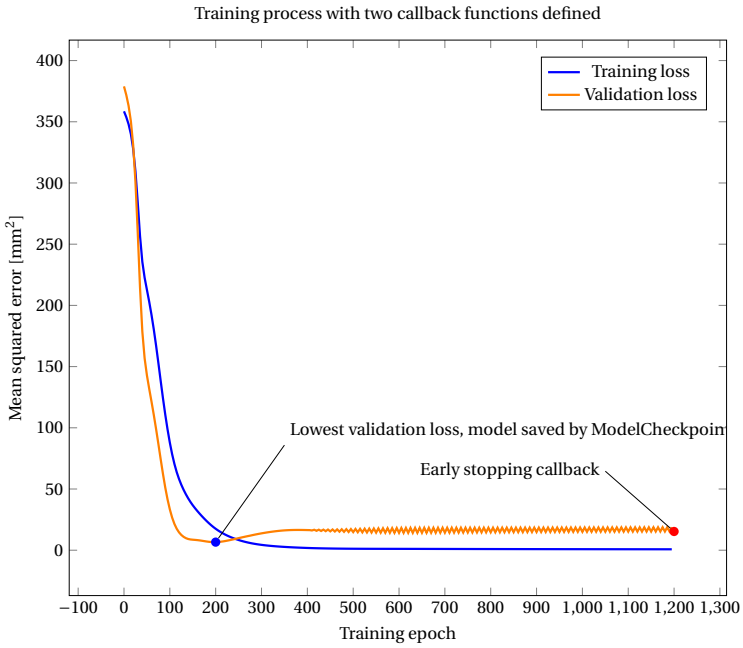


Figure 4.9: In this training process two callbacks are defined. The ModelCheckpoint callback saves the model at the training epoch with the lowest value of the validation loss. The EarlyStopping callback is invoked upon reaching this minimum value of the validation loss and waits for 1000 more epochs before terminating the training process.

4.6.4. Validation

There are two types of validation to be distinguished. The first one is *cross-validation*, which is used during the hyperparameter optimization process. The other one is validation on the testing dataset, which is used for final evaluation and validation of the chosen neural network with the hyperparameters found from the hyperparameter optimization process.

The mean cross validation score is a metric which tells a lot about the *generalization* capacities of the ANN, which is the capacity to predict values on data it has never been trained on. This mean cross validation score will be the scoring metric of the a neural network used in the genetic algorithm

K-fold cross validation K-fold cross validation is a method to evaluate the generalization capacity of the predictive model. The outcome of this validation is a metric for the capability of the model to predict accurately on unseen data, i.e. to give an accurate prediction for situations it has not encountered during the training sessions.

The method might be confusing. Figure 4.10 schematically shows the partitioning of the full dataset. The complete dataset has already been split into a training set and a testing set. This testing set is used for the *final* evaluation of the ANN. For the hyperparameter optimization process, the training dataset is used. But the training dataset is split again into K equally sized partitions. In this project, 5-fold cross validation is applied, meaning that the training dataset is split up in 5 parts.

In K-fold cross-validation, an ANN is trained and evaluated 5 times. Each time, one of the partitions of the training data is used as a *validation* dataset, and the other four parts make up the (sub)training dataset. The ANN is trained on these four training partitions and validated on the validation part left out. The outcome is a validation loss. In the end, each of the five partitions has served as a validation set *once*. After performing this 5 times, the mean value of all 5 separate validation losses is computed.

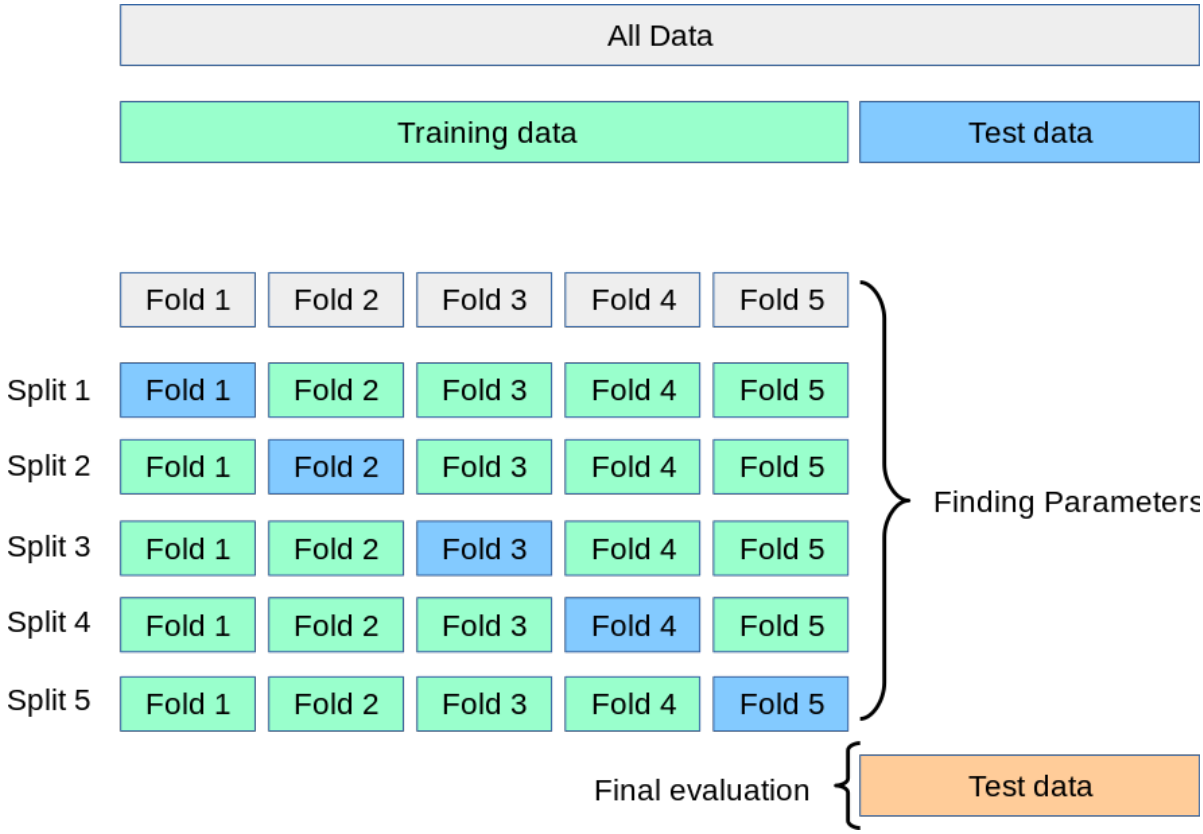


Figure 4.10: Vizualization of the splitting of data in train and test set, after which the test set is split again in 5 folds for cross validation [29]

4.7. Hyperparameter optimization using Genetic Algorithms

The goal of hyperparameter optimization is to find those settings (hyperparameters) for an Artificial Neural Network that produce a network with the best learning capacity for a specific dataset. In this thesis, the hyperparameters will be optimized using genetic algorithms, also called evolutionary algorithms. This essentially means that two machine learning methods are stacked onto eachother. The theoretical concepts are described Chapter 2.

The main idea is to iterate towards an optimal set of hyperparameters for a neural network. These iterations are so-called *generations*, just like a generation of individuals in biological evolution. The *fitness* of the individuals are supposed to improve in each successive generation. In the first generation, a number of random *sets of hyperparameters* is created. These unique sets of hyperparameters are called the individuals in the context of a genetic algorithm. With each set of hyperparameters, a neural network is created, trained and validated. The validation of the neural network returns a certain fitness value, which is assigned to the neural network. The fittest neural networks are selected to survive and pass on their genes to the next generation.

In this section, the deployment of the genetic algorithms for hyperparameter optimization will be discussed. The chosen settings for the genetic algorithms are listed down along with brief motivations. The settings of the genetic algorithms are based on small scale experiments which will not be documented in this thesis. The genetic algorithms are programmed using the DEAP library. The Python code created for the genetic algorithms for this project can be found in Appendix C.

4.7.1. Hyperparameters to optimize

Table 4.11 shows the list of hyperparameters that will be optimized. The available options are listed on the right side of the table. There are five hyperparameters to be optimized. Therefore, the genotype of an individual is represented as a Python list of five elements long. Each element contains the value of the respective hyperparameter. An example of such a genotype is shown in (4.1).

Hyperparameter	Options
Number of neurons	1 - 50
Activation function	Linear, Hyperbolic tangent, ReLu, Elu, Selu, Exponential, Sigmoid, Softplus
Optimizer	RMSprop, Stochastic gradient descent, ADAM, ADAGRAD, ADADELTA, ADAMAX, NADAM
Kernel initializer	Random uniform, Constant, Zeros
Bias initializer	Random uniform, Constant, Zeros

Table 4.11: An overview of the hyperparameters to be optimized and the possible options

$$genotype = [24, \text{sigmoid}, \text{adam}, \text{constant}, \text{random\_uniform}] \tag{4.1}$$

Note about learning rate It was already mentioned in Section 4.6.2 that the learning rate is not included in this list. The reason for that is to avoid issues with defining a genotype. Most of the available optimizers have an *adaptive learning rate*, that automatically changes during the training process. Only a few optimizers have a fixed learning rate which is required to be defined. This would mean that genotypes with different amount of genes would have to be created. An individual with five genes in case of an optimizer with an adaptive learning rate, and six genes in case of an optimizer with a fixed learning rate. This can lead to problems when applying crossover operations between two individuals with different genotype lengths.

An alternative option would be to fix the genotype length to six genes for all individuals, thus including a value for the learning rate for all ANN's. The problem is that the provided learning rate will only have an influence on the individuals with fixed learning rate optimizers and no influence at all on the fitness of the individuals with adaptive learning rate optimizers. It is expected that this will negatively affect the convergence of the genetic algorithm.

Therefore, it is decided to not include the learning rate as a hyperparameter to optimize. Instead, for the optimizers with a fixed learning rate, the default value is maintained.

4.7.2. Settings for genetic algorithm

Before running the genetic algorithm, certain settings have to be chosen. The concepts related to these settings are discussed in the theory section in Chapter 2.

Population size The population size must be chosen large enough, such that there exists a sufficient amount of variety in the gene pool. Too small a population can lead to too fast convergence to a local optimum. At the same time, the larger the population size, the more individuals will have to be evaluated in each generation. If the evaluation of the fitness of an individual takes only a short amount of time, this does not need to be a limitation. But, in the context of this project, the evaluation of an individual means 5-fold cross validation of an ANN (see Section 4.6.4). Since this requires a significant amount of time, the population size needs to be limited in order to limit computation time.

In this project, the population size is set to 40 individuals for each generation.

Number of generations The number of generations needs to be high enough to ensure convergence. But setting a high number of generations also implies evaluating many individuals, therefore increasing computational time.

In this project, the number of generations is limited to 5 generations.

Crossover rate and mutation rate The settings for the crossover rate and mutation rate strongly influence the behaviour of the algorithm. In this project, a comparison has been made between two sets of settings.

- Experiment 1: crossover rate = 0.8, mutation rate = 0.2
- Experiment 2: crossover rate = 0.1, mutation rate = 0.9

The results showed that the settings for the rates in experiment 2 yielded better performing neural networks. Experiment 1 showed a more converged solution, although to a local minimum. The high mutation rate in experiment 2 ensured a more thorough exploration for possibly better solutions, showing a less converged solution, yet the best found individual performed better in this experiment.

In this project the crossover rate will be set to 0.1 and the mutation rate is set to 0.9. This high mutation rate may mean that the algorithm does not really converge to one solution, but it does result in a more thorough exploration of alternative neural networks. It is considered more important to find a best solution from many varied individuals, instead of the algorithm converging to one single solution only. The best hyperparameters for the ANN are selected based on the very best individual from all generations.

Tournament size The size of each group of selected competitors is called the tournament size. The tournament size should have a reasonable proportion to the population size. Simply setting the tournament size equal to the population size would imply that in each tournament, the single best individual of the entire generation would be selected, in which the offspring would contain copies of one single individual only. This will lead to an insufficient variety in the gene pool of the offspring. On the other hand, too low a tournament size would allow weak individuals to win a tournament and pass on their genes to the next generation.

The tournament size in this project is set equal to 10, which is 25% of the population size.

Elite portion In order to avoid the chance of the best individuals not being chosen for competition, elitism is applied in this project.

The elite portion is set equal to a value of 0.1, meaning that the top 10% of a generation is *always* selected to pass to the next generation.

4.7.3. Custom functions

Fitness function For evaluation of the fitness of an individual, a custom evaluation function is defined in the Pyhton script. This function takes the genes of the individual as input, and returns the fitness value of the individual.

In the context of this project, the genes are the hyperparameters of the ANN as demonstrated in Section 4.7.1. The fitness value of the individual is equal to the *mean cross validation loss*. The mean cross validation loss is computed by applying K-fold cross validation which is treared in Section 4.6.4.

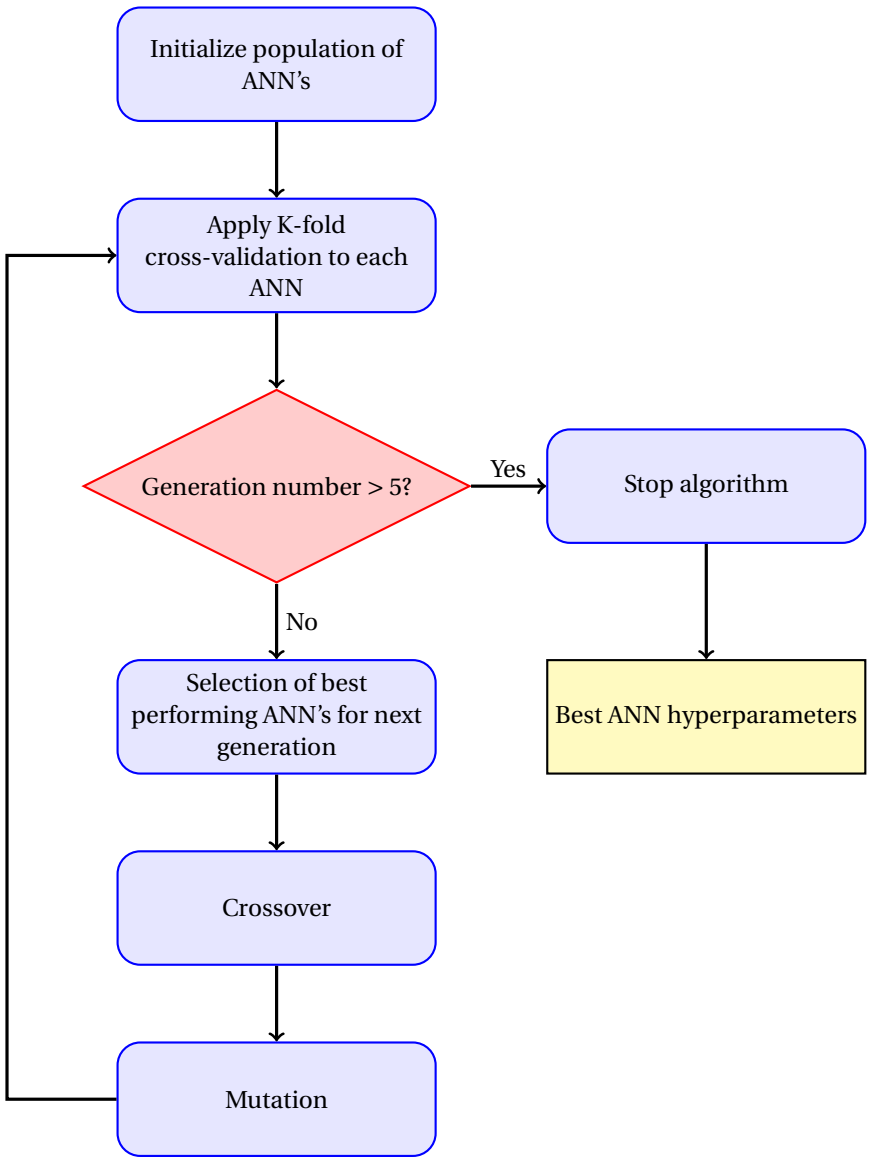


Figure 4.11: Flowchart vizualizing the hyperparameter optimization for neural networks using a genetic algorithm and K-fold cross validation

4.8. Other methods for predictive modeling

The accuracy of the predictions performed by the Artificial Neural Networks are compared with the accuracy of predictions performed by three interpolation techniques. Two of these interpolation techniques are available options in ANSYS for constructing a Response Surface. The first one is Kriging interpolation, which is a commonly used method in geostatistics. The second one is quadratic interpolation, which is the standard option for constructing the Response Surface in ANSYS. Next to quadratic interpolation, also cubic interpolation is performed, which is basically the same technique as quadratic interpolation, although up to a higher degree polynomial. Cubic interpolation is not an available interpolation technique in ANSYS but it is included in this study to make a more thorough comparison. Each of the methods is briefly discussed in the following sections.

4.8.1. Kriging interpolation

Kriging interpolation, also called *Gaussian Process Regression* is a method of interpolation which finds its origin in geostatistics. Figure 4.12 shows an example of Kriging interpolation. The top surface represents the actual prediction over the full domain of the problem. Not only is the method capable of interpolation, the

accuracy of prediction can also be computed. This can be seen in the bottom surface of Figure 4.12 where the white spots correspond to the actual data points. The red regions indicate a higher probability of prediction error. The interpolation technique fits exactly through the data points. Therefore, the error in these data points is always equal to zero.

The exact theoretical background is not discussed here. A brief introduction of the theory on Kriging interpolation can be found on [17] A more in-depth explanation on the theory can be found in the article [28].

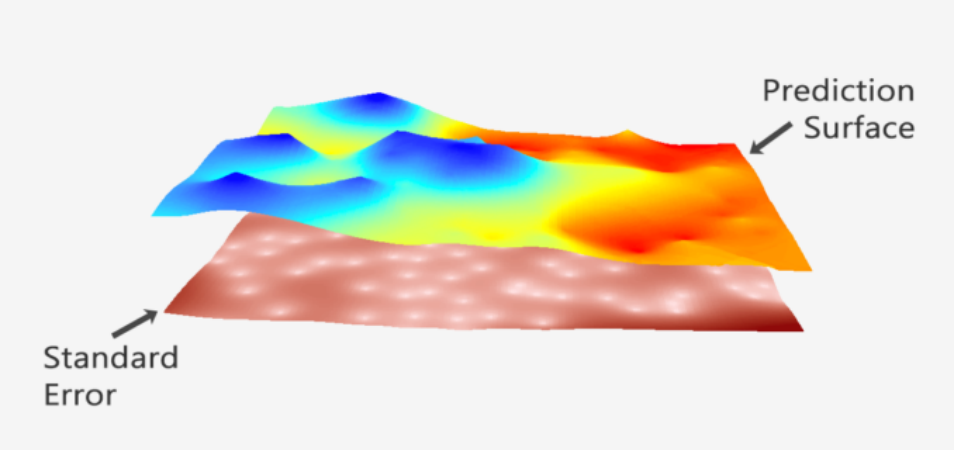


Figure 4.12: 2D vizualization of a prediction surface created with Kriging interpolation. The bottom plot shows the error estimate of the interpolation. [4]

While the applications in geostatistics generally involve predictions in a 2D space, the method is also applicable in more dimensions. In this project, the method is implemented using an open-source Python package, specifically designed for this task. It is included in the powerful Sci-kit Learn library, containing many functions and packages used in machine learning and data science. The class is called *GaussianProcessRegressor* (GPR) and the documentation can be found on [27].

In this project, the GPR is fitted to the exact same data as used for training the Artificial Neural Networks in order to make an exact comparison. The settings for the GPR are left at their default state.

4.8.2. Polynomial interpolation

Polynomial interpolation is a well-known method for fitting a curve through data points. In this project, two Python classes from the Sci-Kit Learn library are used in combination to create polynomial interpolations. These classes can be used for both quadratic and cubic interpolation.

First, a set of polynomial features is constructed which contains all possible polynomial combinations of the variables up to the degree as specified. Then, linear regression is performed in order to determine the values of the coefficients such that the Mean Squared Error is minimized.

Example
Consider a 2-dimensional problem with the two variables x_1 and x_2. A polynomial to the degree 2 is to be constructed for this problem. First, a set polynomial features up to the degree 2 is created. This is done using the Python class *PolynomialFeatures* [6]. This will yield a set of 6 terms as shown in (4.2).

$$f(x_1, x_2) = a_1 \cdot x_1^2 + a_2 \cdot x_1 + a_3 \cdot x_1 x_2 + a_4 \cdot x_2^2 + a_5 \cdot x_2 + a_6 \tag{4.2}$$

The unknown terms a_i are then solved by means of linear regression. The method is based on ordinary least squares regression, aiming at minimizing the Mean Square Error.

5

Results and discussion

5.1. Computational efforts

This section will give a very brief overview of the computational efforts that were required to generate the data and to train and optimize the neural networks.

The total amount of time required for creation of a single predictive model in the form of a neural network, following the methodology as presented in Chapter 4, is composed of the following elements.

1. Creation of the parametric model:
   Setting up an FE model parametrically takes slightly more time and attention than setting up a normal FE model. This is mostly due to the rule based procedures and logic that requires time to implement.

2. Generation of training data:
   The amount of time required for the generation of training data is highly dependent on the nature of the analysis. Within the created framework, a single *linear* FE analysis took approximately 1 second when running in batch mode. Running 160 linear analyses only took 4 minutes.

   Geometrical nonlinear analyses took between 1 to 5 minutes in general, depending on the amount of elements and the convergence behaviour of the analysis. Running 320 nonlinear FE analyses took a little more than 9 hours in total, resulting in an average of 1.7 minutes per analysis.

   Keep in mind that the sole purpose of the *linear* analyses was to experiment with neural networks on a simplified problem. The benefits of creating neural networks are less applicable to linear analyses, since these can be performed very quickly. The time reduction of producing output using a neural network compared to an actual FE analysis is less pronounced than in case of nonlinear analyses.

3. Hyperparameter optimization:
   The hyperparameter optimization process involves 5-fold cross-validation of approximately 200 neural networks in the genetic algorithm as described in Section 4.7. This is equivalent of approximately 1000 training sessions of a neural network. This process took approximately 3 hours for finding an optimal neural network architecture for a *single* dataset. For datasets with 256 training samples corresponding to projects 3-1-2-3 and 3-1-2-4, containing data of maximum stresses in the plate and stiffeners respectively, this optimization process took a little more than 5 hours. When introducing *parallel processing*, as described in Section 5.5, the computational time needed for the genetic algorithm for these datasets were reduced to only 1 hour and 35 minutes.

4. Training of the final neural network:
   The training time of a single neural network itself was found to be relatively short, with training times ranging from 10 to 60 seconds.

The time required for fitting of the standard interpolation techniques to the data is negligible. This step took no more than 1 second.

5.2. Comparison of accuracies predictive models

In this section, an overview will be given where the accuracies of the neural networks are compared to the accuracies of the other interpolation techniques. The datasets are split up for each mechanical model.

5.2.1. Comparison accuracies mechanical model 1

This section shows the resulting accuracies for model 1. Figure 5.2 shows two plots comparing the prediction accuracy of all predictive models for a series of datasets. Both plots have the series of datasets (15 sets in total) on the horizontal axis. These datasets all belong to the simplified mechanical model number 1 and are all predictions on the *test* datasets. So the plotted accuracies/errors are those related to data that the predictive models were *not* trained on.

Both plots in Figure 5.2 show that up to and including dataset number 7, the Artificial Neural Networks outperform the other predictors. The top plot shows that the RMSE suddenly drops for all predictors after dataset number 8. This can be explained by the fact that extra input parameters are added at this point: The dimensions of the flanges of the longitudinal T-stiffeners are included in these models. This introduces higher bending stiffnesses in the structure for all subsequent datasets, reducing the average maximum deflection and also reducing the possible absolute errors.

Therefore, the bottom plot is included plotting the value of the coefficient of determination R^2 on the vertical axis. This accuracy measure is scale independent and therefore is more suitable to compare the accuracies. The closer the value to 1, the better the accuracy. Values below 0.5 are not shown in the plot for better visualization.

In this bottom plot it is clear that the value of R^2 is higher for the neural networks for all datasets except for set number 8. This number corresponds to the dataset for mechanical model 1 with 6 free input parameters and 128 training data points. It is interesting to note that at exactly this point with the highest number of training data (128 training points), the neural network performs worse than the other predictors.

Dataset number 8 Scatter plots belonging to predictions of this particular dataset are shown in Figure 5.3. It is clearly visible that both the predictions belonging to training data *and* testing data deviate a lot from the true value for the ANN. Yet this is the only dataset for mechanical model 1 where the neural network performs worse than the others. A value of $R^2 = 0.88$ as the lowest value is very promising.

Scatter plots dataset 14 Dataset number 14 belongs to the model with the highest amount of input parameters and 128 training datapoints. See Figure 5.4 for the scatterplots belonging to the predictions on this dataset. Here it is clearly visible that the neural network makes the best predictions. It does not perform best on the training data, represented by the blue dots. This is because Kriging *exactly* interpolates through the training dots. The neural network *does* outperform the other predictors on the test data, which is is the most important feature of a predictive model.

Concluding remark about predictions on mechanical model 1 The plots in Figure 5.2 shows that the neural networks perform better on all but one dataset. Furthermore, the accuracies of the neural network look more stable among the different datasets. The other interpolation techniques often have values of R^2 below 0.5, whereas for the neural networks, this value is above 0.88 for all datasets.

Dataset	Project Num.	Num. variables	Num. training samples	Output parameter
0	1-1-1-1	4	32	$u_{x,max}$
1	1-1-1-1	4	64	$u_{x,max}$
2	1-1-1-1	4	128	$u_{x,max}$
3	1-2-1-1	5	32	$u_{x,max}$
4	1-2-1-1	5	64	$u_{x,max}$
5	1-2-1-1	5	128	$u_{x,max}$
6	1-3-1-1	6	32	$u_{x,max}$
7	1-3-1-1	6	64	$u_{x,max}$
8	1-3-1-1	6	128	$u_{x,max}$
9	1-4-1-1	7	32	$u_{x,max}$
10	1-4-1-1	7	64	$u_{x,max}$
11	1-4-1-1	7	128	$u_{x,max}$
12	1-5-1-1	8	32	$u_{x,max}$
13	1-5-1-1	8	64	$u_{x,max}$
14	1-5-1-1	8	128	$u_{x,max}$

Figure 5.1: Overview of datasets and their contents belonging to the analyses of mechanical model 1

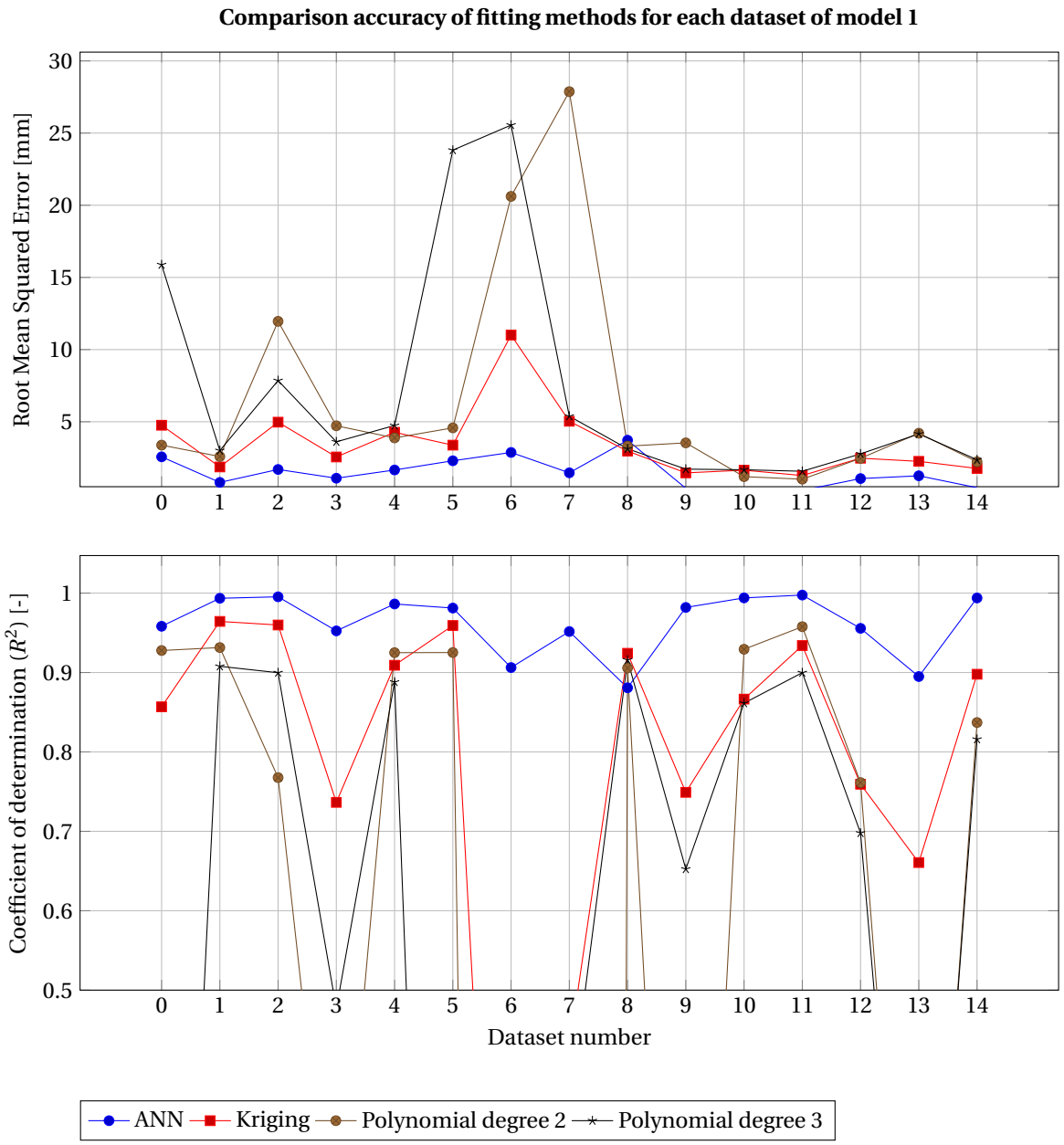


Figure 5.2: Comparison of accuracies of prediction on test data for all datasets of model 1. Model 1 is the simplified mechanical model with maximum plate deflection as output parameter. Each tick on the horizontal axis represents a single dataset belonging to that model. See Table 5.1. On the vertical axes are shown two different error metrics.

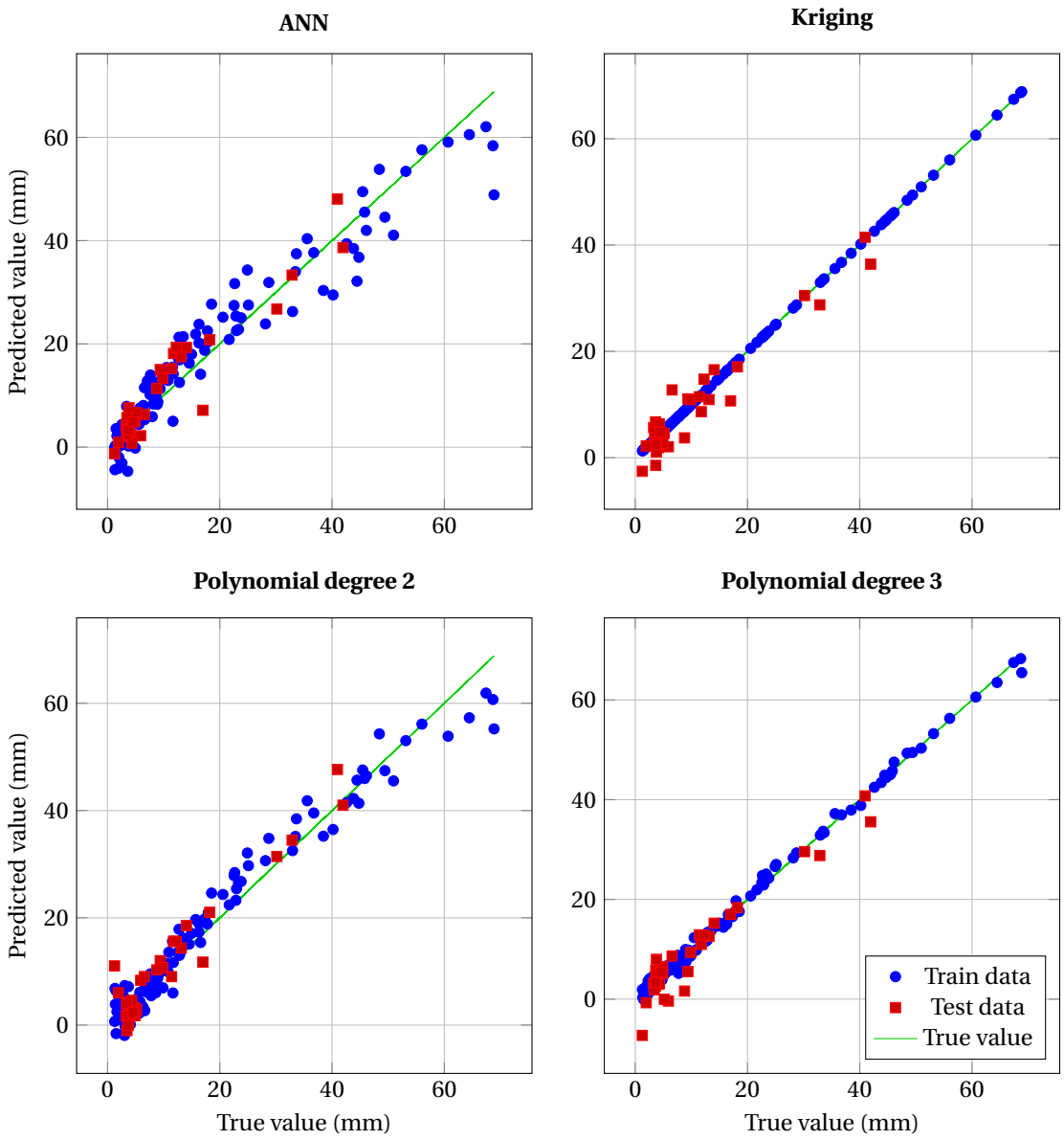


Figure 5.3: Comparison of scatterplots for individual predictors on dataset number 8 of model 1. The Artificial Neural Network performs worse than the other predictive models.

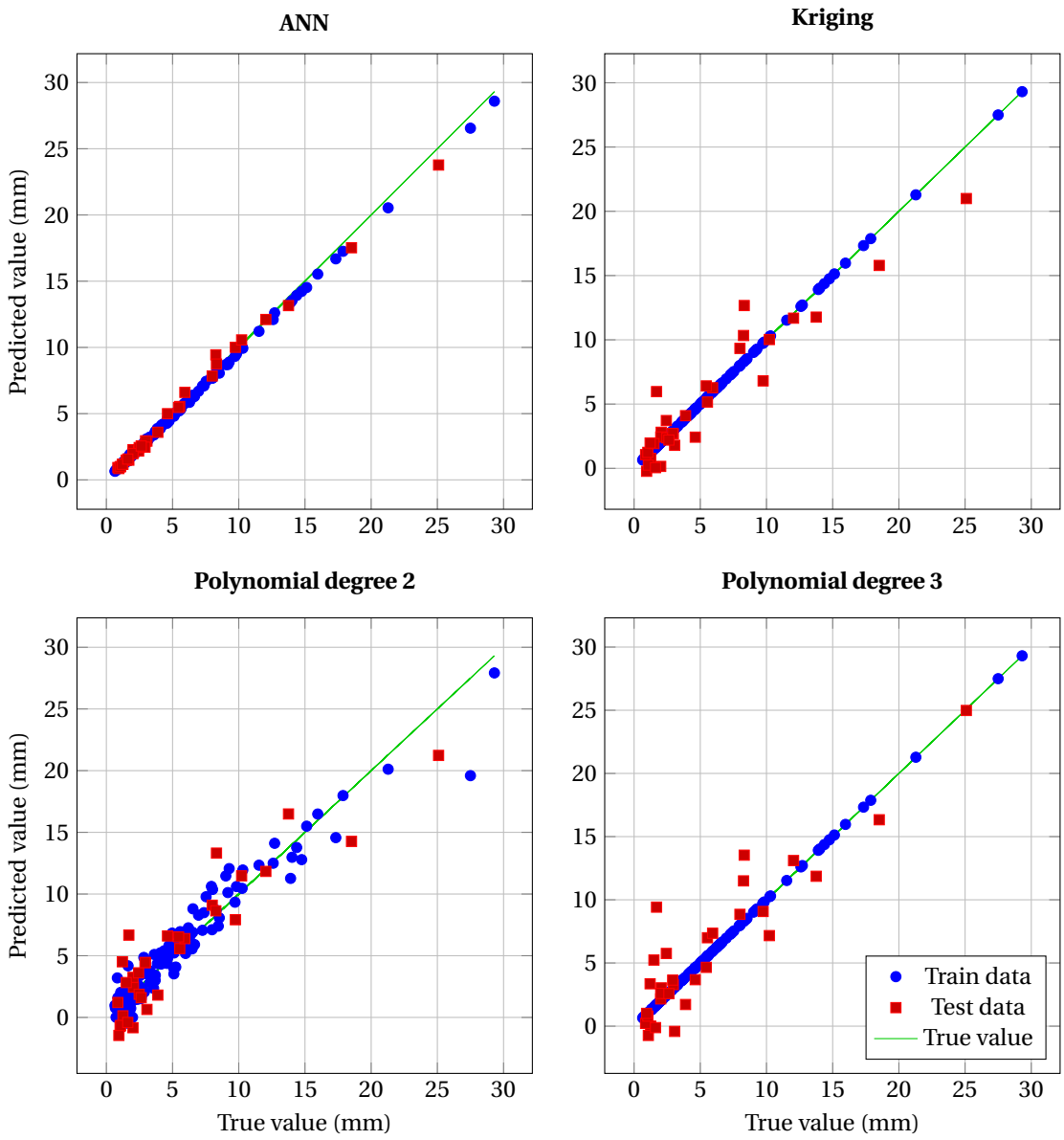


Figure 5.4: Comparison of scatterplots for individual predictors on dataset number 14 of model 1. The Artificial Neural Network performs best on the training data. In the right tail of the spectrum the predictions are slightly underestimated.

5.2.2. Comparison accuracies mechanical model 2

In Figure 5.6 the accuracies of the predictions on test data for mechanical model 2 are compared. The difference in performances in this figure looks less pronounced than for the mechanical model 1 in Figure 5.2. Up to dataset number 6, the neural networks perform better than the other interpolation techniques. But from that point on, the neural networks are either more accurate, or approximately equally accurate as the other interpolation techniques. The only dataset where the neural network performs significantly is dataset number 6, where Kriging interpolation works better. At dataset number 12, all predictive models have a very low coefficient of determination. This dataset corresponds to model 2 with the highest number of input parameters (8 variables) and only 32 training data samples. This is also the first model where the vertical stress σ_{zz} is introduced.

Overall it can be seen that also in these plots, the accuracies of the neural networks are more stable among the different datasets than for the other interpolation techniques. With the extremity of dataset number 12 excluded, all predictions with neural networks have a coefficient of determination of at least 0.7.

Dataset number 12 As already pointed out, all predictive models performed badly on dataset number 12. When looking at the scatterplots in Figure 5.7, we can clearly see that something very strange happens. The predictions of the neural network on the test data are roughly as close to the true values as do the other interpolations. But the predictions on the training data show a significant difference. These predictions do not correlate at all with the true values.

When looking at the convergence plot of the training process of the neural network in Figure 5.8 we can see that the training loss is bigger than the validation loss during the entire training process, while this usually would be the other way around. This plot is in accordance with the observations in the corresponding scatterplot.

Dataset number 14 Dataset number 14 belongs to the model with the highest amount of input parameters and 128 training datapoints. See Figure 5.9 for the scatterplots belonging to the predictions on this dataset. According to the overview plot in Figure 5.6 the differences in accuracies on this dataset are relatively small. This is also visible in the scatterplots. Kriging interpolation and the 3^{rd} degree polynomial interpolation again perform best on the training data, yet the performance on the testing data is slightly better for the neural networks.

Dataset	Project Num.	Num. variables	Num. training samples	Output parameter
0	2-1-1-1	4	32	λ_1
1	2-1-1-1	4	64	λ_1
2	2-1-1-1	4	128	λ_1
3	2-2-1-1	5	32	λ_1
4	2-2-1-1	5	64	λ_1
5	2-2-1-1	5	128	λ_1
6	2-3-1-1	6	32	λ_1
7	2-3-1-1	6	64	λ_1
8	2-3-1-1	6	128	λ_1
9	2-4-1-1	7	32	λ_1
10	2-4-1-1	7	64	λ_1
11	2-4-1-1	7	128	λ_1
12	2-5-1-1	8	32	λ_1
13	2-5-1-1	8	64	λ_1
14	2-5-1-1	8	128	λ_1

Figure 5.5: Overview of datasets and their contents belonging to the analyses of mechanical model 2

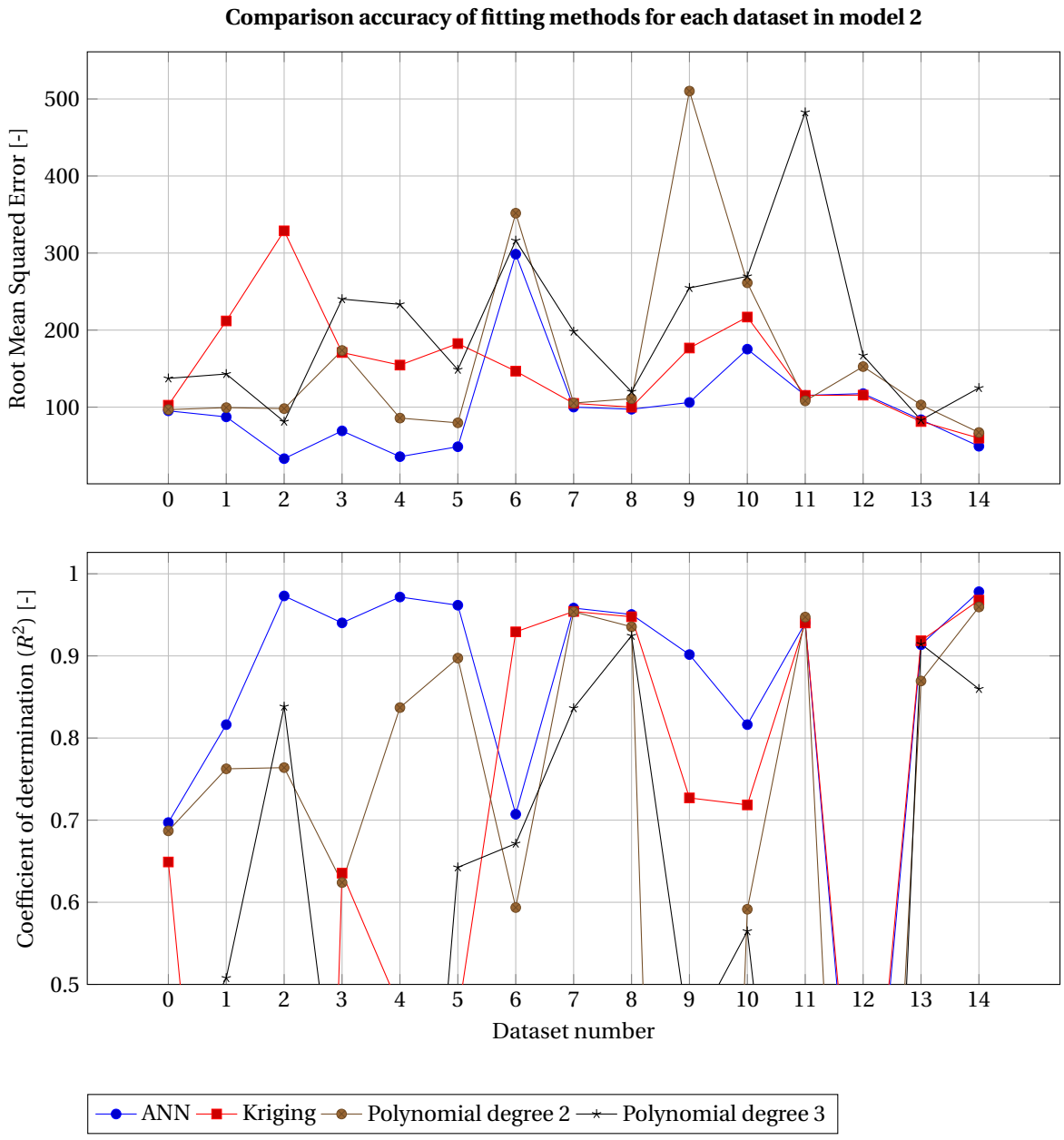


Figure 5.6: Comparison of accuracies of prediction on test data for all datasets of model 1. Model 1 is the simplified mechanical model with maximum plate deflection as output parameter. Each tick on the horizontal axis represents a single dataset belonging to that model. See Table 5.5. On the vertical axes are shown two different error metrics.

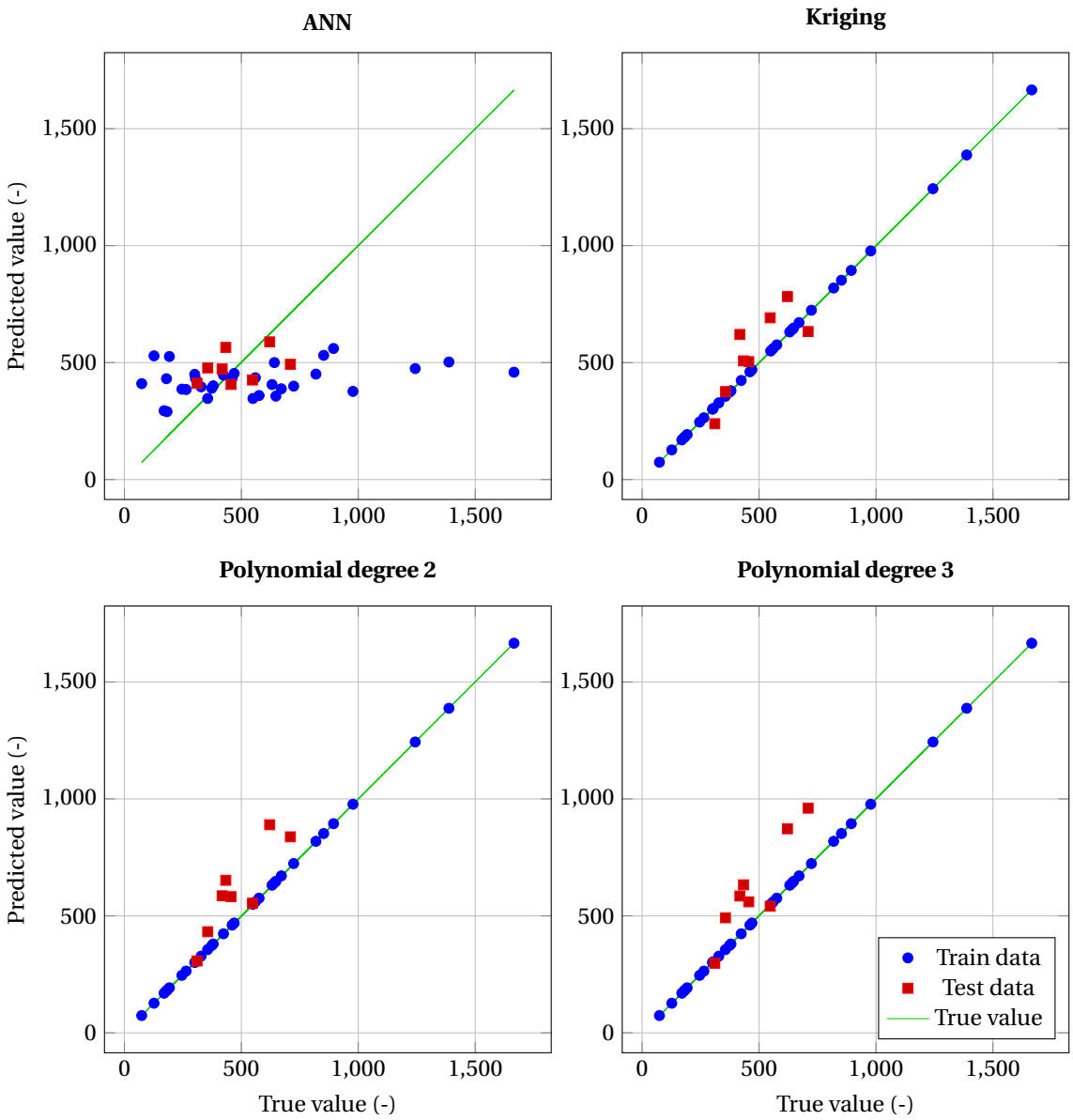


Figure 5.7: Comparison of scatterplots for individual predictors on dataset number 12 of model 2. The coefficient of determination for the *training* data $R^2 = -0,035484122$ which is remarkably bad.

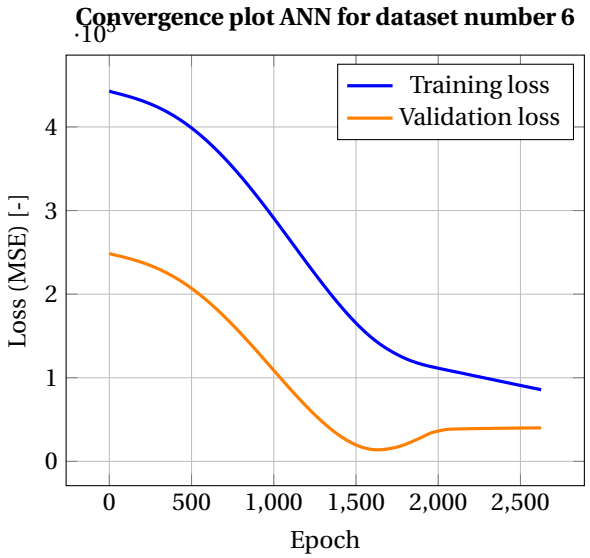


Figure 5.8: Convergence plot of the training process on dataset number 12. Although it looks stable, we see that the training loss is bigger than the validation loss during the entire training process. Also in the end the losses are increasing.

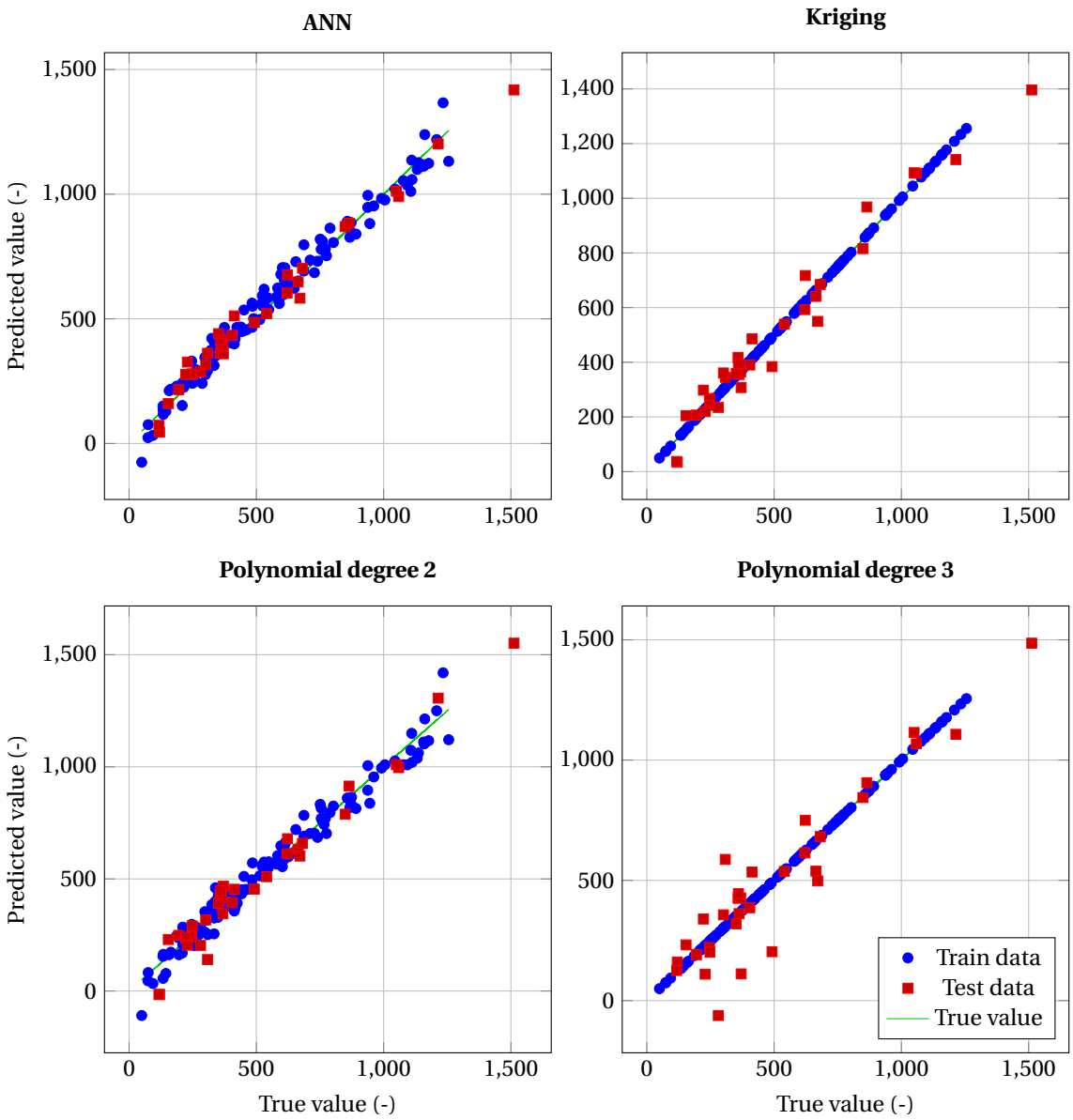


Figure 5.9: Comparison of scatterplots for individual predictors on dataset number 14 of model 2. Accuracy of ANN is approximately equal to Kriging and 2^{nd} degree interpolation

5.2.3. Comparison accuracies mechanical model 3

In Figure 5.11 the accuracies of the predictions on test data for mechanical model 3 are compared. Here, no plot is included comparing the RMSE, since the predicted output values are different among different datasets See Table 5.10 for reference of the dataset numbers. All output values are results of nonlinear FE analyses.

The predictions appear to be less accurate when comparing to mechanical models 1 and 2. This could be due to the fact that the data are based on nonlinear analyses. But it must also be noted that all data for model 3 are dependent on 13 design variables, where the amount of variables for models 1 and 2 was not higher than 8. The relation between prediction accuracy and the number of design variables is discussed further in Section 5.4.

We can see the same phenomenon occurs for predictions on model 3, that the accuracies of the neural networks are far more stable than the other interpolation techniques. The coefficient of determination never drops below 0.58, while the others regularly drop below 0.5.

Predictions of maximum stresses Datasets 0-2 and 6-11 belong to predictions made on maximum stresses in the FE model. Sets 0, 1, 2 contain the output stresses of the complete FE model, whereas the other datasets only contain stresses in either the skin plate or longitudinal stiffeners, both excluding the region close to the columns. The neural networks show more accurate predictions on all of these stress datasets, apart from dataset number 1. On this dataset, Kriging interpolation performs slightly better.

The predictions on maximum stresses in the skin plate, with the regions close to the column excluded, are represented by datasets number 6,7 and 8. We can see that these are by far the most accurate among all stress predictions.

The predictions on the maximum stresses in the stiffeners in datasets 9, 10 and 11 are also slightly more accurate than the predictions on the maximum stresses in the complete mechanical model. This indicates that it can be beneficial to focus on restricted regions to read out stresses, in order to obtain a higher accuracy.

Predictions of maximum deflections It was expected that the predictions on maximum deflection would be more accurate than the predictions of maximum stresses, due to the irregular nature of the location and occurrence of stress peaks in the FE model. However, this is not the case. Both dataset 3 and 5 show very bad results. Dataset 4, trained with 128 data samples, performs best. It is remarkable that these predictions are more accurate than on dataset 5, while the results from dataset 4 are from a neural network that was trained with only half the amount of data.

Figures 5.12 and 5.13 show the scatterplots belonging to predictions on dataset 4 and 5 respectively. In the first figure, it is clearly visible that the neural network predicts the most accurate on the test data. In Figure 5.13, we can observe a very strange behaviour in the predictions. The predicted value seems bounded by an upper value of 20 mm deflection, and a strange curve can be observed below it. Let us take a closer look on the convergence plot of the training process in Figure 5.14. Here we can see a very unstable training process. with sharp alternating values in both training and validation loss. Such a result is not acceptable. This can be an indication that the genetic algorithms that are deployed to optimize the hyperparameters of the neural networks do not always converge to the most stable neural networks.

Dataset	Project Num.	Num. variables	Num. training samples	Output parameter
0	3-1-2-1	13	64	$\sigma_{max,all}$
1	3-1-2-1	13	128	$\sigma_{max,all}$
2	3-1-2-1	13	256	$\sigma_{max,all}$
3	3-1-2-2	13	64	$u_{x,max}$
4	3-1-2-2	13	128	$u_{x,max}$
5	3-1-2-2	13	256	$u_{x,max}$
6	3-1-2-3	13	64	$\sigma_{max,plate,mid}$
7	3-1-2-3	13	128	$\sigma_{max,plate,mid}$
8	3-1-2-3	13	256	$\sigma_{max,plate,mid}$
9	3-1-2-4	13	64	$\sigma_{max,stiff.,mid}$
10	3-1-2-4	13	128	$\sigma_{max,stiff.,mid}$
11	3-1-2-4	13	256	$\sigma_{max,stiff.,mid}$
12	3-1-2-5	13	64	λ_1
13	3-1-2-5	13	128	λ_1
14	3-1-2-5	13	256	λ_1

Figure 5.10: Overview of datasets and their contents belonging to the analyses of mechanical model 3

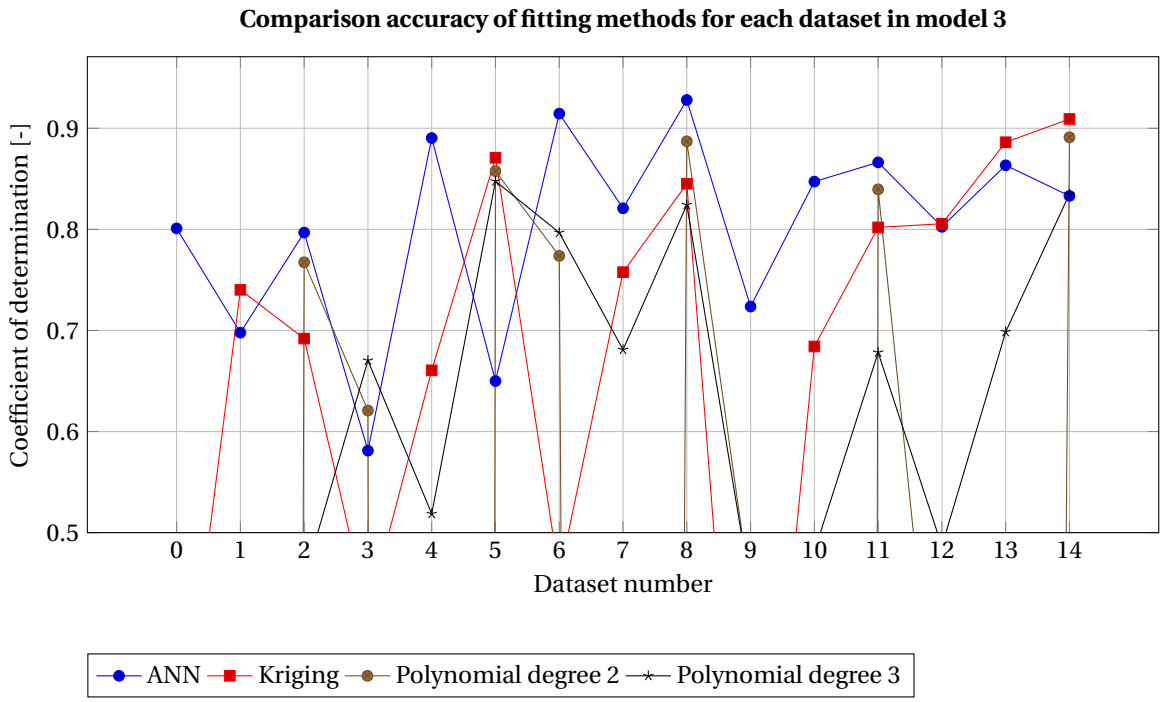


Figure 5.11: Comparison of accuracies of prediction on test data for all datasets of model 3. Each tick on the horizontal axis represents a single dataset belonging to that model. See Table 5.10. Only the coefficient of determination is shown since the units are different among the datasets.

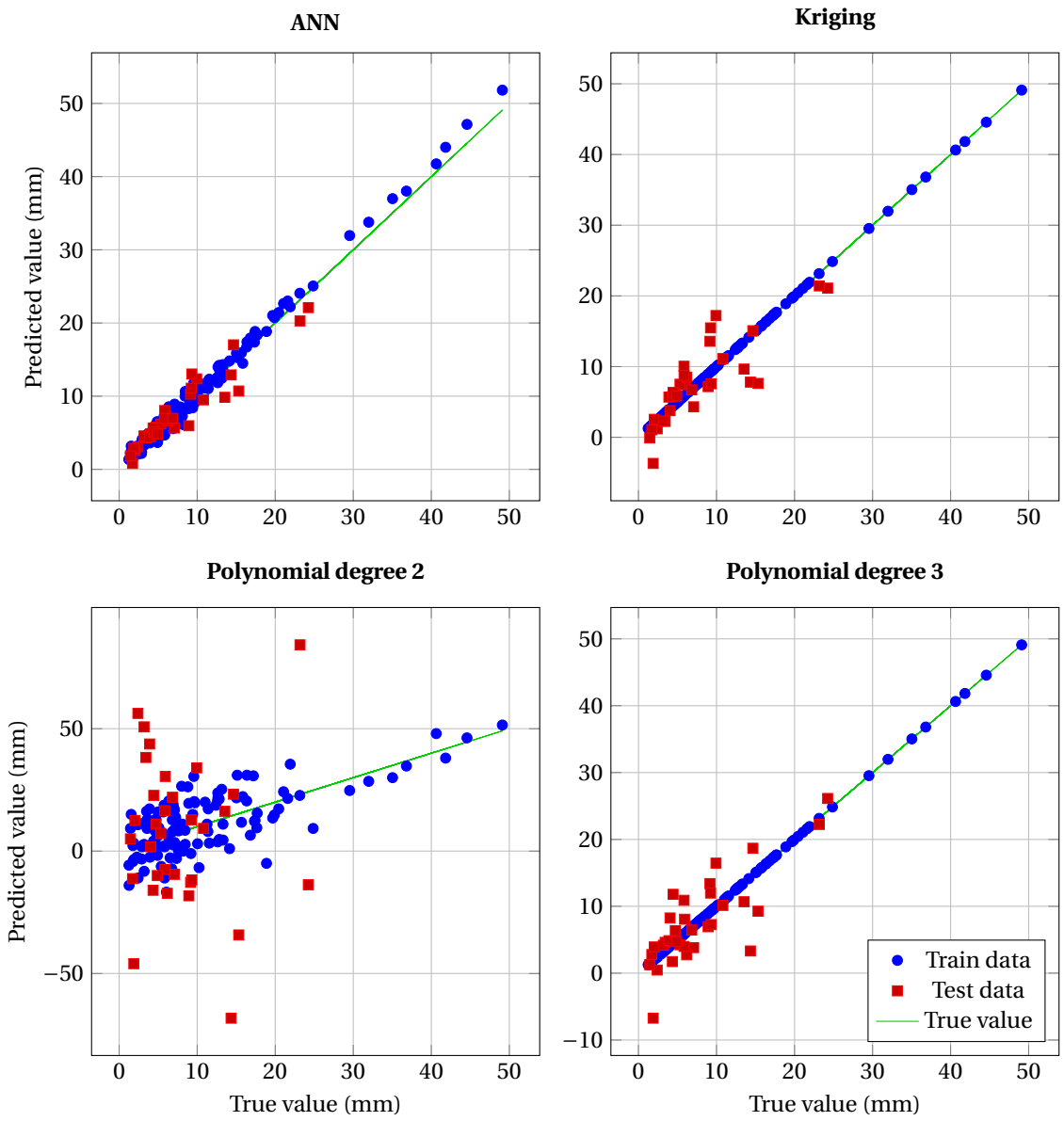


Figure 5.12: Comparison of scatterplots for individual predictors on dataset number 4 of model 3. Outputparameter is maximum deflection $u_{x,max}$. Although trained on less training samples than dataset number 5 in Figure 5.13, its performance is a lot better.

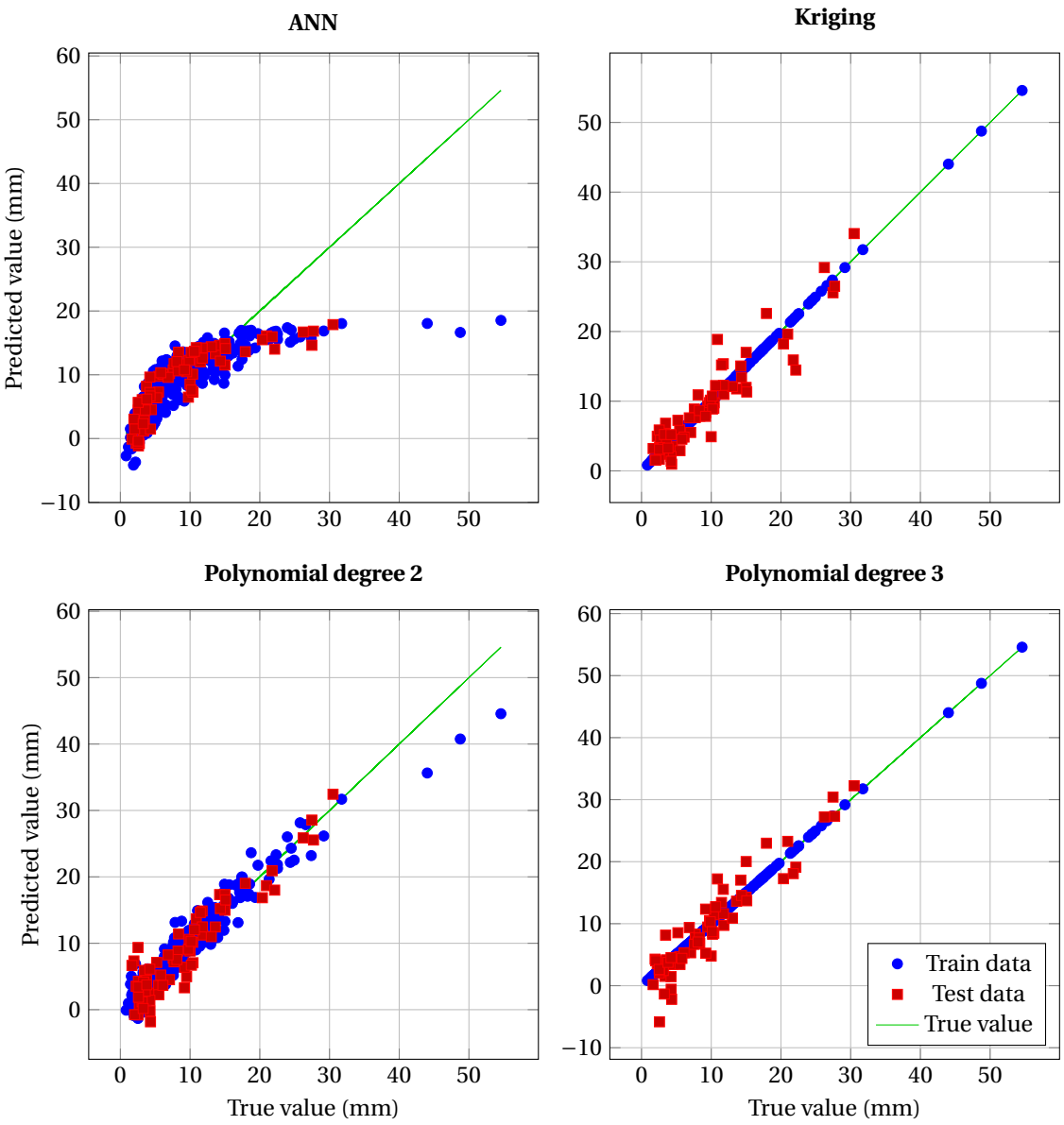


Figure 5.13: Comparison of scatterplots for individual predictors on dataset number 5 of model 3. Neural network performs worse while trained on more training samples than dataset 4. A very strange pattern can be observed in the scatter data.

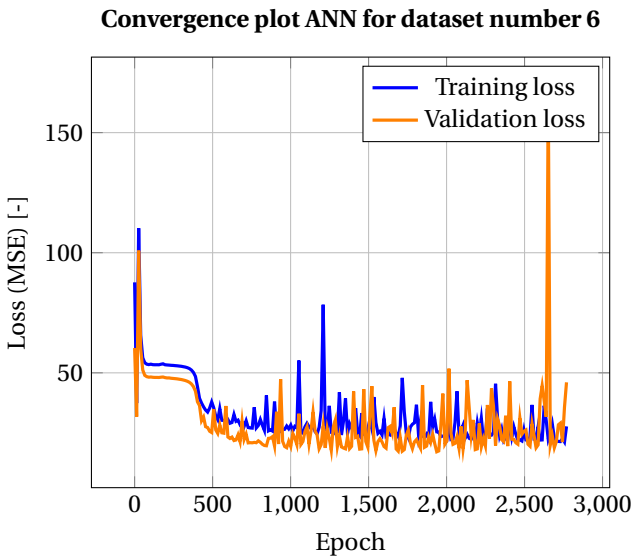


Figure 5.14: Convergence plot of the training process on dataset number 5. The losses show a very unstable pattern.

5.3. Analyses of errors predictions Artificial Neural Networks

This section presents a number of histograms providing insight in the distribution of errors of predictions of the best found neural networks on test data. Each figure shows two plots. The left plot shows the frequency distribution of absolute errors of the predictions on test data. The right plots shows the relative errors in percentage.

Section 5.3.1 presents some error plots of the simplified mechanical models 1 and 2. In Section 5.3.2, the error plots corresponding to the main mechanical model 3 are presented.

Since there are 45 datasets and therefore 45 predictions on test data, only a handful of error distributions will be shown.

5.3.1. Error analysis simplified mechanical models 1 and 2

Simplified model 1 Figure 5.15 shows the errors corresponding to the predictions on test data of the simplest model, namely model number 1 with only 4 design variables. The output parameter is the maximum deflection of the stiffened plate $u_{x,max}$. The absolute errors almost exclusively fall between the range of ± 1.5 mm, apart from one prediction which has an error of -9 mm. The right plot shows that the majority of the predictions have a relative error in the range ± 5 %, which seems very reasonable. Four predictions underestimate the value with more than 5% error. This does not necessarily mean a bad prediction. The minimum value of $u_{x,max}$ in the test data is equal to 1.4 mm. A relative error of -17 % in this case would mean a predicted value of 1.69 mm.

Figure 5.16 shows the error of predictions on test data of project 1-5-1-1, which has the highest number of variables for this simplified model, namely 8 free input parameters. The absolute errors show a nice normal distribution with the mean value around 0 mm error. The absolute errors mostly fall within the range of ± 0.5 mm. This looks like lower errors compared to project 1-1-1-1, but it can be explained by the fact that in this model, flanges are included to the longitudinal stiffeners, increasing the stiffness of the structure. The mean value of the deflections is only 5.34 mm, compared to 13.24 mm. The right plot shows higher relative errors. The minimum value of the true output is only 0.65 mm as compared to 1.4 mm in project 1-1-1-1, which can explain the greater relative errors.

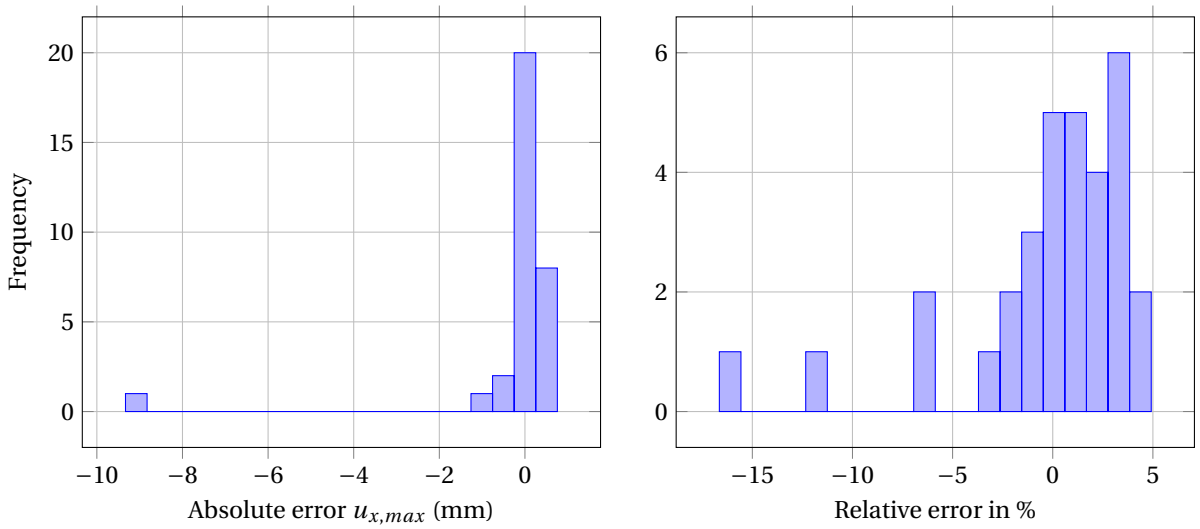


Figure 5.15: Error distribution histograms on test data corresponding to dataset 2 of mechanical model 1

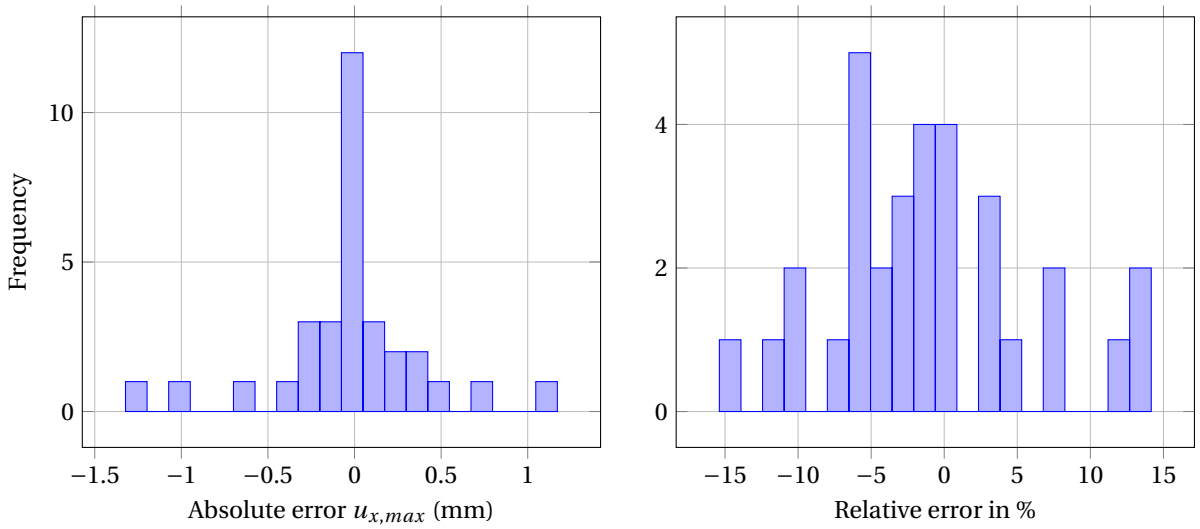


Figure 5.16: Error distribution histograms on test data corresponding to dataset 14 of mechanical model 1

Simplified model 2 Figure 5.18 shows the error distributions for project 2-5-1-1, which corresponds to the eigenvalue analyses with 8 free input parameters. The output values are dimensionless load factors λ_1 corresponding to the first buckling mode. Here the absolute values of the errors are less normally distributed and range between ± 100, while the mean value of the true outputs is $\lambda_1 = 545.5$. The relative errors are between the ranges ± 15% and include extremities up to 60%. These are do not appear to be very accurate predictions.

On the other hand, the plots in Figure 5.17 show a much more concentrated range of errors. These predictions correspond to project 2-1-1-1, eigenvalue buckling with 4 free input parameters. While the mean value is approximately the same with a value of 517.80, the absolute errors in these predictions are a lot smaller, ranging only between ± 50. This improvement is also visible in the relative errors.

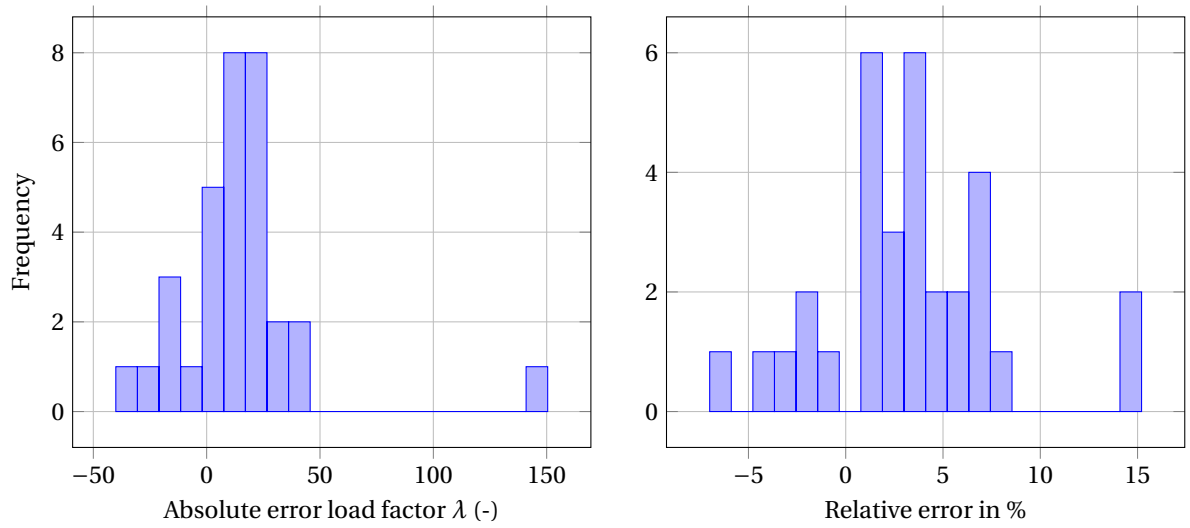


Figure 5.17: Error distribution histograms on test data corresponding to dataset 2 of mechanical model 2

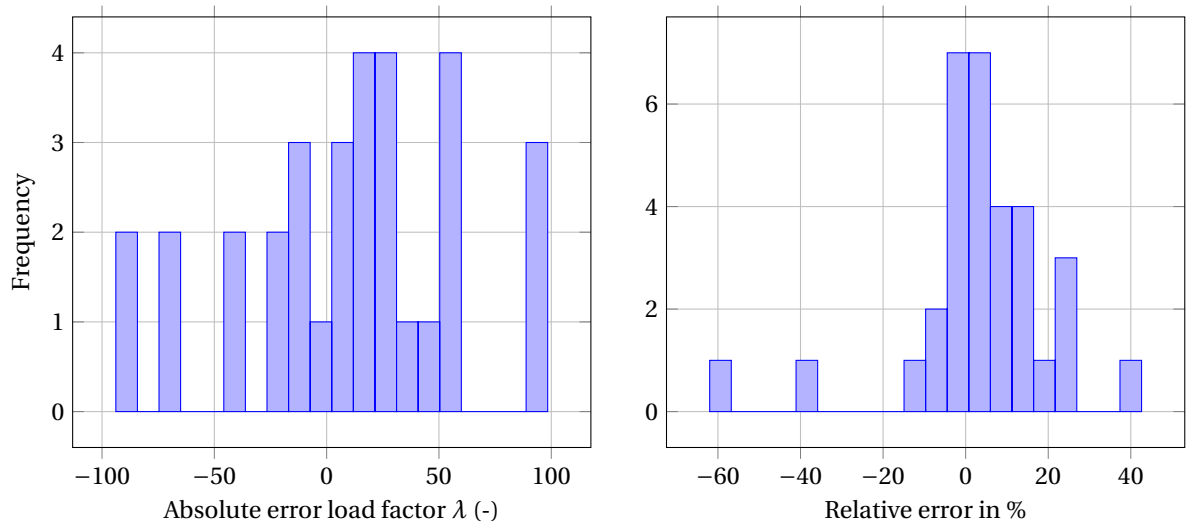


Figure 5.18: Error distribution histograms on test data corresponding to dataset 14 of mechanical model 2

5.3.2. Error analysis main mechanical model 3

In this subsection, the prediction errors on test data for the main mechanical model are presented. All results in these project were obtained by nonlinear analyses. Refer to Table 5.10 for the different output parameters for each of the predictions. Figures 5.19, 5.20 and 5.21 correspond to predictions of maximum stresses, Figure 5.22 and Figure 5.23 correspond to the prediction of maximum deflection $u_{x,max}$, the first one being trained with more training data samples.

Maximum stresses The maximum stresses were extracted from different regions of the structure in order to try to filter out the peak stresses.

- Figure 5.19: The maximum stress of the entire structure
- Figure 5.20: The maximum stress in the skin plate, without the region close to the column
- Figure 5.21: The maximum stress in the longitudinal stiffeners, without the region close to the column

The predictions on maximum stresses in the regions without the columns were expected to be more accurate than the case in which the entire structure is included. This is not directly visible in the three plots when looking at the relative errors. When looking at the absolute errors however, bigger differences are visible. The maximum errors are in the range ± 100 MPa whereas in the other 2 predictions this is mainly between ± 50 MPa.

Also, when looking at the comparative plot in Figure 5.11 of Section 5.2.3, the datasets corresponding to the maximum stresses in the restricted regions, show more accurate predictions than for the predictions predicting the stresses in the total structure, with values $R^2 > 0.8$. for most of the predictions. This more refined extraction of maximum stresses appears to have a positive effect on the accuracy.

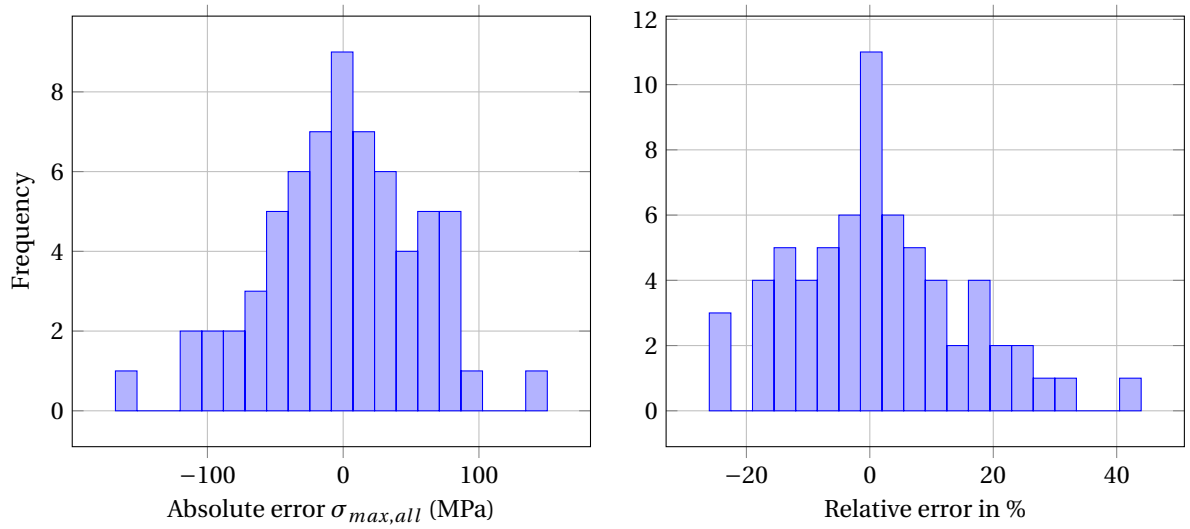


Figure 5.19: Error distribution histograms on test data corresponding to dataset 2 of mechanical model 3

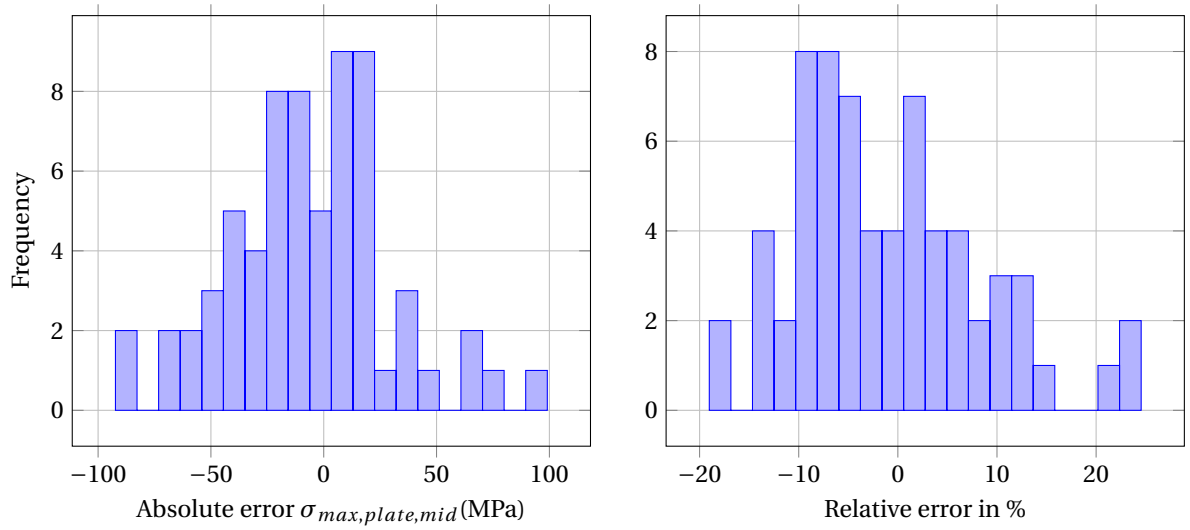


Figure 5.20: Error distribution histograms on test data corresponding to dataset 8 of mechanical model 3

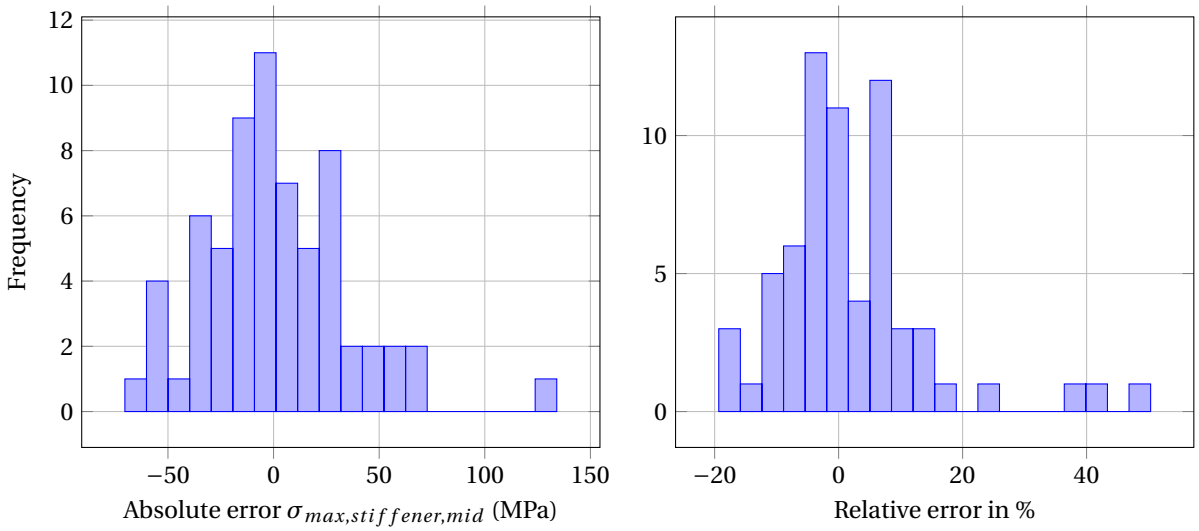


Figure 5.21: Error distribution histograms on test data corresponding to dataset 11 of mechanical model 3

Maximum deflection The predictions of the maximum deflections $u_{x,max}$ for the nonlinear analyses show very big errors in Figure 5.22. The expectation was that predictions on the deflection would be more accurate than the predictions of the stresses, considering the irregular occurrences of stress concentrations.

When looking at Figure 5.23 however, the errors are much lower, while this neural network was trained on 128 samples instead of 256. This unexpected behaviour of larger errors while trained on more data is observed more often. More about this can be found in Section 5.4

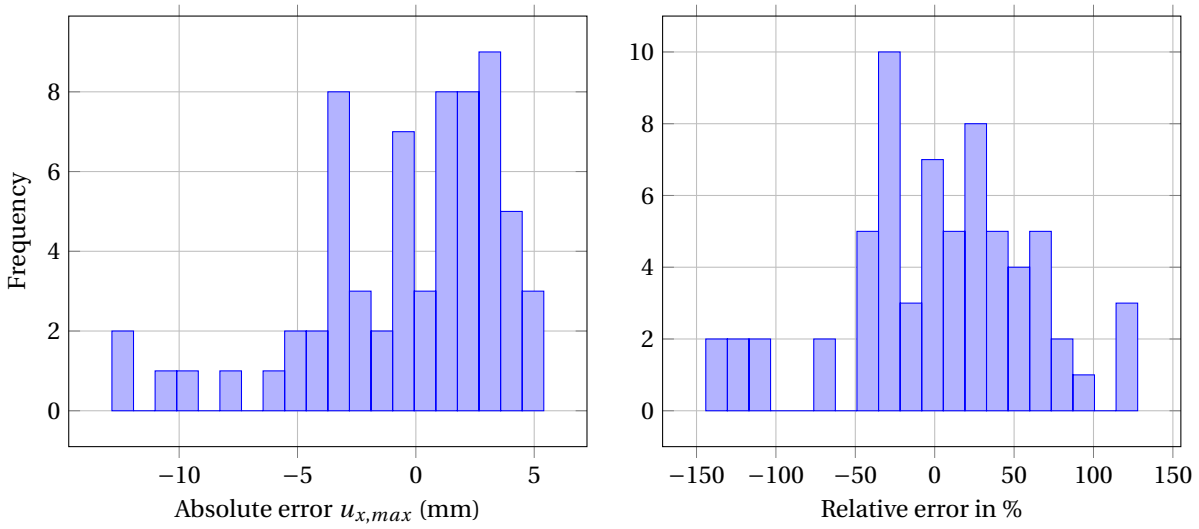


Figure 5.22: Error distribution histograms on test data corresponding to dataset 5 of mechanical model 3

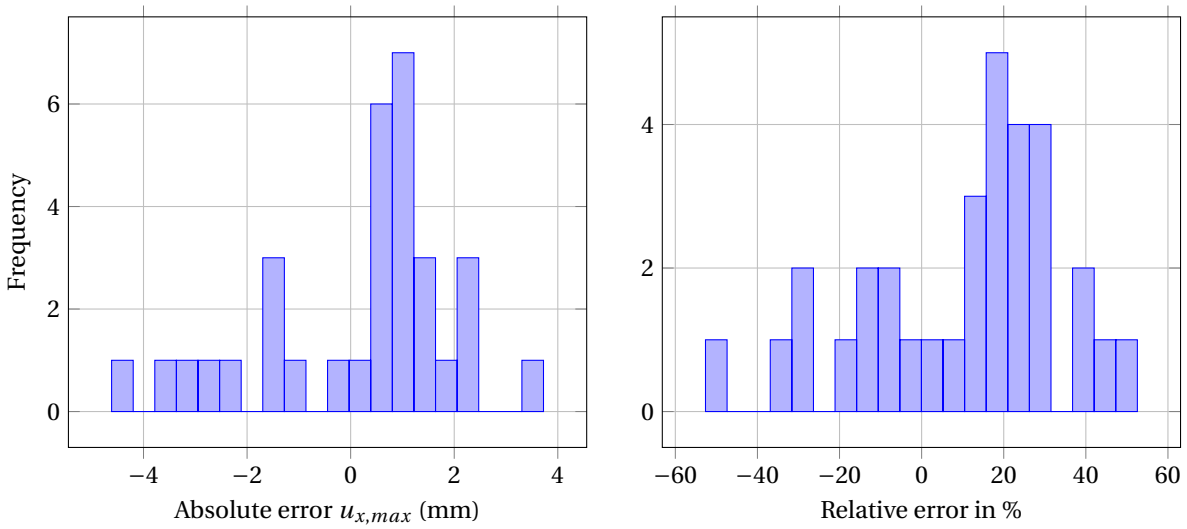


Figure 5.23: Error distribution histograms on test data corresponding to dataset 4 of mechanical model 3

5.4. Influencing parameters on accuracy

This section discusses the influence of the amount of free input variables and the amount of training on the prediction accuracies of the neural networks. Section 5.4.1 will treat the dependence of accuracy on the number of training samples. The dependence on the amount of free input variable is discussed in Section 5.4.2.

5.4.1. Relation accuracy to number of training data samples

See the histograms in Figure 5.24, Figure 5.25 and Figure 5.26. The histograms correspond to mechanical models 1, 2 and 3 respectively. The bars in the histograms are distributed in groups of 3. In case of model 1 and 2, each group of bars corresponds to 3 datasets for a fixed number of free input variables and a varying number of training samples. In case of model 3, each group of bars corresponds to a specific output value (refer to Table 5.10). See the legend for the description.

In all figures, the top plot shows the coefficient of determination as a function of the number of training samples. The bottom plot shows the values of the Root Mean Squared Error (RMSE). One would expect to see an increase of R^2 for increasing number of training samples and a decrease in RMSE. This is not very pronounced in the figures. There does not seems to be a very significant increase in accuracy for increasing number of training samples. Sometimes, neural networks even perform worse for increasing number of training data.

But, according to the histograms with R^2 on the vertical axis, the networks trained on the maximum number of training data (128 samples) almost always perform better than the networks trained on the minimum number of training data (32 samples). The exceptions are the predictions on the mechanical model 1 with 6 free parameters and mechanical model 3 for the prediction of $\sigma_{max,all}$. The network trained on 128 performs a lot worse than the networks trained on 32 and 64 samples.

The predictions of mechanical model 1 with 6 free parameters appears to be the dataset number 8 discussed in Section 5.2.1. This was the only time that the neural network performed worse than the interpolation techniques for mechanical model 1. The convergence plot of the training process showed a very unusual pattern, where the training loss was always higher than the testing loss. This is a clear example of a case where the genetic algorithm converged to a far from optimal neural network.

The histograms in Figure 5.25 are more in correspondence with the expectation that the networks trained on 128 samples perform better than the networks trained on 32 samples. The performance of the networks trained on 64 samples varies, either performing better or worse than the networks trained on 128 samples.

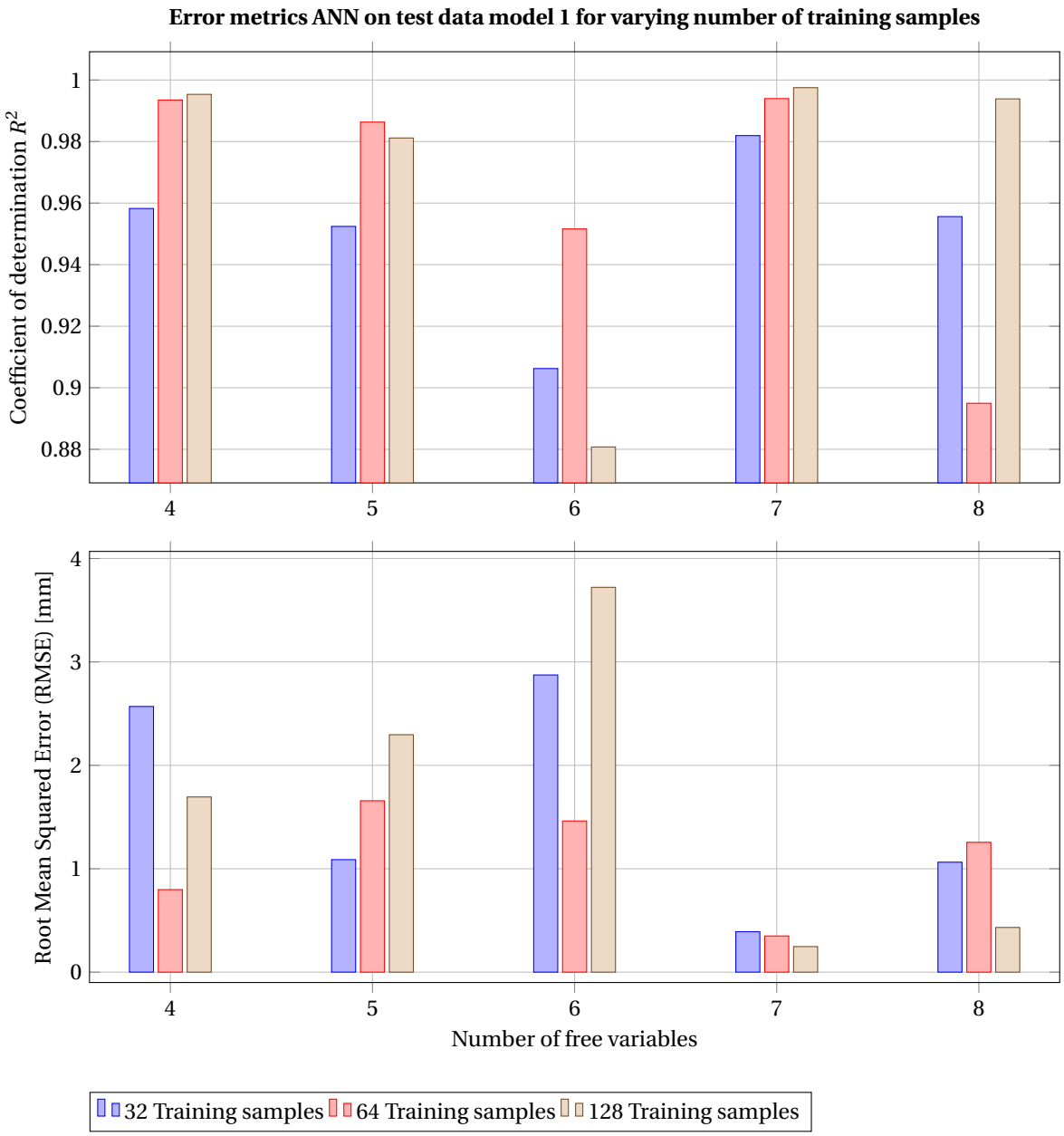


Figure 5.24: Histogram showing varying accuracy of predictions neural networks on results of model 1 for varying number of training samples

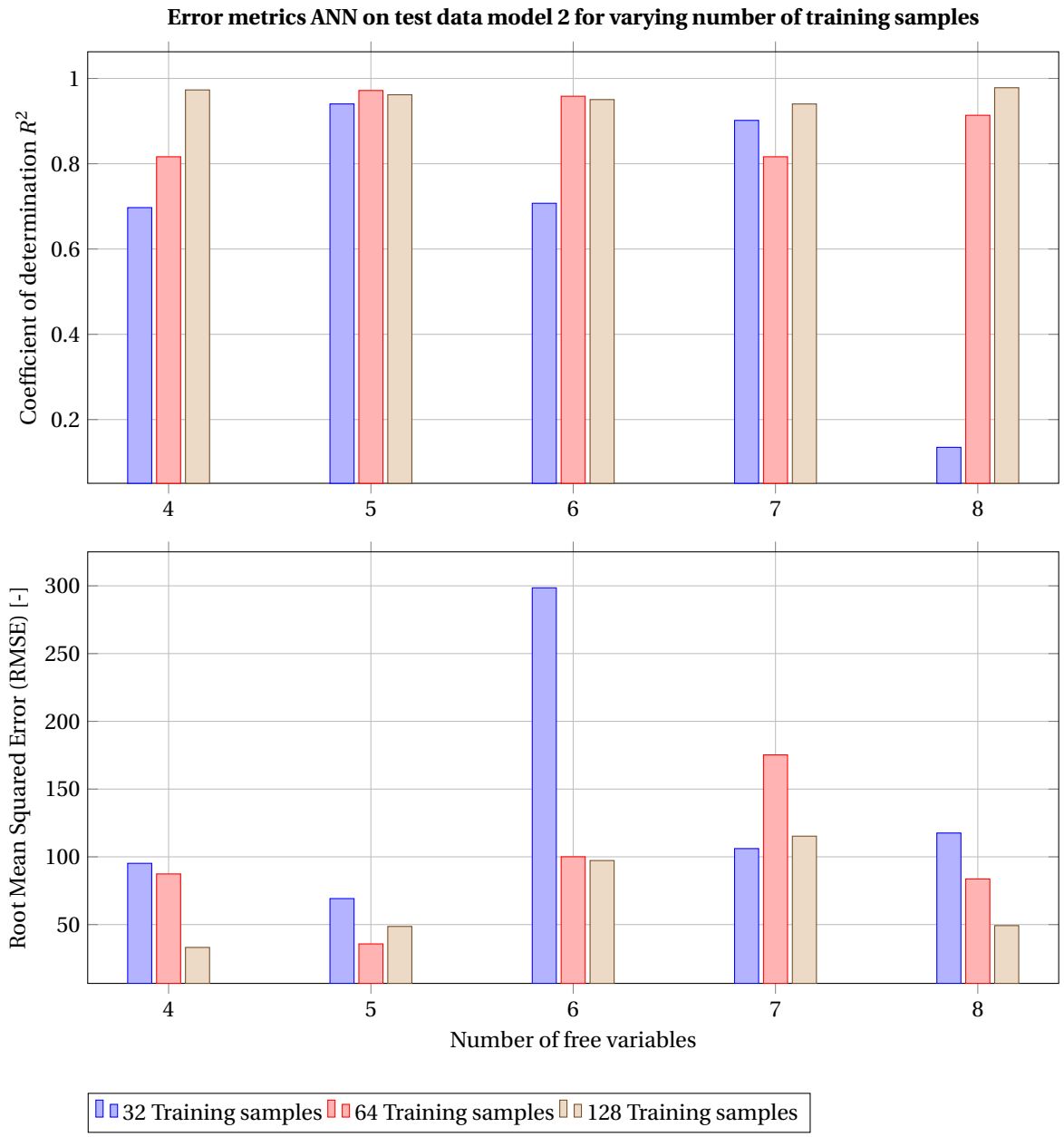


Figure 5.25: Histogram showing varying accuracy of predictions neural networks on results of model 2 for varying number of training samples

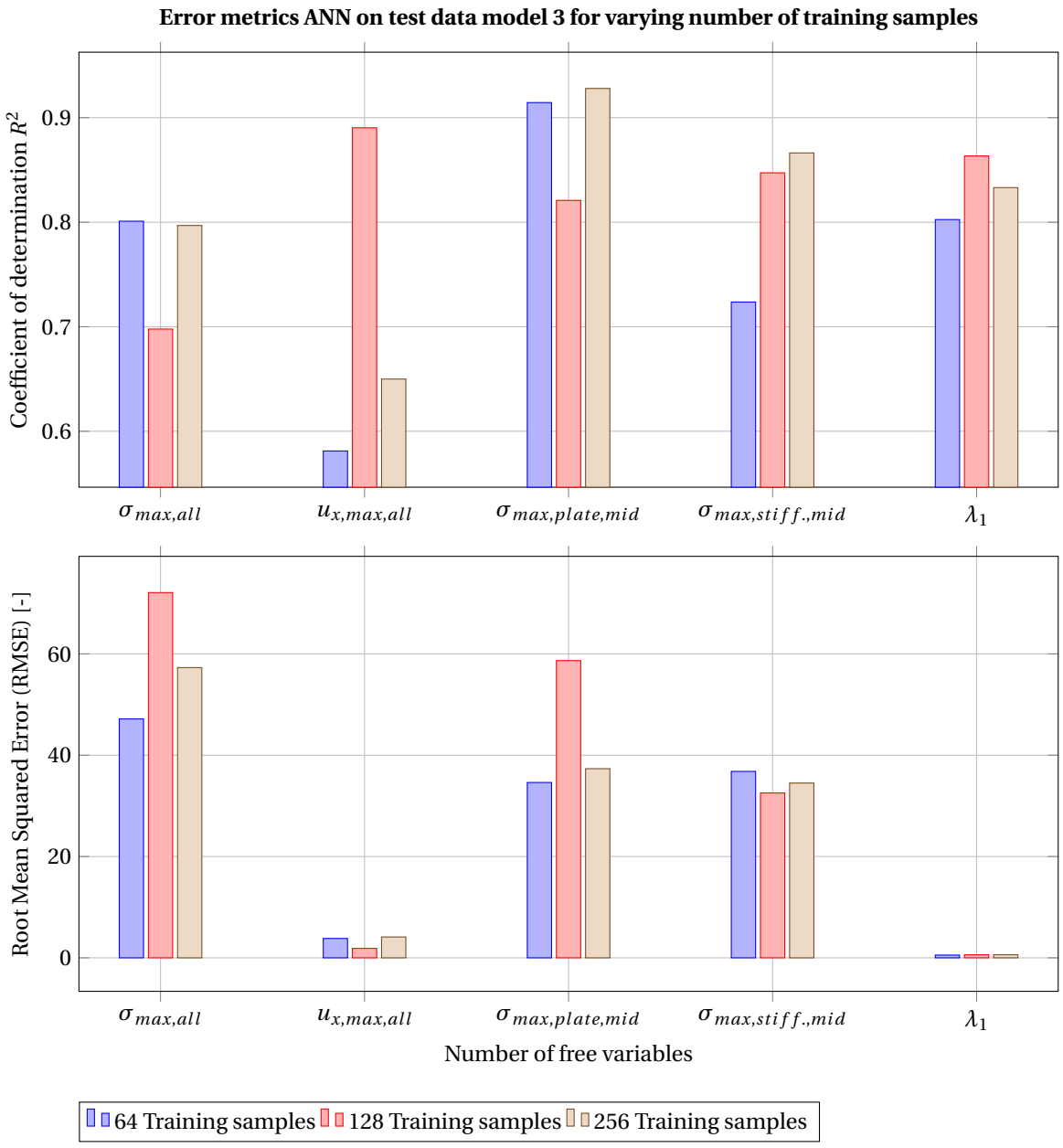


Figure 5.26: Histogram showing varying accuracy of predictions neural networks on results of model 3 for varying number of training samples

5.4.2. Relation accuracy to number of design variables

Figures 5.27 and 5.28 show the response of the accuracies as a function of the number of free input variables for mechanical models 1 and 2 respectively. These plots are based on the same data as the previous figures, although presented in a different way.

The bars in the histograms are distributed in 3 groups of 5. Each group corresponds to 5 datasets for a fixed number of training samples and a varying number of input variables. See the legend for the description.

It is expected that the prediction accuracy decreases when the amount of design variables increases. But also in these plots, this expectation is not very clearly visible. There is no clear relation to be discovered between the dimensionality of the design and the accuracy of the predictions. Only in Figure 5.27, for the datasets with 64 training samples, the coefficient of determination seems to decrease.

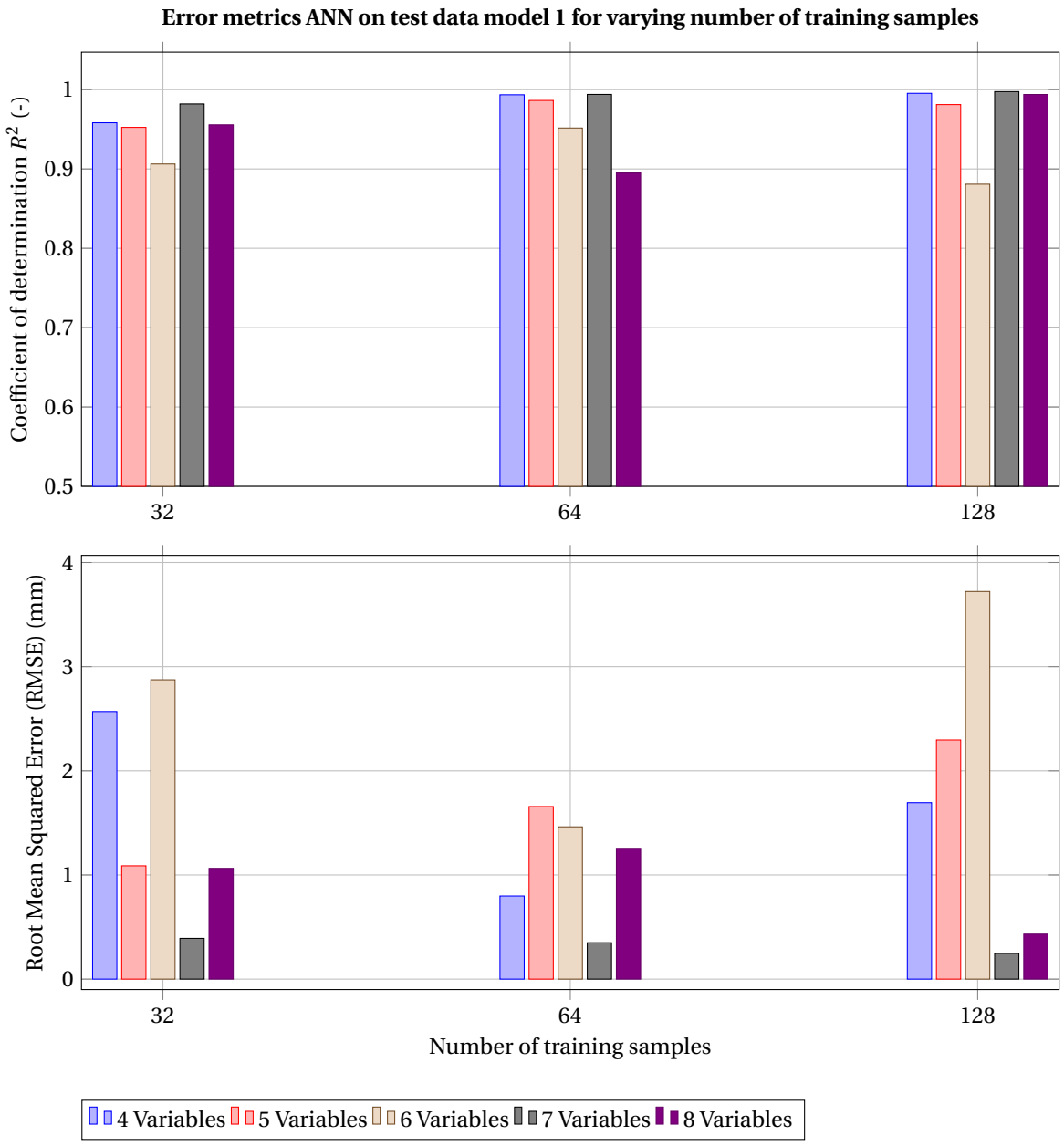


Figure 5.27: Histogram showing varying accuracy of predictions neural networks on results of model 1 for varying number of design variables

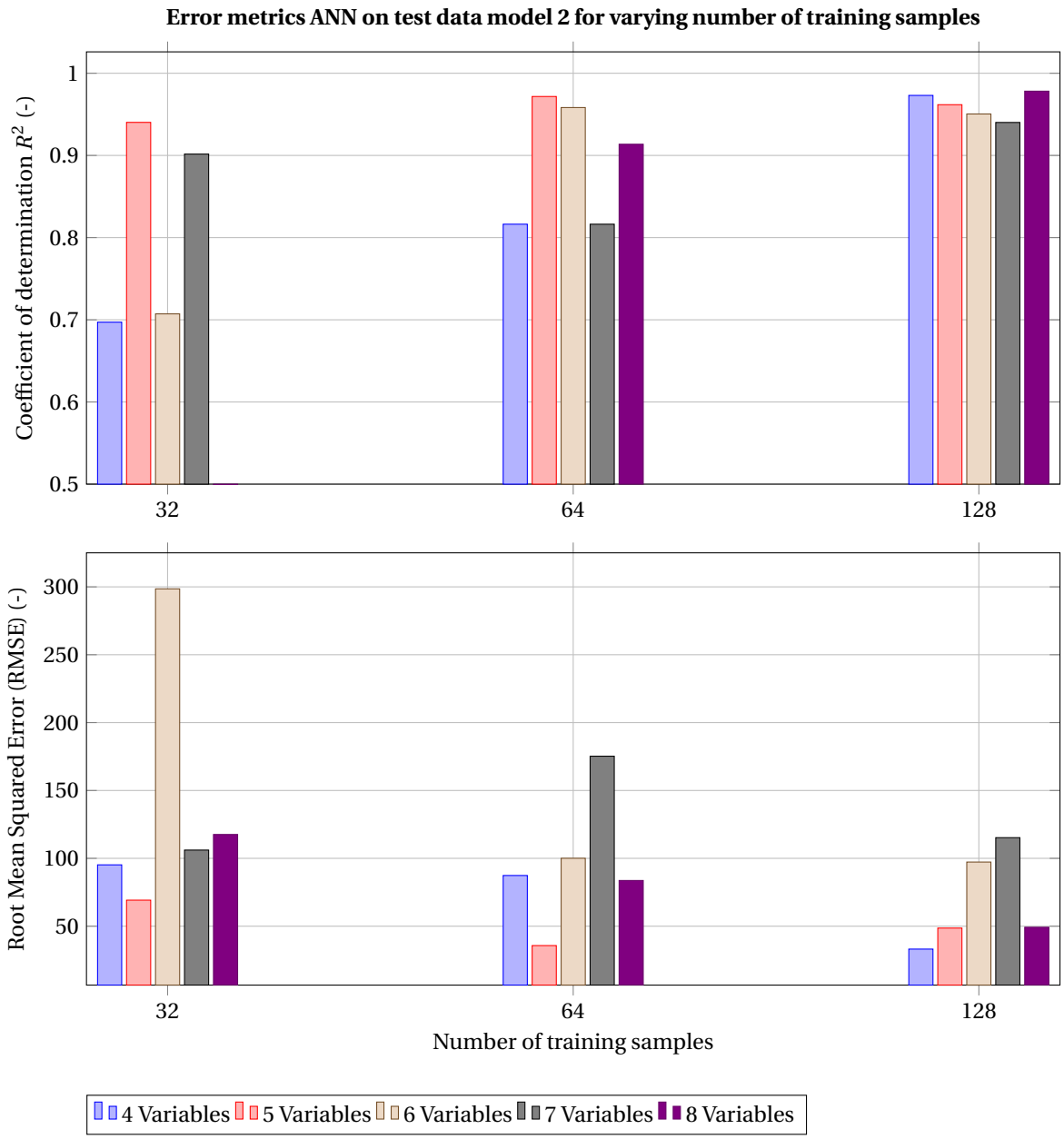


Figure 5.28: Histogram showing varying accuracy of predictions neural networks on results of model 2 for varying number of design variables

5.5. Additional results

After obtaining and analyzing the results as described in the previous sections, this section will give an overview of additional results obtained after applying some small modifications to the possible settings of the neural networks and the addition of some simplified mechanical models.

Section 5.5.1 will explain the improvements made on the neural networks, and the results it yielded on some existing datasets, Section 5.5.2 introduces some new mechanical models which are simplified. The results of these new simple models is presented in Section 5.5.4.

5.5.1. Modifications on Neural Networks training

Some improvements have been applied to the process of the hyperparameter optimization for neural networks. The modifications are listed below.

New predictions have been performed on the datasets of project 3-1-2-3 and 3-1-2-4 after applying these modifications, to see whether the accuracies would improve on predicting the maximum equivalent stresses. The results are shown in Section 5.5.3 Also predictions are made on *new* datasets with simplified models. These new models are described in Section 5.5.2. The results are shown in Section 5.5.4.

- **Increased possible number of hidden layers**
  The maximum possible amount of number of hidden layers is set to 4 instead of only one layer as used previously. This allows for capturing more complex behaviour of a system. The number of neurons is equal in each hidden layer.
- **Removed the option for linear activation function**
  The genetic algorithms of the previous runs show that the linear activation function never yielded good results. In order to avoid unnecessary evaluations of neural networks using linear activation functions, this option has been removed. Furthermore, a neural network containing multiple hidden layers *combined with* a linear activation function is equivalent to just a single-hidden layer neural network [1]. This makes it useless to apply multiple hidden layers.
- **Removed option for Stochastic Gradient Descent (SGD) optimizing scheme**
  The results of the previous runs showed that the neural networks with SGD optimizers never yielded good results. By removing this option, only optimizers with adaptive learning rates are used.
- **Options for kernel initializers modified**
  Two possible methods for the initialization of the weight factors of the neural network are removed as options: namely zero initialization and constant initialization. These options, where all weights have the same value initially, can cause problems in the process of updating the weights during training. If different neurons in the same hidden layer have the same initial weight factors, the gradient descent algorithm will update all these weight factors with the same value. (Page 301 in [19]) In this way all neurons will finally perform the exact same computations, resulting in a useless neural network not able to map complex relations between input and output. A new list of possible initialization methods is added. These can be found in the Python script in Appendix C.
- **All bias matrices initialized as zeros**
  Previously, the bias matrices for the neural networks could be initialized from statistical distributions like a random uniform distribution or a constant value for all bias values. These options are removed and instead *all* bias matrices are initialized with values equal to 0. This is conform convention and eliminates one hyperparameter to be optimized during the genetic algorithm. Zero initialization of bias matrices is compatible with most weight initialization schemes (page 301 in [19]).
- **Parallel processing**
  The genetic algorithm has been modified such, that multiple neural network configurations can be trained at the same time. Up to 8 processes were computed in parallel, reducing the optimization time from 5 hours and 10 minutes to only 1 hour 35 minutes (in case of project number 3-1-2-4). This does not directly influence the accuracy, but more generations or larger populations of neural networks could be generated, possibly resulting in finding a better neural network in the same amount of time. However, the number of generations and the population size is kept the same.

5.5.2. Simplified models. Non-linear analyses of unstiffened plates

To see how the accuracy of the predictions can improve upon reducing the complexity of the models, some additional tests have been performed on strongly simplified problems.

The problems involve *unstiffened* plates under uni-directional loading, applied incrementally using automatic time stepping. The number of free variables was equal to either 1 or 4. All datasets contained 40 data samples, of which 32 samples were used for training, and 8 samples for testing. Table 5.1 shows an overview of the additional simplified problems. Table 5.2 lists the ranges of the parameter values. The same coordinate system and symbol conventions are used as defined in the previous chapters.

- **Geometry**
  The geometry is an *unstiffened*, rectangular, steel plate with a height h_{plate}, a width w_{plate} and a thickness t_{plate}.

- **Boundary conditions**
  The plate is simply supported along all edges. Due to symmetric geometry and loading conditions, symmetric boundary conditions are applied just like the mechanical models 1, 2 and 3 described in the previous sections.
- **Loading**
  The plate is loaded (compressed) in one direction only (y-direction) which is applied directly as a displacement onto the the plate edge.
- **Mesh**
  The mesh size is set to 40 mm for all models
- **Material**
  Models 4 are *linear elastic*. For model 5, the elastic-plastic material model from figure C.2 in the Eurocode [9] is adopted. This is the third material model from Figure 3.17. The yield stress is equal to 355 MPa. The Young's modulus, used to determine the slope of the stress-strain curve after the point of yielding, is equal to 210 GPa.
- **Initial geometrical imperfection:** Instead of performing a linear eigenvalue analysis and applying the first buckling mode shape as geometrical imperfection, a standard equivalent imperfection is applied in order to guarantee that for *each* analysis, the *same* geometrical imperfection is applied. The imperfection shape corresponds to the global buckling shape of the plate as shown in Figure 3.15 from the Eurocode [9] (second item). The amplitude is based on the dimensions of the plate and follows the rules from Figure 3.14 [9]. Before analysis, the nodes of the plate are displaced according to a sinusoidal shape. The implementation using the APDL commands can be found in Appendix J.

For the linear elastic models number 4, datasets with $\sigma_{equ,max}$ and F_y are used as output parameters. For the elastic-plastic models numbers 5, datasets with the maximum equivalent mechanical strain $\epsilon_{equ,max}$ and F_y are used as output parameters. The outputparameter F_y is the reaction force in y-direction, where the displacement is applied along the edge.

Proj. num.	Num. Variables	Variables	Material	Outputparameter
4-1-1-1	1	u_{yy}	Elastic	$\sigma_{max,plate}$
4-1-1-2	1	u_{yy}	Elastic	F_y
4-2-1-1	4	h_{plate}, w_{plate}, t_{plate}, u_{yy}	Elastic	$\sigma_{max,plate}$
4-2-1-2	4	h_{plate}, w_{plate}, t_{plate}, u_{yy}	Elastic	F_y
5-1-1-1	1	u_{yy}	Elastic-plastic	$\epsilon_{max,plate}$
5-1-1-2	1	u_{yy}	Elastic-plastic	F_y
5-2-1-1	4	h_{plate}, w_{plate}, t_{plate}, u_{yy}	Elastic-plastic	$\epsilon_{max,plate}$
5-2-1-2	4	h_{plate}, w_{plate}, t_{plate}, u_{yy}	Elastic-plastic	F_y

Table 5.1: Overview of project numbers, the corresponding input variables and the produced output variables

Proj. num.	h_p [mm]	w_p [mm]	t_p [mm]	u_{yy} [mm]
4-1-1-1	2000	1000	20	0.1, 1.5
4-1-1-2	2000	1000	20	0.1, 1.5
4-2-1-1	2000, 3000	1000, 2000	10, 20	0.1, 1.5
4-2-1-2	2000, 3000	1000, 2000	10, 20	0.1, 1.5
5-1-1-1	2000	1000	20	0.1, 25
5-1-1-2	2000	1000	20	0.1, 25
5-2-1-1	2000, 3000	1000, 2000	10, 20	0.1, 25
5-2-1-2	2000, 3000	1000, 2000	10, 20	0.1, 25

Table 5.2: Overview of project numbers and the ranges of the parameter values. Cells containing a single value imply a fixed value for that parameter. Two values in a cell, separated by a comma, are the minimum and maximum values for that parameter. Note that these value ranges are chosen purely for experimental use. It is not considered whether the resulting designs are sensible in terms of expected efficiency or resistance.

5.5.3. Results on existing datasets

Applying the modifications on the neural network optimization as described in Section 5.5.1, new predictions are made on 2 existing datasets. These are the sets containing 320 data samples each, belonging to projects 3-1-2-3 and 3-1-2-4, containing the output of maximum equivalent stress in the plate or stiffeners respectively. The error histograms are shown in figures 5.29 and 5.30.

When comparing these histograms with the error plots in figures 5.20 and 5.21 from Section 5.3.2, the first thing that can be noticed is that the error bars corresponding to the predictions after the improvements show a slightly more compact bell shape. This in accordance with the results shown in the overview in Table 5.3. The improved optimizations show more predictions to fall within the specified error range compared to before the improvements.

Although some improvements are visible, the differences are not very significant. The bars of the relative errors in Figure 5.30 also show outliers as in the old case of Figure 5.21, with relative errors around 30 to 40 %. Furthermore, although the majority of the 66 predictions have an absolute error smaller than 50 MPa, these errors are still quite significant.

		Num. relative errors		**Num. absolute errors**	
Project	**Settings**	**<5%**	**<10%**	**<25 MPa**	**<50 MPa**
3-1-2-3	**Old**	23	49	39	55
	New	29	54	38	58
3-1-2-4	**Old**	25	51	42	55
	New	30	52	43	60

Table 5.3: Comparison of ranges of prediction errors on test data of projects 3-1-2-3 and 3-1-2-4 (320 samples, of which 66 test samples) before and after improvements of the hyperparameter optimization. The numbers listed is the number of predictions on test data that falls within the error range specified in the column header. 'Old' or 'New' settings refers to predictions before or after improvements of the hyperparameter optimization.

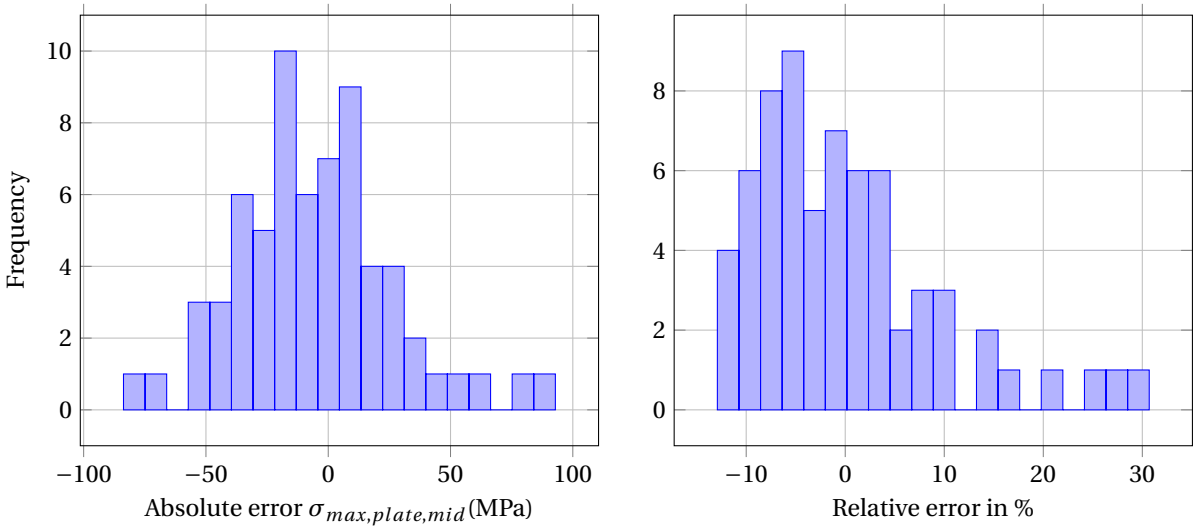


Figure 5.29: Error distribution histograms on test data of mechanical model 3 after applying improvements on the hyperparameter optimization of the neural networks. The predictions are on the maximum equivalent stresses in the *plate*, with the regions neglected that are close to the columns.

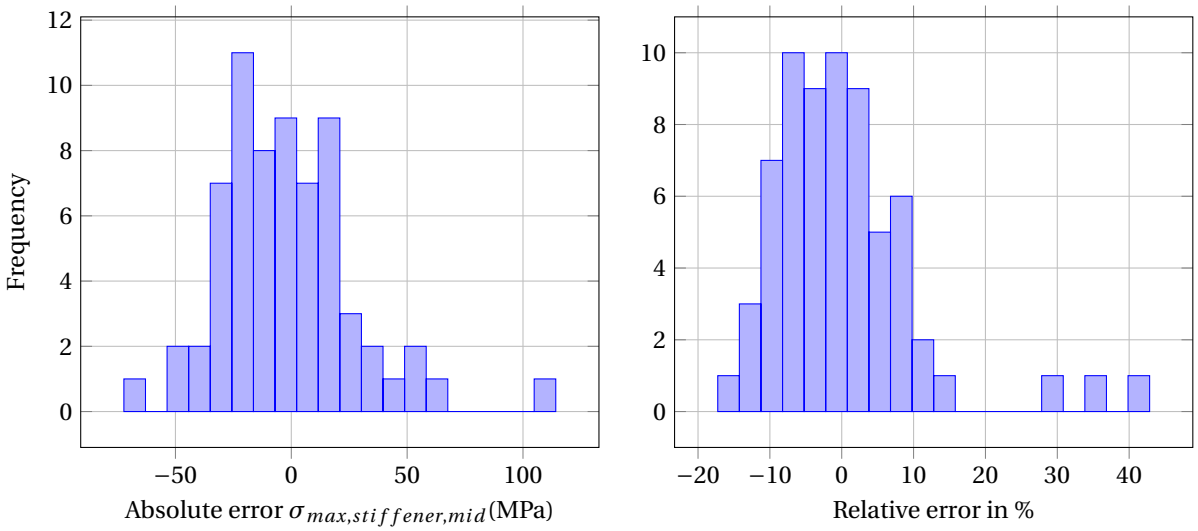


Figure 5.30: Error distribution histograms on test data of mechanical model 3 after applying improvements on the hyperparameter optimization of the neural networks. The predictions are on the maximum equivalent stresses in the *stiffeners*, with the regions neglected that are close to the columns.

5.5.4. Results accuracies unstiffened plates

In this section, the accuracies of predictions of the results of the simplified, unstiffened models are discussed. The accuracies of the neural networks are only compared to Kriging interpolations, since these generally performed better on small datasets compared to the polynomial interpolations. All datasets contain only 40 data samples, of which 8 samples are used for validation. Scatterplots are used to visualize the performance. The true values are on the horizontal axis and the predictions are on the vertical axis. The closer the dots to the line of true value, the more accurate the predictions. Refer to Table 5.1 for an overview of the project numbering and their properties.

Observations scatterplots

- The scatterplots show that both the neural networks and Kriging interpolations perform equally well on datasets with *only one free variable*. Almost all predictions are exactly on the line corresponding to the true value. The execption is in the case of project 5-1-1-2 in Figure 5.36 where the Kriging predictions of the reaction force F_y deviate a lot from the true value.
- In case of linear elastic models with 4 variables, both predictors perform approximately equal, as can be seen in figures 5.33 and 5.34. In the last figure however the neural network performs slightly better, again corresponding to the prediction of the reaction force F_y.
- When including non-linear material and 4 free variables, the neural networks perform significantly better than Kriging interpolations. This is visible in figures 5.37 and 5.38. In this case, the Kriging interpolation shows very strong deviations while the predictions of the neural networks are close to the line of true values.

Next to visual inspection, the performance measures are compared for each dataset between predictions of ANN and Kriging. These are summarized in Table 5.4. This table shows that the neural networks perform best on all datasets of mechanical models 4 and 5.

	MSE		R^2	
Proj num	**ANN**	**Kriging**	**ANN**	**Kriging**
4-1-1-1	**0,30877**	1,673915	**0,999988**	0,999935
4-1-1-2	**4,3E-06**	1,12E-05	**0,999997**	0,999991
4-2-1-1	**98,53278**	162,6066	**0,998378**	0,997324
4-2-1-2	**0,005253**	0,040592	**0,998971**	0,992053
5-1-1-1	**7,87E-07**	2,18E-06	**0,999921**	0,999782
5-1-1-2	**8,97E-06**	0,061075	**0,999966**	0,77046
5-2-1-1	**0,002069**	0,007116	**0,756994**	0,164023
5-2-1-2	**0,062782**	0,866804	**0,904775**	-0,31472

Table 5.4: Comparison of performance measures on all simplified datasets. For each dataset and performance measure, the best value is in bold. The artificial neural network performs best on all datasets. The difference is most significant in dataset 5-1-1-2 and beyond. MSE is the Mean Squared Error value of all predictions on test data. No unit is given since the units vary among the datasets. R^2 is the coefficient of determination.

Linear elastic models. 1 Free variable Figure 5.31 shows the scatter plot of dataset 4-1-1-1, with predictions on the maximum equivalent stress in the plate. Figure 5.32 shows the scatter plot of dataset 4-1-1-2, with predictions on the total reaction force.

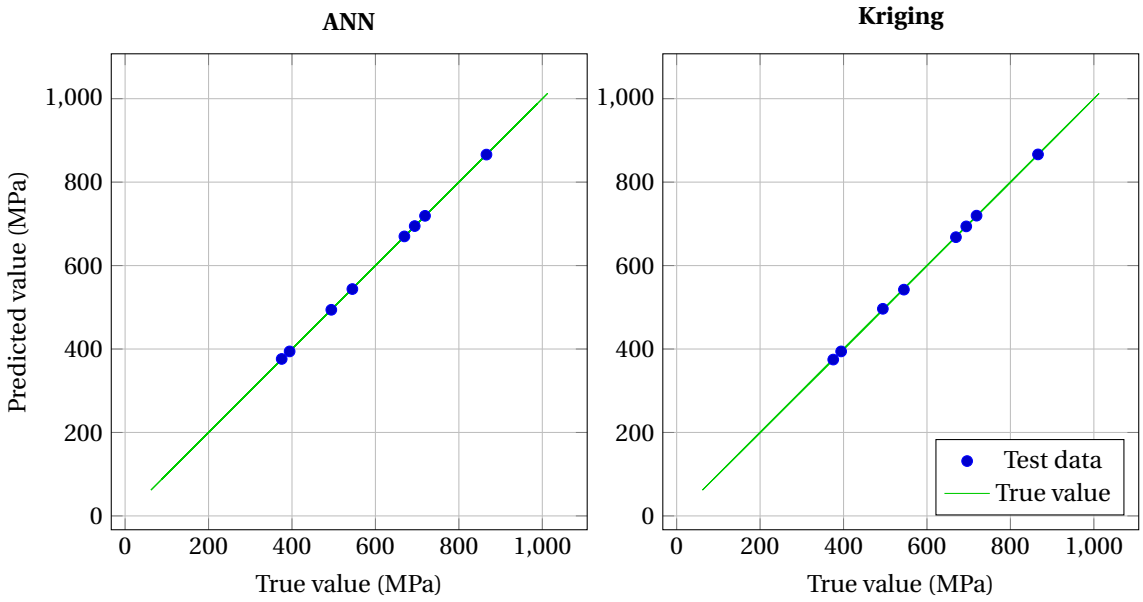


Figure 5.31: Comparison of scatterplots on dataset for project 4-1-1-1. Predicted output parameter is $\sigma_{max,plate}$ in MPa. The predictions of both the ANN and Kriging interpolation lie exactly on the line. The two methods perform equally well on this dataset.

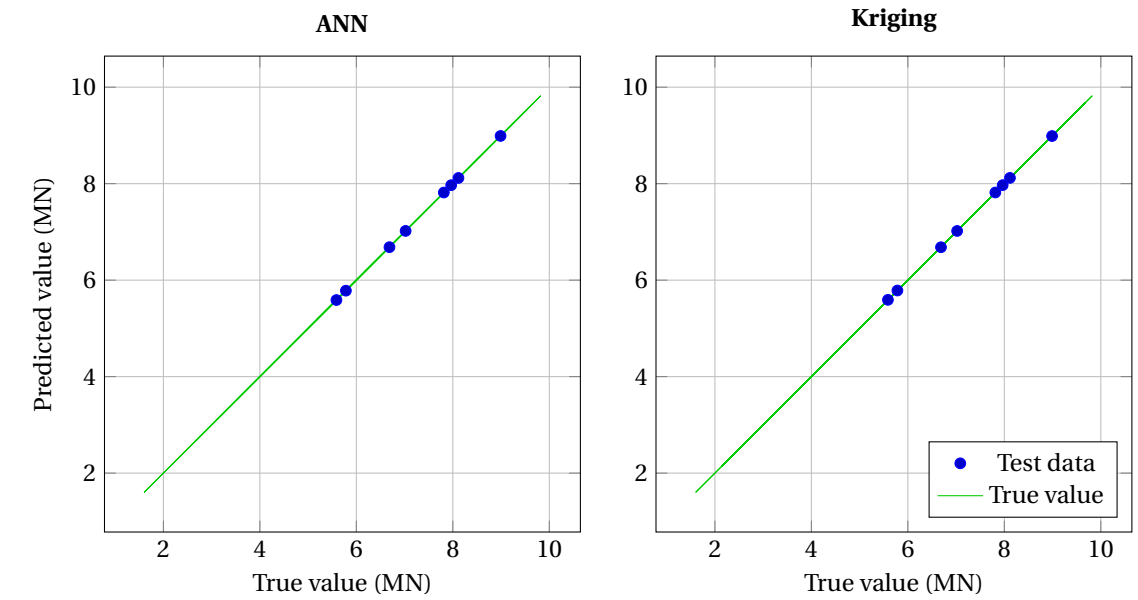


Figure 5.32: Comparison of scatterplots on dataset for project 4-1-1-2. The predicted output parameter is the total reaction force F_y in MN along the loaded edge. The predictions of both the ANN and Kriging interpolation lie exactly on the line. The two methods perform equally well on this dataset.

Linear elastic models. 4 Free variables Figure 5.33 shows the scatter plot of dataset 4-2-1-1, with predictions on the maximum equivalent stress in the plate. Figure 5.34 shows the scatter plot of dataset 4-2-1-2, with predictions on the total reaction force.

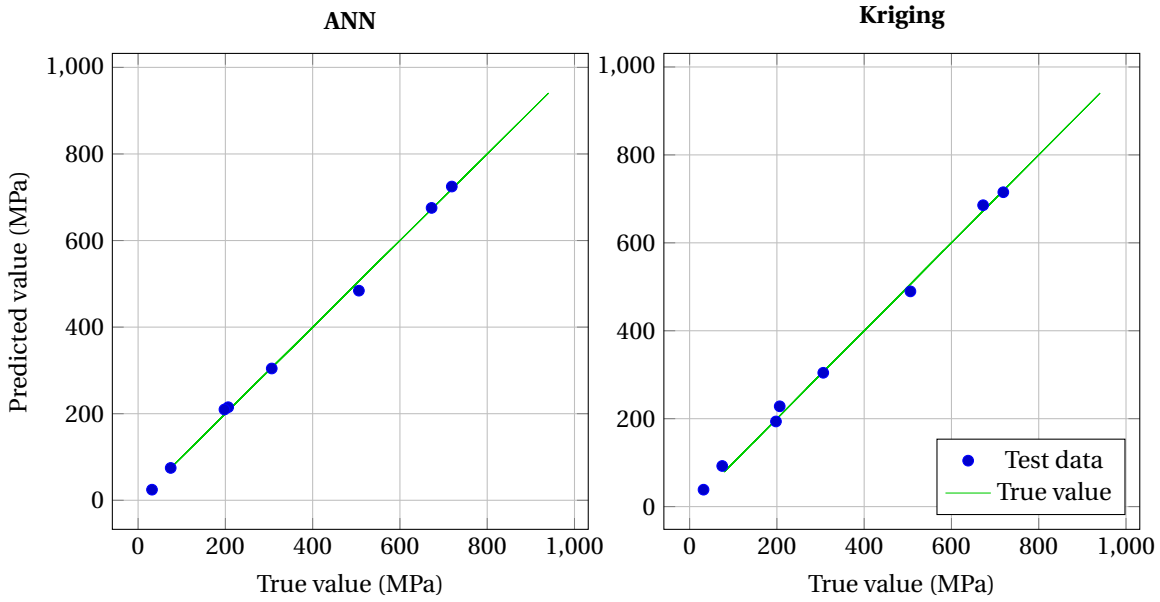


Figure 5.33: Comparison of scatterplots on dataset for project 4-2-1-1. Predicted output parameter is $\sigma_{max,plate}$ in MPa. The predictions of both the ANN and Kriging interpolation are close to the line. The two methods perform approximately equally well on this dataset.

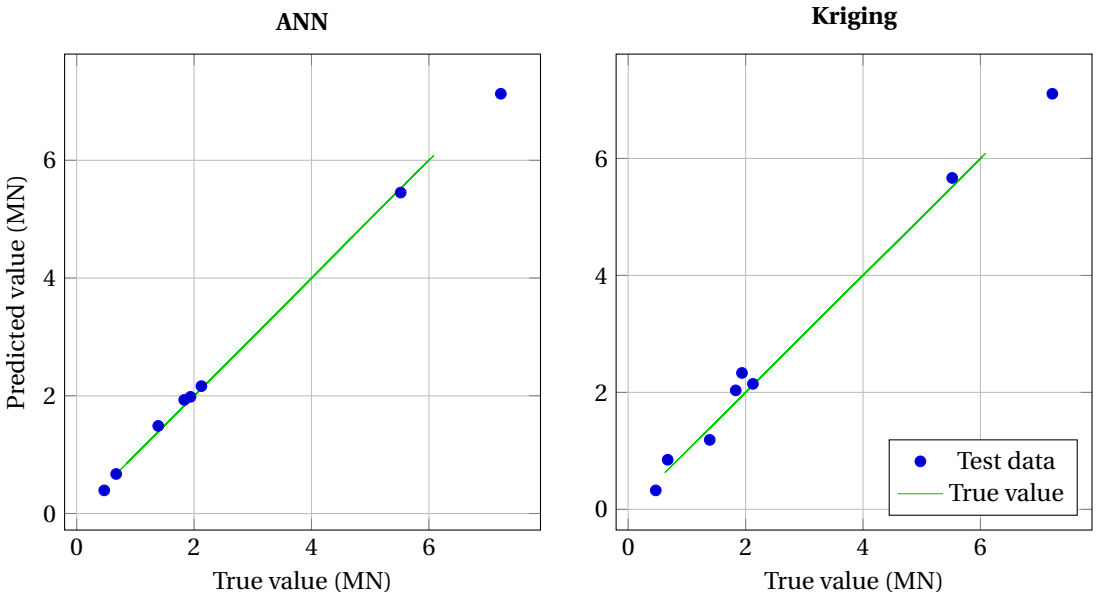


Figure 5.34: Comparison of scatterplots on dataset for project 4-2-1-2. The predicted output parameter is the total reaction force F_y in MN along the loaded edge. The predictions of the ANN are slightly closer to the line compared to Kriging interpolation, implying a better prediction accuracy.

Elastic-plastic material models. 1 Free variable Figure 5.35 shows the scatter plot of dataset 5-1-1-1, with predictions on the maximum equivalent *strain* in the plate. Figure 5.36 shows the scatter plot of dataset 5-1-1-2, with predictions on the total reaction force.

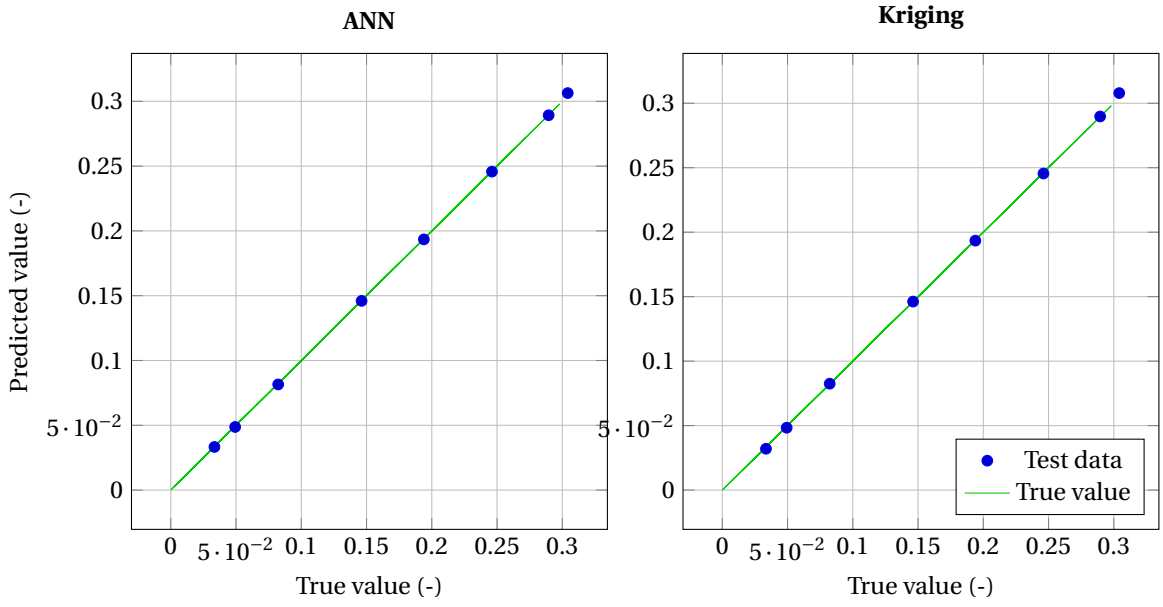


Figure 5.35: Comparison of scatterplots on dataset for project 5-1-1-1. The predicted output parameter is the maximum equivalent mechanical strain $\epsilon_{max,plate}$. The predictions of both the ANN and Kriging interpolation lie exactly on the line. The two methods perform equally well on this dataset.

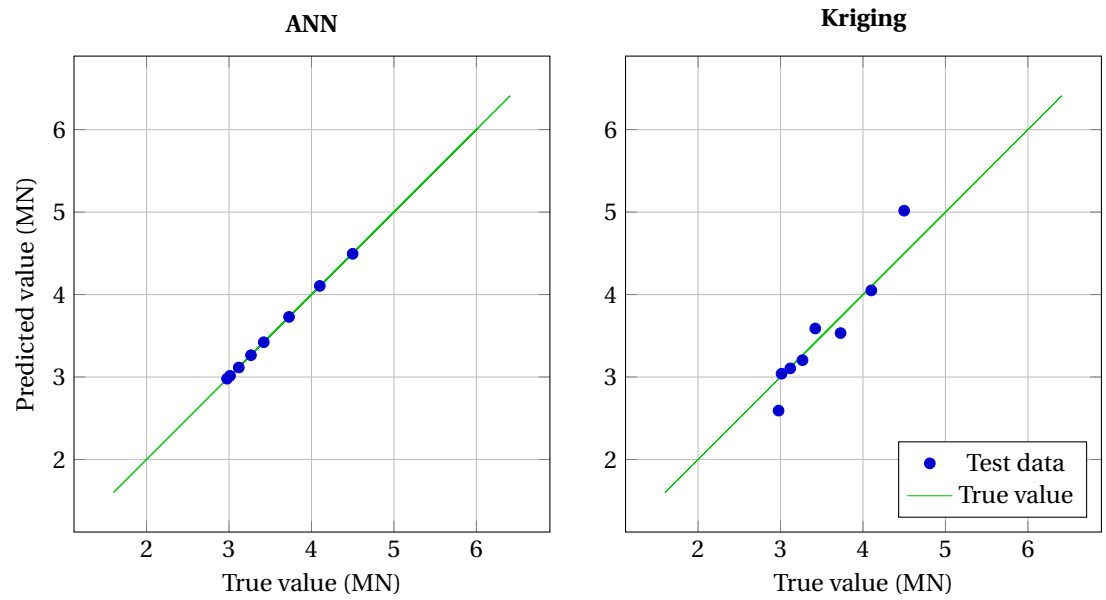


Figure 5.36: Comparison of scatterplots on dataset for project 5-1-1-2. The predicted output parameter is the total reaction force F_y in MN along the loaded edge. The predictions of the ANN lie exactly on the line, implying (near) exact predictions, while the predictions of the Kriging interpolation show very strong deviations. The accuracy of the ANN outperforms that of Kriging on this dataset.

Elastic-plastic material models. 4 Free variables Figure 5.37 shows the scatter plot of dataset 5-2-1-1, with predictions on the maximum equivalent *strain* in the plate. Figure 5.38 shows the scatter plot of dataset 5-2-1-2, with predictions on the total reaction force.

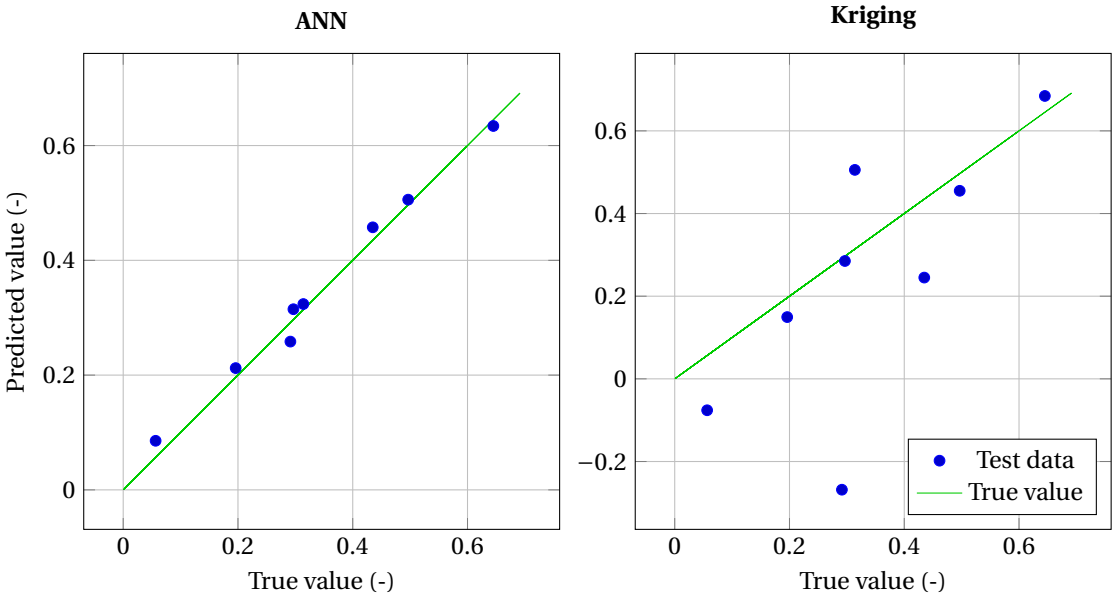


Figure 5.37: Comparison of scatterplots on dataset for project 5-2-1-1. The predicted output parameter is the maximum equivalent mechanical strain $\epsilon_{max,plate}$. The predictions of the ANN lie close to the line, while the predictions of the Kriging interpolation show very strong deviations. The accuracy of the ANN outperforms that of Kriging on this dataset.

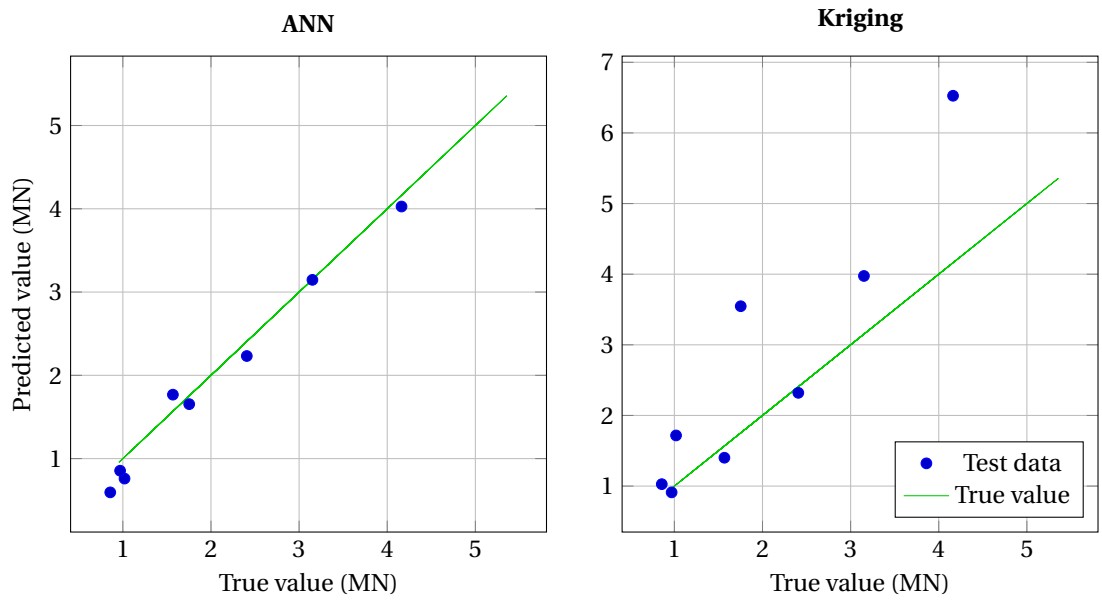


Figure 5.38: Comparison of scatterplots on dataset for project 5-2-1-2. The predicted output parameter is the reaction force F_y in MN. The predictions of the ANN lie close to the line, while the predictions of the Kriging interpolation show very strong deviations. The accuracy of the ANN outperforms that of Kriging on this dataset.

Response of reaction force in elastic-plastic model with one free variable The response plots in Figure 5.39 show the responses of the reaction forces as a function of the applied displacements onto the plate edge. One can clearly see that the first data point on the left with an applied displacement close to zero results in a relatively low reaction force. Kriging interpolation is not able to capture this point, such that it overestimates the reaction force in the early stages of loading. The neural network however is able to capture this point, and therefore shows a load displacement curve with a tipping point after the second data point, where the reaction force decreases for increasing applied displacement. This is in accordance with the expectation of a load displacement curve where the stiffness drops after yielding and further bending.

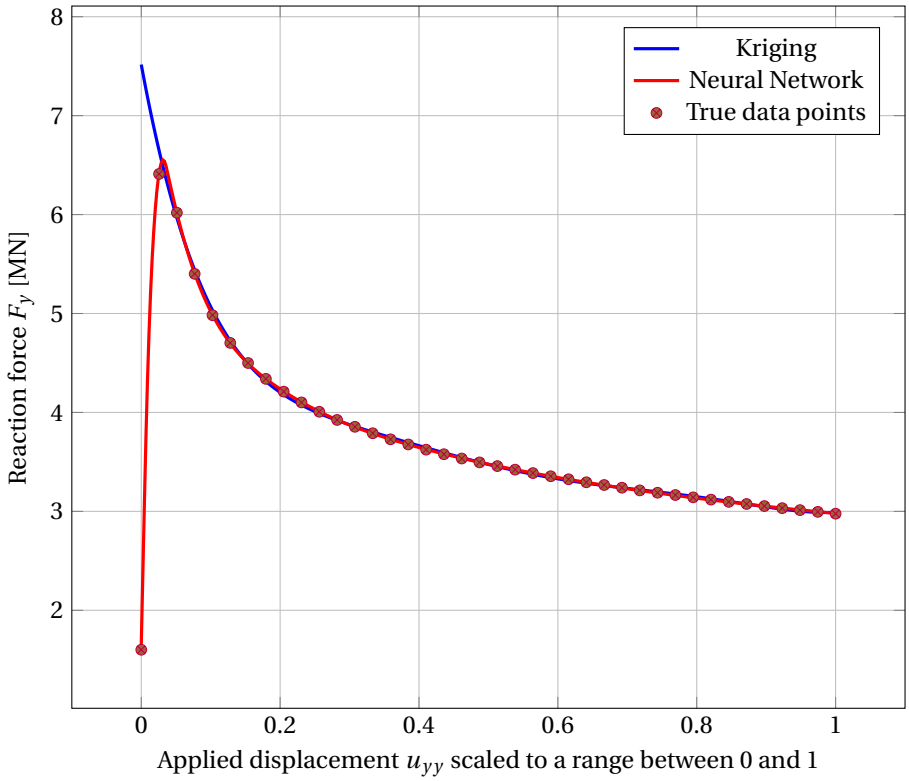


Figure 5.39: Plots showing the predictions of both a trained neural network and a fitted Kriging interpolation. The scatters show the true data points. The neural network is able to fit through all data points, including the first data point. Kriging interpolation overestimates the reaction force in these early load steps.

6

Conclusion

In this chapter, the research questions stated in Chapter 1 will be answered.

Can the results of linear and/or nonlinear finite element analyses be accurately predicted by means of Artificial Neural Networks? Figure 5.2 shows that the maximum linear elastic deflection $u_{x,max}$ of a stiffened steel plate under uniform loading can be accurately predicted. Even for datasets with 8 free design variables and only 32 training data samples (dataset 12), the coefficient of determination on test data was equal to 0.955. The lowest value on datasets of mechanical model 1 was still 0.88. 10 out of 12 predictions resulted in a value higher than 0.9. The error plots in Figure 5.15 show that the majority of predictions have a relative error within the range of ± 5% which is very good. However some outliers are present with relative errors lower than -10%.

The predictions on the eigenvalue were less accurate than for the deflection. See Figure 5.6. Almost all predictions resulted in a coefficient of determination higher than 0.7, except for dataset 12, where all models made very bad predictions.

See Figure 5.11. The predictions on results of geometrical nonlinear analyses was less consistent and yielded both accurate and inaccurate results. The lowest value of R^2 was found to be 0.58, the highest value 0.93. When analyzing the error distributions of the predictions of maximum stresses from nonlinear analyses as shown in the histogram in Figure 5.19 we can see that the absolute errors range up to 100 MPa and relative errors up beyond 20% . This is corresponding to the dataset containing the stresses of the entire structure. However, when focusing on either the plate (Figure 5.20) or the longitudinal stiffeners (Figure 5.21) *and* ignoring a portion of the plate close to the column (to ignore peak stresses), the results look slightly better. The relative errors on plate stresses are mostly within the range of ± 15%.

In Section 5.5, some improvements were described on the hyperparameter optimization process of the neural networks. With these improvements, new neural networks were constructed for datasets 3-1-2-3 and 3-1-2-4. The results are summarized in Table 5.3 and compared with the accuracies before deploying the improvements. The accuracies did improve, however not very significantly. Yet, it can be concluded that 54 out 66 predictions on plate stresses had a relative error smaller than 10%, and 52 out of 66 predictions on stiffener stresses had a relative error smaller than 10% with the improved hyperparameter optimization for neural networks.

Also in Section 5.5, new datasets were generated and new neural networks were constructed for these. These datasets were generated with the results of geometrically nonlinear, and in some cases also physically nonlinear FE analyses of *unstiffened* rectangular plates under uni-directional compressional loading, applied directly as a displacement. The number of free variables was equal to either 1 or 4, which is significantly lower than the 13 design variables of the main mechanical model 3. The performances of the neural networks were compared with Kriging interpolations. Different types of output parameters were predicted. The scatterplots in figures 5.31 till 5.38 show that the predictions of the neural networks are all either exactly on the line of true values, or very close to the true values. For the linear elastic models, the neural networks and Kriging interpolation perform approximately equally well.

Figure 5.39 shows a plot with the reaction force as a function of applied displacement when including material plasticity. One can see that the neural network is able to capture the strong nonlinear response of the system. Even with only one data point which was positioned far from the other data points, the neural network could be fitted such to include this point.

To conclude: Accurate predictions of results of FE analyses are indeed possible, although this is not always the case. Some bad neural networks were found as well. When the goal is to implement the neural network for maximum stresses in a design optimization algorithm, as found for projects 3-1-2-3 and 3-1-2-4, the accuracy is probably not sufficient. Also after improving the neural networks, the accuracies are still not sufficient. The results in Section 5.5 however, showed very good results when the dimensionality of the mechanical problem was reduced.

How does the performance of these neural networks compare to the performance of several well known interpolation techniques? When comparing the accuracy of predictions of the neural networks with the predictions created by either Kriging, 2^{nd} degree or 3^{rd} degree polynomial interpolation, the neural networks generally performed better. Especially on the datasets for mechanical model 1, where the neural networks were more accurate on all but one dataset. Although on some of the datasets, the interpolation methods were more accurate, the neural networks generally showed a more stable accuracy over all datasets, where the coefficient of determination did not drop to very low values as regularly as the other predictors. This can be observed in figures 5.2, 5.6 and 5.11.

This is confirmed by the average values of R^2 over all predictions per predictive model. For the artificial neural networks, an average value of $R^2 = 0.869$ was found over predictions of *all* test datasets of mechanical models 1, 2 and 3. For Kriging interpolation this was $R^2 = 0.640$, for second degree polynomials $R^2 = -1.848$, and a value of $R^2 = 0.356$ for third degree polynomial interpolation.

The predictions of nonlinear results on the unstiffened models (4 and 5) as described in Section 5.5 appear to be a lot more accurate than Kriging interpolation. In Table 5.4 it can be seen that the neural networks perform better on all datasets of the unstiffened models. The difference is most pronounced when including material nonlinearity.

Is it worth the investment of time and computational resources to create the predictive model? One of the main goals of a predictive model is the ability to insert it into a design optimization algorithm, allowing to quickly iterate over a number of design alternatives in order to find the optimal design of a structure quickly. Section 5.1 gives an impression of the time required to create and optimize a neural network for a particular dataset.

The answer to the question highly depends on the application of this procedure in practice. The following points are important to consider.

- How often is the the particular structure/substructure to be designed in a project or among different projects?
- What are the potential gains in terms of time and money when the structure/substructure is optimized using an optimization algorithm compared to manual design optimization?
- What is the required accuracy of the optimization?

When both the repeatability of the structure and the potential gains by optimization of the structure are high, creation of a predictive model could be well worth it. The answer will depend on a cost-benefit analysis. A major benefit of a predictive model is that it can be reused for any project once created.

On the other hand, when the desired accuracy of the optimization is critical, i.e. the design parameters are to be tweaked on a very small scale, a predictive model might not be accurate enough to ensure this amount of precision.

In case of the design of the stiffened skin plate of lock gates, the first two conditions do apply. The lock gates consist of many separate sections that can be optimized individually and also many projects have already been finished by Iv-Infra involving lock gate design. According to the limit state criteria however, the predictive model requires a fair amount of accuracy. In order to be able optimize the design parameters of the skin plate, an accuracy of the predictive model would be required to be in the range of 1 - 5 MPa. According to the

error distribution plots in Chapter 5 however, this amount of accuracy is not reached by the predictive model. So in this particular case, the accuracy of the predictions would have to be improved a lot in order to make the investments in time be worth it.

How does the accuracy of the predictions relate to the number of training samples and the complexity of the model? In figures 5.24, 5.25 and 5.26, we can see that the coefficient of determination R^2 is almost *always* higher for predictions on datasets with the highest number of training samples compared to predictions with datasets with the lowest number of training samples, which is to be expected. There were only 2 exceptions. The accuracies of predictions when trained on the medium amount of training samples (red bars) did not show a very clear pattern.

Also, according to figures 5.27 and 5.28, not a very clear trend has been found that indicates that the predictive models perform worse a higher number of design variables. However, when looking at the drastic improvements of accuracy upon simplification of the mechanical models (models 4 and 5 as described in Section 5.5) and the reduction of their design space, we can safely conclude that the dimensionality must be kept small in order to make accurate predictions. Even when trained with only 32 data samples, the scatterplots in figures 5.31 to 5.38 show very good results. Strong nonlinear behaviours like the one as shown in Figure 5.39 could be captured by keeping the amount of variables limited.

7

Recommendations

This chapter provides an overview of possible improvements and recommendations for further research.

7.1. FE modelling

Handling stress singularities Difficulties were encountered regarding peak stresses. The locations and the actual presence of peak stresses varied among the analyses. Prediction of maximum stresses in the FE model using ANN's was found to be difficult due to the irregular nature of these singularities.

A first step towards improvement of the prediction accuracy of maximum stresses would be to handle these peak stresses. Since the FE analyses for data generation are automated, no personal judgement on the nature of the maximum stresses can be made. Instead, an automated procedure would be necessary to judge whether the maximum stress is a singularity or not. A method must be found to filter out these singularities and find the real, representative maximum stress in the model.

Apart from *handling* peak stresses, it is also possible to take different approaches in which peak stresses are avoided in the first place. These are described in the following paragraphs.

Include plasticity for the main mechanical model All analyses of the main mechanical model in this project were performed with linear elastic material properties. It would be interesting to investigate the influence of including plasticity into the material model. Instead of training an ANN to predict the maximum occurring stress in the model, the network could be trained to predict the maximum strain. The results of mechanical model 5 in Section 5.5 showed promising results for the unstiffened plate.

In regions where stress peaks are present, the material will yield and redistribute the stresses such that these stress peaks disappear. This elimination of stress peaks could be an advantage in terms of prediction capabilities of the neural network. However, divergence of the non-linear analyses may cause issues.

Handling divergence During the automated running of nonlinear FE analyses, no personal judgement can be made about whether an unconverged solution is caused by buckling or by numerical instabilities. A method for properly handling these situations and producing the correct output values would be very useful.

Eigenmodes and geometric imperfections An important step within the nonlinear buckling analysis is selecting an appropriate initial geometric imperfection. In engineering practice this is a very critical step, in which generally multiple combinations of imperfections are combined and analyzed of which the lowest resistance is governing.

In this thesis, the selection of geometric imperfection was solely based on the shape of the first buckling mode, scaled with a certain amplitude. In practice, this amplitude is based on properties of the buckling shape and the geometry of the structure as provided by design guidelines. Since the FE analyses are automated, no judgement can be made on the buckling shape and therefore the correct scaling factor. Therefore the scaling factor was introduced as a free variable, to be input by the user.

It would be interesting to create an algorithm that takes the nodal displacements of the buckling shape as input and produces some key characteristics or classification of the buckling shape as output. This information can then be used to introduce the right scaling factor.

Another possibility would be to create a separate machine learning model that is trained to predict these buckling shape characteristics based on the geometry and loading parameters of the structure.

7.2. Training data

Feature extraction In this project, only the raw values of the input parameters were provided. The accuracy of the predictions could possibly be improved by extracting derived quantities from the input parameters that are expected to have a high influence on the response of the system. In this way not only the dimensionality of the problem is reduced, also the correlation between derived input values and system response may be a lot higher. Experiments can be performed with different sets of derived quantities. Some examples of derived quantities may be the bending stiffness and/or torsional stiffness of the global structure or individual structural elements.

Output parameters based on load-displacement curve Instead of predicting maximum stresses or strains, the neural network could be trained to predict important points in the load-displacement curve of the analysis. The results of mechanical model 5 in Section 5.5 showed promising results for the unstiffened plate. Some examples could include the prediction of the point where the curve first becomes horizontal, or where some threshold value of the stiffness of the structure is reached. It could be the case that these values are more strongly correlated to the provided input parameters than the maximum occurring stresses. Maximum stresses are very localized in nature, depending on location and prone to stress singularities, whereas e.g. the buckling load at which a structure loses stability is expected to be a more global property of the structure.

Nested predictions In line with the previous point about feature extraction, another possibility would be to include predicted output of eigenvalues from one neural network and use these as an input parameter for *another* neural network predicting the buckling load F_{max}.

The procedure can be as follows. It is expected that the (first) eigenvalue λ is a very strong indicator for the buckling load F_{max} of a structure. Therefore, the final neural network could be trained to predict F_{max}, including one or more eigenvalues as *extra* input parameters.

Parallel to this, a separate neural network will be trained to *only* predict the eigenvalues of the structure. This trained neural network will produce the eigenvalue which can be used as extra input in the other neural network that produces F_{max}.

The key advantage in this approach is that training datasets containing only results of *linear* eigenvalue analyses can be produced very fast compared to nonlinear analyses. A lot of data can be generated quickly, providing enough data to train an accurate neural network. When it is indeed the case that predictions of F_{max} are more accurate upon inclusion of λ as an input parameter, the amount nonlinear analyses needed to create an accurate network can be reduced, reducing the computation time.

Narrow down parameter range In this project, the influence of the range of the input parameters was not studied. For each individual mechanical model, the range of possible parameter values was fixed between the same boundaries. It is expected that the accuracy of the predictive model improves for a more narrow parameter range. The drawback is that the design space becomes smaller, reducing the applicability of the trained network. It would be interesting to know to what extend the parameter range influences the accuracy.

7.3. Other machine learning techniques

Network architecture In this project, the possible configurations of neural networks has been limited to feedforward neural networks. However there exists a wide variety of neural network types, or even machine learning algorithms in general. Each network type or machine learning algorithm has its own strengths and weaknesses. It would be interesting to see if other methods can perform better on these types of mechanical problems.

Bibliography

[1] 7 types of neural network activation functions: How to choose? https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/.

[2] Doepy. https://doepy.readthedocs.io/en/latest/#.

[3] Gradient descent method – gradient descent. https://easyai.tech/wp-content/uploads/2019/01/tiduxiajiang-1.png.

[4] Kriging interpolation – the prediction is strong in this one. https://gisgeography.com/kriging-interpolation-prediction/.

[5] Scikit-learn minmaxscaler. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.

[6] *Scikit-Learn PolynomialFeatures.*

[7] *Scikit-Learn PolynomialFeatures.*

[8] Scikit-learn standardscaler. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

[9] *Eurocode 3: Design of steel structures Part 1-5 Plated structural elements*, October 2006.

[10] *ANSYS Mechanical APDL command reference*, November 2013.

[11] *ANSYS Mechanical APDL Element reference*, November 2013.

[12] D. Beg, U. Kuhlmann, L. Davaine, and B. Braun. *Design of Plated Structures.* ECCS, 2010.

[13] J. Brownlee. How to choose loss functions when training deep learning neural networks. https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/, January 2019.

[14] J. Brownlee. How to develop a cnn for mnist handwritten digit classification. https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-ha May 2019.

[15] J. Brownlee. How to use data scaling to improve deep learning model stability and performance. https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/, February 2019.

[16] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015.

[17] A. for Desktop. How kriging works. http://desktop.arcgis.com/en/arcmap/10.3/tools/3d-analyst-toolbox/how-kriging-works.htm, May 2013.

[18] P. Gondár. Preliminary design of longitudinally stiffened skin plate of lock gates (eurocode based). Internship technical report, Delft University of Technology, January 2013.

[19] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016. http://www.deeplearningbook.org.

[20] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org/contents/numerical.html.

[21] P. Hajela and L. Berke. Neural networks in structural analysis and design: An overview. *Computing systems in Engineering*, 3:525–538, 1992.

[22] G. S. Hornby and A. Globus. Automated antenna design with evolutionary algorithms. University of california santa cruz, NASA Ames Research Center, 2006.

[23] Iv-Infra. https://www.nationalestaalprijs.nl/archief/2014/nominaties/infrastructuur/uitbreiding-sluizencomplexen-panamakanaal.

[24] B. Johansson, R. Maquoi, G. Sedlacek, C. Müller, and D. Beg. Commentary and worked examples to en 1993-1-5 plated structural elements. Technical report, October 2007.

[25] K. Koutroumbas and S. Theodoridis. *Pattern Recognition*. Academic Press, 2008.

[26] P. Mehta, M. Bukov, C.-H. Wang, A. G. Day, C. Richardson, C. K. Fisher, and D. J. Schwab. A high-bias, low-variance introduction to machine learning for physicists. http://inspirehep.net/record/1664035/files/parabola-gd.png.

[27] Rasmussen and Williams. Gaussian process regressor. https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.GaussianProcessRegressor.html#sklearn.gaussian_process.GaussianProcessRegressor.

[28] E. Schulz, M. Speekenbrink, and AndreasKrause. A tutorial on gaussian process regression: Modelling, exploring, and exploiting functions. *Journal of Mathematical Psychology*, 85:1–16, 2017.

[29] Scikit-Learn. Cross-validation: evaluating estimator performance. https://scikit-learn.org/stable/_images/grid_search_cross_validation.png.

[30] M. van der Burg. Plate buckling in design codes. Master thesis, Delft University of Technology, 2011.

[31] T. Verhoog. Parametric design tool for longitudinally stiffened plates of lock gates. Technical report, February 2019.

[32] D. H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, pages 1341–1390, 1996.

[33] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, 1:67–82, 1997.

[34] Y. Yao. Do toetsing huidplaten en verstijvers. Technical report, November 2016.

Appendices

A

Main Python script

Script for running the genetic algorithms for hyperparameter optimization

```
1
2  # Keras import statements
3  import keras.optimizers as opt
4  from keras import Sequential
5  from keras import backend as Keras_backend
6  from keras.models import load_model
7  import keras.callbacks as cb
8  import keras.initializers as initializers
9
10 # Deap import statements
11 from deap import base, creator, tools
12
13 from sklearn.model_selection import KFold
14 from data import File, Data, Folder
15 import os
16 import pandas as pd
17
18 import random
19 from scoop import futures
20 import time
21 import numpy as np
22 from files import *
23
24 from multiprocessing import Pool
25
26 np.random.seed(6)
27
28 import concurrent.futures
29
30 """
31 New module containing both ANN and GA classes.
32 Aim is to make a more clean interaction between both and at the same time decouple these from the
33 project- and doe classes.
34 The possible set of hyperparameters will be changed and some validation tests can be run.
35 """
36
37
38
39 def evaluate_ann_single(individual, X_train, y_train, X_test, y_test, callbacks, max_epochs=100000):
40     num_layers, num_neurons, activation, optimizer, kernel_initializer = individual
41     t0 = time.time()
42     ann = ANN()
43     ann.set_hyperparameters(activation, num_layers, num_neurons, optimizer, kernel_initializer)
44     try:
45         ydim = y_train.shape[1]
46     except IndexError:
47         ydim = 1
48     ann.set_dimensions(input_dim=X_train.shape[1],
```

```
                        output_dim=ydim)
    ann.build_model()
    ann.model.fit(x=X_train,
                  y=y_train,
                  batch_size=512,
                  epochs=max_epochs,
                  validation_data=(X_test, y_test),
                  callbacks=callbacks,
                  verbose=0)
    t1 = time.time()


    train_loss = ann.model.evaluate(X_train, y_train)
    validation_loss = ann.model.evaluate(X_test, y_test)
    fit_time = t1 - t0
    return ann, train_loss, validation_loss, fit_time


def evaluate_ann_crossval(individual, X, y, cv=5, max_epochs=50000):
    last_train_losses = []
    last_validation_losses = []
    fit_times = []
    callbacks = [cb.BaseLogger(),
                 cb.TerminateOnNaN(),
                 cb.EarlyStopping(monitor='val_loss',
                                  min_delta=1e-12,
                                  mode='min',
                                  verbose=0,
                                  patience=1000)]

    kf = KFold(n_splits=cv, shuffle=True, random_state=6)
    for train_index, validation_index in kf.split(X):
        ann, train_loss, validation_loss, fit_time = evaluate_ann_single(individual,
                            X_train=X[train_index],
                            y_train=y[train_index],
                            X_test=X[validation_index],
                            y_test=y[validation_index],
                            max_epochs=max_epochs,
                            callbacks=callbacks)
        last_train_losses.append(train_loss)
        last_validation_losses.append(validation_loss)
        fit_times.append(fit_time)
    mean_fit_time = np.mean(fit_times)
    mean_train_loss = np.mean(last_train_losses)
    mean_validation_loss = np.mean(last_validation_losses)
    Keras_backend.clear_session()
    print("Cross validation done: ", str(individual), str(mean_validation_loss))
    return mean_train_loss, mean_validation_loss, mean_fit_time


def refit(individual, X_train, y_train, X_test, y_test, model_filepath, max_epochs=100000):
    print('Refit on: ', str(individual))
    callbacks = [cb.BaseLogger(),
                 cb.TerminateOnNaN(),
                 cb.EarlyStopping(monitor='val_loss',
                                  min_delta=1e-12,
                                  mode='min',
                                  verbose=2,
                                  patience=2000),
                 cb.ModelCheckpoint(model_filepath,
                                    monitor='val_loss',
                                    verbose=0,
                                    save_best_only=True,
                                    save_weights_only=False,
                                    mode='min', period=2)]
    ann, train_loss, validation_loss, fit_time = evaluate_ann_single(individual,
                                                                     X_train, y_train, X_test, y_tes
                                                                     callbacks, max_epochs=max_epoch
    ann.model = load_model(model_filepath)
    return ann

```

```
120
121 def custom_mutation(mutant, toolbox):
122     """Since the individual contains different data types in each attribute, no standard mutation function
123     So instead, this custom mutation function performs a crossover operation between the selected individu
124     and a randomly generated new individual."""
125
126     # for index, attribute in enumerate(random_individual):
127     #     if random.random() < indpb:
128     #         mutant[index] = attribute
129
130     equal_gene = True
131     while equal_gene:
132         # When the selected gene between the mutant and the random individual is equal,
133         # create a new individual and select a new index
134         random_individual = toolbox.individual()
135         index = random.randint(0, len(mutant)-1)      # Swap the attribute of the random individual with i
136         equal_gene = mutant[index] == random_individual[index]
137
138     mutant[index] = random_individual[index]
139     return mutant,
140
141
142 class ANN:
143     def __init__(self):
144         self.model = Sequential()
145
146     def set_hyperparameters(self, activation, num_layers, num_neurons, optimizer, kernel_initializer,
147                             bias_initializer='zeros', loss='mean_squared_error'):
148         self.activation = activation
149         self.hidden_layers = [num_neurons for _ in range(num_layers)]
150         self.optimizer = optimizer
151         self.kernel_initializer = kernel_initializer
152         self.bias_initializer = bias_initializer
153         self.loss = loss
154
155     def set_dimensions(self, input_dim, output_dim):
156         self.input_dim = input_dim
157         self.output_dim = output_dim
158
159     def build_model(self):
160         advanced_optimizers = {
161             'rmsprop': opt.RMSprop(),
162             'adagrad': opt.Adagrad(),
163             'adadelta': opt.Adadelta(),
164             'adam': opt.Adam(),
165             'adamax': opt.Adamax(),
166             'nadam': opt.Nadam()
167         }
168
169         seed = None
170         initializers_dict = {
171                                 'random_normal': initializers.RandomNormal(mean=0.0, stddev=0.05, seed=see
172                                 'random_uniform': initializers.RandomUniform(minval=-0.05, maxval=0.05, se
173                                 'glorot_normal': initializers.glorot_normal(seed=seed),
174                                 'glorot_uniform': initializers.glorot_uniform(seed=seed),
175             'he_normal': initializers.he_normal(seed=seed),
176             'he_uniform': initializers.he_uniform(seed=seed)
177         }
178
179         if self.kernel_initializer in list(initializers_dict):
180             self.kernel_initializer = initializers_dict[self.kernel_initializer]
181
182
183
184
185
186         if self.optimizer in list(advanced_optimizers):
187             self.optimizer = advanced_optimizers[self.optimizer]
188
189         # List the the layer instances
190         from keras.layers import Dense
```

```
        layer_instances = []
        for i, num_neurons in enumerate(self.hidden_layers):
            if i == 0:  # First hidden layer
                layer_instances.append(Dense(num_neurons,
                                             activation=self.activation,
                                             kernel_initializer=self.kernel_initializer,
                                             bias_initializer=self.bias_initializer,
                                             input_dim=self.input_dim))
            else:
                layer_instances.append(Dense(num_neurons,
                                             kernel_initializer=self.kernel_initializer,
                                             bias_initializer=self.bias_initializer,
                                             activation=self.activation))

        layer_instances.append(Dense(self.output_dim))  # Output layer without activation function

        # Add layers and compile model
        for layer in layer_instances:
            self.model.add(layer)
        self.model.compile(optimizer=self.optimizer, loss=self.loss)
        return self.model

    def export_predictions(self, data, prediction_filepath):
        prediction_dict = {
            'Y_train_true': data.Y_train,
            'Y_train_predicted': self.model.predict(data.X_train).reshape(1, -1)[0],
            'Y_test_true': data.Y_test,
            'Y_test_predicted': self.model.predict(data.X_test).reshape(1, -1)[0]
        }
        prediction_df = pd.DataFrame({key: pd.Series(value) for key, value in prediction_dict.items(
        prediction_df.to_csv(prediction_filepath, index=False)


class GeneticAlgorithm:
    def __init__(self):
        pass

    def attach_logfile(self, logfile):
        self.logfile = logfile

    def set_algorithm_settings(self, parameter_grid, num_generations=5, population_size=40, CXPB=0.9
                               elite_portion=0.1, tournament_size=10):
        self.parameter_grid = parameter_grid
        self.num_generations = num_generations
        self.population_size = population_size
        self.CXPB = CXPB
        self.MUTPB = MUTPB
        self.elite_portion = elite_portion
        self.tournament_size = tournament_size


    def set_toolbox_functions(self):
        self.tb = base.Toolbox()
        # creator.create("MinimizeLoss", base.Fitness, weights=(-1.0,))
        # creator.create("Individual", list, fitness=creator.MinimizeLoss)

        self.func_sequence = []
        for key, value in self.parameter_grid.items():
            """Add functions to the sequence which will be called when initializing the individual""
            self.tb.register(key, random.choice, self.parameter_grid[key])
            self.func_sequence.append(getattr(self.tb, key))

        self.tb.register("evaluate", self.evaluate_individual)
        self.tb.register("individual", tools.initCycle, creator.Individual, self.func_sequence, 1)
        self.tb.register("population", tools.initRepeat, list, self.tb.individual)
        self.tb.register("crossover", tools.cxOnePoint)
        self.tb.register("mutate", custom_mutation)
        self.tb.register("select", tools.selTournament, tournsize=self.tournament_size)
```

```
    def evaluate_individual(self, individual):
        X = self.X
        y = self.y
        if str(individual) not in self.evaluated_individuals.keys():
            mean_train_loss, mean_validation_loss, mean_fit_time = evaluate_ann_crossval(individual, X, y)
            self.evaluated_individuals[str(individual)] = (mean_train_loss, mean_validation_loss, mean_fit
        else:
            mean_train_loss, mean_validation_loss, mean_fit_time = self.evaluated_individuals[str(individu

        individual.mean_fit_time = mean_fit_time
        individual.mean_train_loss = mean_train_loss
        individual.mean_validation_loss = mean_validation_loss

        fitness = mean_validation_loss
        individual.fitness.values = fitness,
        return fitness,

    def fit(self, X, y):
        """Data are included as attributes for the mapping function that maps evaluate_individual functior
        to the poupulation. Otherwise errors regarding arguments"""
        self.X = X
        self.y = y

        self.best_individual = self.tb.individual()
        self.best_individual.fitness.values = 1e9,
        self.evaluated_individuals = {}

        generation = 0
        print(f"Start generation {generation}")
        population = self.tb.population(n=self.population_size)
        for ind in population:
            ind.generation = generation


        pool = Pool(os.cpu_count())
        fitnesses = pool.map(self.evaluate_individual, population)
        pool.close()
        pool.join()
        # [self.evaluate_individual(ind) for ind in population]

        for individual, fitness in zip(population, fitnesses):
            individual.fitness.values = fitness
            individual.mean_validation_loss = fitness[0]
            self.logfile.add_individual(individual)
            if individual.fitness.values[0] < self.best_individual.fitness.values[0]:
                self.best_individual = individual

        for generation in range(1, self.num_generations):
            print(f"Start generation {generation}")
            offspring = [self.tb.clone(individual) for individual in population]

            # --------- SELECTION
            elite = tools.selBest(offspring, int(self.elite_portion * len(offspring)))
            tournament_selection = self.tb.select(offspring, int(len(offspring) - len(elite)))
            offspring = elite + tournament_selection  # Offspring for mutation and crossover

            """Create a NEW copy of offspring to ensure that each selected individual remains also
            a UNIQUE python object. This avoids equal mutation for equal individuals after selection."""
            offspring = [self.tb.clone(individual) for individual in offspring]

            # Assign generation number to individuals
            for ind in offspring:
                ind.generation = generation


            # --------- CROSSOVER
            for child1, child2 in zip(offspring[::2], offspring[1::2]):
                if child1 != child2:    # Avoid crossover between identical individuals (unnecessary compu
                    if random.random() < self.CXPB:
                        self.tb.crossover(child1, child2)
                        del child1.fitness.values
```

```
                        del child2.fitness.values


            # --------- MUTATION
            for mutant in offspring:
                if random.random() < self.MUTPB:
                    self.tb.mutate(mutant, self.tb)
                    del mutant.fitness.values


            # Evaluate the individuals for which the fitness values are deleted
            invalid_individuals = [ind for ind in offspring if not ind.fitness.valid]
            pool = Pool(os.cpu_count())
            fitnesses = pool.map(self.evaluate_individual, invalid_individuals)
            pool.close()
            pool.join()
            # [self.evaluate_individual(ind) for ind in invalid_individuals]

            for individual, fitness in zip(invalid_individuals, fitnesses):
                individual.fitness.values = fitness
                individual.mean_validation_loss = fitness[0]
                self.logfile.add_individual(individual)
                if individual.fitness.values[0] < self.best_individual.fitness.values[0]:
                    self.best_individual = individual

            population = offspring
        return population
        #### End of generation


class Logfile:
    def __init__(self, path):
        self.path = path
        self.dataframe = pd.DataFrame()

    def add_individual(self, individual):
        newlog = pd.DataFrame({
            'generation': int(individual.generation),
            'num_layers': [individual[0]],
            'num_neurons': [individual[1]],
            'activation': [individual[2]],
            'optimizer': [individual[3]],
            'kernel_initializer': [individual[4]],
            # 'mean_fit_time': [individual.mean_fit_time],
            # 'mean_train_loss': [individual.mean_train_loss],
            'mean_validation_loss': [individual.fitness.values[0]]
        })
        # Add individual to logfile and export update
        self.dataframe = self.dataframe.append(newlog)
        self.dataframe.to_csv(self.path)

    def export_excel(self):
        # Final frame can be exported to excel file.
        # Only final dataframe exported to excel since modifying an opened excel file raises errors
        writer = pd.ExcelWriter(self.path.replace('.csv', '.xlsx'), engine='xlsxwriter')
        self.dataframe.to_excel(writer, sheet_name='Logfile GA', index=False)
        writer.save()



def export_predictions(Y_train_true, Y_train_pred, Y_test_true, Y_test_pred, prediction_filepath):
    prediction_dict = {
        'Y_train_true': Y_train_true,
        'Y_train_predicted': Y_train_pred,
        'Y_test_true': Y_test_true,
        'Y_test_predicted': Y_test_pred
    }
    prediction_df = pd.DataFrame({key: pd.Series(value) for key, value in prediction_dict.items()})
    prediction_df.to_csv(prediction_filepath, index=False)


```

```
ann_parameter_options = {
    'num_layers': [1, 2, 3, 4],
    'num_neurons': [num for num in range(2,40)],
    'activation': ['tanh', 'relu', 'elu', 'selu', 'exponential', 'sigmoid', 'softplus'],
    'optimizer': ['rmsprop', 'adam', 'adagrad', 'adadelta', 'adamax', 'nadam'],
    'kernel_initializer': ['random_normal', 'random_uniform',
                           'glorot_normal', 'glorot_uniform',
                           'he_normal', 'he_uniform']
    }


"""Define folders for storing and retrieving data"""
main_database = Folder(directory=r'C:\Users\Thomas\Documents\Master Thesis', name='main_database_final')
database_traindata = Folder(main_database.path, 'database_traindata')
database_optimization = Folder(main_database.path, 'database_optimization')




to_train = {
    '4-1-1-1': [40],
    '4-1-1-2': [40],
    '4-2-1-1': [40],
    '4-2-1-2': [40],
    '5-1-1-1': [40],
    '5-1-1-2': [40],
    '5-2-1-1': [40],
    '5-2-1-2': [40],
    '6-1-1-1': [40],
    '6-1-1-2': [40],
    '6-1-1-3': [40],
    '6-2-1-1': [40],
    '6-2-1-2': [40],
    '6-2-1-3': [40],
    '3-1-2-3': [320],
    '3-1-2-4': [320],
}


"""Creator creations must be defined in top level OUTSIDE of main in order for SCOOP to work"""
creator.create("MinimizeLoss", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.MinimizeLoss)


if __name__ == "__main__":
    for project_number in to_train.keys():
        for num_dp in to_train[project_number]:
            # Obtain data file
            project_folder = get_project_folder(database_traindata.path, project_number)
            data_file = get_data_file(database_traindata.path, project_number, num_dp)

            # Read and prepare data
            data = Data()
            data.read_datafile(data_file)
            data.set_output_dim(output_dim=1)
            data.split(test_size=0.2)
            data.scale(scaler='minmax')

            # Define optimization files
            prediction_directory = Folder(database_optimization.path, f'project{project_number}')
            logfilepath = create_logfile_path(prediction_directory.path, project_number, num_dp)
            prediction_filepath_ann = create_prediction_file_path(prediction_directory.path,
                                                                  'ann', project_number, num_dp)

            # # ================================================================================================
            # # --- Genetic algorithm
            GA = GeneticAlgorithm()
            GA.set_algorithm_settings(parameter_grid=ann_parameter_options, num_generations=5, population_
                                      CXPB=0.1, MUTPB=0.9, elite_portion=0.1, tournament_size=8)
            GA.set_toolbox_functions()
            GA.attach_logfile(Logfile(logfilepath))
```

```
475
476
477             pop = GA.fit(data.X_train, data.Y_train)
478             GA.logfile.export_excel()
479
480             bestmodel_file = File(directory=prediction_directory.path,
481                                   name=f'best-ann_project{project_number}_doe-{num_dp}dp',
482                                   extension='.h5')
483
484             print(f"Refit: project {project_number}, {num_dp}dp")
485             best_ann = refit(GA.best_individual, data.X_train, data.Y_train, data.X_test, data.Y_tes
486                              bestmodel_file.path, max_epochs=100000)
487
488             best_ann.export_predictions(data, prediction_filepath_ann)
489             # # ============================================================================================
490
491
492
493
494             # ============================================================================================
495
496             prediction_filepath_gpr = create_prediction_file_path(prediction_directory.path,
497                                                                   'kriging', project_number, num_dp)
498             # ============================================================================
499             #   In this section, predictions are performed and exported with KRIGING
500             # ----------------------------------------------------------------------------
501
502             from sklearn.gaussian_process import GaussianProcessRegressor
503
504             gpr = GaussianProcessRegressor()
505             gpr.fit(data.X_train, data.Y_train)
506
507             Y_train_pred = gpr.predict(data.X_train)
508             Y_test_pred = gpr.predict(data.X_test)
509
510             export_predictions(Y_train_true=data.Y_train,
511                                Y_train_pred=Y_train_pred,
512                                Y_test_true=data.Y_test,
513                                Y_test_pred=Y_test_pred,
514                                prediction_filepath=prediction_filepath_gpr)
515             # ----------------------------------------------------------------------------------------
516             # ========================================================================================
517
518
519             # ============================================================================
520             #   In this section, predictions are performed and exported with 2nd ORDER POLYNOMIALS
521             # ----------------------------------------------------------------------------
522             prediction_filepath_poly2 = create_prediction_file_path(prediction_directory.path,
523                                                                     'poly2', project_number, num_dp)
524
525             from sklearn.linear_model import LinearRegression
526             from sklearn.preprocessing import PolynomialFeatures
527
528             poly = PolynomialFeatures(degree=2)
529             X_poly_train = poly.fit_transform(data.X_train)
530             X_poly_test = poly.fit_transform(data.X_test)
531
532             poly.fit(X_poly_train, data.Y_train)
533
534             lin2 = LinearRegression()
535             lin2.fit(X_poly_train, data.Y_train)
536
537             Y_train_pred = lin2.predict(X_poly_train)
538             Y_test_pred = lin2.predict(X_poly_test)
539
540             export_predictions(Y_train_true=data.Y_train,
541                                Y_train_pred=Y_train_pred,
542                                Y_test_true=data.Y_test,
543                                Y_test_pred=Y_test_pred,
544                                prediction_filepath=prediction_filepath_poly2)
545             # ----------------------------------------------------------------------------------------
```

```
546             # ================================================================================
547
548
549             # ================================================================================
550             #   In this section, predictions are performed and exported with 3rd ORDER POLYNOMIALS
551             # --------------------------------------------------------------------------------
552             prediction_filepath_poly3 = create_prediction_file_path(prediction_directory.path,
553                                                                     'poly3', project_number, num_dp)
554
555
556             poly = PolynomialFeatures(degree=3)
557             X_poly_train = poly.fit_transform(data.X_train)
558             X_poly_test = poly.fit_transform(data.X_test)
559
560             poly.fit(X_poly_train, data.Y_train)
561
562             lin3 = LinearRegression()
563             lin3.fit(X_poly_train, data.Y_train)
564
565             Y_train_pred = lin3.predict(X=X_poly_train)
566             Y_test_pred = lin3.predict(X=X_poly_test)
567
568             export_predictions(Y_train_true=data.Y_train,
569                                Y_train_pred=Y_train_pred,
570                                Y_test_true=data.Y_test,
571                                Y_test_pred=Y_test_pred,
572                                prediction_filepath=prediction_filepath_poly3)
573             # --------------------------------------------------------------------------------
574             # ================================================================================
575     #
```

B

ANN classes

Script containing the classes and functions for creating the Artificial Neural Networks

```
import os

from sklearn.model_selection import KFold
from keras import Sequential
from keras.layers import Dense
import keras.optimizers as opt
import keras.callbacks as callbacks
import numpy as np
import time
import pandas as pd

# A test modification for git testing

# A second test modification for git

class ANN:
    parameter_options = {
         'num_neurons': [num for num in range(1,50)],
         'activation': ['linear', 'tanh', 'relu', 'elu', 'selu', 'exponential', 'sigmoid', 'softplus'],
         'optimizer': ['rmsprop', 'sgd', 'adam', 'adagrad', 'adadelta', 'adamax', 'nadam'],
         'kernel_initializer': ['random_uniform', 'constant', 'zeros'],
         'bias_initializer': ['random_uniform', 'constant', 'zeros'],
    }

    def __init__(self, input_dim, output_dim=1):
        self.input_dim = input_dim
        self.output_dim = output_dim
        self.callbacks = [callbacks.BaseLogger(),
                          callbacks.TerminateOnNaN(),
                          callbacks.EarlyStopping(monitor='val_loss',
                                                  min_delta=1e-12,
                                                  mode='min',
                                                  verbose=2,
                                                  patience=1000)]


    def build_model(self, activation, hidden_layers, optimizer, kernel_initializer, bias_initializer, loss
        # Initiate Sequential instance
        self.model = Sequential()

        advanced_optimizers = {
            'sgd': opt.SGD(),
            'rmsprop': opt.RMSprop(),
            'adagrad': opt.Adagrad(),
            'adadelta': opt.Adadelta(),
            'adam': opt.Adam(),
            'adamax': opt.Adamax(),
            'nadam': opt.Nadam()
```

```
        }
        if optimizer in list(advanced_optimizers):
            optimizer = advanced_optimizers[optimizer]

        # List the the layer instances
        layer_instances = []
        for i, num_neurons in enumerate(hidden_layers):
            if i == 0:  # First hidden layer
                layer_instances.append(Dense(num_neurons,
                                             activation=activation,
                                             kernel_initializer=kernel_initializer,
                                             bias_initializer=bias_initializer,
                                             input_dim=self.input_dim))
            else:
                layer_instances.append(Dense(num_neurons,
                                             kernel_initializer=kernel_initializer,
                                             bias_initializer=bias_initializer,
                                             activation=activation))
        layer_instances.append(Dense(self.output_dim))  # Output layer with linear activation functi

        # Create and compile model
        for layer in layer_instances:
            self.model.add(layer)
        self.model.compile(optimizer=optimizer, loss=loss)
        return self.model

    def fit(self, X, Y, batch_size, epochs, validation_data=None):
        self.model.fit(X, Y,
                       batch_size=batch_size,
                       epochs=epochs,
                       validation_data=validation_data,
                       verbose=2,
                       callbacks=self.callbacks)

    # def fit_kfold_cv(self, X, Y, cv, batch_size, epochs, random_state=None):
    #     # Test
    #     self.train_losses = []
    #     self.validation_losses = []
    #     self.fit_times = []
    #
    #     kf = KFold(n_splits=cv, shuffle=True, random_state=random_state)
    #     for train_index, validation_index in kf.split(X):
    #         t0 = time.time()
    #
    #         initial_weights = self.model.get_weights()
    #         self.model.fit(X[train_index], Y[train_index],
    #                        batch_size=batch_size,
    #                        epochs=epochs,
    #                        validation_data=(X[validation_index], Y[validation_index]),
    #                        verbose=2,
    #                        callbacks=self.callbacks)
    #
    #         # Evaluate and add to lists
    #         t1 = time.time()
    #         training_loss = self.model.evaluate(X[train_index], Y[train_index], batch_size=batch_s
    #         validation_loss = self.model.evaluate(X[validation_index], Y[validation_index], batch_
    #
    #         self.fit_times.append(t1 - t0)
    #         self.train_losses.append(training_loss)
    #         self.validation_losses.append(validation_loss)
    #
    #         self.model.set_weights(initial_weights)     # Reset the weights and biases to the orig
    #
    #     self.mean_fit_time = np.mean(self.fit_times)
    #     self.mean_train_loss = np.mean(self.train_losses)
    #     self.mean_validation_loss = np.mean(self.validation_losses)


    def export_predictions(self, data, prediction_filepath):
        prediction_dict = {
```

```
            'Y_train_true': data.Y_train,
            'Y_train_predicted': self.model.predict(data.X_train).reshape(1, -1)[0],
            'Y_test_true': data.Y_test,
            'Y_test_predicted': self.model.predict(data.X_test).reshape(1, -1)[0]
        }
        prediction_df = pd.DataFrame({key: pd.Series(value) for key, value in prediction_dict.items()})
        prediction_df.to_csv(prediction_filepath, index=False)
```

C

Genetic Algorithm classes

Script containing the classes and functions for creating the Genetic Algorithms

```
1
2  from deap import base, creator
3  from deap import tools
4  import random
5  import pandas as pd
6  from keras import backend as Keras_backend
7  from keras.models import load_model
8  from scoop import futures
9  import multiprocessing
10 import time
11 from sklearn.model_selection import KFold
12
13 import keras.callbacks as callbacks
14
15 from data import *
16 from ann import ANN
17
18 import numpy as np
19
20
21 import pickle
22
23
24 # ============================================================================
25 #
26 # Define functions
27 #
28 # ============================================================================
29
30
31 def evaluate_ann_crossval(individual, data, epochs=50000):
32     # Extract settings from individual and initialize ANN
33     num_neurons, activation, optimizer, kernel_initializer, bias_initializer = individual
34
35     min_train_losses = []
36     min_validation_losses = []
37     fit_times = []
38
39     kf = KFold(n_splits=5, shuffle=True, random_state=None)
40     for train_index, validation_index in kf.split(data.X_train):
41         t0 = time.time()
42
43         ann = ANN(input_dim=data.input_dim, output_dim=data.output_dim)
44         ann.build_model(activation=activation,
45                         hidden_layers=[num_neurons],
46                         optimizer=optimizer,
47                         kernel_initializer=kernel_initializer,
48                         bias_initializer=bias_initializer,
```

```
                          loss='mean_squared_error')

        ann.fit(X=data.X_train[train_index],
                Y=data.Y_train[train_index],
                batch_size=128,
                epochs=epochs,
                validation_data=(data.X_train[validation_index], data.Y_train[validation_index]))
        t1 = time.time()

        history_loss = ann.model.history.history['loss']
        history_val_loss = ann.model.history.history['val_loss']

        min_train_losses.append(min(history_loss))
        min_validation_losses.append(min(history_val_loss))
        fit_times.append(t1 - t0)

    mean_fit_time = np.mean(fit_times)
    mean_train_loss = np.mean(min_train_losses)
    mean_validation_loss = np.mean(min_validation_losses)

    Keras_backend.clear_session()
    return mean_train_loss, mean_validation_loss, mean_fit_time

def refit(individual, data, model_filepath, epochs=50000):
    num_neurons, activation, optimizer, kernel_initializer, bias_initializer = individual
    ann = ANN(input_dim=data.input_dim, output_dim=data.output_dim)

    modelcheckpoint = callbacks.ModelCheckpoint(model_filepath,
                                                monitor='val_loss',
                                                verbose=0,
                                                save_best_only=True,
                                                save_weights_only=False,
                                                mode='min', period=1)
    ann.callbacks.append(modelcheckpoint)

    ann.build_model(activation=activation,
                    hidden_layers=[num_neurons],
                    optimizer=optimizer,
                    kernel_initializer=kernel_initializer,
                    bias_initializer=bias_initializer,
                    loss='mean_squared_error')

    ann.fit(data.X_train, data.Y_train, batch_size=128, epochs=epochs,
            validation_data=(data.X_test, data.Y_test))
    ann.history = ann.model.history.history

    ann.model = load_model(model_filepath)    # Load model with best validation score
    return ann

def export_predictions(self,database, projectname, trainfilename, data):
    prediction_directory = Folder(database.path, projectname)
    prediction_filename = trainfilename.replace('traindata', 'prediction-ann')
    prediction_file = File(prediction_directory.path, prediction_filename, '.csv')
    global prediction_dict
    prediction_dict={
        'Y_train_true': data.Y_train,
        'Y_train_predicted': self.best_ann.predict(data.X_train).reshape(1,-1)[0],
        'Y_test_true': data.Y_test,
        'Y_test_predicted': self.best_ann.predict(data.X_test).reshape(1,-1)[0]
    }
    prediction_df = pd.DataFrame({key: pd.Series(value) for key, value in prediction_dict.items()})
    prediction_df.to_csv(prediction_file.path, index=False)

def custom_mutation(mutant, toolbox):
    """Since the individual contains different data types in each attribute, no standard mutation fu
    So instead, this custom mutation function performs a crossover operation between the selected in
    and a randomly generated new individual."""

    # for index, attribute in enumerate(random_individual):
    #     if random.random() < indpb:
    #         mutant[index] = attribute
```

```
120
121         equal_gene = True
122         while equal_gene:
123             # When the selected gene between the mutant and the random individual is equal,
124             # create a new individual and select a new index
125             random_individual = toolbox.individual()
126             index = random.randint(0, len(mutant)-1)      # Swap the attribute of the random individual with i
127             equal_gene = mutant[index] == random_individual[index]
128
129         mutant[index] = random_individual[index]
130         return mutant,
131
132
133 # ======================================================================================
134 #
135 # Define classes
136 #
137 # ======================================================================================
138
139
140 class Logfile(File):
141     def __init__(self, directory, name, extension):
142         super().__init__(directory, name, extension)
143
144         self.dataframe = pd.DataFrame()
145
146     def log_generation(self, population):
147         log = pd.DataFrame({
148             'parameters': [ind for ind in population],
149             'fitness': [ind.fitness.values[0] for ind in population]
150         })
151
152     def add_individual(self, individual, generation):
153         newlog = pd.DataFrame({
154             'generation': generation,
155             'num_neurons': [individual[0]],
156             'activation': [individual[1]],
157             'optimizer': [individual[2]],
158             'kernel_initializer': [individual[3]],
159             'bias_initializer': [individual[4]],
160             'mean_fit_time': [individual.mean_fit_time],
161             'mean_train_loss': [individual.mean_train_loss],
162             'mean_validation_loss': [individual.fitness.values[0]]
163         })
164         self.dataframe = self.dataframe.append(newlog)
165         # Export intermediate results to csv file
166         self.dataframe.to_csv(self.path)
167
168     def export_excel(self):
169         # Final frame can be exported to excel file.
170         # Only final dataframe exported to excel since modifying an opened excel file raises errors
171         writer = pd.ExcelWriter(self.path.replace('.csv', '.xlsx'), engine='xlsxwriter')
172         self.dataframe.to_excel(writer, sheet_name='Sheet1', index=False)
173         writer.save()
174
175
176
177
178
179
180
181
182
183
184
185 class GeneticAlgorithm:
186
187     def __init__(self, parameter_grid, num_generations=3, population_size=10, CXPB=0.1, MUTPB=0.9,
188                  elite_portion=0.1, tolerance=1e-6, tournament_size=3):
189         self.parameter_grid = parameter_grid
190         self.num_generations = num_generations
```

```
191         self.population_size = population_size
192         self.CXPB = CXPB
193         self.MUTPB = MUTPB
194         self.elite_portion = elite_portion
195         self.tolerance = tolerance
196         self.tournament_size = tournament_size
197 
198         # Create classes for the Fitness and the individual
199         creator.create("MinimizeLoss", base.Fitness, weights=(-1.0,))
200         creator.create("Individual", list, fitness=creator.MinimizeLoss)
201 
202         # Register special functions in the toolbox
203         self.register_toolbox()
204 
205     def register_toolbox(self):
206         self.tb = base.Toolbox()
207 
208         self.func_sequence = []
209         for key, value in self.parameter_grid.items():
210             """Add functions to the sequence which will be called when initializing the individual""
211             self.tb.register(key, random.choice, self.parameter_grid[key])
212             self.func_sequence.append(getattr(self.tb, key))
213 
214         self.tb.register("evaluate", self.evaluate_individual)
215         self.tb.register("individual", tools.initCycle, creator.Individual, self.func_sequence, 1)
216         self.tb.register("population", tools.initRepeat, list, self.tb.individual)
217         self.tb.register("crossover", tools.cxOnePoint)
218         self.tb.register("mutate", custom_mutation)
219         self.tb.register("select", tools.selTournament, tournsize=self.tournament_size)
220 
221         # Register special 'map' function from the futures module to enable parallel processing
222         self.tb.register("map", futures.map)
223 
224     def attach_logfile(self, logfile):
225         self.logfile = logfile
226 
227     def attach_data(self, data):
228         self.data = data
229         self.X_train = data.X_train
230         self.Y_train = data.Y_train
231 
232     def evaluate_individual(self, individual):
233         if str(individual) not in self.evaluated_individuals.keys():
234             mean_train_loss, mean_validation_loss, mean_fit_time = evaluate_ann_crossval(individual,
235             self.evaluated_individuals[str(individual)] = (mean_train_loss, mean_validation_loss, me
236         else:
237             mean_train_loss, mean_validation_loss, mean_fit_time = self.evaluated_individuals[str(in
238 
239         individual.mean_fit_time = mean_fit_time
240         individual.mean_train_loss = mean_train_loss
241         individual.mean_validation_loss = mean_validation_loss
242 
243         fitness = mean_validation_loss
244         individual.fitness.values = fitness,
245         return fitness,
246 
247     def evaluate_population(self, population, generation):
248         # self.tb.map(self.evaluate_individual, population)
249 
250         for individual in population:
251             # individual.generation = generation
252             self.evaluate_individual(individual)
253             self.logfile.add_individual(individual, generation)
254             # Replace best individual with new individual in case it performs better
255             if individual.fitness.values[0] < self.best_individual.fitness.values[0]:
256                 self.best_individual = individual
257 
258     def run_evolution(self):
259         self.best_individual = self.tb.individual()
260         self.best_individual.fitness.values = 1e9,
261         self.evaluated_individuals = {}
```

```
262
263         generation = 0
264         population = self.tb.population(n=self.population_size)  # Create initial population
265         self.evaluate_population(population, generation)
266         # [self.logfile.add_individual(individual, generation) for individual in population]
267
268         # Loop over generations
269         for generation in range(1, self.num_generations):
270             # Make a clone of the population for further manipulation
271             offspring = [self.tb.clone(individual) for individual in population]
272
273             # --------- SELECTION
274             elite = tools.selBest(offspring, int(self.elite_portion*len(offspring)))  # Select elite for c
275             # elite_nextgen = [self.tb.clone(individual) for individual in elite]     # Clone elite for ne
276             tournament_selection = self.tb.select(offspring, int(len(offspring)-len(elite)))
277             offspring = elite + tournament_selection    # Offspring for mutation and crossover
278
279             """Create a NEW copy of offspring to ensure that each selected individual remains also
280             a UNIQUE python object. This avoids equal mutation for equal individuals. """
281             offspring = [self.tb.clone(individual) for individual in offspring]
282
283
284             # --------- CROSSOVER
285             for child1, child2 in zip(offspring[::2], offspring[1::2]):
286                 if child1 != child2:    # Avoid crossover between identical individuals (unnecessary compu
287                     if random.random() < self.CXPB:
288                         self.tb.crossover(child1, child2)
289                         del child1.fitness.values
290                         del child2.fitness.values
291
292
293             # --------- MUTATION
294             for mutant in offspring:
295                 print(mutant)
296                 if random.random() < self.MUTPB:
297                     # print('mutation! : ', mutant)
298                     self.tb.mutate(mutant, self.tb)
299                     # print('after: ', mutant)
300                     del mutant.fitness.values
301                 else:
302                     print('No mutation')
303
304
305             # Compute fitnesses of new individuals with invalid fitness values
306             invalid_inds = [ind for ind in offspring if not ind.fitness.valid]
307             self.evaluate_population(invalid_inds, generation)
308
309             population = offspring
310             # [self.logfile.add_individual(individual, generation) for individual in population]
311             # End of looping over generations
312
313 if __name__ == '__main__':
314     parameter_options = {
315         'num_neurons': [num for num in range(1, 50)],
316         'activation': ['linear', 'tanh', 'relu', 'elu', 'selu', 'exponential', 'sigmoid', 'softplus'],
317         'optimizer': ['rmsprop', 'sgd', 'adam', 'adagrad', 'adadelta', 'adamax', 'nadam'],
318         'kernel_initializer': ['random_uniform', 'constant', 'zeros'],
319         'bias_initializer': ['random_uniform', 'constant', 'zeros'],
320     }
321
322     GA = GeneticAlgorithm(parameter_options)
323
324     tb = GA.tb
325
326     pop = tb.population(10)
327
328
329     import random
330     for ind in pop:
331         ind.fitness.values = random.random(),
332         print(ind, ind.fitness.values[0])
```

```
333
334     gen = 1
335     offspring = [tb.clone(individual) for individual in pop]
336     print(offspring)
337
338     print('Selection:....')
339     elite = tools.selBest(offspring, int(1))  # Select elite for current gene pool
340     # elite_nextgen = [self.tb.clone(individual) for individual in elite]     # Clone elite for next
341     tournament_selection = tb.select(offspring, int(9))
342     offspring = elite + tournament_selection  # Offspring for mutation and crossover
343
344     offspring = [tb.clone(individual) for individual in offspring]
345
346     for i, m in zip(pop, offspring):
347         print(i, '--', m)
348
349     selected_offspring = [tb.clone(ind) for ind in offspring]
350
351
352     print('Crossover:...')
353     for child1, child2 in zip(offspring[::2], offspring[1::2]):
354         if child1 != child2:  # Avoid crossover between identical individuals (unnecessary computing
355             if random.random() < 1:
356                 tb.crossover(child1, child2)
357                 del child1.fitness.values
358                 del child2.fitness.values
359
360
361     print('Mutation:...')
362     for mutant in offspring:
363         if random.random() < 0.9:
364             tb.mutate(mutant, tb)
365             del mutant.fitness.values
366
367     for i,m in zip(selected_offspring, offspring):
368         print(i, '--', m, m.fitness.values)
```

D

APDL classes

Script for creating the generating the datasets by running FE analyses is ANSYS

```
import os
import pandas as pd
import pickle
from data import Folder, File, Data, PrepareTrainData
import doepy
from doepy import build



def initialize_projects_from_file(excel_path, destination_folderpath):
    excelframe = pd.read_excel(excel_path, comment='#')
    projects = []
    for excel_row in excelframe.itertuples():
        if excel_row.include == 'yes':
            project = Project(parent_directory=destination_folderpath,
                              project_number=excel_row.project_number,
                              keyword=excel_row.keyword,
                              template_filename=excel_row.template_filename,
                              template_geometry_filename=excel_row.template_geometry,
                              samples=excel_row.samples)
            project.read_parameters(excel_row)
            projects.append(project)
    return projects


def create_designofexperiments(project_samples, parameters_free, parameters_fixed, parameter_constraints):
    DOE_list = []
    for n_samples in project_samples:
        DOE = DesignOfExperiments(n_samples, parameters_free, parameters_fixed, parameter_constraints)

        # if not os.path.exists(os.path.join(project.folder.path, DOE.name)):
        DOE.folder = Folder(project.folder.path, DOE.name)
        DOE.generate(n_goal=n_samples, max_iterations=50)
        DOE.create_designpoints()

        DOE_list.append(DOE)
    return DOE_list



class Project:
    available_parameters = ['SPH', 'SPW', 'SPT', 'NOTS', 'TSWH', 'TSWT', 'TSFW', 'TSFT',
                            'CWH', 'CWT', 'CFW', 'CFT', 'Q0', 'Q1', 'Syy', 'Dy', 'Szz', 'AMP',
                            'MSHSIZE', 'LONGSUP', 'EQVIMP', 'MATNONL']

    #
    # parameter_constraints = [lambda m: m['SPH'] / (m['NOTS'] + 1) > 240,
    #                          lambda m: m['SPH'] / (m['NOTS'] + 1) < 800,
```

```
49      #                           lambda m: m['SPH'] / (m['NOTS'] + 1) - m['TSFW'] > 150,
50      #                           lambda m: m['TSFT'] > m['TSWT'],
51      #                           lambda m: m['TSWH'] < m['CWH'],
52      #                           lambda m: m['Q0'] > m['Q1'],
53      #                           lambda m: m['Szz'] < 0.5*m['Syy']
54      #                           ]
55
56      parameter_constraints = []
57
58
59      def __init__(self, parent_directory, project_number, keyword, template_filename, template_geomet
60          self.parent_directory = parent_directory
61          self.number = project_number
62          self.keyword = keyword
63          self.template_filename = template_filename
64          self.template_geometry_filename = template_geometry_filename
65          self.samples = [int(num) for num in samples.strip('()').split(',')]
66          self.data_mod = data_mod
67
68          self.index_mech = int(self.number.split('-')[0])
69          self.name = 'project' + str(self.number)
70          self.folder = Folder(parent_directory, self.name)
71
72          self.DOE_list = []
73          self.parameters_free = {}
74          self.parameters_fixed = {}
75
76      def read_parameters(self, excel_row):
77          for parameter_name in self.available_parameters:
78              # Get the raw cell values from a single row in the excel file
79              cellvalue = getattr(excel_row, parameter_name)
80              value_list = [val for val in cellvalue.strip('()').split(',')]
81
82              # Convert the values to a floating point number if possible. Otherwise string
83              for i, val in enumerate(value_list):
84                  try:
85                      value_list[i] = float(val)
86                  except ValueError:
87                      value_list[i] = str(val)
88
89              # Add parameter ranges to the respective dictionaries
90              if len(value_list) == 1:
91                  self.parameters_fixed[parameter_name] = value_list
92              elif len(value_list) == 2:
93                  self.parameters_free[parameter_name] = value_list
94
95      def extract_outputparameters_from_results(self, doe):
96          self.outputparameters = list(doe.resultsframe.drop(
97              labels=list(self.parameters_free.keys()) + list(self.parameters_fixed.keys()), axis=1))
98
99      def create_infofile(self, database_results):
100         self.infofilename = 'info_{}'.format(self.name)
101         self.infofile_folder = Folder(database_results.path, self.name)
102         self.infofile = File(self.infofile_folder.path, self.infofilename, '.csv')
103
104         info = {
105             'project_num': [self.number],
106             'project_keyword': [self.keyword],
107             'num_free_parameters': [len(self.parameters_free)],
108             'free_parameters': [list(self.parameters_free.keys())],
109             'data_mod': self.data_mod
110         }
111         self.info_df = pd.DataFrame(info)
112         self.info_df.to_csv(self.infofile.path)
113
114     def create_picklefile(self, database_results):
115         self.pickle_filename = 'info_{}'.format(self.name)
116         self.picklefile_folder = Folder(database_results.path, self.name)
117         self.picklefile = File(self.picklefile_folder.path, self.pickle_filename, '.p')
118         database_results.project_pickles.append(self.picklefile)
119
```

```
        self.pickle_filename = f'projectobject_{self.name}'
        with open(self.picklefile.path, 'wb') as pf:
            pickle.dump(self, pf)


class DesignOfExperiments:
    # Remove project
    # Add conditions, parameters free, parameters fixed
    # Create resultsfile method with one folder argument. but call the method twice instead of copying
    def __init__(self, n_samples, parameters_free, parameters_fixed, parameter_constraints):
        self.n_samples = n_samples
        self.parameters_free = parameters_free
        self.parameters_fixed = parameters_fixed
        self.parameter_constraints = parameter_constraints

        self.name = 'doe-{}dp'.format(self.n_samples)
        self.designpoints = []


    def generate(self, n_goal, max_iterations):
        n = n_goal
        diff = n
        i = 0
        while diff != 0 and i < max_iterations:
            i+=1
            self.random_sample(n)
            self.filter()
            n_feasible = len(self.matrix)
            ratio_feasible = n_feasible / n
            diff = n_goal - n_feasible
            if i <= 10:
                n +=int(diff / ratio_feasible)
            else:
                n += diff

    def modify_discrete(self, parameter, range, action):
        """Three actions can be performed with this function
        1)  Modify: Modifies the values in parameters_free attribute of project class with a specified ran
            in order to accommodate for correct sampling of discrete variables. The modification action en
            the probability of selecting the extreme values of the discrete variable is equal to the
            prob. of selecting other values in the value options
        2)  Round: The free discrete variable is sampled as a real valued number. The round action rounds
            sampled value to the nearest integer
        3)  Restore: Restores the project attribute parameters free back to the original form;
            without the ranges added to the extreme values of the discrete variables
        """
        original = self.parameters_free
        key = parameter
        if action is 'modify':
            # Add the specified range to the maximum value of discrete variable and subtract it from the m
            if key in self.parameters_free:
                self.parameters_free[key][0] -= range
                self.parameters_free[key][0] += range

        elif action is 'round':
            # Round off values back to integers
            self.matrix[key] = self.matrix[key].apply(round)

        elif action is 'restore':
            # Set project attribute parameters_free back to original
            self.parameters_free = original

    def random_sample(self, n_samples):
        self.modify_discrete(parameter='NOTS', range=0.4999, action='modify')

        # Draw LHS samples from free parameters
        self.matrix = doepy.build.space_filling_lhs(
            self.parameters_free,
            n_samples
        )
```

```
191         # Add fixed parameters to the DOE
192         for key, val in self.parameters_fixed.items():
193             self.matrix[key] = val[0]
194 
195         # For discrete parameters, round off values to integers
196         self.modify_discrete(parameter='NOTS', range=0.499, action='round')
197         self.modify_discrete(parameter='NOTS', range=0.499, action='restore')
198 
199     def filter(self):
200         if self.parameter_constraints is not None:
201             for condition in self.parameter_constraints:
202                 self.matrix = self.matrix[condition(self.matrix)]
203             self.matrix.reset_index(drop=True, inplace=True)
204 
205     def create_designpoints(self):
206         # Create instances of designpoints in DOE and add parameter values
207         for i, row in self.matrix.iterrows():
208             dp = DesignPoint(self, i)
209             dp.inputparameters = self.matrix.iloc[i,:].to_dict()
210             self.designpoints.append(dp)
211 
212     def create_resultsfile(self, destination_folderpath):
213         self.resultsfilename = 'designpoints_{}'.format(self.name)
214         self.resultsfile = File(destination_folderpath, self.resultsfilename, '.csv')
215 
216         self.resultsframe = pd.DataFrame()
217         for designpoint in self.designpoints:
218             print(designpoint.name)
219             newrow = pd.read_csv(designpoint.parametervaluesfile.path, header=None).iloc[:, 1]
220             self.resultsframe = self.resultsframe.append(newrow)
221             print(newrow)
222 
223         columns = list(pd.read_csv(self.designpoints[0].parametervaluesfile.path, header=None).iloc[
224         print(columns)
225         self.resultsframe.columns = columns
226         self.resultsframe.reset_index(drop=True, inplace=True)
227         self.resultsframe.to_csv(self.resultsfile.path)
228 
229     def read_resultsfile(self, resultsfilepath):
230         self.resultsframe = pd.read_csv(resultsfilepath, index_col=0)
231 
232 
233     def create_resultsfile_directfolder(self, destination_folderpath):
234         self.resultsfilename = 'designpoints_{}'.format(self.name)
235         self.resultsfile = File(destination_folderpath, self.resultsfilename, '.csv')
236 
237         self.resultsframe = pd.DataFrame()
238         i=0
239         for dp_foldername in os.listdir(self.folder.path):
240             dp_folderpath = os.path.join(self.folder.path, dp_foldername)
241             for filename in os.listdir(dp_folderpath):
242                 if 'parametervalues' in filename:
243                     parametervaluespath = os.path.join(dp_folderpath, filename)
244 
245 
246 
247 
248                     newrow = pd.read_csv(parametervaluespath, header=None).iloc[:,1]
249                     self.resultsframe = self.resultsframe.append(newrow)
250                     i+=1
251 
252         columns = list(pd.read_csv(parametervaluespath, header=None).iloc[:, 0])
253         self.resultsframe.columns = columns
254         self.resultsframe.reset_index(drop=True, inplace=True)
255         self.resultsframe.to_csv(self.resultsfile.path)
256 
257 
258 class DesignPoint:
259     def __init__(self, DOE, number=0):
260         self.DOE = DOE
261         self.number = number
```

```
262
263         self.name = 'dp{}'.format(str(number))
264         self.folder = Folder(self.DOE.folder.path, self.name)
265
266         #self.project = self.DOE.project
267         #self.DOE.designpoints.append(self)
268
269         # Initiate resultsfile for use later
270         self.parametervaluesfile = File(self.folder.path, 'parametervalues', '.csv')
271
272     def create_inputfile(self, project, templates_folderpath):
273         main_template_path = os.path.join(templates_folderpath, project.template_filename + '.txt')
274         geometry_template_path = os.path.join(templates_folderpath, project.template_geometry_filename +
275
276         self.inputfilename = 'inputfile_{}'.format(self.name)
277         self.inputfile = InputFile(self.folder.path, self.inputfilename, '.inp')
278         self.inputfile.read_templates(main_template_path, geometry_template_path)
279         self.inputfile.add_doc_settings(project, self)
280         self.inputfile.add_parameters(self)
281         self.inputfile.add_geometry()
282         #self.choose_method_nonlinear(self.project)
283         # self.inputfile.add_imperfections(templates_folderpath)  # In case of eqv imperfection
284         self.inputfile.add_upgeomcommand(self)           # In case of eigenvalue analysis
285         self.inputfile.add_writeinputparameters(project)
286         self.inputfile.add_delete_commands(self.name,
287                                            extensions=['rst', 'full', 'esav', 'R001', 'RDB', 'MODE', 'LDHI
288                                                        'db', 'mntr', 'stat'])
289         self.inputfile.write()
290
291         #self.DOE.project.database.inputfiles.append(self.inputfile)
292         self.create_outputfile()
293
294     def choose_method_nonlinear(self, project):
295         if 'eqv_imp' in project.keyword:
296             self.inputfile.add_imperfections(project)
297         elif 'upgeom' in project.keyword:
298                 self.inputfile.add_upgeomcommand(self)
299         else:
300             print('Invalid method')
301
302     def create_outputfile(self):
303         self.outputfilename = 'outputfile_{}'.format(self.name)
304         self.outputfile = File(self.folder.path, self.outputfilename, '.out')
305
306
307 class InputFile(File):
308
309     def __init__(self, directory, name, extension):
310         super().__init__(directory, name, extension)
311
312     def read_templates(self, main_template_path, geometry_template_path):
313         with open(main_template_path, 'r') as f:
314             self.content = f.read()
315         with open(geometry_template_path, 'r') as f:
316             self.geometry_template = f.read()
317
318     def add_doc_settings(self, project, designpoint):
319         self.filename = '{}_{}'.format(project.name, designpoint.name)
320         self.doc_settings_string = '/FILNAME, {} \n/CWD, \'{}\' \n/TITLE, {} \n'.format(
321             designpoint.name, designpoint.folder.path, project.name)
322         self.content = self.content.replace('!input_document_settings', self.doc_settings_string)
323
324     def add_parameters(self, designpoint):
325         self.parameter_strings = []
326         for name, value in designpoint.inputparameters.items():
327             self.parameter_strings.append('{} = {}  \n'.format(name, str(value)))
328         self.content = self.content.replace('!input_parameters', ''.join(self.parameter_strings))
329
330     # def add_imp(self, designpoint):
331     #     if designpoint.inputparameters['EQVIMP'] - 0.0 <= 0.01:
332     #         self.add_upgeomcommand(designpoint)
```

```
333         #     elif designpoint.inputparameters['EQVIMP'] - 1.0 <= 0.01:
334         #         pass
335         #     else:
336         #         pass
337 
338 
339 
340     def add_geometry(self):
341         self.content = self.content.replace('!input_geometry_codes', self.geometry_template)
342 
343     def add_imperfections(self, templates_folderpath):
344         with open(os.path.join(templates_folderpath, 'template_IMP2.txt'), 'r') as f:
345             imperfection_string = f.read()
346         #
347         # delete_string = '\nEWRITE,elementsfile,elem\nALLSEL\nACLEAR,all\nADELE,all\n\n'
348         # redraw_string = '! - Redraw nodes with imperfections\n*do,i,1,ntot\nnodenum = nodes(i,1)\n
349         #                 'x_new = nodes(i,2) + DISPL(i,2)\ny_new = nodes(i,3) + DISPL(i,3)\n' \
350         #                 'z_new = nodes(i,4) + DISPL(i,4)\nN, nodenum, x_new, y_new, z_new\n*enddo\
351         #                 '! Read elements file\nEREAD,elementsfile,elem'
352 
353         self.content = self.content.replace('!input_imperfections', imperfection_string)
354 
355     def add_upgeomcommand(self,designpoint, scale=1):
356 
357         linbuc_string = 'ALLSEL\n/SOL\nANTYPE, static\npstres, on\neqslv, sparse\nSOLVE\n\nFINI\n\nn
358                 '/SOL\nantype,buckle\nbucopt,lanb,n_modes\nmxpand,n_modes,,,yes\nSOLVE\n\nFINI\n\n/
359         upgeom_string = f'/PREP7\nupgeom,AMP,1,1,{designpoint.name},rst\n\n'
360 
361         self.content = self.content.replace('!input_upgeom_command', upgeom_string)
362 
363     def add_writeinputparameters(self, project):
364         write_strings = []
365         for key in project.parameters_free:
366             write_strings.append('*VWRITE, \'{},\', {} \n(A, F)\n'.format(key, key))
367         for key in project.parameters_fixed:
368             write_strings.append('*VWRITE, \'{},\', {} \n(A, F)\n'.format(key, key))
369         self.content = self.content.replace('!input_writeinputparameters', ''.join(write_strings))
370 
371     def add_delete_commands(self, filename, extensions):
372         self.delete_commands = []
373         for ext in extensions:
374             self.delete_commands.append('/DELETE, {}, {},, \n'.format(filename, ext))
375         self.content = self.content.replace('!input_delete_commands', ''.join(self.delete_commands))
376 
377     def add_inputline(self, designpoint):
378         newline = '/INPUT, \'{}\', \'inp\', \'{}\', , 0 \n'.format(designpoint.inputfilename,
379                                                                   designpoint.folder.path)
380         self.content.append(newline)
381 
382 
383 class BatchFile(File):
384     def __init__(self, directory, name, extension):
385         super().__init__(directory, name, extension)
386         self.content = []
387 
388     def add_settings(self, ansyspath, license, num_cores, memory):
389         self.ansyspath = ansyspath
390         self.license = license
391         self.num_cores = num_cores
392         self.memory = memory
393 
394     def add_inputfile(self, working_directory, inputfile):
395         newstring = ("\"{}\" -p \"{}\" -np {} -m {}"
396                      "  -dir \"{}\" -j \"{}\"    -s noread"
397                      " -b -i \"{}\" -o \"{}\" \n").format(self.ansyspath, self.license, self.num_cores,
398                                                           working_directory,
399                                                           inputfile.name,
400                                                           inputfile.path,
401                                                          os.path.join(self.directory, 'main_outputfile.ou
402         self.content.append(newstring)
403 
```

```
404
405
406
407
408
409
410 main_database = Folder(directory=r'C:\Users\Thomas\Documents\Master Thesis', name='main_database_final')
411 database_apdl_files = Folder(main_database.path, 'database_apdl_files')
412 database_apdl_results = Folder(main_database.path, 'database_apdl_results')
413 database_traindata = Folder(main_database.path, 'database_traindata')
414 main_apdl_inputfolder = Folder(database_apdl_files.path, name='main_apdl_input')
415 apdl_templates = Folder(os.getcwd(), name='apdl_templates')
416
417 batchfile = BatchFile(main_apdl_inputfolder.path, 'batchfile', '.bat')
418 batchfile.add_settings(ansyspath=r'C:\Program Files\ANSYS Inc\v192\ansys\bin\winx64\MAPDL.exe',
419                        license='aa_t_a', num_cores=8, memory='2056')
420
421 # Create project objects
422 projects = initialize_projects_from_file(excel_path='project_definition_finalann.xlsx',
423                                          destination_folderpath=database_apdl_files.path)
424
425
426
427
428
429 if __name__ == "__main__":
430     # ============================================================================
431     #   Create DOE's, input files and let ANSYS run the analyses
432     # ============================================================================
433
434     # Generate a DOE's for each project and write individual inputfiles
435     print('Create DOEs')
436     for project in projects:
437         project.DOE_list = create_designofexperiments(project.samples,
438                                                       project.parameters_free,
439                                                       project.parameters_fixed,
440                                                       project.parameter_constraints)
441         for doe in project.DOE_list:
442             for designpoint in doe.designpoints:
443                 designpoint.create_inputfile(project, apdl_templates.path)
444                 batchfile.add_inputfile(working_directory=designpoint.folder.path,
445                                         inputfile=designpoint.inputfile)
446
447     print('Write batchfiles')
448     batchfile.write()
449
450
451
452     # # # ============================================================================
453     # # #   Export the results and create training datasets
454     # # # ============================================================================
455
456     # Export the raw results from the ANSYS analyses as csv file
457     print('Export raw results')
458     for project in projects:
459         copyfolder = Folder(database_apdl_results.path, project.name)
460         project.resultsfiles_folder = copyfolder
461         for doe in project.DOE_list:
462             # doe.create_resultsfile(destination_folderpath=project.folder.path)# Choose this if to use di
463             # doe.create_resultsfile(destination_folderpath=copyfolder.path)
464             doe.create_resultsfile_directfolder(destination_folderpath=copyfolder.path) # Choose this to r
465             project.extract_outputparameters_from_results(doe) # Add list of output parameters to project
466
467
468     # Modify the raw ANSYS results to clean training data files
469     # TODO: Make Data class and PrepareTraindata class more convenient and logical. Maybe combine to one c
470     for project in projects:
471         project.traindata_files = []
472
473         for doe in project.DOE_list:
474             traindata = PrepareTrainData(doe, project)
```

```
475                 traindata.standard_modification(traindata.selected_output[project.number])
476                 traindata.export(database_traindata)
477                 project.traindata_files.append(traindata.traindatafile)
```

E

Data classes

Script containing classes used for convenient data handling

```
import os
import pandas as pd
from sklearn.model_selection import train_test_split
import sklearn.preprocessing as preprocessing



class Folder:
    def __init__(self, directory, name):
        self.directory = directory
        self.name = name
        self.path = os.path.join(self.directory, self.name)

        self.subfolders = []
        self.files = []


        if not os.path.exists(self.path):
            os.makedirs(self.path)

    def add_subfolder(self, folder):
        self.subfolders.append(folder)


class File:
    def __init__(self, directory, name, extension):
        self.directory = directory
        self.name = name
        self.extension = extension
        self.path = os.path.join(self.directory, self.name + self.extension)
        self.content = ''

    def write(self):
        with open(self.path, 'w') as f:
            if type(self.content) is str:
                f.write(self.content)
            elif type(self.content) is list:
                self.content = ''.join(self.content)
                f.write(self.content)
            else:
                print('Invalid file content. Not able to write file.')


class PrepareTrainData:

    selected_output = {
        '0-0-0-0': 'ux_max',
        '1-1-1-1': 'ux_max',
```

```
        '1-2-1-1': 'ux_max',
        '1-3-1-1': 'ux_max',
        '1-4-1-1': 'ux_max',
        '1-5-1-1': 'ux_max',
        '2-1-1-1': 'L1',
        '2-2-1-1': 'L1',
        '2-3-1-1': 'L1',
        '2-4-1-1': 'L1',
        '2-5-1-1': 'L1',
        '3-1-2-1': 'SIGMXALL',
        '3-1-2-2': 'UXMXALL',
        '3-1-2-3': 'SIGMXPLM',
        '3-1-2-4': 'SIGMXSTM',
        '3-1-2-5': 'L1',
        '4-0-1-1': 'SIGMXAL',
        '4-0-1-2': 'FY',
        '4-0-2-1': 'SIGMXAL',
        '4-1-1-1': 'SGMXPL',
        '4-1-1-2': 'FY',
        '4-2-1-1': 'SGMXPL',
        '4-2-1-2': 'FY',
        '5-1-1-1': 'EPMXPL',
        '5-1-1-2': 'FY',
        '5-2-1-1': 'EPMXPL',
        '5-2-1-2': 'FY',
        '6-1-1-1': 'SGMXPL',
        '6-1-1-2': 'SGMXST',
        '6-1-1-3': 'FY',
        '6-2-1-1': 'SGMXPL',
        '6-2-1-2': 'SGMXST',
        '6-2-1-3': 'FY'
    }

    def __init__(self, DOE, project):
        self.DOE = DOE
        self.doe_name = DOE.name
        self.project = project
        self.resultsframe = self.DOE.resultsframe
        self.newframe = self.DOE.resultsframe.copy()

    def keep_free_parameters(self):
        fixed_parameters = list(self.project.parameters_fixed.keys())
        self.newframe.drop(labels=fixed_parameters, axis=1, inplace=True)

    def select_output_parameters(self, selected_output_parameters):
        output_parameters_to_drop = []
        for parameter in self.project.outputparameters:
            if parameter not in selected_output_parameters:
                output_parameters_to_drop.append(parameter)
        self.newframe.drop(labels=output_parameters_to_drop, axis=1, inplace=True)

    def set_new_project_number(self, modnum):
        self.mod_project_name = self.project.name[:-1] + str(modnum)

    def export(self, database_traindata):
        self.traindatafilename = 'traindata_{}_{}'.format(self.project.name, self.doe_name)
        self.traindata_projectfolder = Folder(database_traindata.path, self.project.name)
        self.traindatafile = File(self.traindata_projectfolder.path, self.traindatafilename, '.csv')

        self.newframe.to_csv(self.traindatafile.path)

    def standard_modification(self, selected_output_parameters):
        self.keep_free_parameters()
        self.select_output_parameters([selected_output_parameters])
        self.set_new_project_number(1)


class Data:
    def __init__(self):
        pass

```

```
120     def insert_data(self, X, Y):
121         self.X = X
122         self.Y = Y
123 
124     def read_datafile(self, data_filepath):
125         self.dataframe = pd.read_csv(data_filepath, comment='#', index_col=0)
126 
127     def set_output_dim(self, output_dim):
128         self.output_dim = output_dim
129         self.X = self.dataframe.to_numpy()[:, :-output_dim]
130         self.Y = self.dataframe.to_numpy()[:, -output_dim]
131 
132     def split(self, test_size=0.2, seed=0):
133         X_train, X_test, Y_train, Y_test = train_test_split(self.X, self.Y, test_size=test_size, random_st
134 
135         self.X_train = X_train
136         self.X_test = X_test
137         self.Y_train = Y_train
138         self.Y_test = Y_test
139 
140     def scale(self, scaler='minmax'):
141         scaler_options = {
142             'minmax': preprocessing.MinMaxScaler(feature_range=(0,1)),
143             'standard': preprocessing.StandardScaler(),
144             'robust': preprocessing.RobustScaler()
145         }
146         scaler = scaler_options[scaler]
147         scaler.fit(self.X)
148         self.X_train = scaler.transform(self.X_train)
149         self.X_test = scaler.transform(self.X_test)
150         self.X = scaler.transform(self.X)
151         return scaler
```

F

Summary classes

Script written for summarizing results of all predictions

```
import pandas as pd
import os
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats

import sklearn.metrics.regression as metrics



from data import Folder, File




class Summary:
    def __init__(self):
        self.predictions = []
        self.dataframe = pd.DataFrame()

    def addrow(self, prediction):
        self.predictions.append(prediction)
        newrow = pd.DataFrame({
            'Project number': [prediction.project_num],
            'Mechanical problem': [prediction.index_mech_problem],
            'Index num vars': [prediction.index_num_vars],
            'Index value range': [prediction.index_var_range],
            'Index data modification': [prediction.index_data_mod],
            'Num. variables': [prediction.num_vars],
            'Variable names': [prediction.var_names],
            'Fitting type': [prediction.fitting_type],
            'Num. Design points': [prediction.num_dp],
            'Num. Validation points': [prediction.num_vp],
            'Skewness output': [prediction.skewness_train],
            'Kurtosis output': [prediction.kurtosis_train],
            'Std. output': [''],
            'Transformation': ['None'],
            'Scaling': ['None'],
            'Filtering': ['None'],
            'MSE train': [prediction.MSE_train],
            'MSE validation': [prediction.MSE_test],
            'RMSE train': [prediction.RMSE_train],
            'RMSE validation': [prediction.RMSE_test],
            'R2 train': [prediction.R2_train],
            'R2 validation': [prediction.R2_test],
            'MAE train': [prediction.MAE_train],
            'MAE validation': [prediction.MAE_test]
        })
```

```
49          self.dataframe = self.dataframe.append(newrow, ignore_index=True)
50
51      def export(self, directory):
52          summaryfile_csv = File(directory, 'summary', '.csv')
53          summaryfile_excel = File(directory, 'summary', '.xlsx')
54
55          self.dataframe.to_csv(summaryfile_csv.path)
56          self.dataframe.to_excel(summaryfile_excel.path)
57
58  class Prediction:
59      def __init__(self):
60          pass
61
62      def read_scatterdata(self, prediction_filepath):
63          self.prediction_filepath = prediction_filepath
64          self.Y_train_true = pd.read_csv(self.prediction_filepath, comment='#')['Y_train_true'].to_nu
65          self.Y_train_predicted = pd.read_csv(self.prediction_filepath, comment='#')['Y_train_predict
66          self.Y_test_true = pd.read_csv(self.prediction_filepath, comment='#')['Y_test_true'].dropna(
67          self.Y_test_predicted = pd.read_csv(self.prediction_filepath, comment='#')['Y_test_predicted
68
69      def compute_errors(self):
70          self.MSE_train = metrics.mean_squared_error(self.Y_train_true, self.Y_train_predicted)
71          self.MSE_test = metrics.mean_squared_error(self.Y_test_true, self.Y_test_predicted)
72          self.RMSE_train = np.sqrt(self.MSE_train)
73          self.RMSE_test = np.sqrt(self.MSE_test)
74          self.MAE_train = metrics.mean_absolute_error(self.Y_train_true, self.Y_train_predicted)
75          self.MAE_test = metrics.mean_absolute_error(self.Y_test_true, self.Y_test_predicted)
76          self.R2_train = metrics.r2_score(self.Y_train_true, self.Y_train_predicted)
77          self.R2_test = metrics.r2_score(self.Y_test_true, self.Y_test_predicted)
78
79      def compute_statistics(self):
80          self.skewness_train = stats.skew(self.Y_train_true)
81          self.skewness_test = stats.skew(self.Y_test_true)
82          self.kurtosis_train = stats.kurtosis(self.Y_train_true)
83          self.kurtosis_test = stats.kurtosis(self.Y_test_true)
84          self.std_train = np.std(self.Y_train_true, axis=0)
85          self.std_train = np.std(self.Y_train_true)
86
87      def add_projectinfo(self, project, prediction_filename):
88          self.project_num = project.number
89          self.index_mech_problem = self.project_num.split('-')[0]
90          self.index_num_vars = self.project_num.split('-')[1]
91          self.index_var_range = self.project_num.split('-')[2]
92          self.index_data_mod = self.project_num.split('-')[3]
93          self.description = project.keyword
94          self.num_vars = len(project.parameters_free)
95          self.var_names = list(project.parameters_free.keys())
96          self.data_mod = project.data_mod
97
98          self.fitting_type = prediction_filename.split('_')[0].split('-')[1]
99
100         self.num_dp = len(self.Y_train_true)
101         self.num_vp = len(self.Y_test_true)
102
103     def plot_scatter(self, unit):
104         plt.close('all')
105         fig = plt.figure()
106         ax = fig.add_subplot(111)
107         ax.set(title='Scatterplot true vs. predicted\n{}, n={}'.format(self.fitting_type, self.num_d
108                xlabel='True value {}'.format(unit), ylabel='Predicted value {}'.format(unit))
109         ax.plot(self.Y_train_true, self.Y_train_true, color='g')
110         ax.scatter(self.Y_train_true, self.Y_train_predicted, marker='o')
111         ax.scatter(self.Y_test_true, self.Y_test_predicted, marker='x')
112         plt.show()
113
114     def plot_output_boxplot(self):
115         plt.close('all')
116         fig = plt.figure()
117         ax = fig.add_subplot(111)
118         ax.boxplot(self.Y_train_true)
119
```

```
120
121
122
123
124
125
126
127
128 from apdl import initialize_projects_from_file
129
130
131 main_database = Folder(directory=r'C:\Users\Thomas\Documents\Master Thesis', name='main_database_final')
132 database_apdl_files = Folder(main_database.path, 'database_apdl_files')
133 projects = initialize_projects_from_file(excel_path='project_definition_finalann.xlsx',
134                                          destination_folderpath=database_apdl_files.path)
135
136
137
138 database_optimization = Folder(main_database.path, 'database_optimization')
139
140
141
142 project_nums = [
143     '4-1-1-1', '4-1-1-2', '4-2-1-1', '4-2-1-2',
144     '5-1-1-1', '5-1-1-2', '5-2-1-1', '5-2-1-2',
145 ]
146
147
148
149 from files import *
150
151 if __name__ == "__main__":
152     summary = Summary()
153     for project in projects:
154         project.prediction_files = []
155         print(project.name)
156         for filepath in get_files(os.path.join(database_optimization.path, project.name), '.csv'):
157             if 'prediction' in filepath:
158                 print(filepath)
159                 project.prediction_files.append(filepath)
160         for prediction_filepath in project.prediction_files:
161             prediction = Prediction()
162             prediction.read_scatterdata(prediction_filepath=prediction_filepath)
163             prediction.compute_errors()
164             prediction.compute_statistics()
165             name = prediction_filepath.split(os.path.join(database_optimization.path, project.name))[-1]
166             name = name.strip(r"\\")
167             prediction.add_projectinfo(project, prediction_filename=name)
168             summary.addrow(prediction)
169             summary.export(directory=database_optimization.path)
```

G

APDL template file non-linear buckling analysis stiffened plates

Parts of the commands are re-used from the own produced internship report [31].

```
1
2  !=================================================================
3  ! Start
4  !=================================================================
5
6  /CLEAR,NOSTART
7
8  ! --- Load document settings
9  !input_document_settings
10
11
12
13
14 !=================================================================
15 ! Input parameters
16 !=================================================================
17
18 ! ------------------------------------------------
19 ! Geometry
20 ! ------------------------------------------------
21
22 ! --- Load input parameters
23
24 !input_parameters
25
26
27
28
29
30 ! --- Fixed parameters
31 E=210000
32 rho=0.3
33 sigma_y=355
34 density=7850
35
36
37
38
39
40
41 ! --- Compute derived parameters
42
43 *IF,NOTS,GT,0,THEN
44         CTC_T = SPH/(NOTS+1)            ! CTC distance t stiffeners
45         FOTS = 1.0*CTC_T                ! First offset T stiffeners
```

```
46
47          *ELSEIF,NOTS,EQ,0,THEN
48                  CTC_T=SPH
49  *ENDIF
50
51
52  ! Edge displacements computed from stresses
53  Uyy = Syy*0.5*SPW/E
54  Uzz = Szz*0.5*SPH/E
55
56
57
58
59  ! --- Define pi
60  *afun,rad
61  pi=acos(-1)
62
63
64
65  ! --- Identification numbers necessary for making named selections
66  SPID = 1                          ! Skin Plate ID
67  TSWID = 2                         ! T stiffener web ID
68  TSFID = 3                         ! T stiffener flange ID
69  CWID = 4                          ! Column web ID
70  CFID = 5                          ! Column flange ID
71
72
73
74
75
76
77
78  !=================================================================
79  ! Preprocessor
80  !=================================================================
81  /PREP7
82
83  ! Add shell elements: Element Type - Add
84  ET,1,SHELL181   ! type 1: 4 node shell
85
86
87  ! Set to structural
88  /NOPR
89  KEYW,PR_SET,1
90  KEYW,PR_STRUC,1
91  KEYW,PR_THERM,0
92  KEYW,PR_FLUID,0
93  KEYW,PR_MULTI,0
94  /GO
95
96
97  ! Set material properties: Material Props - Material Models
98  MPTEMP,,,,,,,,
99  MPTEMP,1,0
100 MPDATA,EX,1,,E          ! Youngs modulus
101 MPDATA,PRXY,1,,rho  ! Poissons Ratio
102
103
104 ! Select element key options
105 ETCON,off       ! Let not automatically select applicable keyopts
106 KEYOPT,1,3,2
107 KEYOPT,1,8,2
108
109
110
111 !-----------------------------------------------------------------
112 ! Define element thickness and properties
113 !-----------------------------------------------------------------
114 !EXPLANATION SHELL/MESH ATTRIBUTES
115 ! sect,sectionID,elementType,,sectionName
116 ! secdata, Thickness,MaterialID,0,NumberofIntegrationPoints
```

```
! secoffset,MID
! seccontrol,0,0,0, mass/unit area, 1, 1, 1

sect,SPID,shell,,Skin Plate
secdata, SPT,1,0.0,3
secoffset,MID
seccontrol,,,,0 , , ,

*IF,NOTS,GT,0,THEN
         sect,TSWID,shell,,T-webs
         secdata, TSWT,1,0,3
         secoffset,MID
         seccontrol,,,,0 , , ,

         *IF,TSFW,GT,0,THEN
         sect,TSFID,shell,,T-flanges
         secdata, TSFT,1,0,3
         secoffset,MID
         seccontrol,,,,0 , , ,
         *ENDIF
*ENDIF


sect,CWID,shell,,Column-webs
secdata, CWT,1,0,3
secoffset,MID
seccontrol,,,,0 , , ,

sect,CFID,shell,,Column-flanges
secdata, CFT,1,0,3
secoffset,MID
seccontrol,,,,0 , , ,









!=====================================================================
! Create 3D model
!=====================================================================

!input_geometry_codes




!---------------------------------------------------------------------
! Assign elements to selected components and MESHING
!---------------------------------------------------------------------
AATT, MAT, REAL, ELEMENT TYPE, ESYS, SECTION NUMBER

CMSEL,S,SP                        ! Select component
AATT,       1, ,   1,       0,    SPID

*IF,NOTS,GT,0,THEN
         CMSEL,S,TSW
         AATT,       1, ,   1,       0,    TSWID

         *IF,TSFW,GT,0,THEN
         CMSEL,S,TSF
         AATT,       1, ,   1,       0,    TSFID
         *ENDIF
*ENDIF


*IF,geometry,EQ,3,THEN
         CMSEL,S,CW
         AATT,       1, ,   1,       0,    CWID

```

```
188         CMSEL,S,CF
189         AATT,       1, ,   1,       0,   CFID
190 *ENDIF
191
192
193
194
195 ! --- Meshing
196 ! - Set mesh options
197 MSHKEY,2                ! Use mapped meshing if possible; otherwise, use free meshing
198 MOPT,split,2    ! Quad splitting option for non-mapped meshing. If Value = 2 or WARN,
        quadrilateral
199                                 ! elements in violation of either shape error or
                                        warning limits are split into triangles.
200 MSHAPE,0,2d     ! Mesh with quadrilateral-shaped elements when Dimension = 2D
201
202
203 ! - Set mesh size and mesh areas
204 ESIZE,40
205 ALLSEL
206 AMESH,ALL
207
208
209 seltol,0.0005
210
211 ALLSEL
212
213 ! --- Node components
214 CMSEL,S, TSF
215 CMSEL,A, TSW
216 CMSEL,A, SP
217 NSLA, S, 1
218 NSEL, R, LOC, Y, 0
219 CM, RightEdge, Node
220 NSEL, S, LOC, Y, 0.5*SPW
221 CM, MidEdge, Node
222 NSEL, S, LOC, Z, 0
223 CM, BottomEdge, Node
224 NSEL, S, LOC, Z, SPH
225 CM, TopEdge, Node
226
227 CMSEL, S, RightEdge, Node
228 NSEL, R, LOC, X, 0
229 CM, RightEdgePlate, Node
230 CMSEL, S, MidEdge, Node
231 NSEL, R, LOC, X, 0
232 CM, MidEdgePlate, Node
233 CMSEL, S, BottomEdge, Node
234 NSEL, R, LOC, X, 0
235 CM, BottomEdgePlate, Node
236 CMSEL, S, TopEdge, Node
237 NSEL, R, LOC, X, 0
238 CM, TopEdgePlate, Node
239
240 CMSEL, S, RightEdge, Node
241 NSEL,U,LOC,X,0
242 CM,RightEdgeSection,Node
243 CMSEL, S, MidEdge, Node
244 NSEL,U,LOC,X,0
245 CM,MidEdgeSection,Node
246 CMSEL, S, BottomEdge, Node
247 NSEL,U,LOC,X,0
248 CM,BottomEdgeSection,Node
249 CMSEL, S, TopEdge, Node
250 NSEL,U,LOC,X,0
251 CM,TopEdgeSection,Node
252
253
254 ! - Create node components for reading displacements from eigenmodes in order to apply
        right imperfection
255 NSEL,S,LOC,X,0
```

```
CM,PlateNodes,Node

CMSEL,S,TSW,AREA
NSLA,S,1
NSEL,R,LOC,X,0
CM,BottomStiffenersNodes,Node

CMSEL,S,TSW,AREA
NSLA,S,1
NSEL,R,LOC,X,TSWH
CM,TopStiffenersNodes,Node




!=====================================
! Enter solution processor
!=====================================
/SOL

ALLSEL
! --- Add Boundary conditions

! - Fix horizontal edges in x direction
D, TopEdge,UX, 0
D, BottomEdge,UX, 0

! - Apply deformations
D, RightEdgePlate, UY, Uyy
D, RightEdgeSection, UY, Uyy

! Support column cross sections in y direction
ALLSEL
D, TopEdgeSection, UY, Uyy
D, BottomEdgeSection, UY, Uyy


! Apply Szz as a displacement
D,TopEdge,Uz,-Uzz
D,BottomEdge,Uz,Uzz


! Apply Szz as a stress onto the plate and column section
!CMSEL,S,BottomEdgePLate,Node
!SF,ALL,Pres,Szz*SPT
!CMSEL,S,BottomEdgeSection,Node
!SF,ALL,Pres,Szz*CFT

!CMSEL,S,TopEdgePLate,Node
!SF,ALL,Pres,Szz*SPT
!CMSEL,S,TopEdgeSection,Node
!SF,ALL,Pres,Szz*CFT




! Support in Z direction at a point halfway column web at flange
NSEL,S,LOC,X,CWH
NSEL,R,LOC,Y,0
NSEL,R,LOC,Z,0.5*SPH-25, 0.5*SPH+25
D,ALL,UZ,0



! - Apply symmetric boundary conditions
CMSEL,S,MidEdge,NODE
DSYMM,SYMM,Y
```

```
! --- Add surface loads
sfcum,pres,add

! ----- Load contribution number 1 -----!
Z0 = 0 $ P0 = Q0
Z1 = SPH $ P1 = Q1
delta_p = (P1 - P0)/(Z1 - Z0)
ESEL,S,SEC,,SPID
ESEL,R,CENT,Z,Z0,Z1
SFGRAD,PRES,0,Z,Z0,delta_p
SFE,ALL,1,PRES,0, Q0
!-----------------
/psf,pres,norm,3,1



! - When a nonlinear analysis is performed with an updated geometry following from an
    eigenvalue analysis,
!       the commands will be loaded here

ALLSEL
/SOL
ANTYPE, static
pstres, on
eqslv, sparse
SOLVE

FINI

n_modes=1

/SOL
antype,buckle
bucopt,lanb,n_modes
mxpand,n_modes,,,yes
SOLVE
FINI


/post1
!Retrieve eigenvalues and store in parameters
*DO,mode,1,n_modes
        SET,1,mode
        *GET,lambda_%mode%,FREQ
*ENDDO




!input_upgeom_command




ALLSEL
! Perform nonlinear analysis
/SOLU
ANTYPE, static
OUTRE, all, all
NLGEOM, on
AUTOTS,auto
NSUBST,1000,1000,10
SOLVE
```

```
397
398 !=====================================
399 ! Enter postprocessor
400 !=====================================
401
402
403 /POST1
404 set, last
405
406
407 ! Set view
408 /VIEW,1,1,1,-0.50
409 /ANGLE,1,180
410 /VUP,1,Z
411 /REPLOT
412
413
414
415
416 ! ------------------------------------ Store all results in one array
417 ALLSEL
418 *GET,numNd,NODE,0,COUNT
419 *DIM,selstat,ARRAY,numNd,1
420 *DIM,allresults,ARRAY,numNd,7
421 *VGET,allresults(1,1),NODE,1,LOC,X
422 *VGET,allresults(1,2),NODE,1,LOC,Y
423 *VGET,allresults(1,3),NODE,1,LOC,Z
424 *VGET,allresults(1,4),NODE,1,U,X
425 *VGET,allresults(1,5),NODE,1,U,Y
426 *VGET,allresults(1,6),NODE,1,U,Z
427 *VGET,allresults(1,7),node,1,S,EQV
428
429
430
431
432
433
434 ! ------------------------------------ Take min-max values of results
435
436 ! --- ALL
437 ALLSEL
438 *VSCFUN,SIGMXALL,max,allresults(1,7)
439 *VSCFUN,UXMXALL,max,allresults(1,4)
440
441
442
443 ! --- PLATE
444
445 CMSEL,s,PlateNodes,Node                                  ! Select plate
446 *VGET,selstat(1,1),NODE,1,nsel                  ! Fill selection status array
447 *VMASK,selstat(1,1)                                              ! Create masking vector
448 *VSCFUN,SIGMXPL,max,allresults(1,7)             ! Store max SEQV of plate
449 *VMASK,selstat(1,1)
450 *VSCFUN,UXMXPL,max,allresults(1,4)              ! Store max UX of plate
451
452
453 CMSEL,s,PlateNodes,Node                                  ! Select plate
454 NSEL,R,LOC,Y,0,0.1*SPW                                   ! Reselect 10% of plate width
    next to column
455 *VGET,selstat(1,1),NODE,1,nsel                  ! Fill selection status array
456 *VMASK,selstat(1,1)                                              ! Create masking vector
457 *VSCFUN,SIGMXPLE,max,allresults(1,7)            ! Store max SEQV of plate along column
458
459 CMSEL,s,PlateNodes,Node                                  ! Select plate
460 NSEL,R,LOC,Y,0.1*SPW, 0.5*SPW                   ! Reselect 40% of plate width
461 *VGET,selstat(1,1),NODE,1,nsel                  ! Fill selection status array
462 *VMASK,selstat(1,1)                                              ! Create masking vector
463 *VSCFUN,SIGMXPLM,max,allresults(1,7)
464
465
466 ! --- STIFFENERS
```

```
467
468 CMSEL,s,TSW,AREA
469 CMSEL,a,TSF,AREA
470 NSLA,s,1                                                    ! Select
        Stiffeners nodes
471 *VGET,selstat(1,1),NODE,1,nsel                  ! Fill selection status array
472 *VMASK,selstat(1,1)
473 *VSCFUN,SIGMXST,max,allresults(1,7)            ! Store max SEQV of stiffeners
474 *VMASK,selstat(1,1)
475 *VSCFUN,UXMXST,max,allresults(1,4)             ! Store max ux of stiffeners
476
477
478 CMSEL,s,TSW,AREA
479 CMSEL,a,TSF,AREA
480 NSLA,s,1
481 NSEL,R,LOC,Y,0,0.1*SPW
482 *VGET,selstat(1,1),NODE,1,nsel                        ! Fill selection status array
483 *VMASK,selstat(1,1)
484 *VSCFUN,SIGMXSTE,max,allresults(1,7)           ! Store max SEQV of stiffeners next to
        column
485
486 CMSEL,s,TSW,AREA
487 CMSEL,a,TSF,AREA
488 NSLA,s,1
489 NSEL,R,LOC,Y,0.1*SPW,0.5*SPW
490 *VGET,selstat(1,1),NODE,1,nsel                        ! Fill selection status array
491 *VMASK,selstat(1,1)
492 *VSCFUN,SIGMXSTM,max,allresults(1,7)           ! Store max SEQV of stiffeners
        midsection
493
494
495
496
497
498 ! --- Create and fill output file with input-output parameters
499 *cfopen,parametervalues,csv,,
500 !input_writeinputparameters
501 *VWRITE, 'L1,', lambda_1
502 (A,F)
503 *VWRITE,'UXMXALL,', UXMXALL
504 (A,F)
505 *VWRITE,'UXMXPL,', UXMXPL
506 (A,F)
507 *VWRITE,'UXMXST,', UXMXST
508 (A,F)
509 *VWRITE,'SIGMXALL,', SIGMXALL
510 (A,F)
511 *VWRITE,'SIGMXPL,', SIGMXPL
512 (A,F)
513 *VWRITE,'SIGMXPLM,', SIGMXPLM
514 (A,F)
515 *VWRITE,'SIGMXPLE,', SIGMXPLE
516 (A,F)
517 *VWRITE,'SIGMXST,', SIGMXST
518 (A,F)
519 *VWRITE,'SIGMXSTM,', SIGMXSTM
520 (A,F)
521 *VWRITE,'SIGMXSTE,', SIGMXSTE
522 (A,F)
523 *CFclose
524
525
526 FINI
527
528 ! --- Delete unneccesary files in order to save disk space
529 !input_delete_commands
```

APDL template file non-linear buckling analysis unstiffened plates

Parts of the commands are re-used from the own produced internship report [31].

```
1
2  !=================================================================
3  ! Start
4  !=================================================================
5
6
7
8  /CLEAR,NOSTART
9
10 ! --- Load document settings
11 !input_document_settings
12
13
14
15
16
17
18 !=================================================================
19 ! Input parameters
20 !=================================================================
21
22 ! ------------------------------------------------
23 ! Geometry
24 ! ------------------------------------------------
25
26 ! --- Load input parameters
27
28 !input_parameters
29
30
31
32
33
34 ! --- Fixed parameters
35 E=210000
36 rho=0.3
37 sigma_y=355
38 density=7850
39
40
41 ! Edge displacements computed from stresses
42 Uyy = Syy*0.5*SPW/E
43 Uzz = Szz*0.5*SPH/E
44
45
```

```
46
47
48
49
50
51 ! --- Define pi
52 *afun,rad
53 pi=acos(-1)
54
55
56
57 ! --- Identification numbers necessary for making named selections
58 SPID = 1                          ! Skin Plate ID
59
60
61
62
63
64
65
66 !=================================================================
67 ! Preprocessor
68 !=================================================================
69 /PREP7
70
71 ! Add shell elements: Element Type - Add
72 ET,1,SHELL181   ! type 1: 4 node shell
73
74
75 ! Set to structural
76 /NOPR
77 KEYW,PR_SET,1
78 KEYW,PR_STRUC,1
79 KEYW,PR_THERM,0
80 KEYW,PR_FLUID,0
81 KEYW,PR_MULTI,0
82 /GO
83
84
85 ! Set material properties: Material Props - Material Models
86 MPTEMP,,,,,,,,
87 MPTEMP,1,0
88 MPDATA,EX,1,,E           ! Youngs modulus
89 MPDATA,PRXY,1,,rho  ! Poissons Ratio
90
91
92 *IF,MATNONL,EQ,1,THEN
93         TB,BISO,1,1,2,
94         TBTEMP,0
95         TBDATA,,355,4.7619047,,,,
96 *ENDIF
97
98
99
100 ! Select element key options
101 ETCON,off       ! Let not automatically select applicable keyopts
102 KEYOPT,1,3,2
103 KEYOPT,1,8,2
104
105
106
107 !-----------------------------------------------------------------------
108 ! Define element thickness and properties
109 !-----------------------------------------------------------------------
110 !EXPLANATION SHELL/MESH ATTRIBUTES
111 ! sect,sectionID,elementType,,sectionName
112 ! secdata, Thickness,MaterialID,0,NumberofIntegrationPoints
113 ! secoffset,MID
114 ! seccontrol,0,0,0, mass/unit area, 1, 1, 1
115
116 sect,SPID,shell,,Skin Plate
```

```
117 secdata, SPT,1,0.0,3
118 secoffset,MID
119 seccontrol,,,,0 , , ,
120
121
122
123
124
125
126
127 !*****************************************************************
128 ! Create 3D model
129 !*****************************************************************
130
131 !input_geometry_codes
132
133
134
135
136 !-----------------------------------------------------------------
137 ! Assign elements to selected components and MESHING
138 !-----------------------------------------------------------------
139 AATT, MAT, REAL, ELEMENT TYPE, ESYS, SECTION NUMBER
140
141 CMSEL,S,SP                        ! Select component
142 AATT,       1, ,   1,       0,    SPID
143
144
145
146
147 ! --- Meshing
148 ! - Set mesh options
149 MSHKEY,2                 ! Use mapped meshing if possible; otherwise, use free meshing
150 MOPT,split,2    ! Quad splitting option for non-mapped meshing. If Value = 2 or WARN,
        quadrilateral
151                                 ! elements in violation of either shape error or
                                        warning limits are split into triangles.
152 MSHAPE,0,2d     ! Mesh with quadrilateral-shaped elements when Dimension = 2D
153
154
155 ! - Set mesh size and mesh areas
156 ESIZE,MSHSIZE
157 ALLSEL
158 AMESH,ALL
159
160
161 seltol,0.0005
162
163
164
165 ALLSEL
166
167
168
169
170 ALLSEL
171 !Create arrays to store node information,  create array for displacements for each node
        in case of applied imperfection.
172 *get,ntot,node,,count
173 *dim,nodes,array,ntot,4
174 *vget,nodes(1,1),node,0,nlist
175
176 *do,i,1,ntot
177 nodes(i,2) = nx(nodes(i,1))
178 nodes(i,3) = ny(nodes(i,1))
179 nodes(i,4) = nz(nodes(i,1))
180 *enddo
181
182
183 *dim,DISPL,array,ntot,4
184 *vget,nodes(1,1),node,0,nlist
```

```
185
186
187
188
189
190
191
192
193
194
195 ! ------------------------
196 ! BEGIN IMPERFECTION 2
197 ! ------------------------
198
199
200 ! Input parameters for imperfection
201 !userinput_ImperfectionParameters
202
203 A = SPW
204 B = SPH
205
206 wave_y = 0.5
207 wave_z = 0.5
208
209 ! Eurocode requirement for selecting amplitude of imperfection 2
210 ! The lowest value of (A/400, B/400) is to be selected as maximum displacement of the
      sine and stored as factor C
211 ! In this case A and B represent the total plate width and height respectively
212
213 *DIM,amp_opts,array,2,1
214 amp_opts(1,1) = A/400
215 amp_opts(2,1) = B/400
216 *VSCFUN,C,min,amp_opts
217
218 C = C*EQVIMP
219
220 *do,i,1,ntot
221         x = nodes(i,2)
222         y = nodes(i,3)
223         z = nodes(i,4)
224
225         ! - Skin plate modification
226         *IF,x,EQ,0,THEN
227                 wx = C*sin(2*wave_y*pi*y/A)*sin(2*wave_z*pi*z/B)
228                 wy = 0
229                 wz = 0
230
231                 DISPL(i,2) = DISPL(i,2) + wx
232                 DISPL(i,3) = DISPL(i,3) + wy
233                 DISPL(i,4) = DISPL(i,4) + wz
234         *ENDIF
235
236
237
238 *enddo
239 ! ------------------------
240 ! END IMPERFECTION 2
241 ! ------------------------
242
243
244
245
246
247
248
249
250
251
252
253
254
```

```
ALLSEL
EWRITE,elementsfile,elem
ALLSEL
ACLEAR,all
ADELE,all

! - Redraw nodes with imperfections
*do,i,1,ntot
nodenum = nodes(i,1)
x_new = nodes(i,2) + DISPL(i,2)
y_new = nodes(i,3) + DISPL(i,3)
z_new = nodes(i,4) + DISPL(i,4)
N, nodenum, x_new, y_new, z_new
*enddo

! Read elements file
EREAD,elementsfile,elem









! --- Node components
NSEL, S, LOC, Y, 0
CM, RightEdge, Node
NSEL, S, LOC, Y, 0.5*SPW
CM, MidEdge, Node
NSEL, S, LOC, Z, 0
CM, BottomEdge, Node
NSEL, S, LOC, Z, SPH
CM, TopEdge, Node









!=====================================
! Enter solution processor
!=====================================
/SOL

ALLSEL
! --- Add Boundary conditions

! - Fix horizontal edges in x direction
*IF,LONGSUP,EQ,1,THEN
        D, TopEdge,UX, 0
        D, BottomEdge,UX, 0
*ENDIF

D, RightEdge,UX, 0


! - Apply deformations
*IF,DY,NE,0,THEN        ! Case displacement applied drectly
        D, RightEdge, UY, DY
*ELSE                           ! Case displacement derived from applied stress
        D, RightEdge, UY, Uyy
*ENDIF


! Fix in z-direction
D,BottomEdge,Uz,0


! Apply Szz as a stress onto the plate and column section
```

```
!CMSEL,S,BottomEdgePLate,Node
!SF,ALL,Pres,Szz*SPT


!CMSEL,S,TopEdgePLate,Node
!SF,ALL,Pres,Szz*SPT




! - Apply symmetric boundary conditions
CMSEL,S,MidEdge,NODE
DSYMM,SYMM,Y







ALLSEL
! Perform nonlinear analysis
/SOLU
ANTYPE, static
OUTRES, all, all
NLGEOM, on
AUTOTS,auto
NSUBST,1000,1000,10
SOLVE









!=======================================
! Enter postprocessor
!=======================================


/POST1
set, last


! Set view
/VIEW,1,1,1,-0.50
/ANGLE,1,180
/VUP,1,Z
/REPLOT




/graphics,FULL
! ----------------------------------- Store all results in one array
ALLSEL
*GET,numNd,NODE,0,COUNT
*DIM,selstat,ARRAY,numNd,1
*DIM,allresults,ARRAY,numNd,12
*VGET,allresults(1,1),NODE,1,LOC,X
*VGET,allresults(1,2),NODE,1,LOC,Y
*VGET,allresults(1,3),NODE,1,LOC,Z
*VGET,allresults(1,4),NODE,1,U,X
*VGET,allresults(1,5),NODE,1,U,Y
*VGET,allresults(1,6),NODE,1,U,Z
SHELL,TOP
*VGET,allresults(1,7),node,1,S,EQV
SHELL,BOT
```

```
*VGET,allresults(1,8),node,1,S,EQV
SHELL,MID
*VGET,allresults(1,9),node,1,S,EQV
SHELL,TOP
*VGET,allresults(1,10),node,1,EPTO,EQV
SHELL,BOT
*VGET,allresults(1,11),node,1,EPTO,EQV
SHELL,MID
*VGET,allresults(1,12),node,1,EPTO,EQV




! ----PLATE
ESEL,S,sec,,1
NSLE
*VGET,selstat(1,1),NODE,1,nsel                    ! Fill selection status array

*VMASK,selstat(1,1)                                          ! Create masking vector
*VSCFUN,SGMXPLT,max,allresults(1,7)
*VMASK,selstat(1,1)                                          ! Create masking vector
*VSCFUN,SGMXPLB,max,allresults(1,8)
*VMASK,selstat(1,1)                                          ! Create masking vector
*VSCFUN,SGMXPLM,max,allresults(1,9)
*VMASK,selstat(1,1)                                          ! Create masking vector
*VSCFUN,EPMXPLT,max,allresults(1,10)
*VMASK,selstat(1,1)                                          ! Create masking vector
*VSCFUN,EPMXPLB,max,allresults(1,11)
*VMASK,selstat(1,1)                                          ! Create masking vector
*VSCFUN,EPMXPLM,max,allresults(1,12)


! Get maximum of top/bot/mid stresses
*DIM,maxpl,ARRAY,3
maxpl(1)=SGMXPLT
maxpl(2)=SGMXPLB
maxpl(3)=SGMXPLM
*VSCFUN,SGMXPL,max,maxpl

! Get maximum of top/bot/mid strains
*DIM,maxep,ARRAY,3
maxep(1)=EPMXPLT
maxep(2)=EPMXPLB
maxep(3)=EPMXPLM
*VSCFUN,EPMXPL,max,maxep



*VSCFUN,UXMXALL,max,allresults(1,4)



! Get reaction force

NSEL,s,loc,y,0
FSUM
*GET,FY,fsum,0,item,fy

FY = -1*FY/1000000               ! From N to MN



! --- Create and fill output file with input-output parameters
*cfopen,parametervalues,csv,,
!input_writeinputparameters
*VWRITE, 'L1,', lambda_1
(A,F)
*VWRITE,'UXMXALL,', UXMXALL
(A,F)
*VWRITE,'SGMXPLT,', SGMXPLT
```

```
(A,F)
*VWRITE,'SGMXPLB,', SGMXPLB
(A,F)
*VWRITE,'SGMXPLM,', SGMXPLM
(A,F)
*VWRITE,'SGMXPL,', SGMXPL
(A,F)
*VWRITE,'EPMXPLT,', EPMXPLT
(A,F)
*VWRITE,'EPMXPLB,', EPMXPLB
(A,F)
*VWRITE,'EPMXPLM,', EPMXPLM
(A,F)
*VWRITE,'EPMXPL,', EPMXPL
(A,F)
*VWRITE,'FY,', FY
(A,F)
*CFclose









! =====================================
! Store load displacement data in TABLE


*IF,DY,NE,0,THEN        ! Case displacement applied drectly
        Uymax = Dy
*ELSE                          ! Case displacement derived from applied stress
        Uymax = Uyy
*ENDIF




SET,last
*GET,numSubst,active,0,set,sbst         ! Store number of substeps in parameter
    numSubst
*DIM,LoadDisp,TABLE,numSubst,2,,TIME    ! Initialize table

*DO,i,1,numSubst
        SET,1,i                                         ! Set to loadstep i
        *GET,tt,ACTIVE,0,SET,Time       ! Get time value
        U = tt*Uymax                            ! Get applied displacement value at
            time tt                             ! Get applied displacement value at
            time tt
        LoadDisp(i,0) = tt                      ! Store time in table row
        LoadDisp(i,1) = U                       ! Store time in table row


        NSEL,s,loc,y,0
        FSUM
        *GET,FY,fsum,0,item,fy
        FY = ABS(FY/1000000)            ! From N to MN

        LoadDisp(i,2) = Fy


*ENDDO
! =====================================


! Write table to csv file

*cfopen,loaddisp,csv,,
*VWRITE, 'T,', 'Uy,', 'Fy'
(A, A, A)
*VWRITE, LoadDisp(1,1),', ', LoadDisp(1,2),', ', LoadDisp(1,3)
```

```
(F, A, F, A, F)
*CFCLOSE



FINI

! --- Delete unneccesary files in order to save disk space
!input_delete_commands
```

APDL template file geometry

Parts of the commands are re-used from the own produced internship report [31].

```
1
2  ! --- Indicate which geometry is present based on the input parameters
3
4  *IF,NOTS,GT,0,AND,TSWH,GT,0,THEN         ! Full model
5          *IF,CWH,GT,0,THEN
6                  geometry = 3
7          *ENDIF
8  *endif
9  *IF,NOTS,GT,0,AND,TSWH,GT,0,THEN         ! Stiffened plate without columns
10         *IF,CWH,LT,1,THEN
11                 geometry = 2
12         *ENDIF
13 *ENDIF
14 *IF,NOTS,EQ,0,OR,TSWH,EQ,0,THEN          ! Unstiffened plate, always modelled without
       columns
15         geometry = 1
16 *ENDIF
17
18
19
20 ! --- General approach for each component of the structure:
21 ! Define dimensions of arrays
22 ! Create empty arrays
23 ! Fill each array with correct coordinate values
24 ! Create keypoints in a loop
25 ! Draw areas
26 ! Delete arrays
27
28
29
30
31 ! ==================================== Skin plate
       ================================================= !
32 xdim = 1
33 ydim = 2
34 zdim = 2
35
36
37 *DIM,xval,array,xdim,1,1
38 *DIM,yval,array,ydim,1,1
39 *DIM,zval,array,zdim,1,1
40
41 xval(1,1,1)=0
42
43 yval(1,1,1) = 0                  ! Right edge
44 yval(ydim,1,1) = SPW    ! Left edge
45
46 zval(1,1,1) = 0                  ! Bottom edge
```

```
zval(zdim,1,1) = SPH     ! Top edge



! Create keypoints skin plate
*DO,ii,1,xdim,1
        *DO,jj,1,ydim,1
                *DO,kk,1,zdim,1
                        k,, xval(ii,1,1), yval(jj,1,1), zval(kk,1,1)
                *enddo
        *enddo
*enddo


! Create areas skin plate
x = xval(1,1,1)
*DO,jj,1,ydim-1,1
        y1 = yval(jj,1,1)
        y2 = yval(jj+1,1,1)
        *DO,kk,1,zdim-1,1
                z1 = zval(kk,1,1)
                z2 = zval(kk+1,1,1)

                kp_1 = KP(x,y1,z1)
                kp_2 = KP(x,y1,z2)
                kp_3 = KP(x,y2,z2)
                kp_4 = KP(x,y2,z1)
                A,kp_4,kp_3,kp_2,kp_1
        *ENDDO
*ENDDO


*DEL,xval,,nopr
*DEL,yval,,nopr
*DEL,zval,,nopr




! ======================================= T stiffeners webs
    ================================================== !


*IF,geometry,NE,1,THEN
        xdim = 2
        ydim = 2
        zdim = NOTS

        *DIM,xval,array,ydim,1,1
        *DIM,yval,array,ydim,1,1
        *DIM,zval,array,zdim,1,1

        xval(1,1,1) = 0                 ! Right edge
        xval(xdim,1,1) = TSWH   ! Left edge

        yval(1,1,1) = 0                 ! Right edge
        yval(ydim,1,1) = SPW    ! Left edge

        ! Array z-coordinates T webs

        *DO,q,1,NOTS,1
                kk = q
                zval(kk,1,1) = FOTS + (q-1)*CTC_T
        *enddo

        ! Create keypoints webs
        *DO,ii,1,xdim,1
                *DO,jj,1,ydim,1
                        *DO,kk,1,zdim,1
```

```
                                k,, xval(ii,1,1), yval(jj,1,1), zval(kk,1,1)
                        *enddo
                *enddo
        *enddo


        *DO,kk,1,zdim,1
                z = zval(kk,1,1)
                *DO,jj,1,ydim-1,1
                        y1 = yval(jj,1,1)
                        y2 = yval(jj+1,1,1)
                        *DO,ii,1,xdim-1
                                x1 = xval(ii,1,1)
                                x2 = xval(ii+1,1,1)

                                kp_1 = KP(x1,y1,z)
                                kp_2 = KP(x2,y1,z)
                                kp_3 = KP(x2,y2,z)
                                kp_4 = KP(x1,y2,z)
                                A,kp_1,kp_2,kp_3,kp_4
                        *ENDDO
                *ENDDO
        *ENDDO


        *DEL,xval,,nopr
        *DEL,yval,,nopr
        *DEL,zval,,nopr
*ENDIF




! ==================================== T stiffeners flanges
    ================================================ !


*IF,geometry,NE,1,AND,TSFW,GT,0,THEN
        xdim = 1
        ydim = 2
        zdim = 3*NOTS

        *DIM,xval,array,xdim,1,1
        *DIM,yval,array,ydim,1,1
        *DIM,zval,array,zdim,1,1

        xval(1,1,1)=TSWH

        yval(1,1,1) = 0                  ! Right edge
        yval(ydim,1,1) = SPW    ! Left edge

        ! Fill z-coordinate vector
        *DO,q,1,NOTS,1
                kk = 3*q-1
                z_q = FOTS + (q-1)*CTC_T
                zval(kk,1,1) = z_q
                zval(kk-1,1,1) = z_q - 0.5*TSFW
                zval(kk+1,1,1) = z_q + 0.5*TSFW
        *enddo


        ! Create keypoints flanges
        *DO,ii,1,xdim,1
                *DO,jj,1,ydim,1
                        *DO,kk,1,zdim,1
                                k,, xval(ii,1,1), yval(jj,1,1), zval(kk,1,1)
                        *enddo
                *enddo
        *enddo

```

```
        ! Draw areas flanges
        x = xval(1,1,1)
        *DO,jj,1,ydim-1,1
                y1 = yval(jj,1,1)
                y2 = yval(jj+1,1,1)
                *DO,q,1,NOTS
                        kk = 3*q-1
                        z_q = zval(kk,1,1)

                        z1 = z_q - 0.5*TSFW
                        z2 = z_q
                        z3 = z_q + 0.5*TSFW

                        kp_1 = KP(x,y1,z1)
                        kp_2 = KP(x,y1,z2)
                        kp_3 = KP(x,y2,z2)
                        kp_4 = KP(x,y2,z1)
                        A,kp_1,kp_2,kp_3,kp_4

                        kp_1 = KP(x,y1,z2)
                        kp_2 = KP(x,y1,z3)
                        kp_3 = KP(x,y2,z3)
                        kp_4 = KP(x,y2,z2)
                        A,kp_1,kp_2,kp_3,kp_4
                *ENDDO
        *ENDDO

        *DEL,xval,,nopr
        *DEL,yval,,nopr
        *DEL,zval,,nopr

*ENDIF




! ===================================== Column webs
    ================================================== !
*IF,geometry,EQ,3,THEN

        xdim = 3
        ydim = 2
        zdim = 2+NOTS

        *DIM,xval,array,xdim,1,1
        *DIM,yval,array,ydim,1,1
        *DIM,zval,array,zdim,1,1

        xval(1,1,1) = 0
        xval(2,1,1) = TSWH
        xval(xdim,1,1) = CWH

        yval(1,1,1) = 0
        yval(ydim,1,1) = SPW

        zval(1,1,1) = 0                 ! Bottom edge
        zval(zdim,1,1) = SPH    ! Top edge


        *DO,q,1,NOTS,1
                kk = q+1
                zval(kk,1,1) = FOTS + (q-1)*CTC_T
        *enddo


        *DO,ii,1,xdim,1
                *DO,jj,1,ydim,1
                        *DO,kk,1,zdim,1
                                k,, xval(ii,1,1), yval(jj,1,1), zval(kk,1,1)
                        *enddo
```

```
                *enddo
        *enddo


        *DO,jj,1,ydim
                y = yval(jj,1,1)
                *DO,ii,1,xdim-1
                        x1 = xval(ii,1,1)
                        x2 = xval(ii+1,1,1)

                        *DO,kk,1,zdim-1
                                z1 = zval(kk,1,1)
                                z2 = zval(kk+1,1,1)

                                kp_1 = KP(x1,y,z1)
                                kp_2 = KP(x2,y,z1)
                                kp_3 = KP(x2,y,z2)
                                kp_4 = KP(x1,y,z2)
                                A,kp_1,kp_2,kp_3,kp_4
                        *ENDDO
                *ENDDO
        *ENDDO


        *DEL,xval,,nopr
        *DEL,yval,,nopr
        *DEL,zval,,nopr
*ENDIF




! ==================================== Column flanges
    ================================================ !
*IF,geometry,EQ,3,THEN
        xdim = 1
        ydim = 2*3              ! 2 Columns, both 3 coords in y dir
        zdim = 2+NOTS

        *DIM,xval,array,xdim,1,1
        *DIM,yval,array,ydim,1,1
        *DIM,zval,array,zdim,1,1

        xval(1,1,1) = CWH

        zval(1,1,1) = 0
        zval(zdim,1,1) = SPH

        ! Fill y-coordinate vector
        *DO,q,1,2,1
                jj = 3*q-1
                y_q = (q-1)*SPW
                yval(jj,1,1) = y_q
                yval(jj-1,1,1) = y_q - 0.5*CFW
                yval(jj+1,1,1) = y_q + 0.5*CFW
        *enddo

        ! Fill z-coordinate vector
        *DO,q,1,NOTS,1
                kk = q+1
                zval(kk,1,1) = FOTS + (q-1)*CTC_T
        *enddo


        ! Create keypoints flanges
        *DO,ii,1,xdim,1
                *DO,jj,1,ydim,1
                        *DO,kk,1,zdim,1
                                k,, xval(ii,1,1), yval(jj,1,1), zval(kk,1,1)
```

```
                        *enddo
                *enddo
        *enddo


        ! Draw areas flanges
        x = xval(1,1,1)
        *DO,kk,1,zdim-1
                z1 = zval(kk,1,1)
                z2 = zval(kk+1,1,1)
                *DO,q,1,2
                        y_q =(q-1)*SPW

                        y1 = y_q - 0.5*CFW
                        y2 = y_q
                        y3 = y_q + 0.5*CFW

                        kp_1 = KP(x,y1,z1)
                        kp_2 = KP(x,y1,z2)
                        kp_3 = KP(x,y2,z2)
                        kp_4 = KP(x,y2,z1)
                        A,kp_1,kp_2,kp_3,kp_4

                        kp_1 = KP(x,y2,z1)
                        kp_2 = KP(x,y2,z2)
                        kp_3 = KP(x,y3,z2)
                        kp_4 = KP(x,y3,z1)
                        A,kp_1,kp_2,kp_3,kp_4
                *ENDDO
        *ENDDO


        *DEL,xval,,nopr
        *DEL,yval,,nopr
        *DEL,zval,,nopr
*ENDIF




! ================= FINISH GEOMETRY
! --- Merge areas and keypoints
ALLSELL
*IF,geometry,NE,1,THEN
        AGLUE,all
        NUMMRG,all
*ENDIF



! --- Cut model in half in order to create symmetric model
WPOFFS,,0.5*SPW,                        ! Translate working plane to half of the span
    at Y=0.5*SPW
WPROTA,,-90,                            ! Rotate working plane around x-axis
ASBW,all                                ! Cut model in half at working plane
wpcsys,-1,0                             ! Reset working plane (?)

ASEL,s,loc,y,0.5*SPW,SPW+0.5*CFW
ADELE,all,,1
ALLSEL


! --- CREATE COMPONENTS

ASEL,s,loc,x,0  ! Skin Plate
CM,SP,Area

*IF,geometry,NE,1,THEN
        ASEL,None
```

```
397         *DO,q,1,NOTS,1
398                 zval = FOTS + (q-1)*CTC_T
399                 ASEL,A,loc,z,zval
400         *enddo
401         ASEL,U,LOC,X,TSWH
402         CM,TSW,area
403
404         *IF,TSFW,GT,0,THEN
405                 ASEL,S,loc,x,TSWH
406                 CM,TSF,area
407         *ENDIF
408 *ENDIF
409
410 *IF,geometry,EQ,3,THEN
411         ASEL,none
412         ASEL,A,loc,y,0
413         ASEL,A,loc,y,SPW
414         CM,CW,area
415
416         ASEL,S,loc,x,CWH,
417         CM,CF,area
418 *ENDIF
```

J

APDL template file imperfection type 2

Parts of the commands are re-used from the own produced internship report [31].

```
1
2  ! ------------------------
3  ! BEGIN IMPERFECTION 2
4  ! ------------------------
5
6
7  ! Input parameters for imperfection
8  !userinput_ImperfectionParameters
9
10 A = SPW
11 B = SPH
12
13 wave_y = 0.5
14 wave_z = 0.5
15
16 ! Eurocode requirement for selecting amplitude of imperfection 2
17 ! The lowest value of (A/400, B/400) is to be selected as maximum displacement of the
       sine and stored as factor C
18 ! In this case A and B represent the total plate width and height respectively
19
20 *DIM,amp_opts,array,2,1
21 amp_opts(1,1) = A/400
22 amp_opts(2,1) = B/400
23 *VSCFUN,C,min,amp_opts
24
25
26 *do,i,1,ntot
27          x = nodes(i,2)
28          y = nodes(i,3)
29          z = nodes(i,4)
30
31          ! - Skin plate modification
32          *IF,x,EQ,0,THEN
33                   wx = C*sin(2*wave_y*pi*y/A)*sin(2*wave_z*pi*z/B)
34                   wy = 0
35                   wz = 0
36
37                   DISPL(i,2) = DISPL(i,2) + wx
38                   DISPL(i,3) = DISPL(i,3) + wy
39                   DISPL(i,4) = DISPL(i,4) + wz
40          *ENDIF
41
42
43
44          ! - T-stiffener webs modification
45          *IF,x,GT,0,AND,x,LT,TSWH,THEN
46                   wx = C*sin(2*wave_y*pi*y/A)*sin(2*wave_z*pi*z/B)
47                   wy = 0
```

```
                wz = 0

                DISPL(i,2) = DISPL(i,2) + wx
                DISPL(i,3) = DISPL(i,3) + wy
                DISPL(i,4) = DISPL(i,4) + wz
        *ENDIF



        ! - T-stiffener flanges modification
        *IF,x,EQ,TSWH,THEN
                z_0 = 0.5*CTC_T
                z_1 = SPH - 0.5*CTC_T
                *DO,z_web,z_0,z_1,CTC_T
                        *IF,z,GE,z_web-0.5*TSFW,AND,z,LE,z_web+0.5*TSFW,THEN
                                z = z_web
                                wx = C*sin(2*wave_y*pi*y/A)*sin(2*wave_z*pi*z/B)
                                wy = 0
                                wz = 0

                                DISPL(i,2) = DISPL(i,2) + wx
                                DISPL(i,3) = DISPL(i,3) + wy
                                DISPL(i,4) = DISPL(i,4) + wz
                        *ENDIF
                *ENDDO
        *ENDIF


*enddo
! -----------------------
! END IMPERFECTION 2
! -----------------------
```

K

Complete APDL file nonlinear analysis FEA example

```
1
2
3
4  ! To Do list:
5  !       Make mesh sizes dependent on size of component
6  !       Verify that min-max values in results file are equal to the plotted values
7  !       Add separate results for separate components
8
9  !=================================================================
10 ! Start
11 !=================================================================
12
13
14
15 /CLEAR,NOSTART
16
17 ! --- Load document settings
18 /FILNAME, dp0
19 /CWD, 'C:\Users\Thomas\Documents\Master Thesis\Report\apdl_demos\new_benchmark_zone_A'
20 /TITLE, project3-1-2-1
21
22
23
24
25
26
27
28 !=================================================================
29 ! Input parameters
30 !=================================================================
31
32 ! ------------------------------------------------
33 ! Geometry
34 ! ------------------------------------------------
35
36 ! --- Load input parameters
37
38 SPT = 12
39 NOTS = 6                ! NOT Modified to match CTC distance of approx 510 mm, instead,
       increase DISTANCE FROM EDGES (SEE BELOW)
40 TSWH = 220
41 TSWT = 9
42 TSFW = 50
43 TSFT = 20
44 Q0 = 0.05
45 Syy = 200
```

```
Szz = 75
SPH = 3570.0    ! Based on measuring between 2 keypoints
SPW = 3100
CWH = 515.0
CWT = 12.0
CFW = 250.0
CFT = 15.0
Q1 = 0.03

AMP=10









! --- Fixed parameters
E=210000
rho=0.3
sigma_y=355
density=7850









! --- Compute derived parameters

*IF,NOTS,GT,0,THEN
        CTC_T = SPH/(NOTS+1)            ! CTC distance t stiffeners
        FOTS = 1.0*CTC_T                ! First offset T stiffeners

        *ELSEIF,NOTS,EQ,0,THEN
                CTC_T=SPH
*ENDIF


! Edge displacements computed from stresses
Uyy = Syy*0.5*SPW/E
Uzz = Szz*0.5*SPH/E









! --- Define pi
*afun,rad
pi=acos(-1)



! --- Identification numbers necessary for making named selections
SPID = 1                        ! Skin Plate ID
TSWID = 2                       ! T stiffener web ID
TSFID = 3                       ! T stiffener flange ID
CWID = 4                        ! Column web ID
CFID = 5                        ! Column flange ID









!====================================================================
! Preprocessor
!====================================================================
/PREP7

! Add shell elements: Element Type - Add
ET,1,SHELL181   ! type 1: 4 node shell
```

```
117
118
119 ! Set to structural
120 /NOPR
121 KEYW,PR_SET,1
122 KEYW,PR_STRUC,1
123 KEYW,PR_THERM,0
124 KEYW,PR_FLUID,0
125 KEYW,PR_MULTI,0
126 /GO
127
128
129 ! Set material properties: Material Props - Material Models
130 MPTEMP,,,,,,,,
131 MPTEMP,1,0
132 MPDATA,EX,1,,E           ! Youngs modulus
133 MPDATA,PRXY,1,,rho  ! Poissons Ratio
134
135
136 ! Select element key options
137 ETCON,off       ! Let not automatically select applicable keyopts
138 KEYOPT,1,3,2
139 KEYOPT,1,8,2
140
141
142
143 !-----------------------------------------------------------------
144 ! Define element thickness and properties
145 !-----------------------------------------------------------------
146 !EXPLANATION SHELL/MESH ATTRIBUTES
147 ! sect,sectionID,elementType,,sectionName
148 ! secdata, Thickness,MaterialID,0,NumberofIntegrationPoints
149 ! secoffset,MID
150 ! seccontrol,0,0,0, mass/unit area, 1, 1, 1
151
152 sect,SPID,shell,,Skin Plate
153 secdata, SPT,1,0.0,3
154 secoffset,MID
155 seccontrol,,,,0 , , ,
156
157 *IF,NOTS,GT,0,THEN
158         sect,TSWID,shell,,T-webs
159         secdata, TSWT,1,0,3
160         secoffset,MID
161         seccontrol,,,,0 , , ,
162
163         *IF,TSFW,GT,0,THEN
164         sect,TSFID,shell,,T-flanges
165         secdata, TSFT,1,0,3
166         secoffset,MID
167         seccontrol,,,,0 , , ,
168         *ENDIF
169 *ENDIF
170
171
172 sect,CWID,shell,,Column-webs
173 secdata, CWT,1,0,3
174 secoffset,MID
175 seccontrol,,,,0 , , ,
176
177 sect,CFID,shell,,Column-flanges
178 secdata, CFT,1,0,3
179 secoffset,MID
180 seccontrol,,,,0 , , ,
181
182
183
184
185
186
187
```

```
188
189 !=================================================================
190 ! Create 3D model
191 !=================================================================
192
193
194 ! --- Indicate which geometry is present based on the input parameters
195
196 *IF,NOTS,GT,0,AND,TSWH,GT,0,THEN        ! Full model
197         *IF,CWH,GT,0,THEN
198                 geometry = 3
199         *ENDIF
200 *endif
201 *IF,NOTS,GT,0,AND,TSWH,GT,0,THEN        ! Stiffened plate without columns
202         *IF,CWH,LT,1,THEN
203                 geometry = 2
204         *ENDIF
205 *ENDIF
206 *IF,NOTS,EQ,0,OR,TSWH,EQ,0,THEN         ! Unstiffened plate, always modelled without
        columns
207         geometry = 1
208 *ENDIF
209
210
211
212 ! --- General approach for each component of the structure:
213 ! Define dimensions of arrays
214 ! Create empty arrays
215 ! Fill each array with correct coordinate values
216 ! Create keypoints in a loop
217 ! Draw areas
218 ! Delete arrays
219
220
221
222
223 ! ===================================== Skin plate
        ================================================= !
224 xdim = 1
225 ydim = 2
226 zdim = 2
227
228
229 *DIM,xval,array,xdim,1,1
230 *DIM,yval,array,ydim,1,1
231 *DIM,zval,array,zdim,1,1
232
233 xval(1,1,1)=0
234
235 yval(1,1,1) = 0                  ! Right edge
236 yval(ydim,1,1) = SPW    ! Left edge
237
238 zval(1,1,1) = 0                  ! Bottom edge
239 zval(zdim,1,1) = SPH    ! Top edge
240
241
242
243 ! Create keypoints skin plate
244 *DO,ii,1,xdim,1
245         *DO,jj,1,ydim,1
246                 *DO,kk,1,zdim,1
247                         k,, xval(ii,1,1), yval(jj,1,1), zval(kk,1,1)
248                 *enddo
249         *enddo
250 *enddo
251
252
253 ! Create areas skin plate
254 x = xval(1,1,1)
255 *DO,jj,1,ydim-1,1
256         y1 = yval(jj,1,1)
```

```
        y2 = yval(jj+1,1,1)
        *DO,kk,1,zdim-1,1
                z1 = zval(kk,1,1)
                z2 = zval(kk+1,1,1)

                kp_1 = KP(x,y1,z1)
                kp_2 = KP(x,y1,z2)
                kp_3 = KP(x,y2,z2)
                kp_4 = KP(x,y2,z1)
                A,kp_4,kp_3,kp_2,kp_1
        *ENDDO
*ENDDO


*DEL,xval,,nopr
*DEL,yval,,nopr
*DEL,zval,,nopr




! ==================================== T stiffeners webs
    ================================================ !


*IF,geometry,NE,1,THEN
        xdim = 2
        ydim = 2
        zdim = NOTS

        *DIM,xval,array,ydim,1,1
        *DIM,yval,array,ydim,1,1
        *DIM,zval,array,zdim,1,1

        xval(1,1,1) = 0                 ! Right edge
        xval(xdim,1,1) = TSWH   ! Left edge

        yval(1,1,1) = 0                 ! Right edge
        yval(ydim,1,1) = SPW    ! Left edge

        ! Array z-coordinates T webs

        *DO,q,1,NOTS,1
                kk = q
                zval(kk,1,1) = FOTS + (q-1)*CTC_T
        *enddo

        ! Create keypoints webs
        *DO,ii,1,xdim,1
                *DO,jj,1,ydim,1
                        *DO,kk,1,zdim,1
                                k,, xval(ii,1,1), yval(jj,1,1), zval(kk,1,1)
                        *enddo
                *enddo
        *enddo


        *DO,kk,1,zdim,1
                z = zval(kk,1,1)
                *DO,jj,1,ydim-1,1
                        y1 = yval(jj,1,1)
                        y2 = yval(jj+1,1,1)
                        *DO,ii,1,xdim-1
                                x1 = xval(ii,1,1)
                                x2 = xval(ii+1,1,1)

                                kp_1 = KP(x1,y1,z)
                                kp_2 = KP(x2,y1,z)
                                kp_3 = KP(x2,y2,z)
```

```
                                kp_4 = KP(x1,y2,z)
                                A,kp_1,kp_2,kp_3,kp_4
                        *ENDDO
                *ENDDO
        *ENDDO


        *DEL,xval,,nopr
        *DEL,yval,,nopr
        *DEL,zval,,nopr
*ENDIF




! ===================================== T stiffeners flanges
    ================================================ !


*IF,geometry,NE,1,AND,TSFW,GT,0,THEN
        xdim = 1
        ydim = 2
        zdim = 3*NOTS

        *DIM,xval,array,xdim,1,1
        *DIM,yval,array,ydim,1,1
        *DIM,zval,array,zdim,1,1

        xval(1,1,1)=TSWH

        yval(1,1,1) = 0                 ! Right edge
        yval(ydim,1,1) = SPW    ! Left edge

        ! Fill z-coordinate vector
        *DO,q,1,NOTS,1
                kk = 3*q-1
                z_q = FOTS + (q-1)*CTC_T
                zval(kk,1,1) = z_q
                zval(kk-1,1,1) = z_q - 0.5*TSFW
                zval(kk+1,1,1) = z_q + 0.5*TSFW
        *enddo


        ! Create keypoints flanges
        *DO,ii,1,xdim,1
                *DO,jj,1,ydim,1
                        *DO,kk,1,zdim,1
                                k,, xval(ii,1,1), yval(jj,1,1), zval(kk,1,1)
                        *enddo
                *enddo
        *enddo

        ! Draw areas flanges
        x = xval(1,1,1)
        *DO,jj,1,ydim-1,1
                y1 = yval(jj,1,1)
                y2 = yval(jj+1,1,1)
                *DO,q,1,NOTS
                        kk = 3*q-1
                        z_q = zval(kk,1,1)

                        z1 = z_q - 0.5*TSFW
                        z2 = z_q
                        z3 = z_q + 0.5*TSFW

                        kp_1 = KP(x,y1,z1)
                        kp_2 = KP(x,y1,z2)
                        kp_3 = KP(x,y2,z2)
                        kp_4 = KP(x,y2,z1)
                        A,kp_1,kp_2,kp_3,kp_4
```

```
397
398                             kp_1 = KP(x,y1,z2)
399                             kp_2 = KP(x,y1,z3)
400                             kp_3 = KP(x,y2,z3)
401                             kp_4 = KP(x,y2,z2)
402                             A,kp_1,kp_2,kp_3,kp_4
403                     *ENDDO
404             *ENDDO
405
406             *DEL,xval,,nopr
407             *DEL,yval,,nopr
408             *DEL,zval,,nopr
409
410     *ENDIF
411
412
413
414
415
416     ! ===================================== Column webs
            ================================================= !
417     *IF,geometry,EQ,3,THEN
418
419             xdim = 3
420             ydim = 2
421             zdim = 2+NOTS
422
423             *DIM,xval,array,xdim,1,1
424             *DIM,yval,array,ydim,1,1
425             *DIM,zval,array,zdim,1,1
426
427             xval(1,1,1) = 0
428             xval(2,1,1) = TSWH
429             xval(xdim,1,1) = CWH
430
431             yval(1,1,1) = 0
432             yval(ydim,1,1) = SPW
433
434             zval(1,1,1) = 0                     ! Bottom edge
435             zval(zdim,1,1) = SPH     ! Top edge
436
437
438             *DO,q,1,NOTS,1
439                     kk = q+1
440                     zval(kk,1,1) = FOTS + (q-1)*CTC_T
441             *enddo
442
443
444             *DO,ii,1,xdim,1
445                     *DO,jj,1,ydim,1
446                             *DO,kk,1,zdim,1
447                                     k,, xval(ii,1,1), yval(jj,1,1), zval(kk,1,1)
448                             *enddo
449                     *enddo
450             *enddo
451
452
453             *DO,jj,1,ydim
454                     y = yval(jj,1,1)
455                     *DO,ii,1,xdim-1
456                             x1 = xval(ii,1,1)
457                             x2 = xval(ii+1,1,1)
458
459                             *DO,kk,1,zdim-1
460                                     z1 = zval(kk,1,1)
461                                     z2 = zval(kk+1,1,1)
462
463                                     kp_1 = KP(x1,y,z1)
464                                     kp_2 = KP(x2,y,z1)
465                                     kp_3 = KP(x2,y,z2)
466                                     kp_4 = KP(x1,y,z2)
```

```
                                    A,kp_1,kp_2,kp_3,kp_4
                            *ENDDO
                    *ENDDO
            *ENDDO


            *DEL,xval,,nopr
            *DEL,yval,,nopr
            *DEL,zval,,nopr
*ENDIF




! ======================================= Column flanges
    ================================================= !
*IF,geometry,EQ,3,THEN
            xdim = 1
            ydim = 2*3                  ! 2 Columns, both 3 coords in y dir
            zdim = 2+NOTS

            *DIM,xval,array,xdim,1,1
            *DIM,yval,array,ydim,1,1
            *DIM,zval,array,zdim,1,1

            xval(1,1,1) = CWH

            zval(1,1,1) = 0
            zval(zdim,1,1) = SPH

            ! Fill y-coordinate vector
            *DO,q,1,2,1
                    jj = 3*q-1
                    y_q = (q-1)*SPW
                    yval(jj,1,1) = y_q
                    yval(jj-1,1,1) = y_q - 0.5*CFW
                    yval(jj+1,1,1) = y_q + 0.5*CFW
            *enddo

            ! Fill z-coordinate vector
            *DO,q,1,NOTS,1
                    kk = q+1
                    zval(kk,1,1) = FOTS + (q-1)*CTC_T
            *enddo


            ! Create keypoints flanges
            *DO,ii,1,xdim,1
                    *DO,jj,1,ydim,1
                            *DO,kk,1,zdim,1
                                    k,, xval(ii,1,1), yval(jj,1,1), zval(kk,1,1)
                            *enddo
                    *enddo
            *enddo


            ! Draw areas flanges
            x = xval(1,1,1)
            *DO,kk,1,zdim-1
                    z1 = zval(kk,1,1)
                    z2 = zval(kk+1,1,1)
                    *DO,q,1,2
                            y_q =(q-1)*SPW

                            y1 = y_q - 0.5*CFW
                            y2 = y_q
                            y3 = y_q + 0.5*CFW

                            kp_1 = KP(x,y1,z1)
```

```
537                             kp_2 = KP(x,y1,z2)
538                             kp_3 = KP(x,y2,z2)
539                             kp_4 = KP(x,y2,z1)
540                             A,kp_1,kp_2,kp_3,kp_4
541
542                             kp_1 = KP(x,y2,z1)
543                             kp_2 = KP(x,y2,z2)
544                             kp_3 = KP(x,y3,z2)
545                             kp_4 = KP(x,y3,z1)
546                             A,kp_1,kp_2,kp_3,kp_4
547                     *ENDDO
548             *ENDDO
549
550
551             *DEL,xval,,nopr
552             *DEL,yval,,nopr
553             *DEL,zval,,nopr
554     *ENDIF
555
556
557
558
559
560     ! ================= FINISH GEOMETRY
561     ! --- Merge areas and keypoints
562     ALLSELL
563     *IF,geometry,NE,1,THEN
564             AGLUE,all
565             NUMMRG,all
566     *ENDIF
567
568
569
570     ! --- Cut model in half in order to create symmetric model
571     WPOFFS,,0.5*SPW,                        ! Translate working plane to half of the span
            at Y=0.5*SPW
572     WPROTA,,-90,                            ! Rotate working plane around x-axis
573     ASBW,all                                ! Cut model in half at working plane
574     wpcsys,-1,0                             ! Reset working plane (?)
575
576     ASEL,s,loc,y,0.5*SPW,SPW+0.5*CFW
577     ADELE,all,,1
578     ALLSEL
579
580
581     ! --- CREATE COMPONENTS
582
583     ASEL,s,loc,x,0  ! Skin Plate
584     CM,SP,Area
585
586     *IF,geometry,NE,1,THEN
587             ASEL,None
588             *DO,q,1,NOTS,1
589                     zval = FOTS + (q-1)*CTC_T
590                     ASEL,A,loc,z,zval
591             *enddo
592             ASEL,U,LOC,X,TSWH
593             CM,TSW,area
594
595             *IF,TSFW,GT,0,THEN
596                     ASEL,S,loc,x,TSWH
597                     CM,TSF,area
598             *ENDIF
599     *ENDIF
600
601     *IF,geometry,EQ,3,THEN
602             ASEL,none
603             ASEL,A,loc,y,0
604             ASEL,A,loc,y,SPW
605             CM,CW,area
606
```

```
607         ASEL,S,loc,x,CWH,
608         CM,CF,area
609 *ENDIF
610
611
612
613
614 !-----------------------------------------------------------------
615 ! Assign elements to selected components and MESHING
616 !-----------------------------------------------------------------
617 AATT, MAT, REAL, ELEMENT TYPE, ESYS, SECTION NUMBER
618
619 CMSEL,S,SP                      ! Select component
620 AATT,       1, ,   1,        0,   SPID
621
622 *IF,NOTS,GT,0,THEN
623         CMSEL,S,TSW
624         AATT,       1, ,   1,        0,   TSWID
625
626         *IF,TSFW,GT,0,THEN
627         CMSEL,S,TSF
628         AATT,       1, ,   1,        0,   TSFID
629         *ENDIF
630 *ENDIF
631
632
633 *IF,geometry,EQ,3,THEN
634         CMSEL,S,CW
635         AATT,       1, ,   1,        0,   CWID
636
637         CMSEL,S,CF
638         AATT,       1, ,   1,        0,   CFID
639 *ENDIF
640
641
642
643
644 ! --- Meshing
645 ! - Set mesh options
646 MSHKEY,2                ! Use mapped meshing if possible; otherwise, use free meshing
647 MOPT,split,2    ! Quad splitting option for non-mapped meshing. If Value = 2 or WARN,
        quadrilateral
648                                 ! elements in violation of either shape error or
                                        warning limits are split into triangles.
649 MSHAPE,0,2d     ! Mesh with quadrilateral-shaped elements when Dimension = 2D
650
651
652 ! - Set mesh size and mesh areas
653 ESIZE,40
654 ALLSEL
655 AMESH,ALL
656
657
658 seltol,0.0005
659
660 ALLSEL
661
662 ! --- Node components
663 CMSEL,S, TSF
664 CMSEL,A, TSW
665 CMSEL,A, SP
666 NSLA, S, 1
667 NSEL, R, LOC, Y, 0
668 CM, RightEdge, Node
669 NSEL, S, LOC, Y, 0.5*SPW
670 CM, MidEdge, Node
671 NSEL, S, LOC, Z, 0
672 CM, BottomEdge, Node
673 NSEL, S, LOC, Z, SPH
674 CM, TopEdge, Node
675
```

```
676 CMSEL, S, RightEdge, Node
677 NSEL, R, LOC, X, 0
678 CM, RightEdgePlate, Node
679 CMSEL, S, MidEdge, Node
680 NSEL, R, LOC, X, 0
681 CM, MidEdgePlate, Node
682 CMSEL, S, BottomEdge, Node
683 NSEL, R, LOC, X, 0
684 CM, BottomEdgePlate, Node
685 CMSEL, S, TopEdge, Node
686 NSEL, R, LOC, X, 0
687 CM, TopEdgePlate, Node
688
689 CMSEL, S, RightEdge, Node
690 NSEL,U,LOC,X,0
691 CM,RightEdgeSection,Node
692 CMSEL, S, MidEdge, Node
693 NSEL,U,LOC,X,0
694 CM,MidEdgeSection,Node
695 CMSEL, S, BottomEdge, Node
696 NSEL,U,LOC,X,0
697 CM,BottomEdgeSection,Node
698 CMSEL, S, TopEdge, Node
699 NSEL,U,LOC,X,0
700 CM,TopEdgeSection,Node
701
702
703 ! - Create node components for reading displacements from eigenmodes in order to apply
        right imperfection
704 NSEL,S,LOC,X,0
705 CM,PlateNodes,Node
706
707 CMSEL,S,TSW,AREA
708 NSLA,S,1
709 NSEL,R,LOC,X,0
710 CM,BottomStiffenersNodes,Node
711
712 CMSEL,S,TSW,AREA
713 NSLA,S,1
714 NSEL,R,LOC,X,TSWH
715 CM,TopStiffenersNodes,Node
716
717
718
719
720
721 !====================================
722 ! Enter solution processor
723 !====================================
724 /SOL
725
726 ALLSEL
727 ! --- Add Boundary conditions
728
729 ! - Fix horizontal edges in x direction
730 D, TopEdge,UX, 0
731 D, BottomEdge,UX, 0
732
733 ! - Apply deformations
734 D, RightEdgePlate, UY, Uyy
735 D, RightEdgeSection, UY, Uyy
736
737 ! Support column cross sections in y direction
738 ALLSEL
739 D, TopEdgeSection, UY, Uyy
740 D, BottomEdgeSection, UY, Uyy
741
742
743 ! Apply Szz as a displacement
744 D,TopEdge,Uz,-Uzz
745 D,BottomEdge,Uz,Uzz
```

```
746
747
748 ! Apply Szz as a stress onto the plate and column section
749 !CMSEL,S,BottomEdgePLate,Node
750 !SF,ALL,Pres,Szz*SPT
751 !CMSEL,S,BottomEdgeSection,Node
752 !SF,ALL,Pres,Szz*CFT
753
754 !CMSEL,S,TopEdgePLate,Node
755 !SF,ALL,Pres,Szz*SPT
756 !CMSEL,S,TopEdgeSection,Node
757 !SF,ALL,Pres,Szz*CFT
758
759
760
761
762 ! Support in Z direction at a point halfway column web at flange
763 NSEL,S,LOC,X,CWH
764 NSEL,R,LOC,Y,0
765 NSEL,R,LOC,Z,0.5*SPH-25, 0.5*SPH+25
766 D,ALL,UZ,0
767
768
769
770 ! - Apply symmetric boundary conditions
771 CMSEL,S,MidEdge,NODE
772 DSYMM,SYMM,Y
773
774
775
776 ! --- Add surface loads
777 sfcum,pres,add
778
779 ! ----- Load contribution number 1 -----!
780 Z0 = 0 $ P0 = Q0
781 Z1 = SPH $ P1 = Q1
782 delta_p = (P1 - P0)/(Z1 - Z0)
783 ESEL,S,SEC,,SPID
784 ESEL,R,CENT,Z,Z0,Z1
785 SFGRAD,PRES,0,Z,Z0,delta_p
786 SFE,ALL,1,PRES,0, Q0
787 !-----------------
788 /psf,pres,norm,3,1
789
790
791
792 ! - When a nonlinear analysis is performed with an updated geometry following from an
      eigenvalue analysis,
793 !       the commands will be loaded here
794
795 ALLSEL
796 /SOL
797 ANTYPE, static
798 pstres, on
799 eqslv, sparse
800 SOLVE
801
802 FINI
803
804 n_modes=1
805
806 /SOL
807 antype,buckle
808 bucopt,lanb,n_modes
809 mxpand,n_modes,,,yes
810 SOLVE
811 FINI
812
813
814 /post1
815 !Retrieve eigenvalues and store in parameters
```

```
*DO,mode,1,n_modes
        SET,1,mode
        *GET,lambda_%mode%,FREQ
*ENDDO


/PREP7
upgeom,AMP,1,1,dp0,rst



ALLSEL
! Perform nonlinear analysis
/SOLU
ANTYPE, static
OUTRE, all, all
NLGEOM, on
AUTOTS,auto
NSUBST,1000,1000,10
SOLVE

!=====================================
! Enter postprocessor
!=====================================


/POST1
set, last


! Set view
/VIEW,1,1,1,-0.50
/ANGLE,1,180
/VUP,1,Z
/REPLOT




! ------------------------------------ Store all results in one array
ALLSEL
*GET,numNd,NODE,0,COUNT
*DIM,selstat,ARRAY,numNd,1
*DIM,allresults,ARRAY,numNd,7
*VGET,allresults(1,1),NODE,1,LOC,X
*VGET,allresults(1,2),NODE,1,LOC,Y
*VGET,allresults(1,3),NODE,1,LOC,Z
*VGET,allresults(1,4),NODE,1,U,X
*VGET,allresults(1,5),NODE,1,U,Y
*VGET,allresults(1,6),NODE,1,U,Z
*VGET,allresults(1,7),node,1,S,EQV







! ---------------------------------- Take min-max values of results

! --- ALL
ALLSEL
*VSCFUN,SIGMXALL,max,allresults(1,7)
*VSCFUN,UXMXALL,max,allresults(1,4)



! --- PLATE

CMSEL,s,PlateNodes,Node                                  ! Select plate
*VGET,selstat(1,1),NODE,1,nsel                  ! Fill selection status array
*VMASK,selstat(1,1)                                          ! Create masking vector
```

```
887 *VSCFUN,SIGMXPL,max,allresults(1,7)             ! Store max SEQV of plate
888 *VMASK,selstat(1,1)
889 *VSCFUN,UXMXPL,max,allresults(1,4)             ! Store max UX of plate
890
891
892 CMSEL,s,PlateNodes,Node                                ! Select plate
893 NSEL,R,LOC,Y,0,0.1*SPW                                 ! Reselect 10% of plate width
        next to column
894 *VGET,selstat(1,1),NODE,1,nsel                   ! Fill selection status array
895 *VMASK,selstat(1,1)                                          ! Create masking vector
896 *VSCFUN,SIGMXPLE,max,allresults(1,7)            ! Store max SEQV of plate along column
897
898 CMSEL,s,PlateNodes,Node                                ! Select plate
899 NSEL,R,LOC,Y,0.1*SPW, 0.5*SPW                  ! Reselect 40% of plate width
900 *VGET,selstat(1,1),NODE,1,nsel                  ! Fill selection status array
901 *VMASK,selstat(1,1)                                          ! Create masking vector
902 *VSCFUN,SIGMXPLM,max,allresults(1,7)
903
904
905 ! --- STIFFENERS
906
907 CMSEL,s,TSW,AREA
908 CMSEL,a,TSF,AREA
909 NSLA,s,1                                                                ! Select
        Stiffeners nodes
910 *VGET,selstat(1,1),NODE,1,nsel                   ! Fill selection status array
911 *VMASK,selstat(1,1)
912 *VSCFUN,SIGMXST,max,allresults(1,7)             ! Store max SEQV of stiffeners
913 *VMASK,selstat(1,1)
914 *VSCFUN,UXMXST,max,allresults(1,4)              ! Store max ux of stiffeners
915
916
917 CMSEL,s,TSW,AREA
918 CMSEL,a,TSF,AREA
919 NSLA,s,1
920 NSEL,R,LOC,Y,0,0.1*SPW
921 *VGET,selstat(1,1),NODE,1,nsel                         ! Fill selection status array
922 *VMASK,selstat(1,1)
923 *VSCFUN,SIGMXSTE,max,allresults(1,7)            ! Store max SEQV of stiffeners next to
        column
924
925 CMSEL,s,TSW,AREA
926 CMSEL,a,TSF,AREA
927 NSLA,s,1
928 NSEL,R,LOC,Y,0.1*SPW,0.5*SPW
929 *VGET,selstat(1,1),NODE,1,nsel                         ! Fill selection status array
930 *VMASK,selstat(1,1)
931 *VSCFUN,SIGMXSTM,max,allresults(1,7)            ! Store max SEQV of stiffeners
        midsection
932
933
934
935
936
937
938
939 ! --- Create and fill output file with input-output parameters
940 *cfopen,parametervalues,csv,,
941 *VWRITE, 'SPH,', SPH
942 (A, F)
943 *VWRITE, 'SPW,', SPW
944 (A, F)
945 *VWRITE, 'SPT,', SPT
946 (A, F)
947 *VWRITE, 'NOTS,', NOTS
948 (A, F)
949 *VWRITE, 'TSWH,', TSWH
950 (A, F)
951 *VWRITE, 'TSWT,', TSWT
952 (A, F)
953 *VWRITE, 'TSFW,', TSFW
```

```
(A, F)
*VWRITE, 'TSFT,', TSFT
(A, F)
*VWRITE, 'Q0,', Q0
(A, F)
*VWRITE, 'Q1,', Q1
(A, F)
*VWRITE, 'Syy,', Syy
(A, F)
*VWRITE, 'Szz,', Szz
(A, F)
*VWRITE, 'AMP,', AMP
(A, F)
*VWRITE, 'CWH,', CWH
(A, F)
*VWRITE, 'CWT,', CWT
(A, F)
*VWRITE, 'CFW,', CFW
(A, F)
*VWRITE, 'CFT,', CFT
(A, F)

*VWRITE, 'L1,', lambda_1
(A,F)
*VWRITE,'UXMXALL,', UXMXALL
(A,F)
*VWRITE,'UXMXPL,', UXMXPL
(A,F)
*VWRITE,'UXMXST,', UXMXST
(A,F)
*VWRITE,'SIGMXALL,', SIGMXALL
(A,F)
*VWRITE,'SIGMXPL,', SIGMXPL
(A,F)
*VWRITE,'SIGMXPLM,', SIGMXPLM
(A,F)
*VWRITE,'SIGMXPLE,', SIGMXPLE
(A,F)
*VWRITE,'SIGMXST,', SIGMXST
(A,F)
*VWRITE,'SIGMXSTM,', SIGMXSTM
(A,F)
*VWRITE,'SIGMXSTE,', SIGMXSTE
(A,F)
*CFclose
```