Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

Andreas Stene Dehlin Gudbrandsen

Predicting the Buckling Load of Stiffened Plates by Surrogate Modeling

Master's thesis in Marine Technology
Supervisor: Bernt Johan Leira
June 2024

NTNU
Norwegian University of
Science and Technology

Andreas Stene Dehlin Gudbrandsen

Predicting the Buckling Load of Stiffened Plates by Surrogate Modeling

Master’s thesis in Marine Technology
Supervisor: Bernt Johan Leira
June 2024

Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

NTNU Trondheim
Norwegian University of Science and Technology *Faculty of Engineering*
Department of Marine Technology

Master Thesis Spring 2024

for

Stud. Tech. Andreas Stene Gudbrandsen

Predicting the Buckling Load of Stiffened Plates by Surrogate Modeling

Prediksjon av knekklast for avstivede plater ved bruk av surrogatmodeller

Traditionally, finite element analysis (FEA) has been the go-to method for determining the loadbearingcapacity of stiffened plates. However, FEA can be time-consuming and complex, which can be a disadvantage, particularly when quick yet precise assessments are needed. Fortunately, new technologies such as machine learning (ML) and artificial intelligence (AI) may provide a more efficient and speedy way of analyzing structures.

Researchers and experts are working on using artificial neural networks (ANN) to predict the buckling load of stiffened plates. ANNs are computer models that can learn to recognize patterns and make predictions based on data input. They have the potential to replace traditional FEA methods for some applications, providing a faster and more efficient way of analyzing structures without compromising accuracy.

The goal of this thesis is to assess the applicability of surrogate models that utilize data obtained from FEA simulations in order to predict the buckling load of stiffened plates, exploring the intersection between AI and structural engineering. However, the accuracy and dependability of the surrogate model are heavily reliant on the quality, diversity, and comprehensiveness of the data used to train it. Additionally, the current model only deals with specific types of stiffened plates, which means it may not apply to all structural elements or more complex loading conditions.

The following items are to be addressed as part of the present master thesis:

1. Perform a literature survey on methods and procedures for construction of digital twin models (also referred to as surrogate models, meta-models or cyber-physical models), for structures and structural systems. Both data-based methods and physics-based methods (and combinations of these) are to be considered. Examples of methods are: Neural networks, Response surfaces, Polynomial Chaos Expansion and Gaussian Process Regression

2. As an example of application, a stiffened plate panel is to be studied. Relative to the study in the project thesis, this example case is to be explored in further details based on the Neural Network approach. This could also consist of incorporating alternative and more diversified neural network architectures.

3. As a next step the model of the stiffened plate is to be expanded in term of number of parameters that are varied. This could also comprise an expansion of outer dimensions of the

panel thereby increasing the number of stiffeners and possibly multi-bay panels with tranverse frames forming part of the model.

4. The same stiffened plate example as in item 2 is also to be analysed by application of the combination of a physics based model and a neural network (i.e. a so-called PINN-model).

5. To the extent that time allows, alternative approaches (according to item 1) for constructing a digital twin of the plate example are to be investigated. As part of this, statistical models of parameter uncertainties and their propagation to system level can also be addressed.

The work scope may prove to be larger than initially anticipated. Subject to approval from the supervisors, topics may be deleted from the list above or reduced in extent. This is to be notified to the reader in the introduction.

In the master report, the candidate shall present his/her personal contribution to the resolution of problems within the scope of the master work

Theories and conclusions should be based on mathematical derivations and/or logic reasoning identifying the various steps in the deduction.

The candidate should utilise the existing possibilities for obtaining relevant literature.

Master report format
The master report should be organised in a rational manner to give a clear exposition of results, assessments, and conclusions. The text should be brief and to the point, with a clear language. Telegraphic language should be avoided. The report shall contain the following elements: A text defining the scope (this document to be included), preface, list of contents, summary, main body of thesis, conclusions with recommendations for further work, list of symbols and acronyms, references and (optional) appendices. All figures, tables and equations shall be numerated.

The supervisors may require that the candidate, in an early stage of the work, presents a written plan for the completion of the work.

The original contribution of the candidate and material taken from other sources shall be clearly defined. Work from other sources shall be properly referenced using an acknowledged referencing system.

The report shall be submitted in electronic format (.pdf):
- Signed by the candidate
- The text defining the scope shall be included (this document)
- Drawings and/or computer models that are not suited to be part of the report in terms of appendices shall be provided on separate (.zip) files.

Ownership
NTNU has according to the present rules the ownership of the master reports. Any use of the report has to be approved by NTNU (or external partner when this applies). The department has the right to use the report as if the work was carried out by a NTNU employee, if nothing else has been agreed in advance.

Thesis supervisors:
Prof. Bernt J. Leira, NTNU, bernt.leira@ntnu.no

Deadline: June 10, 2024

Trondheim, January 14 , 2024

Bernt J. Leira

Bernt J. Leira

ABSTRACT

This master's thesis investigates the application of surrogate models, especially artificial neural networks, to the structural analysis of stiffened plate panels, a common component in marine engineering. Compared to traditional Finite Element Analysis, the study aims to reduce computational time and improve prediction accuracy in predicting buckling loads.

The thesis begins with a comprehensive literature review on surrogate modeling techniques and their applications in structural engineering. A neural network model is then developed and trained using a dataset generated from FEA simulations. The stiffened plate was modeled in Abaqus and exported to a Python code by using the Python library abqpy. The model was then simulated for 1944 different geometric configurations to create an extensive dataset for the neural network to train on before being able to predict the buckling load for unseen data.

The results show that the neural network model significantly reduces computational time while maintaining high accuracy, achieving a mean prediction error of approximately 2.94% within the training range. While the model performs well within the trained parameters, its accuracy decreases for data outside the training range with an average error of 5.41%. These are still decent results but highlight the need for a more diverse dataset and advanced modeling techniques. The study demonstrates the potential of neural networks to transform structural analysis by offering a scalable and flexible approach. However, it also underscores challenges such as data dependency, model interpretability, and issues with extrapolation. The thesis concludes with recommendations for future work, including expanding the dataset, exploring hybrid models that combine machine learning with physics-based approaches, and validating the models in real-world engineering projects.

Overall, this thesis contributes to the ongoing development of efficient and accurate predictive models in structural engineering, paving the way for more sophisticated applications of digital twin technology in engineering design and analysis.

SAMMENDRAG

Denne masteroppgaven undersøker anvendelsen av surrogatmodeller, spesielt kunstige nevrale nettverk, til strukturanalyse av stive platepaneler, en vanlig komponent innen marin ingeniørkunst. Sammenlignet med tradisjonell Finite Element Analysis (FEA), har studien som mål å redusere beregningstiden og forbedre prediksjonsnøyaktigheten i å forutsi knekkbelastninger.

Oppgaven begynner med en omfattende litteraturgjennomgang om surrogatmodelleringsteknikker og deres anvendelser innen strukturingeniørkunst. En nevrale nettverksmodell utvikles deretter og trenes ved hjelp av et datasett generert fra FEA-simuleringer. Det stive platepanelet ble modellert i Abaqus og eksportert til en Python-kode ved å bruke Python-biblioteket abqpy. Modellen ble deretter simulert for 1944 forskjellige geometriske konfigurasjoner for å skape et omfattende datasett som det nevrale nettverket kunne trene på før det kunne forutsi knekkbelastningen for ukjente data.

Resultatene viser at den nevrale nettverksmodellen betydelig reduserer beregningstiden samtidig som den opprettholder høy nøyaktighet, og oppnår en gjennomsnittlig prediksjonsfeil på omtrent 2,94% innenfor treningsområdet. Selv om modellen presterer godt innenfor de trente parameterne, reduseres nøyaktigheten for data utenfor treningsområdet med en gjennomsnittlig feil på 5,41%. Dette er fortsatt gode resultater, men understreker behovet for et mer variert datasett og avanserte modelleringsteknikker. Studien viser potensialet til nevrale nettverk for å transformere strukturanalyse ved å tilby en skalerbar og fleksibel tilnærming. Den fremhever imidlertid også utfordringer som dataavhengighet, modellfortolkning og problemer med ekstrapolering. Oppgaven avsluttes med anbefalinger for fremtidig arbeid, inkludert utvidelse av datasettet, utforsking av hybride modeller som kombinerer maskinlæring med fysikkbaserte tilnærminger, og validering av modellene i reelle ingeniørprosjekter.

Totalt sett bidrar denne oppgaven til den pågående utviklingen av effektive og nøyaktige prediktive modeller innen strukturingeniørkunst, og baner vei for mer sofistikerte anvendelser av digital tvillingteknologi i ingeniørdesign og analyse.

PREFACE

This thesis for a Master's degree in Marine Technology at the Norwegian University of Science and Technology (NTNU) with a specialization in Marine Structures is the final piece of the program. It builds upon the work completed in the project thesis in the autumn of 2023, and it was completed in the spring of 2024. This thesis represents the culmination of the five-year study program.

The thesis investigates the possibility of using Neural Networks and other types of surrogate modeling to predict the buckling load of stiffened plates. A Neural Network model was made in Python, and the data the model was trained on was obtained from the Finite Element software ABAQUS. Using analytical skills in combination with structural knowledge has been very beneficial. It has been a rewarding journey to gain an understanding of machine learning, neural networks, and surrogate modeling, starting from little to no knowledge about nothing it.

I am thankful for Professor Bernt Johan Leira's help during the process, and his guidance and support have been highly appreciated. I am very grateful that I got to shape my own thesis and received guidance and reassurance from Bernt while writing it.

Lastly, I want to thank my classmates for the last five years, and especially my two office roommates, Harald Horne Fosseng and Christian Ohm. We have complained, laughed, and struggled together to complete our degree.

Trondheim, 07.06.24

Andreas Stene Dehlin Gudbrandsen

Contents

List of Figures

List of Tables

NOMENCLATURE

$\bar{\lambda}$	Reduced slenderness
ϵ	Strain
Γ	Gaussian distribution
$\hat{\mathcal{M}}$	Surrogate of the complex model M
λ_i	Eigenvalues
$\mathcal{B}$	Beta distribution
$\mathcal{M}$	Function representing a complex model
$\mathcal{N}$	Gaussian distribution
$\mathcal{U}$	Uniform distribution
μ	Factor
∇^4	Laplace operator
σ_b	Design bending stress
σ_E	Euler stress
σ_P	X
σ_x	Axial stress
σ_Y	Yield stress
σ_{xcr}	Critical stress in compression
τ_{crg}	Critical shear stress
A	Cross sectional area of stiffener
A_e	Cross sectional area of plate
A_e	Effective area
A_w	Cross sectional area of web

ANN	Artificial Neural Network
b	Width
b_e	Effective width
D	Plate Stiffness
D	Plate stiffness
DOF	Degrees of Freedom
$E[]$	Expectation
E	Youngs modulus
e	Eccentricity of the stiffener to the plate flange
E_s	Secant Modulus
E_T	Tangent Modulus
FEA	Finite Element Analysis
FEM	Finite Element Method
I	Moment of inertia
I_e	Effective moment of inertia
i_e	Effective radius of gyration
$K(r)$	Secant Stiffness
K	Stiffness Matrix
k	Element Stiffness Matrix
K_0^{NM}	Stiffness in the base state
K_Δ^{NM}	Differential stiffness
$K_I(r)$	Incremental Stiffness
l	Length
ML	Machine Learning
N, M	Degrees of freedom
N_x, N_y, N_z	Force in x, y, and z-direction
N_τ	Additional shear force
NN	Neural Network
P^N	X

Q^N	Incremental load
R	External Loads
S	Nodal Forces Vector
t	Plate Thickness
U	Elastic Strain Energy
v	Nodal Displacements Vector
v^M	X
W	Elastic section modulus
z_p	Distance from the plate flange to the original neutral axis

CHAPTER

ONE

INTRODUCTION

Structural engineering deals with designing and analyzing load-bearing structures, and analyzing these systems under complex loading conditions often requires advanced computational methods to ensure safety and performance. While traditional methods like the Finite Element Method are accurate, they are computationally intensive and time-consuming, limiting real-time applications and iterative design processes. An example of this is predicting and managing buckling behavior in stiffened plate panels due to their nonlinear nature, requiring detailed and computationally expensive simulations.

To overcome these challenges, surrogate models, such as Artificial Neural Networks and Polynomial Chaos Expansion, offer computationally efficient approximations of complex models. These models reduce the computational resources required for structural analysis while maintaining high accuracy. This thesis examines the use of surrogate models in analyzing the structural behavior of stiffened plate panels. Specifically, it focuses on the accuracy of predicting buckling loads and attempts to predict the buckling load of a stiffened plate using an Artificial Neural Network.

The thesis is divided into these chapters:

- **Chapter2** - Background
- **Chapter3** - Theory
  - Chapter 3.1 - Finite Element Method, Stiffened Plates and Buckling
  - Chapter 3.2 - Surrogate Modelling and Artificial Neural Networks
  - Chapter 3.3 - Polynomial Chaos Expansion
- **Chapter4** - Method
  - Chapter 4.2 - Building the Stiffened Plate in Abaqus and Python
  - Chapter 4.3 - Building the Neural Network
- **Chapter5** - Results
- **Chapter6** - Discussion and Further Work
- **Chapter7** - Conclusion

CHAPTER TWO

BACKGROUND

2.1 Use of Surrogate Models in Structural Analysis

An established approach to simulation of structural mechanics is the finite element method (FEM), which is renowned for its precision in structural mechanics. However, achieving higher resolution in simulations necessitates increased computational effort. Surrogate modeling provides a dependable solution to address this challenge. Nonetheless, selecting the appropriate surrogate model and its hyperparameters for a specific use case is complex [1]. This section investigates and compares different mesh-free surrogate models based on traditional and emerging machine learning (ML) and deep learning methods.

As given by Wang and Shan [2], there are various roles of surrogate modeling in support of design process:

- Model approximation. Approximating computation-intensive processes throughout or just a part of the design space is employed to lower computation expenses.
- Design space exploration. Surrogates provide insight into the functional relationship between design parameters and criteria, addressing obstacles in understanding numerical model behavior.
- Problem formulation. With a deeper comprehension of a design optimization issue, it is possible to decrease the quantity and scope of design variables; some ineffective constraints can be eliminated; a single objective optimization problem can be transformed into a multi-objective optimization problem, or vice versa. The use of metamodels can aid in creating an optimization issue that is either easier to solve or more precise than it would be otherwise.
- Optimization support. The business sector has different optimization requirements, such as global optimization, multi-objective optimization, multidisciplinary design optimization, probabilistic optimization, and more. Each optimization type presents unique challenges. The integration of metamodeling can address varied optimization problems with computationally intensive functions.

Designing intricate thin-walled structures (e.g., ships) is a large-scale problem involving multiple design goals, numerous design variables, and thousands of design constraints. In this study, the focus is on the potential for using surrogate modeling to predict the structural responses of stiffened plates.

2.1.1 Previous studies using surrogate modeling in structural analysis

In this section, various machine learning techniques and their applications to topics in structural engineering are to be presented. Among these methods, Neural Networks, the most widely used method in the field, will be discussed separately.

A widely employed technique for surrogate modeling is response surfaces (RS). Its popularity in modeling deterministic computer experiments stems from its origins in classical Design of Experiments theory, where RS was utilized for describing physical phenomena [3]. Simpson et al. (2001) discusses the statistical challenges of applying RS to deterministic computer experiments [4]. Some of the engineering applications of RS include structural optimization, as discussed by Araia and Shimizu (2001) [5], and in the PhD thesis of Prebeg (2011) [6].

A recent method based on neural networks is physics-informed neural networks (PINNs). These networks are trained simultaneously on both data and governing differential equations, and they can be used to solve and invert equations that govern physical systems. Haghighat and Juanes used PINNs to replace a specific FEM simulation of a perforated strip under uniaxial extension [7]. In 2021, Haghighat et al. introduced a surrogate modeling approach using PINNs and a specific application [7]. Shin examined results related to PINNs with Poisson's equation and the heat equation, focusing on consistency [8]. Yin et al. employed PINNs to forecast permeability and viscoelastic modulus from thrombus deformation data, described by the fourth-order Cahn–Hilliard and Navier–Stokes equations [9].

A research study by Bui et al. delves into using Polynomial Chaos Expansion (PCE) to analyze the stochastic vibration and buckling of functionally graded materials [10]. The researchers used PCE to assess stochastic responses, such as natural frequencies and critical buckling loads. Their results illustrate that PCE offers efficient and precise evaluations, making it a valuable tool for examining structural behavior under uncertainty.

Another notable contribution on the subject is the research on uncertainty quantification and global sensitivity analysis for structural buckling by Chen et al. [11]. In this study, a data-driven PCE model is developed to quantify uncertainties and carry out sensitivity analysis on eigenvalue buckling problems. The efficacy of the PCE model is confirmed using statistical moments and Sobol sensitivity indices, providing insights into the parameters that have the most significant impact on buckling behavior. The authors emphasize the effectiveness of PCE in providing a comprehensive understanding of structural stability under uncertain conditions.

Gaspar and Soares have investigated the potential use of PCE in the stochastic analysis of a steel rectangular plate element under lateral pressure, representing a ship bottom plate element. The results demonstrate that the adopted PCE technique in the analysis could accurately predict the stochastic plate response [12].

2.2 Use of Artificial Neural Networks in FEA

Because of the multitude of geometric, material, and load variations in designing stiffened panels, FEM delivers precise results and clearly defined failure modes. However, it requires time-consuming modeling and calculations. Therefore, there is a need for a new approach that can achieve both speed and accuracy. Artificial Neural Networks show promise in addressing such engineering challenges.

A number of studies that utilize machine learning in structural engineering are emerging. However, the use of ANNs in marine structures is quite limited. An effort was made by Wei and Zhang (1999) to apply ANNs to predict the ultimate strength of stiffened plates [13], and they obtained positive results.

Another study done on marine structures was carried out by Li and colleagues [14]. The study provides a detailed nonlinear investigation of the ultimate compressive strength properties of ship plates containing several cracks under longitudinal compression using FEM and ANN methods. It was shown in this study that the ANN effectively predicted the decrease in Ultimate Strength.

Limited research has been conducted regarding steel plates, thin shells, and stiffened panels. However, Yongchang Pu and Ehsan Mesbahi [15] utilized an ANN to assess the ultimate strength of an unstiffened panel under uniaxial compression. The results of their study demonstrate the potential of ANN, which generally yields superior outcomes compared to empirical formulas derived from traditional regression analysis. This study accurately forecasted compressive strength by leveraging data from prior experiments. Additionally, ML was utilized to analyze plates made of non-linear materials such as aluminum and stainless steel alloys in the research conducted by Cevik and Guzelbey [16], demonstrating that ANNs outperformed other ML techniques.

Zareei et al. used machine learning methods to predict the ultimate strength of flat-bar aluminum stiffened panels. They included welding effects in their model to improve accuracy [17]. Mallela and Upadhyay employed a neural network trained on a small set of finite element models to forecast the ultimate strength of laminated composite stiffened panels, achieving high accuracy and efficiency [18]. Kumar et al. created a model to determine the ultimate strength of hat-stiffened composite panels under axial-compressive loading, with most models yielding errors below 2% [19]. ANNs have been utilized in structural engineering to predict deflection and ultimate buckling strength in beams and columns. For instance, Tohidi and Sharifi used this approach to predict the buckling strength of steel I-beams with restrained buckling [20]. Another study by Abambres et al. [21]

explored the ultimate buckling strength of Cellular I-profiles using ANNs, with promising results.

The work of Bisagni and Lanzi [22] acknowledges the challenges faced by other researchers striving for minimum weight optimization for stiffened composite panels. In such optimization tasks, key considerations often include minimum panel strength and the critical buckling load as output criteria. Input parameters typically encompass the layup definition, stiffener count, and dimensions involving discrete and continuous variables. To tackle the complexities arising from these variables and the potential non-convex nature of the cost function, Bisagni and Lanzi opted for a genetic algorithm (GA) as the optimization algorithm.

Due to the computational intensity of this GA-based approach, Bisagni and Lanzi proposed an optimization procedure that employs ANNs to predict the desired structural response outputs. Initially, these neural networks are trained using a series of FEM simulations. Once trained, these networks eliminate the need for time-consuming calculations during optimization. A distinct ANN is trained for each output variable, covering pre-buckling stiffness, buckling load, collapse load, and even the load-displacement curve.

It is important to note that many of these studies primarily evaluated the performance of ML models within the range of their training data, with limited exploration of their generalizability and scalability.

CHAPTER
THREE

THEORY

3.1 Finite Element Method

The finite element method is utilized in the design of buildings, vessels, aircraft, and spacecraft. It is a numerical approach used to estimate solutions to differential equations. The fundamental idea is to address the behavior of each finite element individually after breaking down a complex problem into smaller, more manageable segments known as finite elements, each with its own governing equations. A set of nodal values, which represent the values of the solution at specific points within the element, define the behavior of each component. A system of linear equations can be established by applying boundary conditions and consolidating the equations for each element in the mesh. This system of equations can then be numerically solved to approximate the overall solution. It will not provide exact solutions, but it is precise enough for engineering purposes.

3.1.1 Principles

The finite element method is based on three principles [23]:

- Equilibrium
- Kinematic compatibility
- Stress-strain relationship

The equation for equilibrium is defined by stress, while kinematic compatibility is defined by strains, and the final principle involves the connection between stresses and strains. When dealing with small displacements, equilibrium is achieved using linear strain functions, resulting in a linear stress-strain relationship in line with Hooke's Law. However, when calculating the ultimate strength of structures that buckle and collapse, the linear assumptions often need to be adjusted.

3.1.2 Main Steps

The theory written about in this section is taken from the compendium in TMR4190 - Finite Element Modelling and Analysis of Marine Structures by Torgeir Moen [23].

3.1.2.1 Discretization

Discretization involves dividing the structure into multiple smaller elements, with all elements and nodes being numbered. Figure 3.1.1 illustrates the discretization of a transverse frame beam model. The nodes are positioned along the connections between the structural elements. In the case of a plane stress problem, the structure's geometry is subdivided into smaller elements using a mesh, with each element being interconnected at the corners and along the edges. The number of elements typically determines the accuracy of the results.

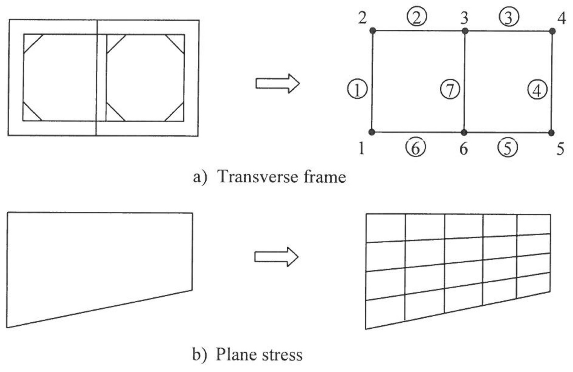


Figure 3.1.1: Discretization of frame and plane stress problems [23]

3.1.2.2 Element Analysis

Element analysis consists of two primary aspects: expressing the displacements within the elements and ensuring equilibrium within the elements. It is also crucial to maintain compatibility by utilizing stress-strain relationships.

To derive the stiffness matrix for each element, the element stiffness relation is acquired with respect to a local reference system. The equation utilized is $S = kv$, where S represents a vector consisting of nodal forces, k is the element stiffness matrix in the local reference system, and v is the vector containing the nodal displacements. For beam elements, the relationship between forces, moments, and corresponding displacements is employed to derive this equation. This relationship can be understood as obtained from the governing differential equation and boundary condition of the beam elements. However, it is not feasible to employ the exact solution for a plane stress problem.

Expressions for the shape functions are utilized to represent the displacements within the element, which are then scaled by the node displacements. By assuming these shape functions, it becomes feasible to compute the displacement at any point within the element using the displacement of the nodal points.

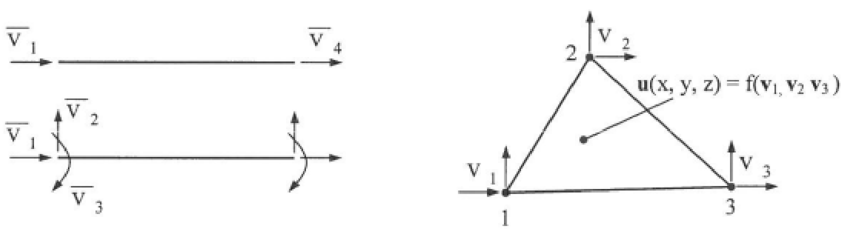


Figure 3.1.2: General Beam Element and Plane Stress Element [23]

The part of the design that the element represents is supported by the forces acting along the edges. When conducting finite element analysis, it is useful to calculate nodal point forces by ensuring that the element is in integrated equilibrium through work or energy considerations. This procedure establishes a connection between the nodal point displacements and forces, which can be expressed as:

$$\mathbf{S} = \mathbf{k} \cdot \mathbf{v} + \mathbf{S^0} \tag{3.1}$$

Where S^0 is the nodal point forces for the external load.

3.1.2.3 System Analysis

The system's stiffness relations are established using an "address table," which outlines the connections between the local degrees of freedom of all the elements and the system's DOFs. A connection is formed between the load and the nodal displacements to maintain equilibrium for all nodal points in the structure.

$$\mathbf{R} = \mathbf{K} \cdot \mathbf{r} + \mathbf{R^0} \tag{3.2}$$

$$\mathbf{K} = \sum_{\mathbf{j}} \mathbf{a_j^T k_j a_j} \tag{3.3}$$

$$\mathbf{R^0} = \sum_{\mathbf{j}} \mathbf{a_j^T S_j^0} \tag{3.4}$$

3.1.2.4 Boundary Conditions

The equation system is adjusted by introducing boundary conditions. Boundary conditions can be applied either by setting nodal displacements to known values or by including spring stiffnesses. The equation system is then expressed as $R_{mod} = K_{mod} r_{mod}$.

3.1.2.5 Finding Global Displacements

The global displacements are found by solving the linear set of equations stated below.

$$\mathbf{r} = \mathbf{K^{-1}} \cdot (\mathbf{R} - \mathbf{R^0}) \tag{3.5}$$

3.1.2.6 Calculation of Stresses

Hooke's law is used to calculate the stresses based on the strains. Strains are obtained from the displacement functions within the element, which are then combined with Hooke's law. The following equation provides a general expression for this process.

$$\sigma(x, y, z) = \mathbf{D} \cdot \mathbf{B}(x, y, z) \cdot \mathbf{v} \tag{3.6}$$

Here, $\mathbf{D}$ is Hooke's law on matrix form and $\mathbf{B}$ is derived from $\mathbf{u}(x, y, z)$.

3.1.3 Nonlinearities

When steel structures collapse along with buckling, it typically occurs because of one or both of two nonlinear behaviors. These include geometric nonlinearity, which is connected with buckling or significant deflection, and material nonlinearity, which is triggered by yielding or plastic deformation [23].

3.1.3.1 Geometric Nonlinearity

When solving for equilibrium equations and calculating strains from displacement, Geometrical Nonlinear Behavior is taken into account. In problems involving geometric nonlinearity, deformations are significant enough to invalidate linear assumptions. This indicates that the connection between loads and deformations becomes nonlinear, even when the material's behavior is linear. The stiffness relationship in linear theory is represented as shown in Equation (3.7).

$$R = Kr \tag{3.7}$$

K, the stiffness matrix, remains constant regardless of the structure's deformation, while r represents the displacement vector. This is equal to the external loads R. When geometric nonlinearities are introduced, the geometric stiffness relation is altered to:

$$R = K(r)r \tag{3.8}$$

K(r) is denoted secant stiffness. Equation (3.8) can be solved analytically for a given R, but it is normally done using iterative methods. To do this, the equation is expressed on a differential form.

$$dR = \frac{d}{dr}(K(r)r)dr = K_I dr \tag{3.9}$$

and

$$K_I(r) = \frac{d}{dr}(K(r)) \tag{3.10}$$

where $K_I(r)$ is the incremental stiffness.

3.1.3.2 Material Nonlinearity

Material nonlinearity occurs when the connection between stress and strain in a material is not linear. In linear materials, stress is directly proportional to strain (Hooke's Law). However, in nonlinear materials, this relationship becomes more intricate. Tests on metal materials demonstrate that linearity no longer holds true when stress exceeds a certain level, denoted as σ_P. Dealing with nonlinear material properties requires the introduction of additional terminology.
From Figure 3.1.3 material properties can be defined.

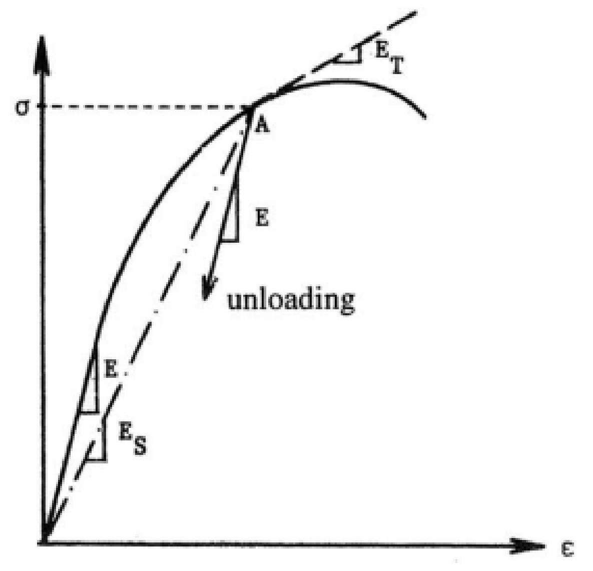


Figure 3.1.3: Material Properties [23]

The stress at A is calculated from Equation (3.11):

$$\sigma = E_s \epsilon \tag{3.11}$$

Here, E_s is the secant modulus which is dependent on the stress level. For loading and unloading from point A, the change of stress can be described as in Equation (3.12) and (3.13), respectively. E_T is the tangent modulus, while Hookes's law applies to the unloading.

$$\Delta\sigma = E\Delta\epsilon \tag{3.12}$$

$$\Delta\sigma = E_T\Delta\epsilon \tag{3.13}$$

3.1.4 Solution Techniques

3.1.4.1 General

The characteristic features of some types of nonlinear response are illustrated in Figure 3.1.4 and 3.1.5.

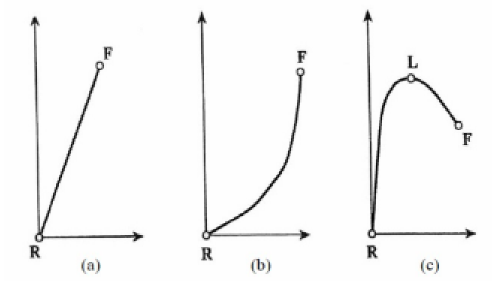


Figure 3.1.4: [23]

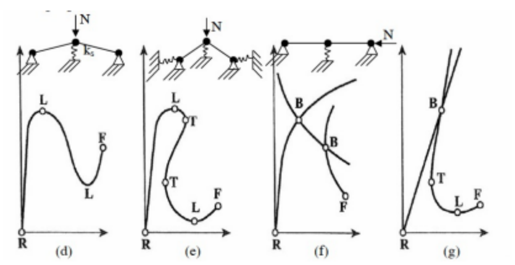


Figure 3.1.5: [23]

The cases illustrated are listed from a-g. L and F describe limit points and failure, while B and T shows bifurcation and turning points.

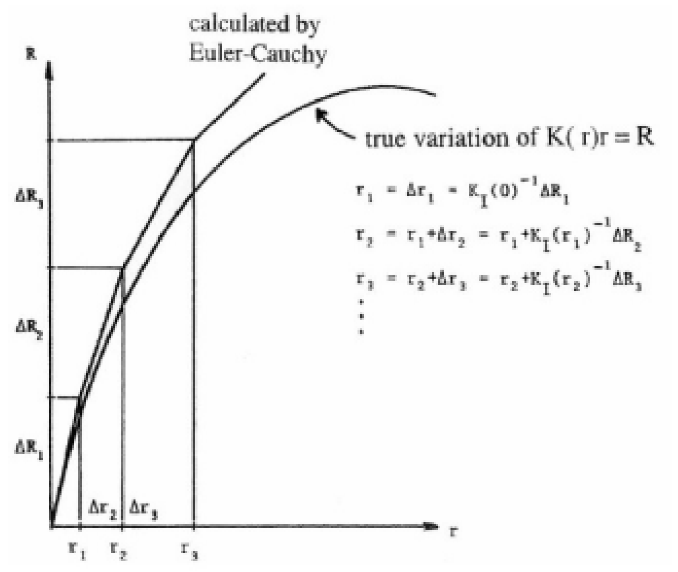
- Linear until brittle failure
- Stiffening or hardening
- Softening
- Snap-Through
- Snap-Back
- Bifurcation combined with limit points and snap-back (f and g)

For nonlinear analyses, the solutions are no longer unique indicating that the achieved solution might not align with the intended one. This section will cover three fundamental methods to address the variety of possible solutions. The three methods are the Euler-Cauchy method, Newton-Rapshon iterative method, and a combined method.

3.1.4.2 Euler-Cauchy Method

The increments in displacement from the Euler-Cauchy method are based on external loading increments. By adding together the new displacement increments the total displacement can be calculated. This is explained mathematically in Equation (3.14) and illustrated in Figure 3.1.6.

$$\begin{aligned} \Delta R_{m+1} &= R_{m+1} - R_m \\ \Delta r_{m+1} &= K_I(r_m)^{-1}\Delta R_{m+1} \\ r_{m+1} &= r_m + \Delta r_{m+1} \end{aligned} \tag{3.14}$$



calculated by Euler-Cauchy
true variation of K(r)r = R
$r_1 = \Delta r_1 = K_I(0)^{-1}\Delta R_1$
$r_2 = r_1+\Delta r_2 = r_1+K_I(r_1)^{-1}\Delta R_2$
$r_3 = r_2+\Delta r_3 = r_2+K_I(r_2)^{-1}\Delta R_3$
R
ΔR_3
ΔR_2
ΔR_1
Δr_2
Δr_3
r_1
r_2
r_3
r

Figure 3.1.6: Euler-Cauchy increment [23]

In this approach, the stiffness from the previous step is utilized to calculate the displacement increment for the subsequent step. The accuracy of this method can be enhanced by using smaller load increments.

3.1.4.3 Newton-Raphson Method

The most frequently used iterative method for solving non-linear structural problems is the Newton-Raphson method. In the Newton-Raphson method, the displacement increment at each step is determined by the difference between the approximate values and the true values at that step and can be written mathematically as described in Equation (3.15).

$$R - R_{int} = K_{I(n)}\Delta r_{n+1} \tag{3.15}$$

The basic principle for this iteration is illustrated in Figure 3.1.7 for a single d.o.f. system. This method requires that K_I is established and that Δr_{n+1} is solved in each iterative step. This is time-consuming. By updating K_I less frequently reduced efforts are needed. Since this approach implies only a limited loss of rate of convergence, such modified Newton-Raphson iteration is beneficial.

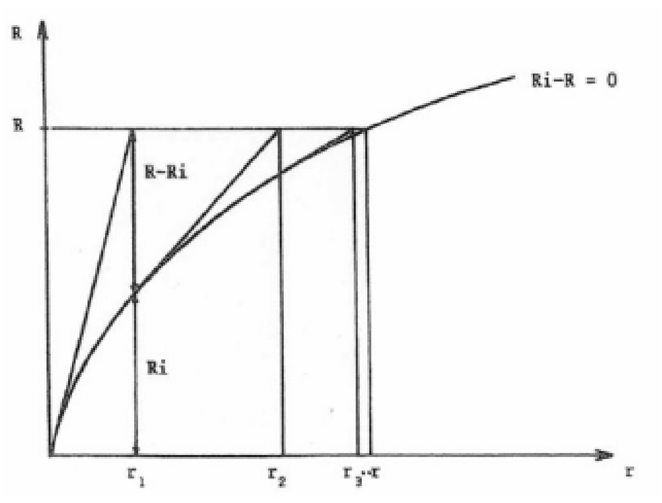


Figure 3.1.7: Newton-Raphson [23]

Figure 3.1.8 illustrates two alternatives for modified Newton-Raphson methods, one with no updating of K_I and one method where K_I is updated after the first iteration. The iteration is stopped when the accuracy is acceptable. The convergence criterion may be based on the change of displacement from one iteration to the next.

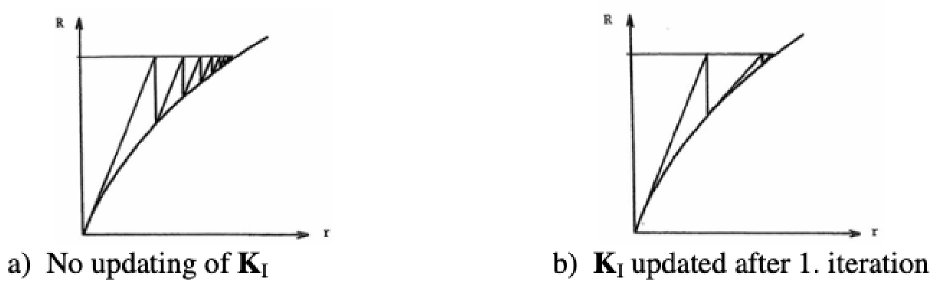

R
r
a) No updating of $\mathbf{K}_I$
R
r
b) $\mathbf{K}_I$ updated after 1. iteration

Figure 3.1.8: Modified Newton-Raphson [23]

3.1.4.4 Combined Methods

Incremental and iterative methods can also be combined and can be done by applying the external load in increments. By doing this, equilibrium in each increment is achieved by iteration as shown in Figure 3.1.9.

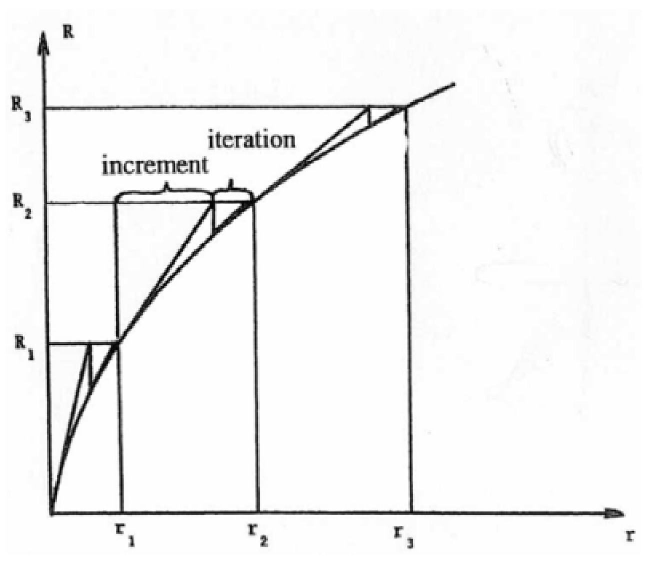


Figure 3.1.9: Combined Method [23]

The process involves applying loads as specified by Equation (3.14), followed by iterations at each load level using Equation (3.15). Typically, the modified Newton-Raphson method is used keeping the gradient K_I constant across several iteration cycles.

3.1.5 Stiffened Plates

The stiffened panel is crucial to the structural integrity of ship hulls due to its ease of fabrication and excellent strength-to-weight ratio. Typically, stiffened panels in ships are subject to combined in-plane and lateral pressure loads. The study of stiffened plates experiencing compressive loads is important in structural engineering, especially in applications where stability under high compressive stress is critical, such as in ship hulls, bridge decks, and aerospace structures [24].

The primary framing system for hull girders consists of closely spaced longitudinal stiffeners, along with more widely spaced, heavier girders in the transverse direction. Plates play a vital role in transferring hydrostatic loads to the stiffeners, which then transfer the loads to the transverse girders, forming the transverse frames of the hull girder. The loads are introduced as membrane stresses in the side from the vertical girders.

The side also experiences hydrostatic loads. Generally, the bottom plate of a ship is subjected to biaxial in-plane loads caused by longitudinal bending of the hull girder and hydrostatic pressure on the sides, as shown in Figure 3.1.10 [24]. Performing a rigorous analysis of such panels subjected to simultaneous action of lateral pressure as well as in-plane loads is quite challenging. For design purposes, the problem is often divided so that the critical load is first determined for each of the loads acting alone.

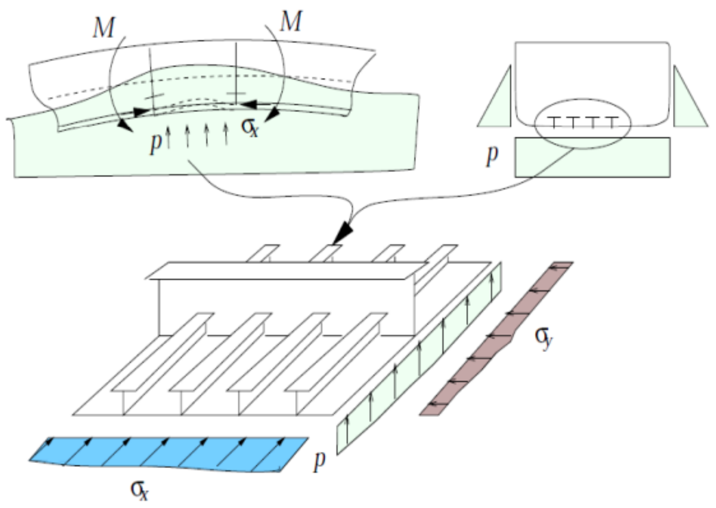


Figure 3.1.10: Stiffened Panels in a Bottom Structure [24]

For a stiffened plate under compressive loading, the primary concern is buckling. The buckling can occur in both the plate itself and in the stiffeners.

3.1.5.1 Failure Modes

The possible failure modes for a stiffened plate is illustrated in Figure 3.1.11.

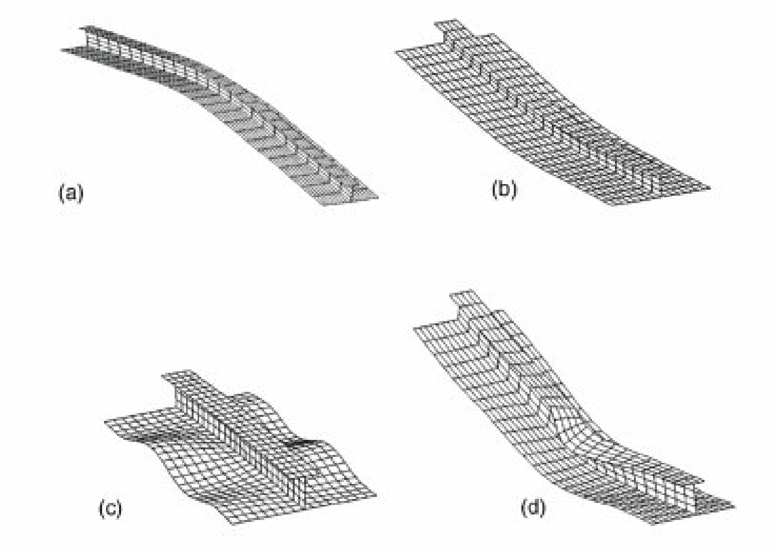


Figure 3.1.11: Failure Modes [24]

Plate buckling and ultimate collapse refer to when the plate load exceeds the maximum capacity, followed by unloading, resulting in the collapse of the stiffened panel without significant yield occurring in the stiffeners. Interframe flexural buckling occurs in the longitudinal stiffeners along with associated plating. This type of failure involves the yielding of the stiffeners, which is hastened by the loss of stiffness due to buckling or yielding of the plate. Stiffener-restrained torsional buckling is caused by the elastic or elasto-plastic loss of stiffness, which depends

on the slenderness of the stiffeners, the rotational restraint provided by the plating, and the initial out-of-shape. Overall grillage buckling involves the bending of transverse girders and longitudinal stiffeners. Most structures are designed to prevent overall grillage buckling, therefore this failure mode is unlikely except for lightly stiffened panels found in superstructure decks.
The failure of a stiffened plate can occur in different forms, including local buckling, plate-induced failure, stiffener-induced failure, overall buckling, and tripping. However, in some cases, the junction of the plate and stiffener may slightly move out of its usual flat plane. This movement is typically small, only about one-tenth of the overall movement of the structure. This supports the idea that it generally doesn't extend outwards significantly.

When plates buckle overall, the junction between the web and the line goes beyond the point that is out of the plane, causing the plate to lose its flatness. This results in a half-wave projection resembling a half-sine wave. The failure of both the plate and the stiffener happens simultaneously, which is known as Euler type buckling. If the stiffener is on the tension side, the overall buckling failure is referred to as plate-induced overall buckling. If the stiffener is on the compression side, it is called stiffener-induced overall buckling. Lateral torsional buckling is a sudden reduction in load-carrying capacity due to the tripping failure of the stiffened plate around the stiffener to plate junction. The tripping stiffener may undergo tension or compression due to bending. It is a significant failure due to the drop in load-carrying capacity [23].

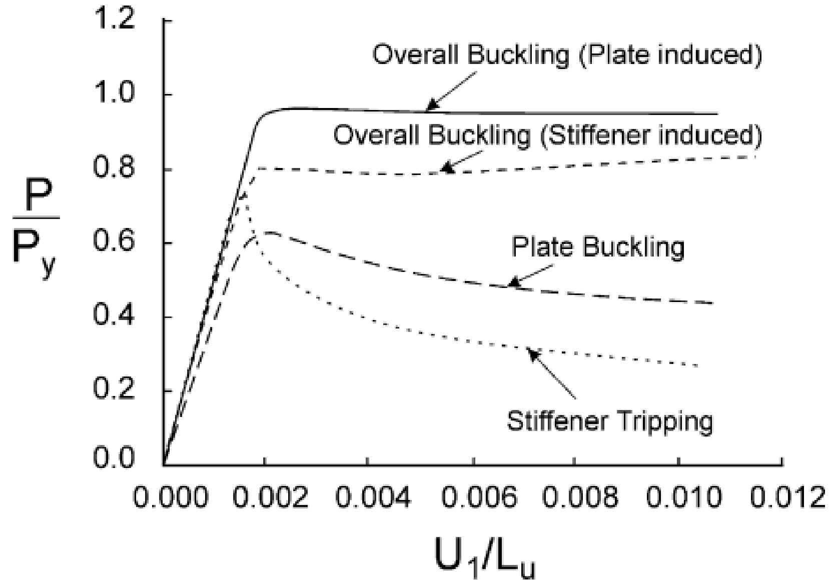


Figure 3.1.12: Failure Modes [24]

3.1.6 Buckling of Plates

Buckling in a structural context can be characterized as the abrupt, unstable deflection observed in slender structures when subjected to compression or shear loads. This instability phenomenon can manifest itself at loads considerably lower than the material failure thresholds of the structure's constituent materials. Once buckling occurs, the structure loses its stability, leading to unanticipated deformations and eventual failures. Consequently, it is crucial to account for the impact of buckling in a structural assembly and to calculate the premature failures resulting from this phenomenon. The approach described here is taken from the compendium in TMR4205 Buckling and Ultimate Strength of Marine Structures [24] and also written about in the author's project thesis [25].

The classical approach to elastic plate buckling problems is either by solving the differential equation of equilibrium or applying energy methods.

The equilibrium equation for a plate is given by Equation (3.16).

$$\nabla^4 w = \frac{1}{D}\left(q + N_x \frac{\partial^2 w}{\partial x^2} + 2N_{xy}\frac{\partial^2 w}{\partial x \partial y} + N_y \frac{\partial^2 w}{\partial y^2}\right) \tag{3.16}$$

With D being the plate stiffness, given as:

$$D = \frac{Et^3}{12(1-v^2)} \tag{3.17}$$

The N-quantities, N_x, N_y and N_{xy}, are the membrane stress resultants. For the problem with uniaxial compression illustrated in Figure 3.1.13 Equation 3.16 looks like:

$$\nabla^4 w = \frac{N_x}{D}\frac{\partial^2 w}{\partial x^2} \tag{3.18}$$

This leads to a solution for the buckling stress on the form:

$$\sigma = \frac{\pi^2 E}{12(1-v^2)}(\frac{t}{b})^2 \cdot k \tag{3.19}$$

Here, k is a factor that depends on the plate aspect ratio, illustrated in Figure 3.1.14.

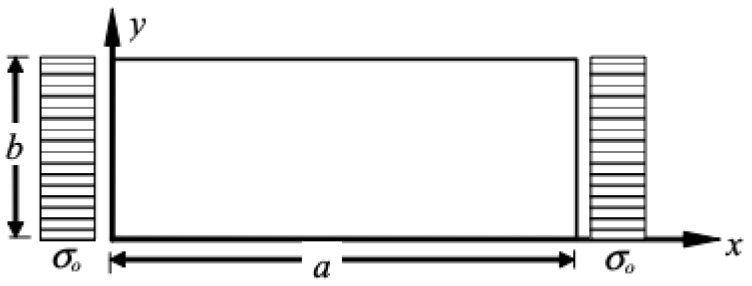


Figure 3.1.13: Simply Supported Plate Under Uniaxial Compression [24]

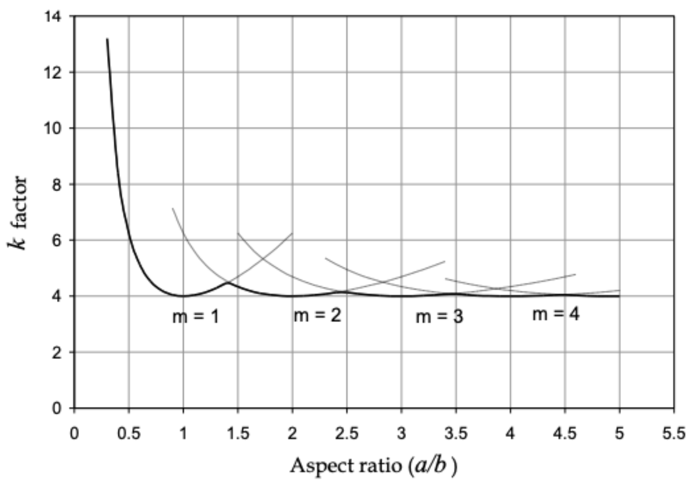


Figure 3.1.14: Buckling Coefficient versus Plate Aspect Ratio [24]

The other approach to solving plate buckling problems is the energy method. The elastic strain energy at the critical load can be expressed as:

$$U = \frac{D}{2}\int_0^a \int_0^b \left\{ \left(\nabla^2 w\right)^2 - 2(1-v)\left(\frac{\partial^2 w}{\partial x^2}\frac{\partial^2 w}{\partial y^2} - \left(\frac{\partial^2 w}{\partial x \partial y}\right)^2\right)\right\} dxdy \tag{3.20}$$

By setting two conditions along the boundaries to

$$w = 0\frac{\partial w}{\partial n} = 0 \tag{3.21}$$

the critical load can be found by applying the principle of minimum potential energy, leading to the same critical load equation as for the solution of the differential equation.

3.1.7 Buckling of Stiffened Plates

Most of the theory in this section is taken from the compendium in TMR4205 Buckling and Ultimate Strength of Marine Structures by Professor Jørgen Amdahl [26].

3.1.7.1 Effective Width Method According to Faulkner

The effective width method was proposed by Faulkner and is based on the elastic critical load for a strut with pinned ends described in Equation (3.22).

$$\sigma_E = \frac{\pi^2 E I_e'}{l^2(A_w + A_e)} \tag{3.22}$$

It is modified for plasticity according to the Johnson-Ostenfield formulation:

$$\frac{\sigma_e}{\sigma_y} = 1 - \frac{\bar{\lambda}^2}{4} \quad , \bar{\lambda}^2 \leq 2 \tag{3.23}$$

The ultimate strength is reduced to account for loss of plate stiffness,

$$\sigma_u = \sigma_e \frac{A_w + A_e}{A_w + A_p} \tag{3.24}$$

The effective moment of inertia of the stiffener is calculated for a tangent effective width of the plate given by:

$$\frac{b_e}{b} = \frac{1}{b}\sqrt{\frac{\sigma_y}{\sigma_e}} \tag{3.25}$$

3.1.7.2 Initial Yield Method (DNV Classification Note 30.1)

The buckling check in stiffened plates is based upon a beam-column approach,

$$\frac{\sigma_x}{\sigma_{xcr}} + \frac{\sigma_b}{(1 - \frac{\sigma_x}{\sigma_E})\sigma_Y} \tag{3.26}$$

Where

- σ_x is the axial stress
- σ_{xcr} is the critical stress for the plate/stiffener in pure compression
- σ_b is the design bending stress
- σ_Y is the yield stress
- σ_E is the Euler buckling stress for plate/stiffener

The method for finding the critical stress for pure compression is the same as the one described for columns in Equation(3.27).

$$\frac{\sigma_{xcr}}{\sigma_Y} + \frac{\sigma_{xcr}}{1 - \frac{\sigma_{xcr}}{\sigma_E}} \frac{w_{eq}A}{\sigma_Y W} = 1 \tag{3.27}$$

w_{eq} is the equivalent imperfection accounting for the true out-of-straightness and the effect of fabrication stresses, and l is the member length. The critical axial stress can be calculated by introducing a factor, μ.

$$\mu = \frac{w_{eq}A}{W} = 0.0015 l \frac{z}{i^2} \tag{3.28}$$

Now the critical stress can be found from Equation (3.29)

$$\frac{\sigma_{xcr}}{\sigma_Y} = \frac{1 + \mu + \bar{\lambda}^2 - \sqrt{(1 + \mu + \bar{\lambda}^2) - 4\bar{\lambda}^2}}{2\bar{\lambda}^2} \tag{3.29}$$

$$\bar{\lambda} = \sqrt{\frac{\sigma_Y}{\sigma_E}} \quad , \quad \sigma_E = \frac{\pi^2 E i_e^2}{l_e^2} \tag{3.30}$$

i_e, the effective radius of gyration, is defined by

$$i_e = \sqrt{\frac{I_e}{A + b_e t}} \tag{3.31}$$

The effective moment of inertia can be written as

$$I_e = I + e^2 A(l + \frac{A}{b_e t})^{-1} \tag{3.32}$$

With

- e being the eccentricity of the stiffener to the plate flange
- I the moment of inertia of the stiffener without the plate flange
- b_e the effective width of the plating
- t the plate thickness

In the case of plate-induced failure, there is a displacement of the neutral axis resulting from a reduction in effective width which is illustrated in Figure 3.1.15. This displacement introduces additional eccentricity for the plate or stiffener, which must be considered in the analysis.

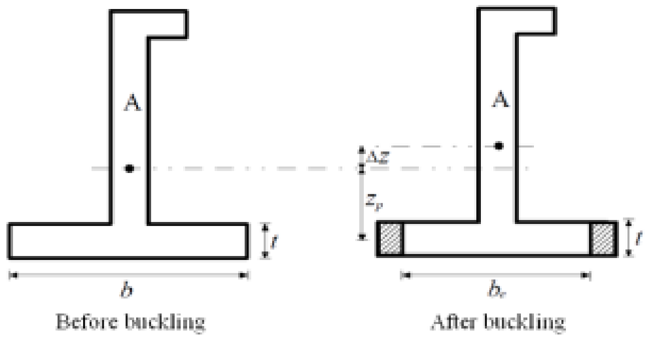


Figure 3.1.15: Shift of Effective Neutral Axis After Plate Buckling [24]

The shift can be calculated by Equation (3.33). The effective buckling length is influenced by lateral pressure.

$$\Delta z = z_p \frac{(b - b_e)t}{A + bt} = z_p(1 - \frac{A + b_e t}{A + bt}) \tag{3.33}$$

In the absence of lateral pressure, the effective length is considered to be equivalent to the frame spacing. However, when lateral pressure is present, two failure modes must be considered: asymmetric buckling and symmetric buckling relative to the frame. Generally, the over-pressure may be on either the plate side or the stiffener side. This gives four potential buckling modes as shown in Figure 3.1.16.

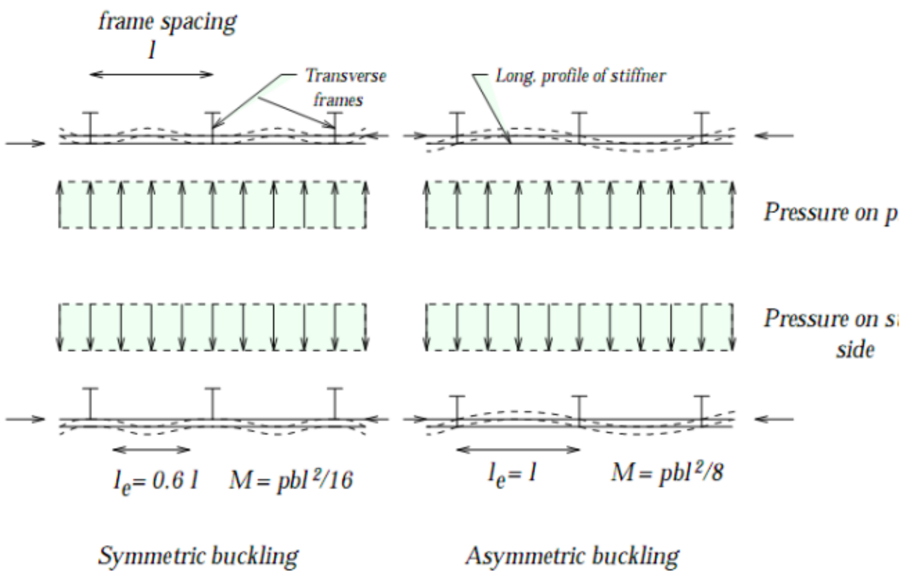


Figure 3.1.16: Buckling Modes Plate/Stiffener [24]

3.1.8 Buckling of Stiffeners - DNV RPC201

3.1.8.1 Equivalent Load Effects

The equivalent axial force combines the actual axial force with a tension field action. The tension field concept enables shear stresses to exceed the critical stress level, τ_{crl}, for plate shear buckling occurring between stiffeners and girders. These shear stresses are supported by tension forces among the stiffeners, as depicted by the shaded areas in Figure 3.1.18. The additional shear force that the stiffener must support through compression is calculated as follows [24]:

$$N_\tau = (\tau - \tau_{crg})st \tag{3.34}$$

Keep in mind that τ_{crg} refers to the critical shear stress between the girders when the stiffeners are not present. We use $\tau_{\text{crg}} < \tau_{\text{crlr}}$ because, in the post-buckled condition, we assume that the plate can only bear the shear force between the girders.

It's worth noting that the calculation of the axial force is based on the stresses acting on the entire plate flange, as derived from linear elastic analysis. The axial capacity, on the other hand, is determined using the effective plate flange.

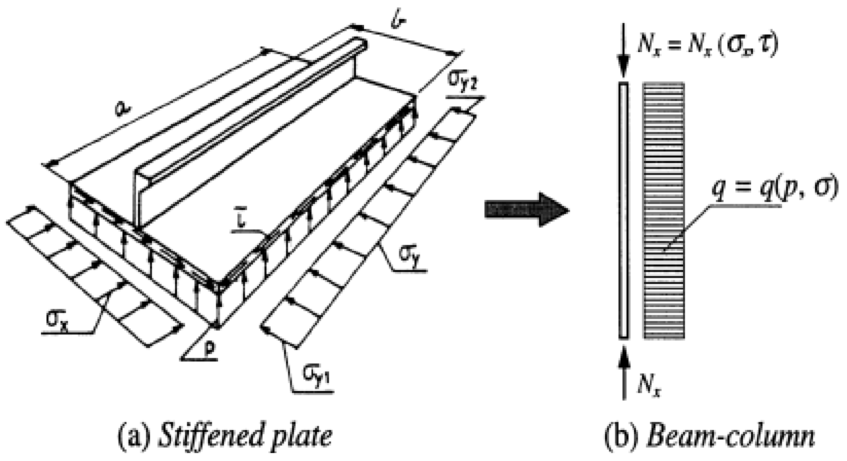


Figure 3.1.17: Equivalent beam-column model of stiffened plate [24]

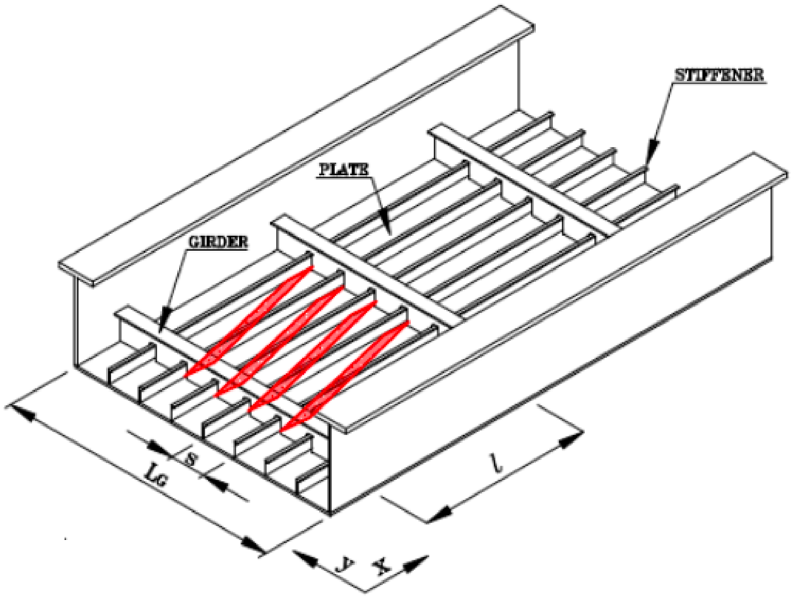


Figure 3.1.18: Tension Field in Stiffened Plate [24]

In addition, the transverse stresses, σ_y, are considered to have a driving effect on plate and stiffener buckling. The buckling stress of a transversely stiffened plate is given by Equation (3.35).

$$\sigma_{yE} = \frac{\pi^2 D}{1^2 t} \left[\left(\frac{ml}{L_G} + \frac{L_G}{ml} \right)^2 + i_s \left(\frac{L_G}{ml} \right)^2 \right] \tag{3.35}$$

3.1.9 ABAQUS

Abaqus was used to develop and analyze the stiffened panel. Its vast element library provides a powerful set of tools for solving a number of different problems.

Most of the theory that ABAQUS is built on that is presented in this section is taken from the ABAQUS documentation [27].

Each element in ABAQUS can be characterized by 5 different categories:

- Family
- Degrees of Freedom
- Number of Nodes
- Formulation
- Integration

The most common families in ABAQUS are shown in Figure 3.1.19, where it can be seen that the main difference between the families is if they are one, two, or three-dimensional.

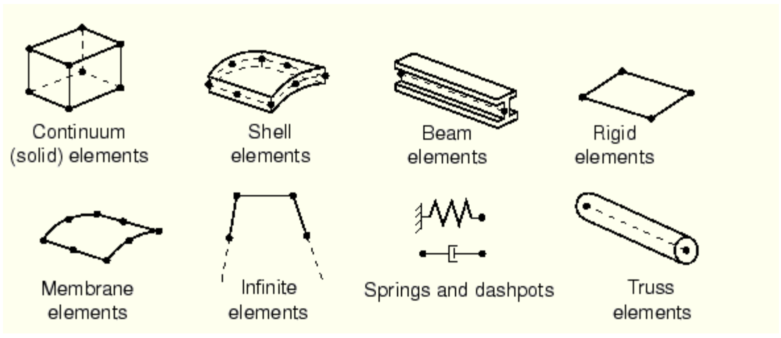


Figure 3.1.19: Element Families [27]

The degrees of freedom serve as the essential variables computed during the analysis. In stress/displacement simulations, they represent the translations and, in the case of shell and beam elements, the rotations at each node. Displacements and rotations are determined exclusively at the nodes of the element, and at other points within the element, the displacements are derived through interpolation from the nodal displacements. An element's formulation refers to the mathematical theory utilized to define the element's behavior. In ABAQUS/Explicit, in the absence of adaptive meshing, all deformable elements are based on the Lagrangian or material description of behavior, where the element deforms with the material. Alternatively, elements in the Eulerian or spatial description are stationary in space as the material flows through them. ABAQUS incorporates numerical techniques to integrate various quantities across the volume of each element. Gaussian quadrature is typically employed by ABAQUS to evaluate the material response at each integration point in each element [27].

3.1.9.1 Shell Elements

The element family that will be introduced here is shell elements. Shell elements are used to model structures where one dimension, in this case the thickness, is significantly smaller than the other dimensions and the stresses in the thickness direction are negligible [27].

The ABAQUS Shell Element Library allows for the modeling of curved, intersecting shells that can demonstrate nonlinear material response and can undergo significant translations and rotations. The library can be categorized into three sections: general-purpose, thin, and thick shell elements. Thick shell elements are based on Mindlin shell theory, thin shell elements are derived from classical Kirchoff shell theory, and general-purpose shell elements provide solutions for both thin and thick shell elements. As a result, general-purpose shell elements are the preferred choice for most applications and are the category of shell elements recommended by ABAQUS [28].

3.1.9.2 Buckling in Abaqus

This section is the same as in the author's project thesis [25].

The determination of the buckling load can be achieved through FEA using software like ABAQUS [28]. This analysis has the objective of extracting both the eigenvalues and the corresponding eigenvectors, where the eigenvalues signify the buckling load and the eigenvectors represent the associated mode shape. It's important to emphasize that the eigenvectors are normalized, indicating that they express relative displacement values rather than exact displacements. To determine the buckling load, the initial conditions are scaled by the eigenvalues since the eigenvalue acts as a load multiplier. This analytical approach relies on a linear perturbation method to estimate the critical buckling load for a given structure. The buckling analysis procedure revolves around identifying the load that results in a singular stiffness matrix, thus requiring the solution of the following eigenvalue problem:

$$K^{NM}v^{M} = 0 \tag{3.36}$$

In this context, K^{NM} denotes the stiffness matrix, while v^M stands for nontrivial displacement solutions. During the eigenvalue-based buckling prediction stage, an incremental loading pattern, Q^N, is established. This pattern is then adjusted by load multipliers λ_i, rendering its specific magnitude irrelevant. The general equation for the eigenvalue problem is as follows:

$$(K_0^{NM} + \lambda_i K_\Delta^{NM})v_i^M = 0 \tag{3.37}$$

The stiffness in the base state is denoted by the symbol K_0^{NM}, while the differential stiffness is represented by K_Δ^{NM}. The base state stiffness is the result of a combination of hypoelastic tangent stiffness, initial stress stiffness, and load stiffness, as described by ABAQUS.

The eigenvalues λ_i serve as multipliers, providing an estimate of the generalized buckling load as $P^N + \lambda_i Q^N$, while the corresponding eigenvectors v_i^N illustrate the associated buckling modes.

Though the lowest mode is typically the primary concern in most analyses, ABAQUS is capable of extracting multiple modes simultaneously. Furthermore, it is noteworthy that ABAQUS can easily handle the common scenario of an antisymmetric buckling mode on a symmetric base state and buckling load.

If the prediction of the tangent stiffness is inaccurate using $K_0^{NM} + \lambda_i K_\Delta^{NM}$, a nonlinear analysis utilizing the Riks method becomes necessary to obtain a reliable estimate of the structure's load-carrying capacity.

3.2 Machine Learning by Artificial Neural Network

According to the textbook "Machine Learning" by Zhi-Hua Zhou, Machine Learning can be described as "The technique that improves system performance by learning from experience via computational methods" [29]. It is a broad term which is including all algorithms that can extract patterns from a data set.

This chapter will mainly focus on Machine Learning (ML), surrogate models, Artificial Neural Networks (ANN) and an introduction to Polynomial Chaos Expansion (PCE). The a goal is to establish a high-level overview and the most important aspects of surrogate modelling, in addition to an introduction to machine learning.

3.2.1 Machine Learning

In the following section, the aim is to provide an intuitive definition of a machine learning problem, which forms the core concept behind self-learning computational units like ANN. The problems can be broken into three essential components: the task, the performance measure, and the experience [30]. Parts of the following section are extracted and rewritten from the author's project thesis [25].

3.2.1.1 Task

When designing projects and solving problems in machine learning, it is important to define the task, which refers to the object the model is set to accomplish [30].

Regression is the foundational method for establishing a trendline from a dataset. It aims to predict a continuous numerical output by identifying a relationship between input variables and the target variable. The algorithm generates a function to predict the output value based on the input values. This function can be linear or non-linear, with several trainable parameters to represent complex problems. Thus, regression is not only the most basic but also one of the most versatile tasks an algorithm can perform.

On the other hand, classification is another widely used method that shares similarities with regression. However, instead of producing a continuous range of values, it predicts the class to which an input example belongs. In classification tasks, a machine learning model assigns a label or class to each input data point based on patterns learned from the training data. For instance, in an ANN, classification can be used to categorize input images of animals, such as distinguishing between dogs and cats.

3.2.1.2 Performance

The performance measure is a crucial aspect of a machine learning problem, defining how the success or quality of the model's predictions or outcomes will be evaluated. It quantifies the effectiveness of the machine learning algorithm in performing the specified task. This measure is related to the task at hand and is a function of the discrepancy between the predicted and actual values in the dataset.

For problems that yield continuous outputs, performance measures typically include cost functions such as mean squared error (MSE), R-squared (R^2), and mean absolute error (MAE), which will be discussed in more detail later. These metrics assess the accuracy and goodness of fit between the predicted values and the actual observed values.

Selecting the appropriate performance metric is essential in machine learning because it influences how the model is trained and evaluated. It allows for the comparison of different models, adjustment of parameters, and determination of the model's suitability for a particular application. The choice of performance measure depends on the nature of the problem, as well as the specific objectives and requirements of the task [30].

3.2.1.3 Experience

Experience in machine learning refers to the set of variables that characterize a computational unit, derived from the data points it has been trained on and the training process itself. This experience enables the model to improve its performance on a specified task over time.

Machine learning models learn from experience by analyzing and processing a dataset containing relevant examples or instances. This dataset includes input data and corresponding target values, which inform the model about relationships and patterns in the data. The learning process involves iteratively adjusting the model's internal parameters or structure to better fit the observed patterns in the training data, allowing the model to make more accurate predictions or classifications when presented with new, unseen data.

There are two primary approaches to gaining experience: supervised learning and unsupervised learning. In supervised learning, the model learns from labeled examples, where the correct answers are provided alongside the input data during training. The model's goal is to generalize from these labeled examples to make accurate predictions for unseen data. In contrast, unsupervised learning involves the model learning from unlabeled data, seeking to identify hidden patterns or structures within the data. For ANNs, supervised learning is the preferred method [31].

3.2.2 Surrogate Models

Surrogate/approximation/metamodeling, is the key to surrogate assisted optimization. Surrogate models are typically created as an affordable alternative to a costly-to-evaluate function $\mathcal{M}$. A surrogate model, denoted as $\hat{\mathcal{M}}$, is usually configured using a specific set of N evaluation points, called the training set or experimental design as described by Thomas Sauder in his PhD thesis [32]. When discussing surrogate models, the two parameters ε and $\mathcal{F}$ are defined as

$$\varepsilon := (x^{(1)}, x^{(2)}, ..., x^{(N)})^T \tag{3.38}$$

$$\mathcal{F} := [\mathcal{M}(x^{(1)}), \mathcal{M}(x^{(2)}), ..., \mathcal{M}(x^{(N)})]^T \tag{3.39}$$

From the values of $(\varepsilon, \mathcal{F})$, $\hat{\mathcal{M}}$ can be used to predict the value of $\mathcal{M}(x)$ at points x that are not included in ε. An important point to emphasize in this chapter is that suitable surrogate models can also be utilized to study specific properties of $\mathcal{M}$.

The simplest example of a surrogate model is the linear interpolator between the evaluation points. In this case, $\hat{\mathcal{M}}$ is exactly equal to $\mathcal{M}$ at the evaluation points. A linear regressor provides an alternative type of surrogate model. In this case, an assumption is made regarding the overall behavior of $\mathcal{M}$ (that it is linear across the entire range of evaluation points), which goes beyond the requirement for $\hat{\mathcal{M}}$ to be exactly equal to $\mathcal{M}$ at the evaluation points. There are a number of different surrogate modeling techniques; however, a common aspect is that they are all parameterized functions, and their parameters are optimized in some way to ensure that $\hat{\mathcal{M}}$ mimics the original function $\mathcal{M}$.

3.2.3 Artificial Neural Networks

ANNs have become a powerful tool in various fields in recent years, used for tasks such as data categorization, pattern recognition, and prediction. ANNs have proven to be as effective as traditional statistical models [29].

The main advantage of ANNs is their ability to handle data quickly, efficiently, and reliably. They excel in accuracy, processing speed, latency, overall performance, fault tolerance, large data handling, scalability, and problem-solving. This makes them especially suitable for complex tasks like image recognition and natural language processing. What makes ANNs unique is their capacity for independent learning, adaptation, fault tolerance, processing non-linear information, and effective input-to-output mapping. Their widespread use in various numerical fields today highlights these outstanding characteristics [31].

Based on the human brain, ANNs have acquired many favorable traits over time [29]. While most computers now outperform humans in most numerical simulations, the human brain still excels in other areas. Humans can rapidly recognize faces even in challenging conditions such as poor lighting, comprehend speech in noisy environments, and most importantly, the human brain has the ability to learn. This is the foundation of ANNs, inspired by early models of sensory processing by the brain, specifically the neural structure of the central nervous system.

The brain's calculations take place through a highly intricate and interconnected network of neurons. These neurons communicate by sending electrical signals along neural pathways, which consist of axons, synapses, and dendrites. Artificial neurons seek to mimic this process, as shown in Figure 3.2.1. While a biological neuron has dendrites to receive signals, a cell body to process signals, and an axon to transmit signals to other neurons, an artificial neuron has multiple input channels, a processing stage, and a single output that can connect with multiple other artificial neurons [31].

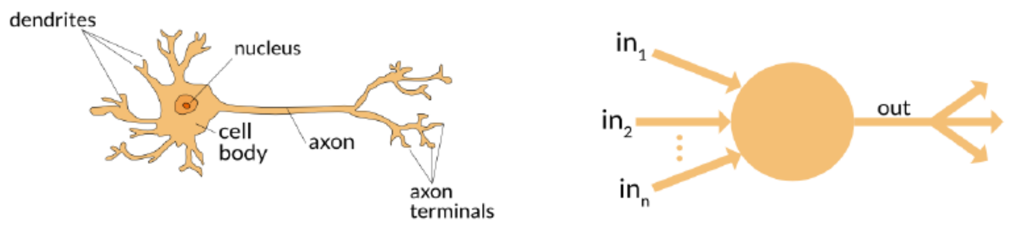


Figure 3.2.1: Brain Neuron vs Artificial Neuron [31]

3.2.3.1 The Neuron

The concept of the artificial neuron originates from the work of McCulloch and Pitts in 1943 [33] and is shown in Figure 3.2.2. This concept can be divided into three parts: weighting, summing, and activation. McCulloch and Pitts modeled a neuron as a switch that receives inputs from other neurons and, based on the total weighted input, either activates or remains inactive. Inspired by Hebb's design of the single-layer neuron, Rosenblatt later developed the multi-layer neuron shown in the Figure [34].

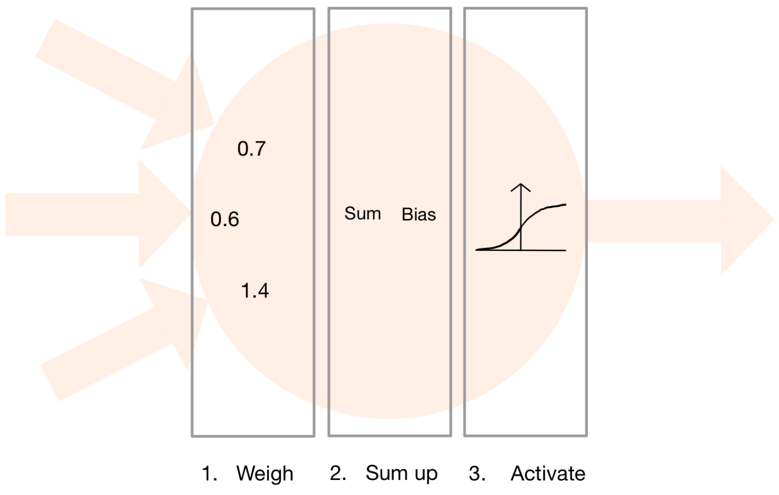


Figure 3.2.2: Inside an artificial neuron

When a signal enters the neuron, it is assigned a weight value and then multiplied by it. In this example, the neuron has three inputs, each assigned a distinct weight. These weights are adjusted based on the errors observed during the learning phase. Next, the weighted input signals are summed into a single value, to which an offset known as bias is added [31]. This can be formulated as Equation (3.40).

$$z = w_1x_1 + w_2x_2 + ... + w_nx_n + b_1 \tag{3.40}$$

3.2.3.2 The Network

To recognize and analyze intricate patterns within a dataset, multi-layer neurons play a crucial role. An ANN is formed by combining multiple artificial neurons. For complex problems, the network can be structured with numerous parallel nodes per layer, potentially integrating multiple layers based on the issue's complexity.

Effectively training an ANN begins with establishing the correct outputs for a series of inputs from a sizable and representative dataset. Following this, appropriate values are assigned to the weight matrices and biases, and the inputs are passed through the network. An error function is then employed to evaluate the network's performance for single or multiple inputs, and adjustments to the weight matrices and bias vectors are made accordingly. Once trained, the network can forecast outcomes for inputs similar to those in the training data. A fundamental comprehension of these principles is essential for setting up and efficiently training an ANN, as detailed in the subsequent section.

3.2.4 Modelling the Network

This section will explain the key concepts and elements behind modeling Artificial Neural Networks.

3.2.4.1 Training and Test Data

In order for an ANN to perform effectively, it needs to be trained using a comprehensive dataset. The engineer's decisions regarding the data are pivotal in constructing a capable ANN. By adjusting the offsets and weights for each node, the network is trained, and prediction errors are minimized.

Training an ANN successfully necessitates a dataset that is both sufficiently large and representative. This dataset should be labeled, indicating that the output for each data point is known. Typically, the available data is split into two subsets: the training set and the testing set. The training set, usually the largest, is utilized to train the network, while the test set assesses the effectiveness of the trained network. This evaluation is carried out by computing the error of the test data outputs from the trained network. The process of choosing and dividing the data into sets is not fixed but relies on experience and experimentation. A common approach is to use an 80%/20% or 90%/10% split for training and testing [35].

The size of the dataset needs to be sufficiently large to accurately represent the behavior of the task. Insufficient volume will result in significantly higher testing errors compared to training errors. Furthermore, the domain of the data is critical for the optimal performance of an ANN. It needs to be broad and diverse enough to prevent predictions outside of the training domain.

3.2.4.2 Activation Function

An activation function applies a function to the combined output and passes the resulting value, aiming to imitate the activation/deactivation process. The state of a neuron is determined by the activation function, which computes the combined input [36].

After the signals are weighted and summed, the weighted sum is compared to a threshold or bias to decide whether the neuron is activated, as shown in Figure 3.2.2 [29].

The activation function is nonlinear, which allows the ANN to capture non-linear patterns in the data. This ability enables it to recognize trends such as parabolic or tangential patterns in the dataset. According to Cybenko's Universal Approximation Theorem, an ANN with a single hidden layer and a nonlinear activation function can approximate any non-linear relationship [37].

The range is an essential factor when choosing the activation function. Depending on the function, the range of the function may be in the interval $[0, 1]$, $[-1, 1]$, or $[0, \infty >$. Activation functions with a small range are not ideal for the hidden layers as they can slow down the learning rate. However, they have advantageous properties for the output layer, contributing to improved stability. This section will introduce some of the most commonly used neuron activation functions.

The step function is the most basic type of activation function in neural networks, generating a binary output based on the input. If the input exceeds a specific threshold, the output is '1'; otherwise, it is '0'. This function serves as an on/off switch and does not support partial activation, which can be limiting when a neuron needs to capture complex, non-binary relationships in the data. Initially used in early neural network models like the perceptron for binary classification due to its simplicity, the step function is not suitable for modern, advanced networks. The step function is shown in Equation (3.41) and Figure 3.2.3 [38].

$$H(z) = \begin{cases} 1, & \text{for } z \geq 0 \\ 0, & \text{for } z < 0 \end{cases} \tag{3.41}$$

The logistic function, also known as the sigmoid function, generates an S-shaped curve that converts any real-valued input into a value ranging from 0 to 1. This property is particularly valuable for evaluating probabilities or levels of confidence. Unlike the step function, the sigmoid function offers a smooth transition in its output, which can be understood as the likelihood of a neuron being activated. This is especially beneficial for binary classification tasks, such as determining whether an email is spam. The smooth gradient of the sigmoid function also simplifies the backpropagation process, improving the effectiveness of neural network training. However, the sigmoid function has a tendency to become saturated for very high or very low input values, leading to extremely small gradients. This saturation can significantly slow down the learning process, as adjustments to weights and biases become minimal. Additionally, the output of the sigmoid function is not centered around zero, which can result in zig-zagging of gradient updates during training [38].

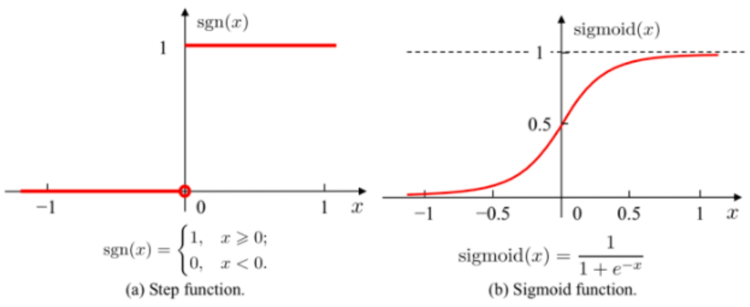


Figure 3.2.3: Step function and Sigmoid function [31]

The hyperbolic tangent function, often referred to as the tanh function and shown in Figure (3.42), yields a result within the -1 to 1 range. An important characteristic of the tanh function is its centering at zero, resulting in an output spanning from -1 to 1 with zero as the midpoint. This property helps mitigate the vanishing gradient problem that may arise during deep neural network training. The zero-centered nature of the tanh function promotes more reliable and effective training. The shape of the tanh function is reminiscent of the sigmoid function, exhibiting an S-shaped curve. However, unlike the sigmoid function, the tanh function provides an output range of [-1, 1] rather than [0, 1]. This zero-centered range can be advantageous for specific neural network architectures, as shown in Figure 3.2.4.

The tanh function's output range makes it particularly valuable in situations where a neuron's output must encompass both positive and negative values [38].

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3.42}$$

The last activation function presented here is The Rectified Linear Unit (ReLu), which can be written mathematically as in Equation (3.43) [38].

$$ReLu(z) = Max(0, x) \tag{3.43}$$

The ReLU function, portrayed in Figure 3.2.4, is a commonly utilized activation function in neural networks. It functions by producing the input value if it is positive, and zero if it is negative or zero. This results in a linear function for all positive values and introduces non-linearity at x=0, making it suitable for complex tasks.

In comparison to other non-linear functions like sigmoid or tanh, ReLU is computationally efficient because of its straightforward thresholding at zero. This results in sparse activation, where only a subset of neurons is activated at any given time. Sparse activation improves model efficiency and reduces the risk of overfitting. Additionally, ReLU helps tackle the vanishing gradient problem, which arises when gradients are too small for effective learning during backpropagation. This problem is more pronounced with functions like sigmoid or tanh, but ReLU helps mitigate it.

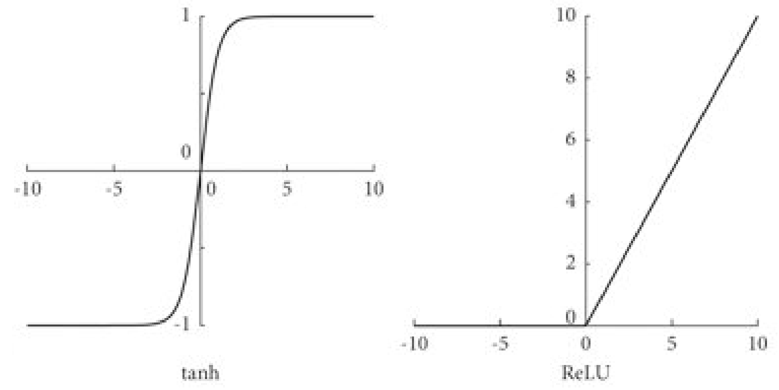


Figure 3.2.4: tanh and ReLu function [39]

3.2.4.3 Cost Functions

The metric Mean Squared Error (MSE) is commonly used in regression tasks. It involves calculating the average of the squared differences between predicted and actual values. The equation for MSE is provided in Equation (3.44). MSE is beneficial because it gives more weight to larger errors, making it useful for minimizing large errors. In neural networks for regression tasks, MSE is often utilized as the loss function, and during training, the network's weights are adjusted to minimize it [40].

Nevertheless, MSE does have a drawback, which is its susceptibility to outliers. If the data contains outliers, the presence of large errors can dominate the MSE, potentially resulting in overfitting or a model that overly focuses on the outliers.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.44}$$

R-Squared quantifies the proportion of the variance in the dependent variable that is predictable from the independent variables and is expressed as Equation 3.45. It is a relative evaluation of fit and offers insight into how much better the model performs compared to a simple average. An R-squared value of 1 signifies perfect prediction, while 0 indicates that the model performs no better than a model that consistently predicts the mean value. While R-squared is commonly used to evaluate regression models, it should be approached with caution. A high R-squared does not always indicate a good model, particularly if the model is overly complex and overfits the data. Additionally, R-squared does not consider the number of predictors in the model and can sometimes lead to misinterpretations when irrelevant predictors are included [40].

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \tag{3.45}$$

The Mean Absolute Error (MAE) is a measurement used to calculate the average of the absolute variances between the predicted and actual values and is expressed as shown in Equation (3.46). This metric offers a straightforward indication of the average error made by the model in the units of the variable being predicted. Unlike MSE, MAE treats all errors equally and is not overly influenced by outliers. This characteristic makes it a dependable measure of model performance, particularly when working with datasets containing anomalies or outliers. Furthermore, MAE is commonly employed as a loss function when training regression models [40].

$$MAE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i) \tag{3.46}$$

3.2.4.4 Topology

The structure of an ANN is illustrated in Figure 3.2.5. The network's nodes can be interconnected in various ways, resulting in complex behavior. The most basic topology is the feed-forward network, which typically consists of input layers, output layers, and potentially one or more hidden layers. In a feed-forward network, signals move in only one direction without loops. Determining the network's architecture involves establishing the node count within each layer, the number of layers in the network, and the connections between nodes. Initially, these parameters are often set based on intuition and then refined through multiple rounds of experimentation [29].

The first layer in a neural network is the input layer, which receives the input data. Each neuron in the input layer represents a feature of the input data. The input layer typically does not perform any computations or transformations on the data; it simply passes the data to the next layer in its raw form.

The layers between the input and output layers are known as the hidden layers. Neurons in the hidden layers carry out various computations and transformations on the input data. Each neuron in a hidden layer receives input from multiple neurons of the previous layer, applies a weighted sum followed by an activation function, and passes the result to the next layer. The term "hidden" is used because these layers are not directly exposed to the input or output, but they are where the network learns to interpret the input data, extracting features and patterns necessary for accurate predictions. The number and size of the hidden layers determine the network's complexity and capability [29].

The final layer in a neural network is the output layer. It receives input from the previous layer and converts it into a format suitable for the network's intended output. The number of neurons in this layer usually corresponds to the number of classes or outputs the network is designed to predict.

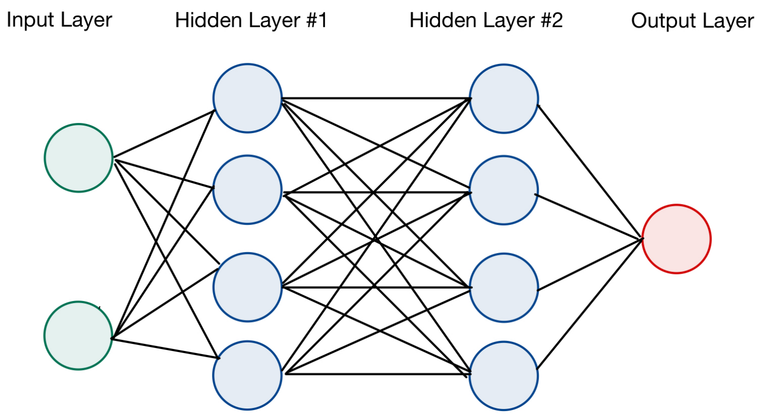


Figure 3.2.5: General layout of ANN

3.2.4.5 Fitting

Fitting a model is a crucial step in machine learning, involving training the model with a dataset to determine the best possible mapping of inputs to outputs. This process entails adjusting the model's parameters to accurately capture the underlying patterns and relationships in the data. Model fitting aims to calibrate the model to produce precise predictions for new, unseen data. The quality of model fitting significantly impacts the success of any machine learning algorithm. Well-fitted models can effectively achieve the objectives of machine learning tasks by accurately predicting outcomes.

Model fitting goes beyond just minimizing errors on the training data; it also involves finding the right trade-off between simplicity and complexity. Striking this balance is vital to ensure that the model performs well not only on seen data but also on new data. The primary challenges in this process are underfitting and overfitting.

Underfitting occurs when the model is too simplistic to capture the underlying data patterns, often observed in linear models dealing with complex or nonlinear data. An underfitted model performs poorly on both training and testing data, resulting from insufficient parameters or layers to learn the data's structure. Underfitting can also result from inadequate training, which can be addressed by adding extra neurons and layers and extending the training time by increasing the number of epochs.

Overfitting happens when the model learns the training data too well, leading to a model that captures patterns specific only to the training set and does not generalize well to unseen data. Overfitting is often caused by insufficient data, overtraining, or an excessive number of parameters relative to the number of observations. Figure 3.2.6 illustrates the concepts of underfitting and overfitting in a model.

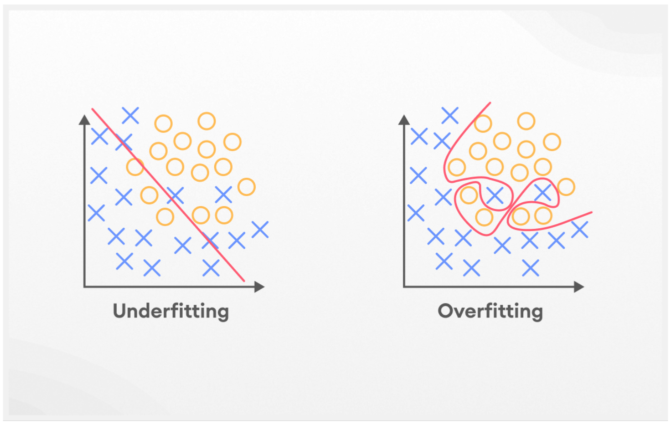


Figure 3.2.6: Underfitting and Overfitting [41]

3.3 Polynomial Chaos Expansion

In modern engineering, uncertainty quantification is becoming increasingly important. Instead of using deterministic scenario-based predictive modeling, there is a gradual shift towards stochastic modeling to accommodate the inherent uncertainty in physical phenomena and measurements. However, this transition requires dealing with significantly larger amounts of information, such as when using Monte-Carlo simulation, leading to the need for repetitive and costly computational model evaluations.

Polynomial chaos expansions (PCE) represent a potent metamodelling technique that seeks to provide a functional approximation of a computational model by using its spectral representation on a suitably constructed basis of polynomial functions.

Most of the theory described in this section is taken from the documentation of the PCE-program UQLab [42] and the Article "Application of polynomial chaos expansions in stochastic analysis of plate elements under lateral pressure" by Gaspar and Guedes Soares [12].

3.3.1 General About PCE

Consider a structural system for which the response is obtained as the solution of a deterministic numerical model $\mathcal{M}$, which can be

$$Y = \mathcal{M}(x) \tag{3.47}$$

of a n-dimensional vector $x = \{x_1, ..., X_M\}^T$ of input variables, and vector of quantities of interest provided by the model $y = \{y_1, ..., x_M\}^T$, which is referred to as the model response in the sequel.

In analyzing the randomness of structural systems, the input variables of the numerical model are treated as random variables, representing the uncertainties in the essential properties of the structure (such as dimensions, material properties, and loads). A joint probability density function (PDF) f_X is linked to the set of input random variables. It is also assumed that the elements of this set are independent, allowing f_X to be defined as the product of the individual marginal PDFs f_{Xi} (where i = 1, ..., n) [12]. The uncertainty or randomness in the input variables will propagate through the deterministic numerical model in Equation (3.47), resulting in an output response variable that is a scalar random variable with PDF f_Y describing the structural system's response (such as displacements and stress components). The output random variable of the numerical model can be expressed in a suitable space spanned by a PCE basis. This basis is a series of multivariate orthogonal polynomials concerning the joint PDF of the input random variables [43]. This PCE representation can be formulated as:

$$Y = \mathcal{M}(X) = \sum_{\alpha \in \mathbb{N}^M} \mathbf{a}_\alpha \boldsymbol{\Psi}_{\mathbf{a}}(\mathbf{X}) \tag{3.48}$$

where $\mathbf{a_a} : \mathbf{a} \epsilon N^M$ are coefficients of the multivariate orthonormal polynomials ψ_a to be determined and $a = a_1, ..., a_n$ are vectors with ordered lists of integers defining the indices of the expansion.

3.3.2 Orthonormal Polynomials

The core idea behind polynomial chaos expansion is to represent a random variable or a stochastic process as a series expansion over a set of polynomial basis functions. These polynomials are chosen to be orthonormal with respect to the probability density function of the input random variables. This orthonormality ensures that the polynomials are uncorrelated, which simplifies the computation of the coefficients in the series expansion.

Different families of orthonormal polynomials exist and are used as the basis for PCE and are dependent on the distribution of the input variables [42].

- Hermite polynomials for Gaussian distributions, $X \sim \mathcal{N}(0, 1)$
- Legendre polynomials for uniform distributions, $X \sim \mathcal{U}(-1, 1)$
- Laguerre polynomials for exponential distributions, $X \sim \Gamma(1, k)$
- Jacobi polynomials for beta distributions, $X \sim \mathcal{B}(r, s, -1, 1)$

3.3.3 Multivariate Orthonormal Polynomials

The multivariate polynomials from Equation (3.48) are then assembled as the tensor product of their univariate counterparts. This set of multivariate polynomials forms a basis where other functions can be represented [44]. Given n independent input variables, creating the multivariate polynomial basis by tensor product of the univariate polynomials is a possibility according to:

$$\Psi_\alpha(\mathbf{x}) = \prod_{i=1}^{n} \phi_{\alpha_i}^{(i)}(\mathbf{x}_i) \tag{3.49}$$

where ϕ_{α_i} is an univariate polynomial of degree α_i in the input variable $x_i(i = 1, ..., n)$.

Due to the orthonormality relations in the polynomials, it follows that also the multivariate polynomials thus constructed are orthonormal:

$$\langle \Psi_\alpha(x), \Psi_\beta(x) \rangle = \delta_{\alpha\beta} \tag{3.50}$$

with $\delta_{\alpha\beta}$ being the Kronecker symbol to the multi-dimensional case [42].

3.3.4 Truncated PCE

The explicit representation of the numerical model output random variable can be used as a surrogate for the true deterministic numerical model $\mathcal{M}$, allowing the solution of the underlying uncertainty quantification problem at reduced computational cost. The exact PCE representation of the true numerical model response requires an infinite series expansion of polynomials. However, an infinite series representation cannot be implemented in practice and therefore a truncated series has to be considered. Different truncation schemes can be adopted for this purpose.

3.3.4.1 Basis truncation schemes

Given the polynomials listed above, it is straightforward to define a “standard truncation scheme”, which corresponds to all polynomials in the M input variables of total degree less than or equal to p:

$$\begin{aligned} \mathcal{A}^{M,p} &= \{\alpha \in \mathbb{N}^M : |\alpha| \leq p\} \\ \mathcal{A}^{M,p} &\equiv P = \begin{pmatrix} M+p \\ p \end{pmatrix} \end{aligned} \tag{3.51}$$

Several additional truncation schemes can be built that are better suited to various types of applications.

3.3.4.2 Maximum interaction truncation scheme

This truncation scheme is based on choosing a subset of the terms defined in Equation (3.51), such that the α's have at most r non-zero elements (low-rank α):

$$\mathcal{A}^{M,p,r} = \{\alpha \in \mathcal{A}^{M,p} : ||\alpha||0 \leq r\} \tag{3.52}$$

where $||\alpha||_0 = \sum_{i=1}^{M} 1_{\alpha_i > 0}$ is the rank of the multi-index α. This truncation scheme can be used to significantly reduce the cardinality of the polynomial basis by limiting the number of interaction terms, which is particularly effective in high dimension.

3.3.4.3 Hyperbolic truncation scheme

A modification of the standard scheme, the hyperbolic (or q-norm) truncation scheme was proposed by Blatman and Sudret (2010) [44]:

$$\mathcal{A}^{M,p,q} = \{\alpha \in \mathcal{A}^{M,p} : ||\alpha||_q \leq p\} \tag{3.53}$$

where:

$$||\alpha|| = (\sum_{i=1}^{M} \alpha_i^q)^{1/q} \tag{3.54}$$

Note that for q = 1 hyperbolic truncation corresponds exactly to the standard truncation scheme in Equation (3.51). For q < 1, hyperbolic truncation includes all the high-degree terms in each single variable, but discourages equivalently high order interaction terms. An example of the behaviour of the hyperbolic norm in two dimensions for different values of p and q is shown in Figure 3.3.1.

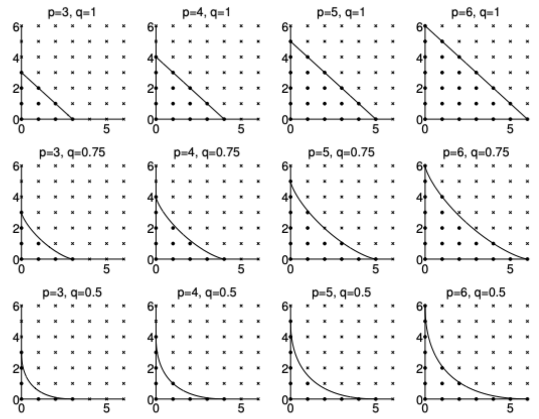


Figure 3.3.1: Behaviour of the hyperbolic norm [42]

3.3.5 Coefficients

In the truncated PCE representation in Equation (3.48) the coefficients $\{\mathbf{a}_\alpha : \boldsymbol{\alpha} \in \mathrm{A} \subset \mathrm{N}^n\}$ related with $\mathbf{\Psi}_\alpha(\mathbf{X})$ can be determined using intrusive and non-intrusive techniques.

3.3.5.1 Projection Approach

By exploiting the orthonormality of the PC basis the coefficients can be calculated by using the projection approach. By taking the expectation of Equation (3.48) multiplied by $\psi_a(\mathbf{X})$ one gets the expression of each coefficient a_a [44].

$$\begin{aligned} a_a &= \langle \mathcal{M}(\mathbf{X}), \Psi_a(\mathbf{X}) \rangle_{L2} = E[\mathcal{M}(\mathbf{X})\Psi_\alpha(\mathbf{X})] \\ &= \int_{D_X} \mathcal{M}(\mathbf{x})\Psi_\alpha(\mathbf{x}) f_X(\mathbf{x}) d\mathbf{x} \end{aligned} \tag{3.55}$$

The resulting multidimensional integral can be computed by utilizing quadrature schemes. This can be schemes based on random sampling like Monte Carlo, or multivariate Gauss quadrature techniques like full tensor product quadrature. However, the projection approach is computationally expensive due to it often requiring a large set of realizations of the input random variables to provide sufficient accuracy in addition to the cost of the quadrature techniques heavily increasing with the number of input parameters [44].

3.3.5.2 Regression Approach

The regression approach consists of minimizing the mean square error, and was proposed by Berveiller et al. (2006) [45]. These regression methods are more efficient, and have the upper hand over other techniques due to not needing implementations at the level of the numerical model formulation.

The approach starts by considering the PC expansion described in Equation (3.56), ordered degree p.

$$\mathcal{M}_p(\mathbf{X}) = \sum_{0\leq|a|\leq p} a_a \Psi_a(\mathbf{X}) \tag{3.56}$$

When rewritten, it becomes:

$$\mathcal{M}_p(\mathbf{X}) = a^\top \Psi(\mathbf{X}) \tag{3.57}$$

The coefficients of the truncated PCE representations are then determined as the solution of the underlying least squares minimization problem:

$$\hat{\boldsymbol{a}} = \arg\min \mathbb{E}\left[\left(\boldsymbol{a}^\top \Psi(\boldsymbol{X}) - \mathcal{M}(\boldsymbol{X})\right)^2\right] \tag{3.58}$$

with,

$$(\boldsymbol{\Psi}^\top \boldsymbol{\Psi})\hat{\boldsymbol{a}}_\alpha = \boldsymbol{\Psi}^\top \mathbf{Y} \tag{3.59}$$

Ψ is the data matrix which assembles the values of all the orthonormal polynomials in $\mathbf{X}$, and is given by:

$$\Psi_{ij} = \Psi_j(\mathbf{x}^{(i)}) \tag{3.60}$$

3.3.6 Polynomial chaos approximation

Both the projection approach and the regression approach provide a stochastic response surface with assessments of the performance that must be done. The approximation accuracy of the PCE can be evaluated based on different error measures.

3.3.6.1 Generalization Error

The notion of generalization error is a basic concept of statistical learning theory. It refers to the discrepancy between a model's performance on the training dataset and its performance on an unseen dataset. The generalization error is defined as its mathematical expectation [44]:

$$I[\mathcal{M}_X] = E[(\mathcal{M}(X) - \mathcal{M}_X(X))^2] = \int (\mathcal{M}(X) - \mathcal{M}_X(X))^2 f_X(x)dx \tag{3.61}$$

where $(\mathcal{M}(X) - \mathcal{M}_X(X))^2$ being that loss function measuring the prediction error for a given x.

3.3.6.2 Leave-one-out Cross-validation

An accurate and efficient estimator for the generalization error is Leave-one-out (LOO) Cross-validation [12]. It is a technique where the data sample is divided into two subsamples. ples. A metamodel is built from one subsample, i.e. the training set, and its performance is assessed by comparing its predictions to the other subset, similar to the technique used for neural networks. In ANN LOO helps in verifying the model's ability to generalize beyond the training data, whereas for PCE it provides an estimate of the generalization error, guiding the choice of model parameters like polynomial degree and selection of basis functions. The error is formulated as the following:

$$\varepsilon_{LOO} = \frac{\sum_{k=1}^{m} \left(\mathcal{M}\left(x^{(k)}\right) - \mathcal{M}_{\mathrm{A}(k)k}\left(x^{(k)}\right)\right)^2}{\sum_{k=1}^{m} \left(\mathcal{M}\left(x^{(k)}\right) - \hat{\mu}_Y\right)^2} \tag{3.62}$$

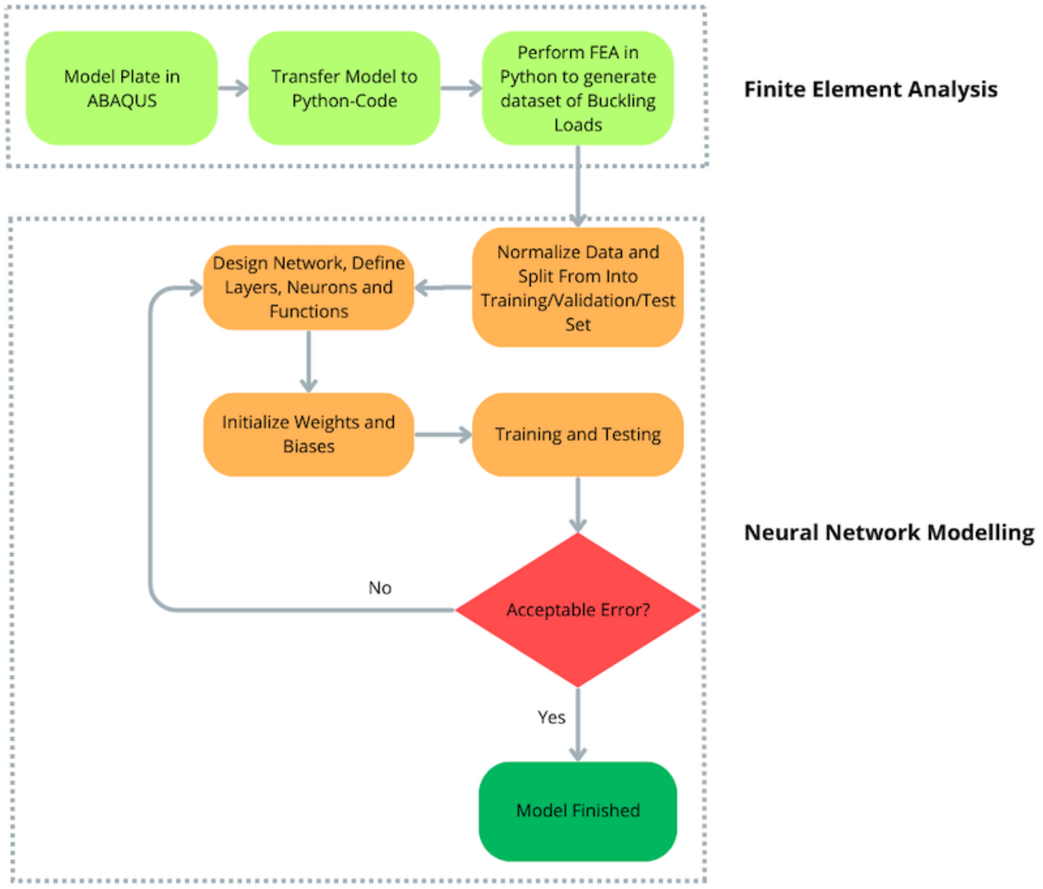
CHAPTER

FOUR

METHODS

4.1 Method

The approach for this thesis can be summarized as the flowchart in Figure 4.1.1. It is split into two parts, where the first part is modeling the plate in Abaqus before generating different configurations of the plate in Python. The second part of the process was to model the neural network. Both parts will be explained in depth in this chapter.



Model Plate in ABAQUS
Transfer Model to Python-Code
Perform FEA in Python to generate dataset of Buckling Loads
Finite Element Analysis
Design Network, Define Layers, Neurons and Functions
Normalize Data and Split From Into Training/Validation/Test Set
Initialize Weights and Biases
Training and Testing
Neural Network Modelling
No
Acceptable Error?
Yes
Model Finished

Figure 4.1.1: Flow Chart of Method

4.2 Abaqus Model

The first part of the problem was to model the stiffened plate in Abaqus and then generate enough variants of the model to train the neural network.

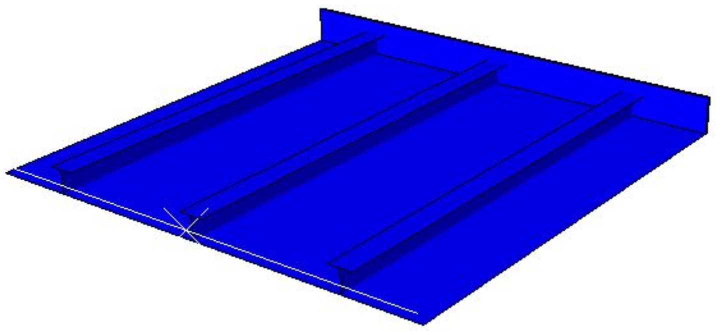


Figure 4.2.1: Stiffened Steel Panel modelled in ABAQUS

4.2.1 Assembly and Geometry

The model that was analyzed consists of a plate, a number of longitudinal stiffeners, and a transverse stiffener. The web height and length of the plate were kept consistent for every iteration, while the thickness of the plate and stiffeners, the width of the flange, and the number of stiffeners were the parts of the tested geometry. The values used for the iterations are listed in Table 4.2.1. In addition, the number of stiffeners also varied. This led to 1944 different configurations after material properties were taken into account.

Parameter	Values
Plate Thickness	2, 3, 4
Flange Width	30, 40
Number of Stiffeners	2, 3, 4, 5
Plate Width	220, 260, 300
Stiffener Thickness	3.50, 4.00, 4.50

Table 4.2.1: Model Parameters

In addition to this, analysis of buckling loads for parameters outside of the original range was also done. This was done to test the ANN on input parameters it was not originally trained on. Table 4.3.1 lists the geometry for those iterations, a dataset of 864 configurations.

Parameter	Values
Thickness of Plate	2, 3, 4
Thickness of Stiffeners	2.5, 3.5
Number of Stiffeners	2, 3, 4, 5
Flange Width	30, 32, 37
Plate Width	220, 300, 340

Table 4.2.2: Test Outside

4.2.2 Element Types

The shell is meshed with S4R elements, which are 4-node, quadrilateral, stress/displacement shell elements with reduced integration and a large-strain formulation. ABAQUS recommends using the S4R element type for modeling general purpose shells. When dealing with thin shells, S4R elements utilize discrete Kirchhoff thin shell theory, while thick shell theory is employed for thick shells [46]. ABAQUS documentation specifies that the theory used depends on the thickness of the element in calculations involving S4R elements. Furthermore, changes in thickness during calculations can lead to a change in the selected theory [46].

The reason for choosing the conventional S4R element is its ability to accurately model both thin and thick elements. In practice, S4R elements generally gives reasonable and accurate results in most scenarios.

4.2.3 Mesh Convergence

A convergence test decided the mesh size, and the results are presented in Table 4.2.3 and Figure 4.2.2. The model used for the convergence testing is the model seen in Figure 4.2.1. From the results, it can be seen that the only major change in buckling load when changing mesh size is from 20 to 17.5, and for the rest of the decrease in mesh size, the change in buckling load is insignificant. Still, the mesh chosen for the model was 10mm to ensure that the results were correct and with an acceptable computational time.

Mesh Size	Buckling Load [kN]	Change [%]
20	200.56	-
17.5	190.34	5.37
15	188.81	0.82
12.5	188.18	0.33
10	187.52	0.35
7.5	187.49	0.016
5	187.00	0.26

Table 4.2.3: Mesh Convergence

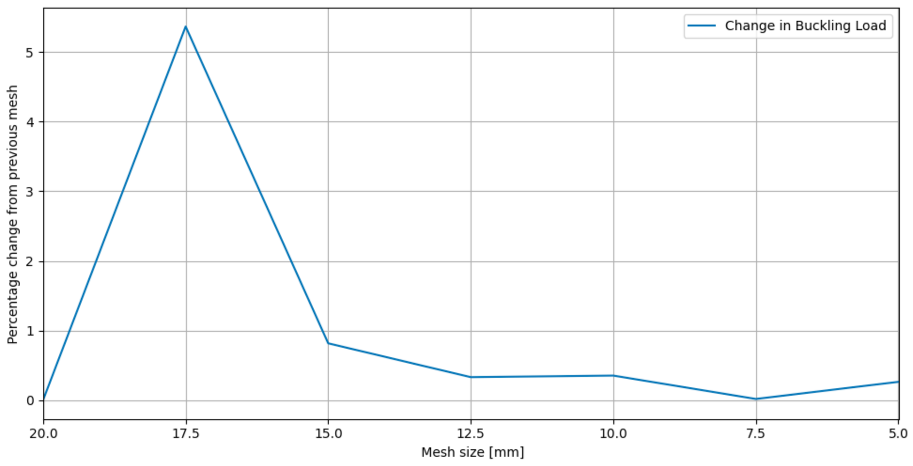


Figure 4.2.2: Mesh Convergence

In addition, with a mesh size of 10mm for the entire model, both the longitudinal and transverse stiffeners got a satisfactory mesh without needing to have a separate mesh size. The full mesh and a closer look at the mesh are illustrated in Figure 4.2.3 and 4.2.4.

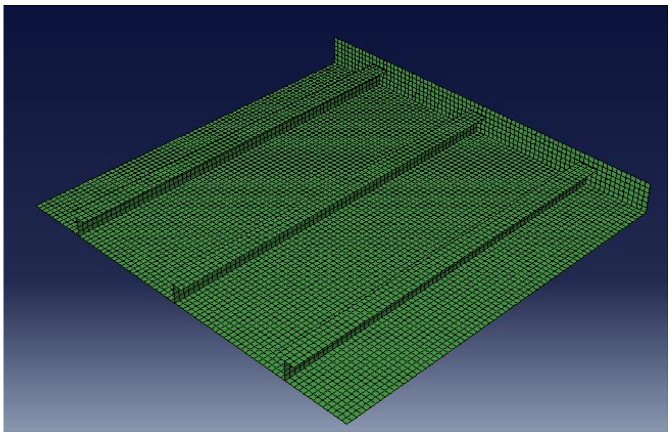


Figure 4.2.3: Full Model Mesh

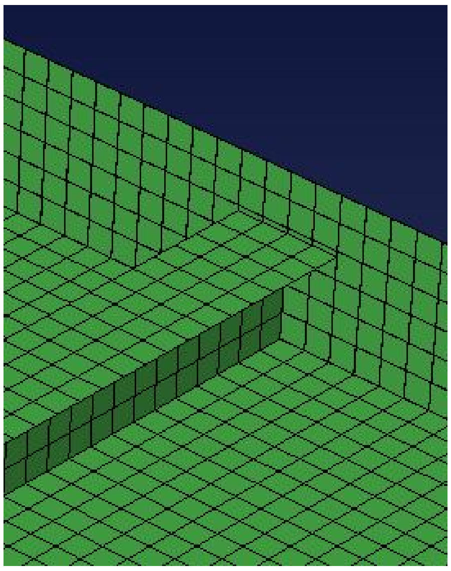


Figure 4.2.4: Close-up of Mesh

4.2.4 Material Properties

The material chosen for the model was steel. The Youngs Modulus was set to 210 GPa for every iteration, with varying yield strengths. The types of steel assumed tested were Mild Steel, High-Tensile Steel, and High Strength Steel. The yield strength tested for each type of steel is listed in Table 4.2.4 [47], with some of them being used for the dataset within the training range and some for the simulations outside of it. The rest of the material properties were kept the same in every simulation and are listed in Table 4.2.5.

Steel Type	Yield Strength [MPa]
Mild Steel	190, 235, 245
High-Tensile Steel	315, 355
High Strength Steel	400, 450

Table 4.2.4: Yield Strengths

Parameter	**Value**	**Unit**
E	210	GPa
v	0.3	-
ρ	7850	kg/m^3

Table 4.2.5: Material Parameters

4.2.5 Boundary Conditions and Constraints

The boundary conditions on the two ends were coupled to two reference points, R1 and R2, as seen in Figure 4.2.5, and the "tie"-constraint is used for this. The reference points, the constraints, and the implementation of Boundary Conditions in the Python Code were made after inspiration by code by PhD candidate Xintong Wang.

The boundary conditions applied in Abaqus are listed in Table 4.2.6. The end under load is fixed in all directions except for the load direction, while the opposite end is fixed for all six degrees of freedom to making it a clamped end.

Degree of Freedom	Clamped End	End Under Load	Sides
U1	x		
U2	x	x	
U3	x	x	x
UR1	x		
UR2	x	x	
UR3	x	x	x

Table 4.2.6: Boundary Conditions

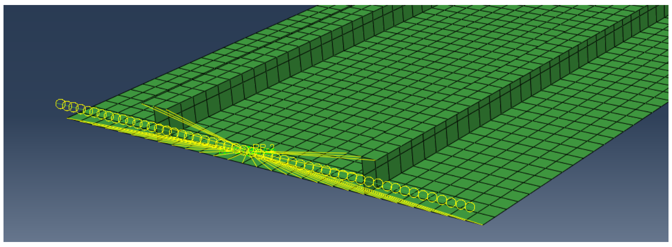


Figure 4.2.5: Constraints

4.2.6 Loads

The loads were shell edge loads applied on the edge of the shell elements, as seen in Figure 4.2.6. Shell Edge Load was used to induce compression. I was chosen because it accurately represents real-world loading conditions where plates are subjected to compressive forces along their edges and it ensures a uniform distribution of compressive stress along the edge.

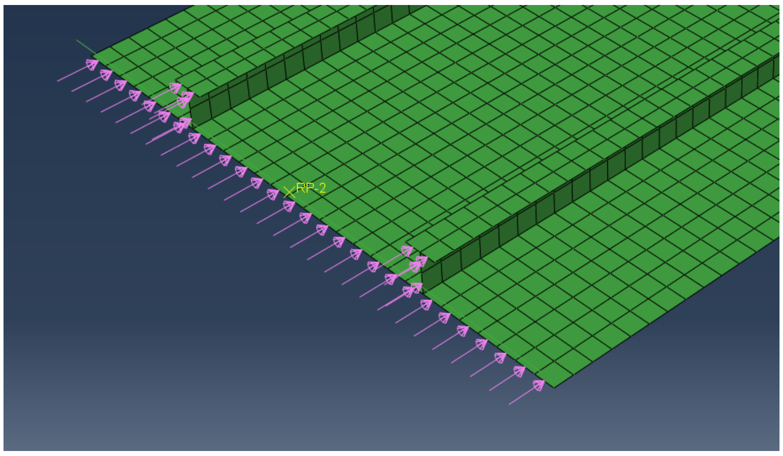
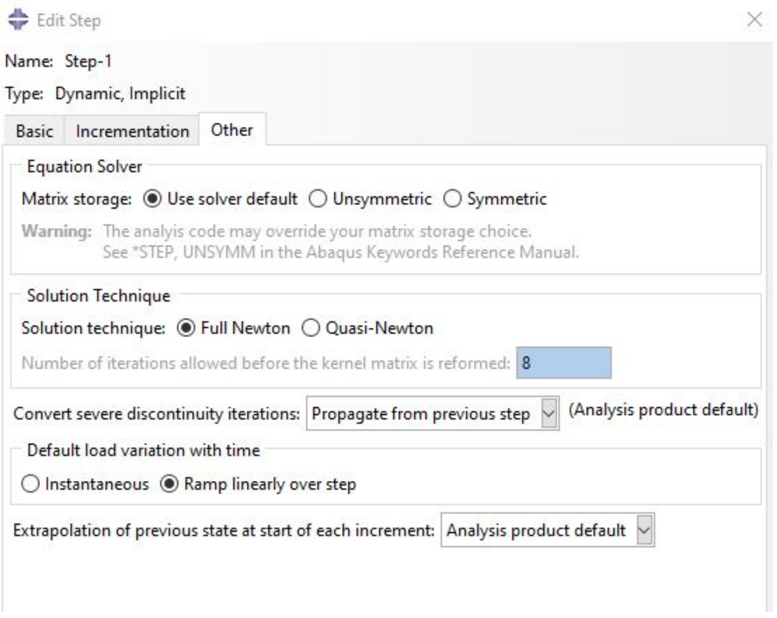


Figure 4.2.6: Load

4.2.6.1 Step and Solution Technique

The step chosen for this analysis was the dynamic implicit step with nonlinear geometry turned on. This was chosen due to the simplicity of this step type in running and coding the model in Python. The solution technique is Newton-Raphson, as seen in Figure 4.2.7 and described in Section 3.1.4.3.


Edit Step

Name: Step-1

Type: Dynamic, Implicit

Basic | Incrementation | Other

Equation Solver

Matrix storage: [✓ Use solver default] Unsymmetric Symmetric

Warning: The analyis code may override your matrix storage choice. See *STEP, UNSYMM in the Abaqus Keywords Reference Manual.

Solution Technique

Solution technique: [✓ Full Newton] Quasi-Newton

Number of iterations allowed before the kernel matrix is reformed: 8

Convert severe discontinuity iterations: Propagate from previous step (Analysis product default)

Default load variation with time

Instantaneous [✓ Ramp linearly over step]

Extrapolation of previous state at start of each increment: Analysis product default

Figure 4.2.7: Step and Solution Method

4.2.7 ABAQUS in Python

4.2.7.1 Generating Models

To generate enough models for the neural network to work, the process had to be automated in Python. To do this the Python extension abqpy was used. It is a Python package providing type hints for Python scripting of Abaqus, you can use it to write your Python script of Abaqus fluently, even without doing anything in Abaqus. It also provides some simple APIs to execute the Abaqus commands so that you can run your Python script to build the model, submit the job and extract the output data in just one Python script, even without opening the Abaqus/CAE [48]. The model generator code is in Appendix A1.

The loop to generate the different models is shown in the code below.

```python
for no in no_stiffeners_gen:
    nosti = no
    for y_p in yld_plate_gen:
        yieldp = y_p
        for y_s in yld_stiffener_gen:
            yields = y_s
            for fw in flange_width_gen:
                flangew = fw
                for t in plate_thickness_gen:
                    thick = t
                    for sd in plate_width_gen:
                        platew = sd
                        for st in stiff_thickness_gen:
                            stithi = st
                            jname = JOBNAME
```

4.2.7.2 Post-Processing

The post-processing was done in Python; the code is attached in the Appendix A2. The buckling load was found by extracting reaction forces and displacement and plotting them against each other. Then, a code was written to identify where the gradient of the curve seen in Figure 4.2.8 changed sign. The figure shows when the buckling happens, and two of the buckling modes are shown in Figure 4.2.9 and 4.2.10.

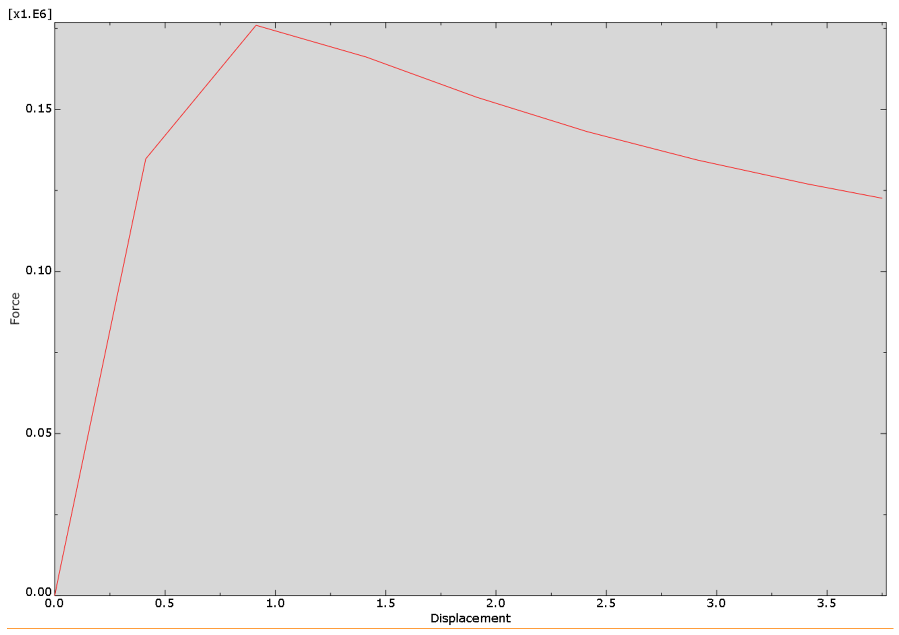


Figure 4.2.8: Post-Processing

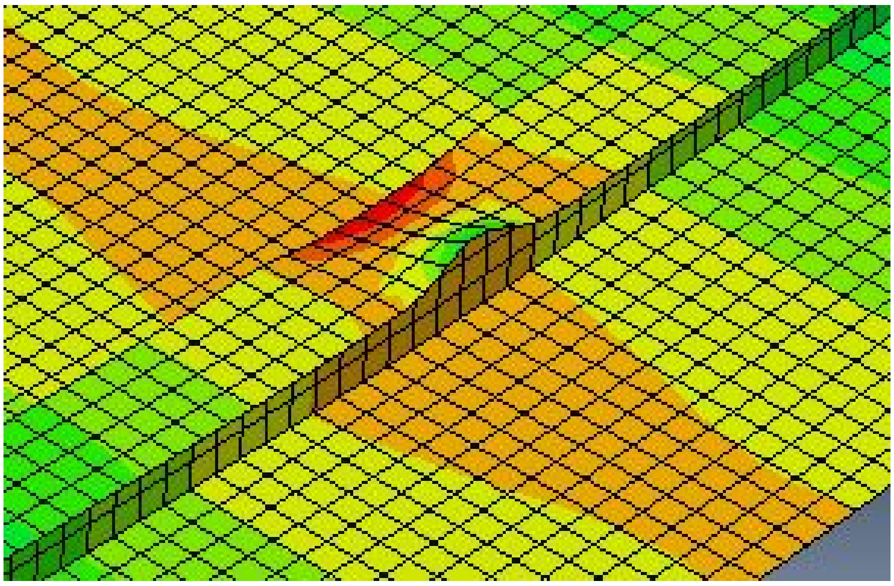


Figure 4.2.9: Post-Processing

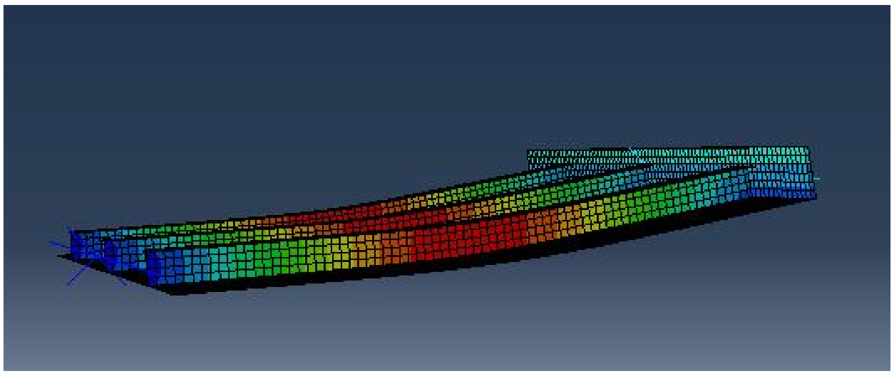


Figure 4.2.10: Post-Processing

4.3 Neural Network Code

The ANN was coded in Python, with the help of the Python libraries TensorFlow and Keras [49]. The overall structure of a ANN is developed using the model object in Keras, which provides a simple way to create a stack of layers by adding new layers one after the other. Most of the description of how Keras is built is from "Learn Keras for Deep Neural Networks" by Jojo Moolayil [36]. Parts of the code will be presented here, the full code is in Appendix B1.

The type of neural network used for this thesis is a feedforward neural network, often referred to as a multilayer perceptron. This type of network consists of multiple layers of neurons, each fully connected to all neurons in the previous layer. It is suitable for a wide range of problems and especially complex regression tasks like predicting buckling loads.

4.3.1 Software

Originally, Keras was developed as an independent project, but it has since been integrated into TensorFlow, and serves as TensorFlow's high-level API. Keras enables the construction and training of neural networks. Neural network architectures in Keras are defined by stacking layers, and these can range from fully connected layers to convolutional layers and recurrent layers.

Tensorflow is an open-source software library, and was developed Google in 2015 [50]. The core concept of TensorFlow is the computation graph. TensorFlow is a mathematical library which is used for Machine Learning.

4.3.2 Description of Code

The results from the analysis in Abaqus were transferred to a CSV file. The was structured as seen below, with 1944 combinations of input parameters and buckling loads structured as seen below.

PlateStrength	StiffStrength	Thickness	FlangeWidth	Stiffeners	BucklingLoad	PlateWidth	BucklingLoad
235.000000	235.00	2.00	30.00	2.00	220	3.50	180512.95
235.000000	235.00	2.00	40.00	2.00	220	4.00	202128.06
235.000000	235.00	2.00	30.00	2.00	220	4.50	179438.84
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:

Table 4.3.1: Test Outside

4.3.3 Preprocessing of Data

After importing the dataset, it was split into training, validation and test sets by using the built-in *train_test_split*()-function. After trying different variations of splitting the data, 75% of the data sets were used for training, 15% for validation and 10% for testing. The shapes of the different split data were tested to validate the sets, with the results listed in Table 4.3.2.

Set Type	Shape
Training, Input	(1485, 7)
Validation, Input	(263, 7)
Testing, Input	(195, 7)
Training, Output	(1485, 1)
Validation, Output	(263, 1)
Testing, Output	(195, 1)

Table 4.3.2: Set Shapes

This was followed up by scaling the data. Given the variability in the magnitude of the input features, for instance, the thickness is small while the plate strength is large, the data must be normalized to ensure that no single feature disproportionately influences the model's learning process. To combat this StandardScaler from Scikit-learn was used as seen in the code snippet below. It subtracts the mean and divides it by the standard deviation for each feature, ensuring that each feature contributes equally to the prediction.

```
scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

4.3.4 Layers and Neurons

After the data was split, the model could be built. The model chosen was Sequential. It is the easiest way to define a model, allowing easy creation of a linear stack of layers [36].

The layers used for this network was the 'Dense'-layer in Keras. A dense layer is a regular layer that connects every neuron in the current layer to every neuron in the previous layer. Due to the layer accommodating every possible connection between the layers, it is called a dense layer.

Testing of different combinations of the number of layers and amount of neurons was done, and some of the configurations tested are listed in Table 4.3.3. Each number represents the number of neurons in each layer. The input layer (the number of input parameters) and output layer (predicted buckling load) are not included in the test table. The full list of tests for configurations is in Appendix C2.

Layer	R^2 Inside	R^2 Outside	MAPE Inside	MAPE Outside
4-8	0.9668	0.8602	7.19	16.71
4-32	0.9792	0.9320	5.39	11.46
16-8	0.9794	0.9096	5.52	10.57
16-16	0.9828	0.9166	4.90	13.18
4-4-2	0.9840	0.9646	4.56	10.15
16-8-4	0.9865	0.9640	4.07	8.91
16-8-8	0.9883	0.9612	3.90	9.04
16-16-8	0.9887	0.9521	3.67	9.04
32-16-8	0.9893	0.9723	3.37	8.63
128-64-32	0.9998	0.8634	1.28	17.42

Table 4.3.3: Joint Dataset

The final definition of the model and layers is shown in the code snippet attached below. The best results for testing for the data the model was trained on is as low as an average error of 1.28 %, but this results in a mean error of 17.42 % for the outside range. This happens due to overfitting and not being able to adjust well to new data. For this reason, the configuration chosen was the one that gave reasonably good results both inside and outside of the training range. The results from both the best test within the training range and the best combination of inside and outside the range will b represented in the results section of the thesis.

Furthermore, L2 regularization is utilized. L2 regularization involves adding the squared weights to the loss function. This process aims to reduce the weights to values close to 0 (but not exactly 0) in order to enhance the model's generalization - hence it is also referred to as the "weight decay" technique. In most scenarios, L2 regularization is preferred over L1 to mitigate overfitting. The formula for it is shown in Equation (4.1) [36].

$$\text{Cost Function} = \text{Loss (as defined)}\frac{\lambda}{2m} * ||Weights||^2 \tag{4.1}$$

```
model = tf.keras.Sequential([
    Dense(32, activation='relu', input_shape=(7,)),
    Dense(16, activation='relu', #kernel_regularizer=regularizers.l2(0.001)),
    Dense(8, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
    Dense(1)
])
```

4.3.5 Weighting of Neurons

In Keras, the weights of neurons are managed through layers, with each type of layer handling weights according to its specific function and architecture. Two neurons of successive layers have a connection with an associated weight, where the weight would define the influence of the input for the output to the next neuron and in the end its influence on the final output. The weights start at random in the training phase, before they are improving and updated iteratively until they can predict the correct output [36].

4.3.6 Activation Function

The ReLU activation function was chosen for this model, due to its effectiveness in tasks involving regression with neural networks. ReLU is a factor in helping the network train faster because its derivative is 1 for all positive inputs which leads to a simplification of the computation of the gradient. This can be useful in networks with many layers where training can become computationally expensive.

ReLU introduces non-linearity into the model, enabling it to capture complex relationships between the input features and the target value such as non-linear patterns that occur in the data for buckling loads. It also helps minimizing the vanishing gradient problem, which means that for positive inputs, the gradient of ReLU is constant, a problem that can occur during backpropagation in neural networks. This leads to effective learning in all layers of the network.

4.3.7 Optimization and Cost/Loss Function

The compile method in Keras is used to configure the learning process of the neural network. Each component specified in the compile method plays a role in how the model learns from the data. For the optimizer, ADAM (Adaptive Moment Estimation) was chosen. ADAM is efficient in terms of computation and memory requirements, and it's suitable for a wide range of optimization problems [36].

The reason ADAM was chosen over other optimizers is because of a study done by Kingma and Ba in 2014 on optimizers for multi-layer neural networks. As seen from Figure 4.3.1, ADAM proved to have better convergence than other methods [51].

The gradient of the loss defines the momentum and variance, which leads to updated weight parameters. The combination of momentum and variance is an effective improvement of the learning process, and it helps smooth the learning curve. The optimizer can be described like this in math terms: $Weights = Weights - (Momentum + Variance)$.

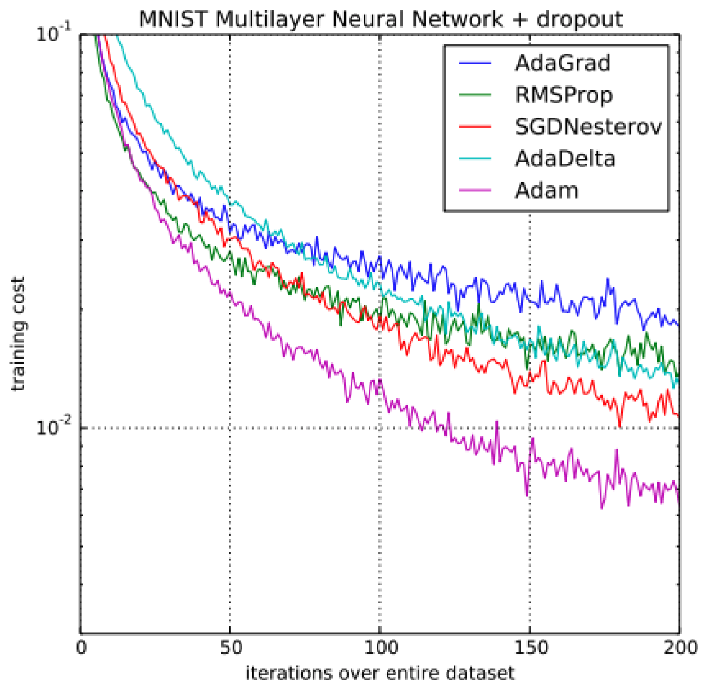


Figure 4.3.1: Convergence for different optimizers

The reason for using MSE as the cost function is because squaring heavily penalizes larger errors, making it particularly suitable for cases where large errors are more detrimental than smaller ones. Throughout training, the model aims to minimize MSE to reduce the average squared difference between the estimated values and the actual values. Additionally, MAE is incorporated in the metrics to monitor the model's performance. While the model focuses on minimizing the loss function (MSE), other metrics can be specified for monitoring. MAE provides an easily interpretable average absolute error between the model's predictions and the actual values and is not biased towards any particular type of error. It is less sensitive to outliers than MSE, making it a great choice for gaining a more intuitive understanding of the model's prediction error. The neural network is designed to handle regression tasks and is specifically focused on efficiently minimizing prediction errors while providing a clear metric for evaluating the model's performance.

4.3.8 Training the Model

The fit()-function provided by Keras trains the model on the provided training data. The fit function is for the model object to train with the provided training data.

The different x-values feature the training data which have been scaled, while the y-values are the target values corresponding to the scaled x-values. The validation data is used to evaluate the model after each epoch of training, providing a measure of performance on data not used in training. The fit()-function captures the model's performance metrics at each epoch during training. It includes parameters like training loss, validation loss, and any other additional metrics specified during model compilation.

To evaluate the model's performance, evaluate() is used. This is done on a dataset that the model has never seen during training and it returns the loss value and metrics values for the evaluation.

Lastly, the predictions are made. This is done by the predict()-function, and it is used to generate predictions from the input data. Here, it's being used to predict outcomes based on the unseen input parameters from the test set.

4.3.9 Validation of Model

In an ideal training situation for a neural network, it's expected that both the training loss and the validation loss will decrease as the model learns from the data. In addition, the model's accuracy should increase. However, the specific shapes and features of these curves may differ, providing important insights into the model's learning behavior and possible issues such as overfitting or underfitting.

Ideally, the training and validation loss and accuracy should either converge or exhibit a similar trend, indicating that the model is effectively generalizing to unseen data. If the training loss continues to decrease while the validation loss starts to increase or remains constant, it's a clear indication of overfitting. This occurs when the model learns the training data too well, incorporating its noise and specifics, at the cost of its ability to generalize to new data. If both the training and validation loss remain high and remain constant, the model might be underfitting. This implies that the model is too simplistic to capture the underlying patterns in the data.

A rapid decrease in loss followed by a plateau could suggest that the learning rate is appropriately set - initially large enough to show rapid progress, but subsequently small enough to allow fine-tuning as the model converges.

In order to prevent underfitting and overfitting, careful examination of the graphs for Training Loss and Validation is necessary. As shown in Figure 4.3.2, both the training and validation loss/error decrease rapidly initially before overfitting leads to increased error in validation. The selected number of epochs for this project is 1200 because, as seen in Figure 4.3.3, this is where the perfect fit from Figure 4.3.2 is.

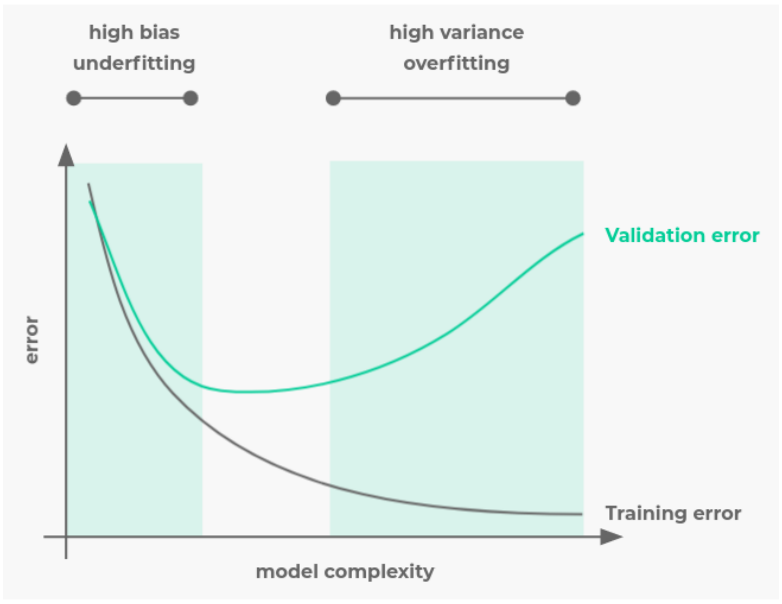


Figure 4.3.2: Underfitting vs Overfitting [52]

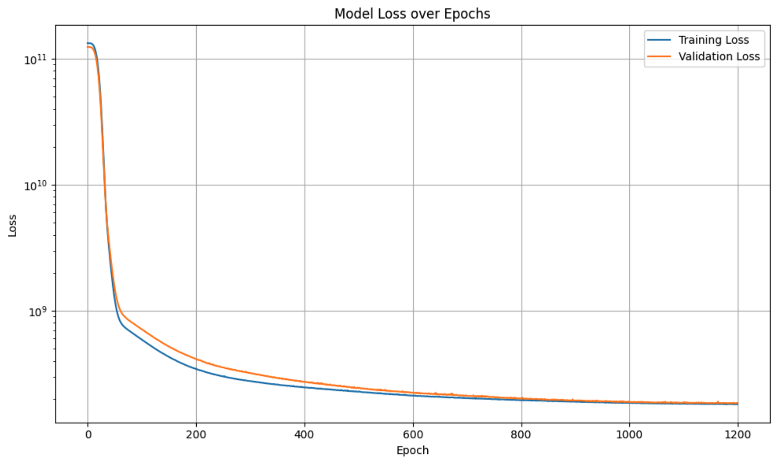


Figure 4.3.3: Training Loss vs Validation Loss

CHAPTER

FIVE

RESULTS

5.1 Comparing Computational Time

Before presenting the results from the neural network, the difference in computational time between the FEM analysis and ANN will be shown. Table 5.1.1 illustrates the clear improvement in computational time and proves that using an ANN is vastly faster than running simulations in FEM programs like Abaqus.

Analysis Type	**Computational Time**
FEM	27h32m15s
ANN	3m11s

Table 5.1.1: Computational Time

5.2 Testing Within The Training Range

For the first part of the testing, the network was tested with a dataset consisting of input parameters the network was already trained on. The testing was split into two parts; testing for the whole dataset and testing where the data sets were split for each number of stiffeners.

5.2.1 One Dataset

For the testing with one dataset, the network was trained on all variations of stiffeners. As seen in table 5.3.1, the mean error is 3.67% with a maximum error of 10.74%. The comparison between the true buckling load and the predicted buckling load is illustrated in figure 5.2.1.

Number of Stiffeners	**Mean Error [%]**	**Max Error [%]**
All Stiffeners	3.67	10.74

Table 5.2.1: Error results for split dataset

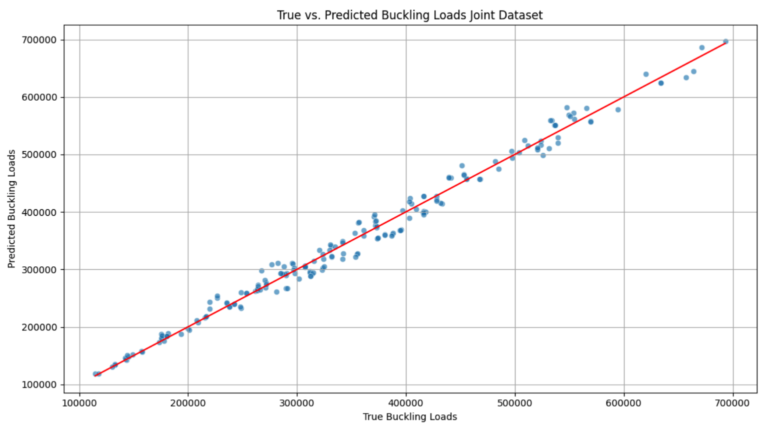


Figure 5.2.1: Training Loss vs Validation Loss

The histogram in Figure 5.2.2 shows the distribution of percentage errors between predicted values and actual values. From the histogram, multiple observations can be made. Most of the data is clustered around 0% suggesting that for most of the predictions the error small. This indicates that the model predictions are often close to the actual values. The same can be seen from Table 5.2.2. The errors spread out from around -9% to around +10.74%. The distribution is close to symmetrical around the center, which is an indication that the model does not have a systematic bias and therefore is neither underfitting nor overfitting.

As from Table 5.2.2, the most common error range is between 0% and 3%, with almost half the predictions being under 3% away from the FEM results. There are fewer instances as the error percentage moves away from 0%, which is typical in well-performing models. Around a third of the errors are over 5% of the analytical results.

There are a few bars at the tails of the distribution, showing that there are a few cases with high error percentages. These could be outliers or cases that are more difficult for the model to predict accurately. The presence of errors across a wide range of percentages suggests that while the model generally performs well, there could be room for improvement, especially in reducing the occurrence of those larger errors.

Error	**<3%**	**3%-5%**	**>5%**
All Stiffeners	96	37	62

Table 5.2.2: Error results for joint dataset

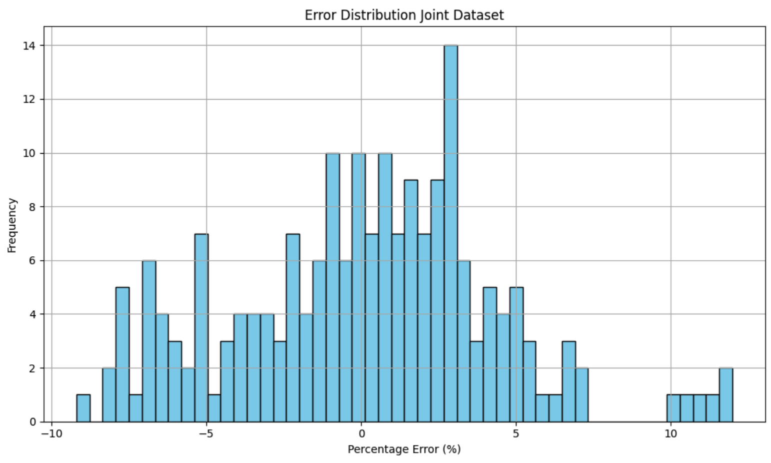


Figure 5.2.2: Training Loss vs Validation Loss

A code was also written to see the input parameters of the largest errors in the predictions. Table 5.3.3 describes the largest errors for each number of stiffeners. It can be seen that the maximum error for each combination of stiffeners decreases with increasing stiffeners. This shows that the model predicts higher buckling loads better than low buckling loads.

Number of Stiffeners	Max Error [%]
2	11.68
3	9.58
4	8.13
5	6.47

Table 5.2.3: Error results for each stiffener for joint dataset

5.2.2 Split Dataset

To avoid the effect of different mode shapes for different numbers of stiffeners influencing the prediction of buckling load, the dataset was split after the number of stiffeners.

The results for when the network was trained on a split dataset is noticeably better than for the joint dataset. The average error for the new testing is 2.94%, which is a little better than for the previous test, while the maximum error is 7.90%. When looking at the individual stiffeners, all of them have significant improvements, apart from the instance with 4 stiffeners. The maximum error is almost half the size with 2 stiffeners for the split dataset compared to the joint dataset.

From the error distribution in Table 5.3.5 and a major improvement can be seen from the testing with one dataset. Whereas over 30% of the predictions were missed with more than 5% in the first test only 6% of the predictions fell within that range.

Number of Stiffeners	**Mean Error [%]**	**Max Error (ME)[%]**	**ME Change [%]**
2	3.11	5.91	49.4
3	2.72	5.09	46.9
4	2.99	7.90	2.8
5	2.86	5.76	11.0
Total	**2.94**	**7.90**	

Table 5.2.4: Error results for split dataset

Error	**<3%**	**3%-5%**	**>5%**
2 Stiffeners	19	22	6
3 Stiffeners	26	21	1
4 Stiffeners	24	22	3
5 Stiffeners	25	22	2
Total	**94**	**87**	**12**

Table 5.2.5: Error results for split dataset

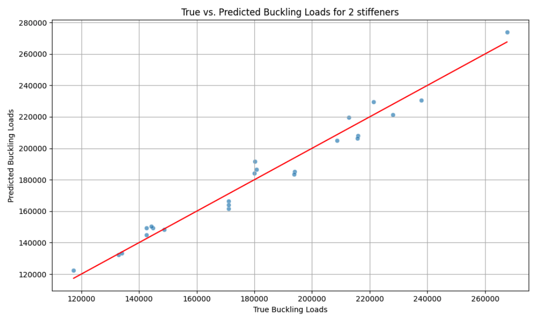


Figure 5.2.3: Correct vs Predicted Buckling load for 2 stiffeners

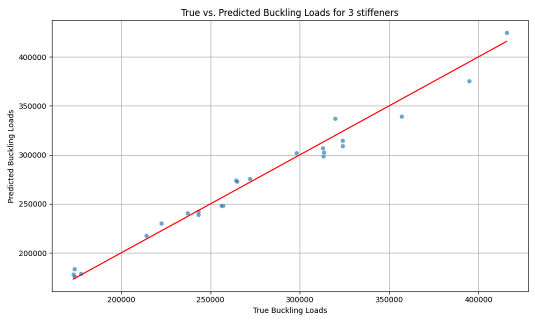


Figure 5.2.4: Correct vs Predicted Buckling load for 3 stiffeners

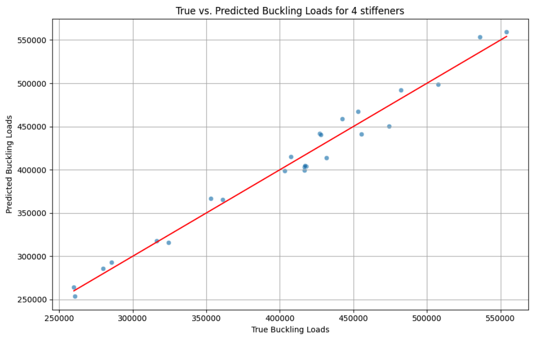


Figure 5.2.5: Correct vs Predicted Buckling load for 4 stiffeners

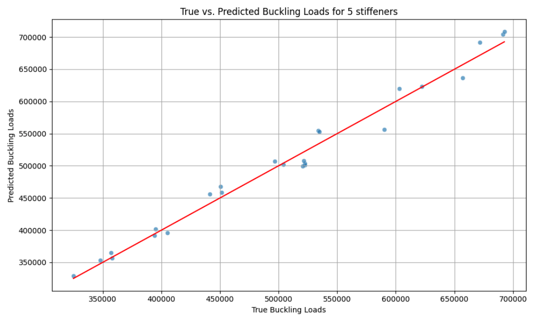


Figure 5.2.6: Correct vs Predicted Buckling load for 5 stiffeners

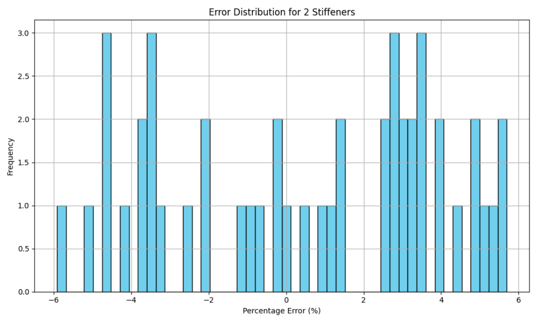


Figure 5.2.7: Error Distribution 2 Stiffeners

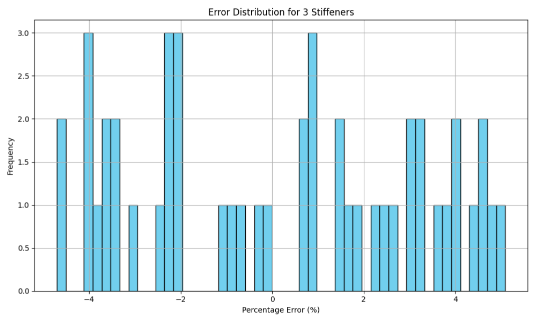


Figure 5.2.8: Error Distribution 3 Stiffeners

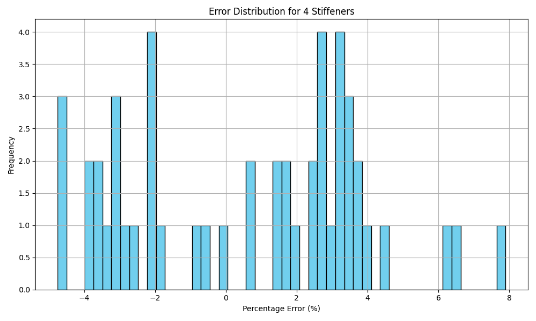


Figure 5.2.9: Error Distribution 4 Stiffeners

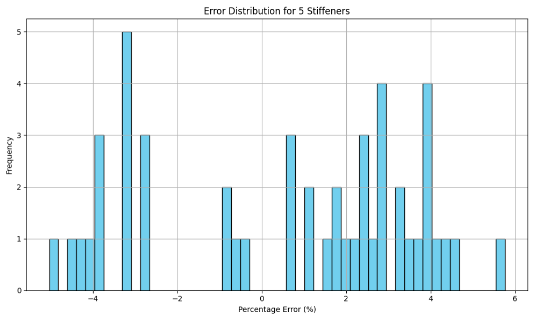


Figure 5.2.10: Error Distribution 5 Stiffeners

5.3 Testing Outside the Training Range

After testing the model on data it was trained on, the model was tested on data outside of the training range

5.3.1 One Dataset

When testing with one dataset, it can be seen from Figure 5.3.1 that the errors here are visibly larger than when testing within the training range. From the error results listed in Appendix C1, the the most common input parameters for the largest errors are for the combination of a yield strength of 190 MPa for the stiffener strength and a configuration of 2 stiffeners. Even with larger errors, the error distribution is still looking good with regards to over- and underfitting. It is mostly centered around 0% and symmetrical on each side but with a small bias towards overpredicting the buckling load. It ranges from overpredicting by a little over 30% to underpredicting by 20%. When testing which parameters are affecting the buckling load the most, the number of stiffeners is the most important followed by the yield strength of the stiffeners and flangewidth.

When compared to the testing within the training range with one dataset, the results are significantly worse. Both the mean error and max error are almost three times as large for this test.

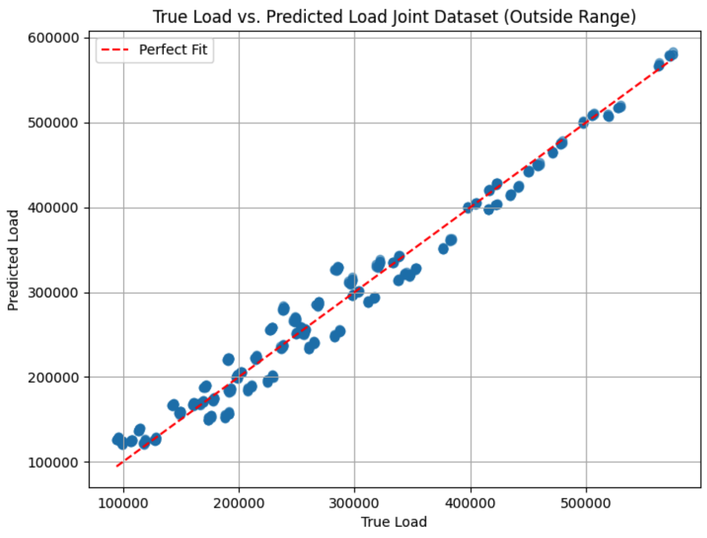


Figure 5.3.1: Training Loss vs Validation Loss

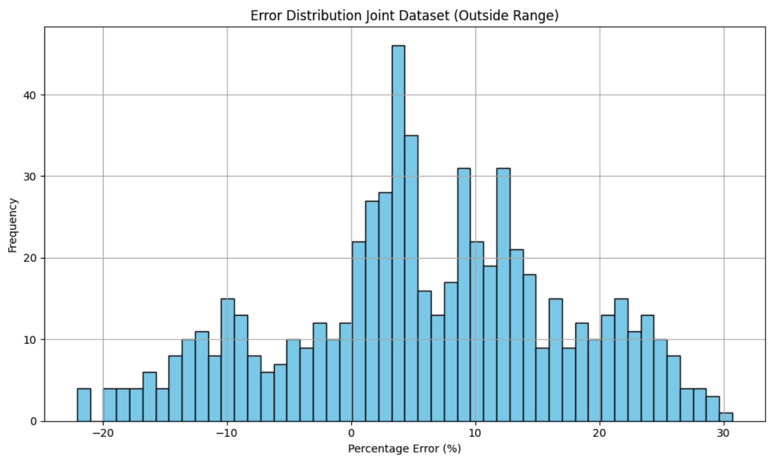


Figure 5.3.2: Training Loss vs Validation Loss

Number of Stiffeners	Mean Error [%]	Max Error [%]
All Stiffeners	8.68	31.55

Table 5.3.1: Error results for split dataset

Error	<5%	5%-10%	>10%
All Stiffeners	293	216	355

Table 5.3.2: Error results for joint dataset

When looking at the results from the joint dataset for each stiffener, it is again clear that the model has a problem with predicting buckling for two stiffeners. The error for two stiffeners is around three times as large for testing outside of the training range as it was for training inside, while the error is only twice as large for three and four stiffeners.

Number of Stiffeners	Max Error [%]
2	31.62
3	18.37
4	17.42
5	20.69

Table 5.3.3: Error results for each stiffener for joint dataset

5.3.2 Split Dataset

Then, the model was trained only on one configuration of stiffeners. For 2 out of the 4 stiffener configurations there is a significant improvement in the maximum error while there is a increase in maximum error for 4 stiffeners. However, when looking at Figure 5.3.5 this is only one outlier. When looking at the average mean error, it has notably improved from training on the entire dataset. It has reduced from 8.68% to 5.41%.

Number of Stiffeners	Mean Error [%]	Max Error (ME) [%]	ME Change [%]
2	5.85	20.75	34.38
3	5.83	18.02	1.91
4	5.17	19.79	-13.67
5	4.80	16.73	19.14
Total	**5.41**	**20.75**	**34.38**

Table 5.3.4: Error results for split dataset outside range

From Table 5.3.5 there is also a noticeable difference that can be seen in how the errors are distributed. From only 33.9% of the errors being under 5% and 58.9% under 10% for the joint dataset, the results are now 56.1% of the predictions are within 5% of the FEA result, and 85.3% of the predictions under 10% away from the correct result.

Error	<5%	5%-10%	>10%
2 Stiffeners	107	73	36
3 Stiffeners	115	66	35
4 Stiffeners	126	68	22
5 Stiffeners	137	45	34
Total	**485**	**252**	**144**

Table 5.3.5: Error results for split dataset

However, there is a clear tendency where the model overpredicts, and this is especially the case for 3, 4, and 5 stiffeners as seen in Figures 5.3.4-5.3.6. For all these tests there are a substantial group of errors standing out from the others. When looking at Figures 5.3.7-5.3.10 the model overpredicts the buckling load for a yield strength of 190MPA for the stiffeners, which shows that the model struggles to predict input parameters outside of the training range.

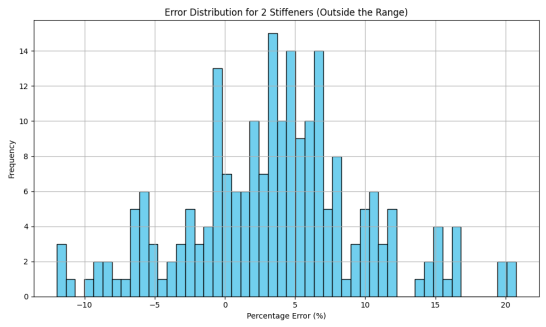


Figure 5.3.3: Error Distribution 2 Stiffeners

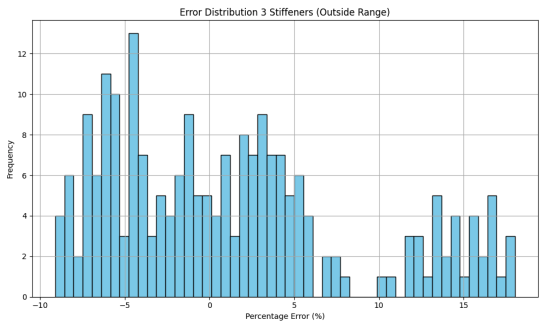


Figure 5.3.4: Error Distribution 3 Stiffeners

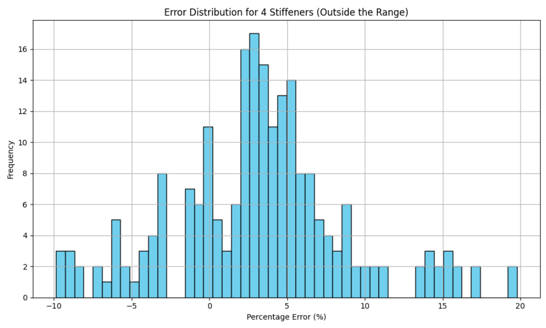


Figure 5.3.5: Error Distribution 4 Stiffeners

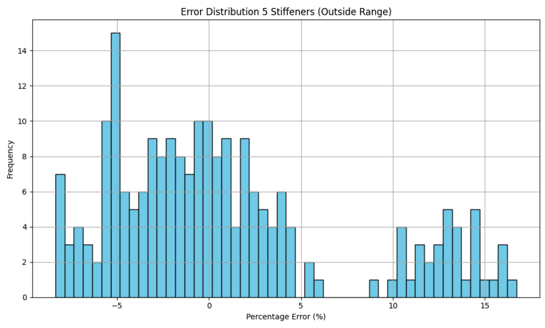


Figure 5.3.6: Error Distribution 5 Stiffeners

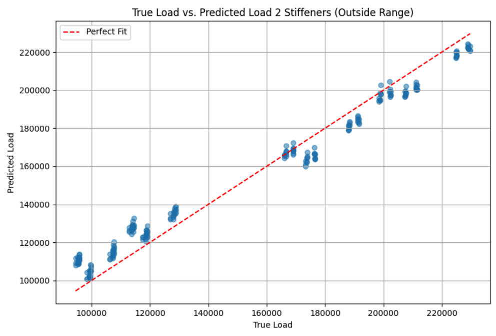


Figure 5.3.7: Error Distribution 2 Stiffeners

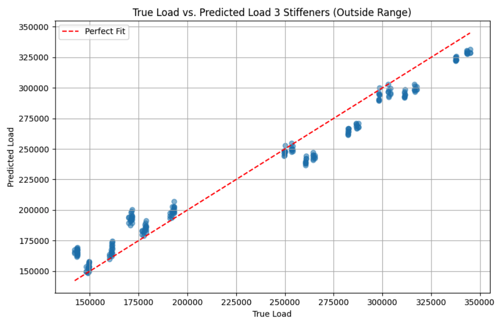


Figure 5.3.8: Error Distribution 3 Stiffeners

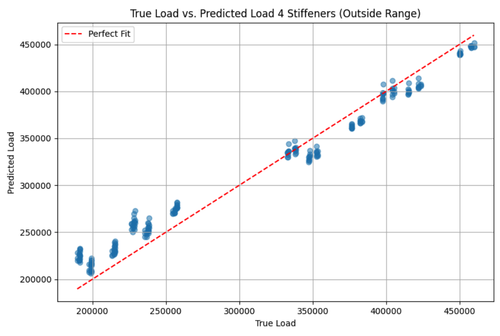


Figure 5.3.9: Error Distribution 4 Stiffeners

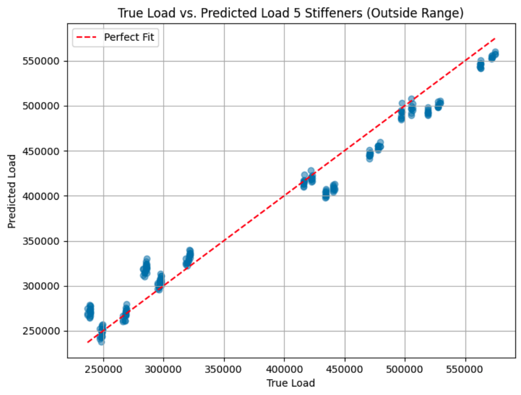


Figure 5.3.10: Error Distribution 5 Stiffeners

5.3.3 Example Overfitting

When testing with significantly more neurons in each layer than for the rest of the testing, much better results when testing within the range of the dataset the model was trained on. An average error of 1.28% was achieved, and as seen in Figure 5.3.11 the model predicts the buckling load remarkably well. However, as seen from the model loss plot in Figure 5.3.12, something is not right. The model has been too used to the data it was trained on and fails to generalize to unseen data. This results in the plot in Figure 5.3.13. The model does not predict well to new input parameters and gives an average error of 17.42%, with a maximum error of 95.42%. This proves that the model can predict data it is trained on really well, but this results in a model that does not adjust well to new data.

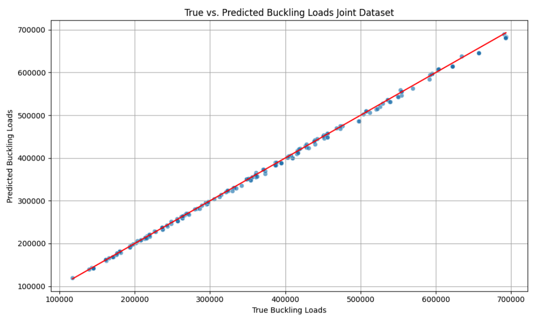


Figure 5.3.11: Overfitted prediction

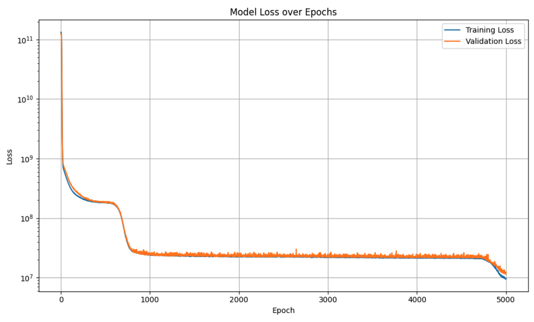


Figure 5.3.12: Overfitted Validation Plot

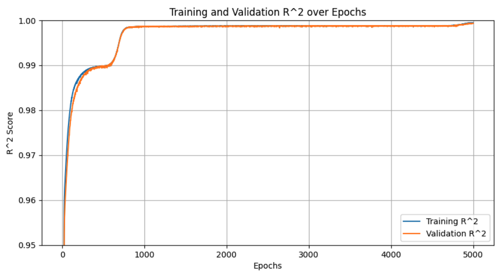


Figure 5.3.13: Overfitted R^2 score

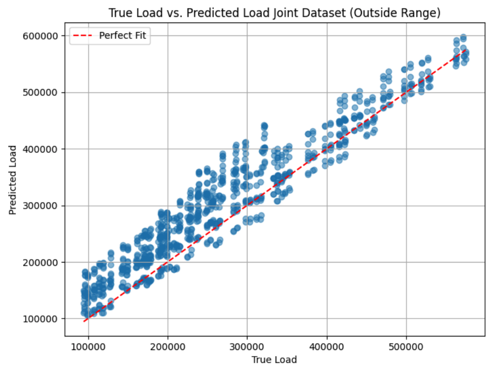


Figure 5.3.14: Overfitted Outside Range Prediction

CHAPTER

SIX

DISCUSSION

6.1 Discussion and Future work

The exploration of surrogate models, particularly neural networks, in the structural analysis of stiffened plate panels has yielded significant insights and identified several key areas for future research and development. This discussion will delve into the implications of the findings, the limitations encountered, and potential directions for further investigation.

The successful application of neural networks to predict buckling loads in stiffened plate panels underscores the potential of machine learning techniques to transform traditional engineering analysis. The primary benefits observed include:

- One of the most notable advantages of using neural networks is the substantial decrease in computational time. This efficiency enables faster iterations in the design process, allowing engineers to explore a wider range of design variables and scenarios without the prohibitive time costs associated with full-scale FEA.
- The neural network models demonstrated high accuracy in predicting buckling loads within the training range. This indicates that, with adequate and representative training data, machine learning models can serve as reliable surrogates for complex computational methods.
- Neural networks offer scalability and flexibility, adapting to different problem sizes and configurations. This characteristic is particularly valuable in structural engineering, where the complexity and scale of problems can vary widely.

Despite the promising results, several limitations and challenges were encountered during the study:

- The neural network models exhibited reduced accuracy when predicting values outside the training range. This limitation highlights the challenge of generalizing machine learning models to unseen data, particularly in the context of highly nonlinear and complex structural behavior.

- The performance of neural networks is heavily dependent on the quality and quantity of the training data. Inadequate or unrepresentative data can lead to poor model performance and limit the applicability of the model to real-world scenarios.

- Neural networks, especially deep learning models, often function as "black boxes," providing little insight into the underlying mechanics of their predictions. This lack of interpretability can be a drawback in engineering applications where understanding the rationale behind predictions is essential.

6.1.1 Future Work

To overcome these limitations and expand on the discoveries of this study, future research can explore several possibilities. The improvement of the model's resilience and generalization abilities can be achieved by gathering and integrating a wider variety of training data, including diverse setups, material characteristics, and loading conditions. This extension could encompass both empirical data and high-fidelity simulations. Hybrid approaches that meld machine learning with physics-based models can capitalize on the strengths of both techniques. For example, the incorporation of physical laws into the training process through Physics-Informed Neural Networks (PINNs) could potentially enhance extrapolation and interoperability. The exploration of more sophisticated neural network structures, such as convolutional neural networks or recurrent neural networks, may yield enhanced performance for specific types of structural analysis issues. Furthermore, the combination of multiple models through ensemble methods might produce more precise and resilient forecasts. The incorporation of techniques for quantifying uncertainty into neural network models could furnish more dependable predictions and aid engineers in assessing the confidence level in the model's results. In addition, applying and validating neural network models in real-world engineering projects would offer practical insights and help bridge the gap between theoretical research and practical application. Collaboration with industry partners can facilitate this process and ensure that the models meet the demands of actual engineering workflows.

CHAPTER
SEVEN

CONCLUSIONS

7.1 Conclusion

This study has explored the construction and application of surrogate models in the structural analysis of stiffened plate panels. The primary focus was on the integration of neural networks to predict buckling loads, examining their effectiveness in reducing computational time and enhancing prediction accuracy.

The research commenced with a comprehensive literature review on surrogate modeling and the utilization of neural networks in structural analysis, finite element analysis and buckling of plates. This foundational knowledge was crucial for understanding the theoretical underpinnings and practical applications of the methods employed.

The experimental work involved creating a neural network model trained on a dataset derived from FEA results from ABAQUS and Python. An extensive dataset was created, with 1944 different combinations of the input parameters such as number of stiffeners, plate thickness and yield strength for the stiffened plate. In addition, a dataset of 864 new variations of input parameters was created to test the Neural Network on data it was not trained on.

Several different neural network designs were experimented with, and their effectiveness was assessed using metrics such as Mean Absolute Error and Mean Squared Error. The performance of the ANN was tested using two different sets of data. The first evaluation was carried out using the same dataset that was used to train the ANN. Discrepancies between the results produced by the ANN and those from the FEA were calculated, revealing that the deviation was minimal for the data points in the training set, indicating that the ANN results are in line with the analysis results. When the ANN was tested on the full dataset within the training range it achieved an average error of 3.37%, with a maximum error of 11.68%. When the data was split for each combination of stiffener to avoid the effects of buckling modes the average error decreased to 2.94% with a maximum error of 7.90%. This proves that the ANN can be utilized for nonlinear structural analysis of a stiffened plate as long as the input parameters remain within the limits of the input parameter set.

When testing outside of the data the model was trained on, the mean error was 8.68% and max error 31.55% for the full dataset. After the data was split, the mean error decreased to 5.41% with a maximum error of 20.75%. This shows promise of also being able to predict the buckling load for structures outside of the input parameters the ANN is trained on, but the model still needs improvements to make more accurate predictions, with the maximum errors still being quite large.

This thesis shows the potential of using machine learning and surrogate modeling in the world of structural engineering, while still proving that there is a need and room for improvement in the field.

REFERENCES

[1] Pero Prebeg, Vedran Zanic, and Bozo Vazic. “Application of a surrogate modeling to the ship structural design”. In: *Ocean Engineering* 84 (2014), pp. 259–272.

[2] G. Gary Wang and S. Shan. “Review of Metamodeling Techniques in Support of Engineering Design Optimization”. In: *Journal of Mechanical Design* 129.4 (2006), pp. 370–380.

[3] D. C. Montgomery. *Designs and Analysis of Experiments.* 2012.

[4] Simpson T.W. et al. “Metamodels for computer-based engineering design: survey and recommendations”. In: *Eng. Comput.* 17 (2001), pp. 129–150.

[5] M. Arai and T. Shimizu. “Optimization of the design of ship structures using response surface methodology. Practical design of ships and other floating structures.” In: *Proceedings of the Eighth International Symposium on Practical Design of Ships and Other Floating Structures* (2001), pp. 331–339.

[6] P. Prebeg. “Multi-criteria Design of Complex Thin-walled Structures (Ph.D. Doctoral dissertation)”. In: (2011).

[7] E. Haghighat and R. Juanes. “SciANN: A Keras/TensorFlow wrapper for scientific computations and physics-informed deep learning”. In: *Comput. Methods Appl. Mech. Eng.* (2021), p. 371.

[8] Y. Shin. “On the Convergence of Physics Informed Neural Networks for Linear Second-Order Elliptic and Parabolic Type PDEs.” In: *CiCP* 28 (2020), pp. 2042–2074.

[9] M. Yin et al. “Non-invasive Inference of Thrombus Material Properties with PhysicsInformed Neural Networks.” In: *Comput. Methods Appl. Mech. Eng.* (2021), p. 375.

[10] Trung-Kien Nguyen Xuan-Bach Bui and Phong T. T. Nguyen. “Stochastic vibration and buckling analysis of functionally graded sandwich thin-walled beams”. In: *Mechanics Based Design of Structures and Machines* 52.4 (2024), pp. 2017–2039.

[11] Ming Chen et al. “Uncertainty quantification and global sensitivity analysis for composite cylinder shell via data-driven polynomial chaos expansion”. English. In: *Journal of Physics: Conference Series* 2174.1 (Jan. 2022), p. 012085.

[12] B. Gaspar and C. Guedes Soares. "Application of polynomial chaos expansions in stochastic analysis of plate elements under lateral pressure". English. In: *Maritime Technology and Engineering 3* (2016).

[13] D. Wei and S. Zhang. "Ultimate compressive strength prediction of stiffened panels by counterpropagation neural networks (CPN)". In: (1999).

[14] Dongyang Li et al. "Ultimate strength assessment of ship hull plate with multiple cracks under axial compression using artificial neural networks". In: *Ocean Engineering* 263 (2022), pp. 387–411.

[15] Yongchang Pu and Ehsan Mesbahi. "Application of artificial neural networks to evaluation of ultimate strength of steel panels". In: (2005).

[16] A. Cevik and I.H. Guzelbey. "A soft computing based approach for the prediction of ultimate strength of metal plates in compression." In: *Engineering Structures* 29 (2007), pp. 383–394.

[17] M. Zareei and M.R. Iranmanesh. "Ultimate strength formulation of stiffened panels under inplane compression or tension with cracking damage." In: *Journal of Naval Architecture and Marine Engineering* 15 (2018), pp. 1–16.

[18] M. Zareei and M.R. Iranmanesh. "U.K. Mallela and A. Upadhyay. Buckling load prediction of laminated composite stiffened panels subjected to in-plane shear using artificial neural networks." In: *Thin-walled Structures* 102 (2016), pp. 158–164.

[19] S. Kumar et al. "The prediction of buckling load of laminated composite hat-stiffened panels under compressive loading by using of neural networks." In: *The Open Civil Engineering Journal* 12 (2018), pp. 468–480.

[20] S. Tohidi and Y. Sharifi. "A new predictive model for restrained distortional buckling strength of half through bridge girders using artificial neural network". In: *KSCE Journal of Civil Engineering* 20 (2016), pp. 1392–1403.

[21] M. Abambres et al. "Neural network-based formula for the buckling load prediction of i-section cellular steel beams." In: *Computers* 8 (2018), pp. 1–26.

[22] C. Bisagni and L. Lanzi. "Post-buckling optimisation of composite stiffened panels using neural networks". In: *Composite Structures* 58 (2002), pp. 237–247.

[23] Torgeir Moen. *TMR4190 - Finite Element Modelling and Analysis of Marine Structures.* 2003.

[24] Jørgen Amdahl. *TMR4205 Buckling and Ultimate Strength of Marine Structures.* 2013.

[25] A. D. Gudbrandsen. "Implementing Artificial Neural Network to Predict Buckling Load of Stiffened Plates". In: (2023).

[26] Jørgen Amdahl. *TMR4205 - Buckling and Ultimate Strength of Marine Structures.* 2012.

[27] ABAQUS. *Element library: overview.* URL: `https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/usb/default.htm?startat=pt06ch23s06alm15.html`.

[28] ABAQUS. *Eigenvalue buckling prediction.* URL: `https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/stm/default.htm?startat=ch02s03ath17.html` (visited on 10/15/2023).

[29] Zhi-Hua Zhou. *Machine Learning.* Tsinghua University Press, 2016.

[30] Issam El Naqa and Martin J. Murphy. *What Is Machine Learning?* URL: `https://link.springer.com/chapter/10.1007/978-3-319-18305-3_1` (visited on 11/08/2023).

[31] AppliedGo. *Perceptrons - the most basic form of a neural network.* URL: `https://appliedgo.net/perceptron/` (visited on 09/28/2023).

[32] T. Sauder. "Fidelity of cyber-physical empirical methods". In: (2018).

[33] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133.

[34] F Rosenblatt. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms.* 1962.

[35] Encord. *Training, Validation, Test Split for Machine Learning Datasets.* URL: `https://encord.com/blog/train-val-test-split/` (visited on 10/15/2023).

[36] Jojo Moolayil. *Learn Keras for Deep Neural Networks.* 2019.

[37] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems* 2 (1989), pp. 303–314.

[38] Siddharth Sharma, Simone Sharma, and Jaipur Anidhya Athaiya. "ACTIVATION FUNCTIONS IN NEURAL NETWORKS". In: *International Journal of Engineering Applied Sciences and Technology* 4 (2020), pp. 310–316.

[39] *Curves-of-the-Sigmoid-Tanh-and-ReLu-activation.* URL: `https://www.researchgate.net/figure/Curves-of-the-Sigmoid-Tanh-and-ReLu-activation-functions_fig1_354971308` (visited on 10/02/2023).

[40] Ahmad Jafarian et al. "On artificial neural networks approach with new cost functions". In: *Applied Mathematics and Computation* 339 (2018), pp. 546–555.

[41] SuperAnnotate. *Overfitting and underfitting in machine learning.* URL: `https://www.superannotate.com/blog/overfitting-and-underfitting-in-machine-learning` (visited on 10/15/2023).

[42] S. Marelli and B. Sudret. *UQLab user manual – Polynomial Chaos Expansions.* Tech. rep. Report UQLab-V0.9-104. Chair of Risk, Safety & Uncertainty Quantification, ETH Zurich, 2015.

[43] R.G. Ghanem and P.D. Spanos. "Stochastic finite elements: A spectral approach". English. In: (1991).

[44] Geraud Blatman and Bruno Sudret. "An adaptive algorithm to build up sparse polynomial chaos expansions for stochastic finite element analysis". In: *Probabilistic Engineering Mechanics* 25 (2010), pp. 183–197.

[45] M. Berveiller, B. Sudret, and M. Lemaire. “Stochastic finite element: A non intrusive approach by regression”. English. In: *European Journal of Computational Mechanics* 15 (2006), pp. 81–92.

[46] ABAQUS. *23.6.2 Choosing a shell element.* URL: https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/usb/default.htm?startat=pt06ch23s06alm15.html (visited on 09/28/2023).

[47] xometry. *Yield Strength: Definition, Importance, Graphs, and How to Calculate.* URL: https://www.xometry.com/resources/3d-printing/yield-strength/ (visited on 10/18/2023).

[48] haiiliin. *abqpy 2024.* URL: https://pypi.org/project/abqpy/.

[49] TensorFlow. *Keras: The high-level API for TensorFlow.* URL: https://www.tensorflow.org/guide/keras (visited on 10/05/2023).

[50] Tensorlow. URL: https://www.tensorflow.org/ (visited on 10/05/2023).

[51] Diederik P. Kingma and Jimmy Lei Ba. “ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION”. In: (2014).

[52] T. M. Mitchell. *Machine Learning.* McGraw Hill, Mar. 1997, p. 2.

APPENDICES

A - ABQPY

A1 - Abaqus Model Generator

```
import numpy as np
from abaqus import *
from abaqusConstants import *
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from optimization import *
from job import *
from sketch import *
from visualization import *
from WorkingInputGen import *


def generate_jobs(jname, yld_plate, yld_stiffener, plate_thickness,
↪  flange_width, no_stiffeners, plate_width, web_thickness):
    # Use coordinates instead of GUI notations
    session.journalOptions.setValues(replayGeometry=COORDINATE,
     ↪  recoverGeometry=COORDINATE)

    web_height_ctc = web_height-web_thickness/2-plate_thickness/2
    plate_half_width = plate_width/2

    mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
     ↪  sheetSize=200.0)
    mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.0,
     ↪  0.0),
        point2=(plate_width, 0.0))
```

```
mdb.models['Model-1'].sketches['__profile__'].Line(point1=
    (plate_half_width, 0.0), point2=(
    plate_half_width, web_height_ctc))


mdb.models['Model-1'].sketches['__profile__'].Line(point1=
(plate_half_width-flange_width/2, web_height_ctc),
 ↪  point2=(plate_half_width+flange_width/2, web_height_ctc))


mdb.models['Model-1'].Part(dimensionality=THREE_D,
 ↪  name='Part-1', type=
    DEFORMABLE_BODY)
mdb.models['Model-1'].parts['Part-1'].BaseShellExtrude
    (depth=unit_length, sketch=
    mdb.models['Model-1'].sketches['__profile__'])
del mdb.models['Model-1'].sketches['__profile__']
mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
 ↪  sheetSize=200.0)
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.0,
 ↪  0.0), point2=(
    plate_width*no_stiffeners, 0.0))
mdb.models['Model-1'].Part(dimensionality=THREE_D,
 ↪  name='Part-2', type=
    DEFORMABLE_BODY)
mdb.models['Model-1'].parts['Part-2'].BaseWire(sketch=
    mdb.models['Model-1'].sketches['__profile__'])
del mdb.models['Model-1'].sketches['__profile__']
mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
 ↪  sheetSize=200.0)
mdb.models['Model-1'].sketches['__profile__'].rectangle(
point1=(0.0, 0.0), point2=(plate_width*no_stiffeners,
 ↪  trans_height))
mdb.models['Model-1'].Part(dimensionality=THREE_D,
 ↪  name='Part-4', type=
    DEFORMABLE_BODY)
mdb.models['Model-1'].parts['Part-4'].BaseSolidExtrude(depth=5,
 ↪  sketch=
    mdb.models['Model-1'].sketches['__profile__'])
del mdb.models['Model-1'].sketches['__profile__']

# Material building
mdb.models['Model-1'].Material(name='rigid_beam')

 ↪  mdb.models['Model-1'].materials['rigid_beam'].Density(table=((7.85,
 ↪  ), ))
```

```
mdb.models['Model-1'].materials['rigid_beam'].
    DeformationPlasticity(table=((E_modulus,
    poison_ratio, yld_plate, n_plate, offset_plate), ))

mdb.models['Model-1'].Material(name='alu_plate')

↪   mdb.models['Model-1'].materials['alu_plate'].Density(table=((density,
↪   ), ))
mdb.models['Model-1'].materials['alu_plate'].
    DeformationPlasticity(table=((E_modulus,
    poison_ratio, yld_plate, n_plate, offset_plate), ))

mdb.models['Model-1'].Material(name='alu_stiffener')

↪   mdb.models['Model-1'].materials['alu_stiffener'].Density(table=((density,
↪   ), ))
mdb.models['Model-1'].materials['alu_stiffener'].
    DeformationPlasticity(table=((E_modulus,
    poison_ratio, yld_stiffener, n_stiffener,
     ↪   offset_stiffener), ))

mdb.models['Model-1'].Material(name='trav_stiffener')

↪   mdb.models['Model-1'].materials['trav_stiffener'].Density(table=((density,
↪   ), ))
mdb.models['Model-1'].materials['trav_stiffener'].
    DeformationPlasticity(table=((E_modulus,
    poison_ratio, yld_stiffener, n_stiffener,
     ↪   offset_stiffener), ))



# Section building

↪   mdb.models['Model-1'].HomogeneousShellSection(idealization=NO_IDEALIZATION,
    integrationRule=SIMPSON, material='alu_plate',
     ↪   name='plate', nodalThicknessField=
    '', numIntPts=5, poissonDefinition=DEFAULT,
     ↪   preIntegrate=OFF, temperature=
    GRADIENT, thickness=plate_thickness, thicknessField='',
     ↪   thicknessModulus=None,
    thicknessType=UNIFORM, useDensity=OFF)

↪   mdb.models['Model-1'].HomogeneousShellSection(idealization=NO_IDEALIZATION,
    integrationRule=SIMPSON, material='alu_stiffener',
     ↪   name='web', nodalThicknessField=''
    , numIntPts=5, poissonDefinition=DEFAULT, preIntegrate=OFF,
     ↪   temperature=
    GRADIENT, thickness=web_thickness, thicknessField='',
     ↪  thicknessModulus=None,
    thicknessType=UNIFORM, useDensity=OFF)
mdb.models['Model-1'].HomogeneousShellSection(
    idealization=NO_IDEALIZATION,
    integrationRule=SIMPSON, material='alu_stiffener',
     ↪  name='flange',
    nodalThicknessField='', numIntPts=5,
     ↪  poissonDefinition=DEFAULT,
    preIntegrate=OFF, temperature=GRADIENT,
     ↪  thickness=web_thickness, thicknessField='',
    thicknessModulus=None, thicknessType=UNIFORM,
     ↪  useDensity=OFF)
mdb.models['Model-1'].HomogeneousSolidSection(
    material='TRAV_STIFFENER', name='travstiff',
     ↪  thickness=None)
mdb.models['Model-1'].CircularProfile(name='beam', r=5.0)
mdb.models['Model-1'].BeamSection(consistentMassMatrix=False,
 ↪  integration=
    DURING_ANALYSIS, material='rigid_beam', name='beam',
     ↪  poissonRatio=0.0,
    profile='beam', temperatureVar=LINEAR)

# Mesh

 ↪  mdb.models['Model-1'].parts['Part-1'].seedPart(deviationFactor=0.1,
    minSizeFactor=0.1, size=mesh_size)
mdb.models['Model-1'].parts['Part-1'].setElementType(elemTypes=
    (ElemType(
    elemCode=S4R, elemLibrary=STANDARD,
     ↪  secondOrderAccuracy=OFF,
    hourglassControl=DEFAULT), ElemType(elemCode=S3,
     ↪  elemLibrary=STANDARD)),
    regions=(mdb.models['Model-1'].parts['Part-1'].faces.
    getSequenceFromMask((
    '[#f ]', ), ), ))

 ↪  mdb.models['Model-1'].parts['Part-1'].setMeshControls(elemShape=QUAD,
 ↪  regions=
    mdb.models['Model-1'].parts['Part-1'].faces.
    getSequenceFromMask(('[#f ]',
    ), ), technique=STRUCTURED)
mdb.models['Model-1'].parts['Part-1'].generateMesh()

 ↪  mdb.models['Model-1'].parts['Part-2'].seedPart(deviationFactor=0.1,
    minSizeFactor=0.1, size=10)

 ↪  mdb.models['Model-1'].parts['Part-2'].setElementType(elemTypes=(ElemTyp
    elemCode=B31, elemLibrary=STANDARD), ), regions=(

     ↪  mdb.models['Model-1'].parts['Part-2'].edges.getSequenceFromMask(('[#f
     ↪  ]', ),
    ), ))
mdb.models['Model-1'].parts['Part-2'].generateMesh()


 ↪  mdb.models['Model-1'].parts['Part-4'].seedPart(deviationFactor=0.1,
    minSizeFactor=0.1, size=mesh_size)

 ↪  mdb.models['Model-1'].parts['Part-4'].setElementType(elemTypes=(ElemType(
    elemCode=S4R, elemLibrary=STANDARD,
     ↪  secondOrderAccuracy=OFF,
    hourglassControl=DEFAULT), ElemType(elemCode=S3,
     ↪  elemLibrary=STANDARD)),
    regions=(mdb.models['Model-1'].parts['Part-4'].faces.
    getSequenceFromMask(('[#f ]', ), ), ))

mdb.models['Model-1'].parts['Part-4'].generateMesh()

# Set assignment
mdb.models['Model-1'].parts['Part-1'].Set(faces=

     ↪  mdb.models['Model-1'].parts['Part-1'].faces.findAt(((plate_half_width-
    flange_width/2, web_height_ctc,
    0.0), (plate_half_width, web_height_ctc, 1.0)),
     ↪  ((plate_half_width+1, web_height_ctc, 0.0),
     ↪  (plate_half_width+flange_width/2, web_height_ctc,
     ↪  1.0)),
    ), name='flange')



mdb.models['Model-1'].parts['Part-1'].Set(faces=

     ↪  mdb.models['Model-1'].parts['Part-1'].faces.findAt(((plate_half_width,
     ↪  0,
    0), (plate_half_width, web_height_ctc, 1)), ),
                                    name='web')

mdb.models['Model-1'].parts['Part-4'].Set(faces=
    mdb.models['Model-1'].parts['Part-4'].faces.findAt(((0, 0,
    0), (plate_width*no_stiffeners, trans_height, 0)), ),
                                    name='travstiff')


mdb.models['Model-1'].parts['Part-1'].Set(faces=
    mdb.models['Model-1'].parts['Part-1'].faces.findAt(((0.0,
     ↪  0.0,
    0.0), (plate_half_width, 0.0, 1.0)), ((plate_half_width+1,
     ↪  0.0, 0.0), (plate_width, 0.0, 1.0)),
    ), name='plate')


mdb.models['Model-1'].parts['Part-2'].Set(edges=
    mdb.models['Model-1'].parts['Part-2'].edges.findAt(((0, 0,
     ↪  0), )),
    name='beam')



# Section assignment

↪   mdb.models['Model-1'].parts['Part-1'].SectionAssignment(offset=0.0,
                        offsetField='',
                         ↪  offsetType=MIDDLE_SURFACE, region=

                         ↪  mdb.models['Model-1'].parts['Part-1'].sets[
                            'web'],
                        sectionName='web',
                         ↪  thicknessAssignment=FROM_SECTION)

↪   mdb.models['Model-1'].parts['Part-1'].SectionAssignment(offset=0.0,
    offsetField='', offsetType=MIDDLE_SURFACE, region=
    mdb.models['Model-1'].parts['Part-1'].sets['flange'],
    sectionName='flange', thicknessAssignment=FROM_SECTION)

↪   mdb.models['Model-1'].parts['Part-1'].SectionAssignment(offset=0.0,
                        offsetField='',
                         ↪  offsetType=MIDDLE_SURFACE, region=

                         ↪  mdb.models['Model-1'].parts['Part-1'].sets[
                            'plate'],
                        sectionName='plate',
                        thicknessAssignment=FROM_SECTION)

↪   mdb.models['Model-1'].parts['Part-4'].SectionAssignment(offset=0.0,
                        offsetField='',
                         ↪  offsetType=MIDDLE_SURFACE, region=

                         ↪  mdb.models['Model-1'].parts['Part-4'].sets[
                            'travstiff'],
                        sectionName='travstiff',
                        thicknessAssignment=FROM_SECTION)
```

```
# Assembly into structures
mdb.models['Model-1'].rootAssembly.Instance(dependent=ON,
 ↪  name='Part-1-1',
    part=mdb.models['Model-1'].parts['Part-1'])
mdb.models['Model-1'].rootAssembly.Instance(dependent=ON,
 ↪  name='Part-4-1',
    part=mdb.models['Model-1'].parts['Part-4'])


 ↪  mdb.models['Model-1'].rootAssembly.LinearInstancePattern(direction1=(1.0,
 ↪  0.0,
    0.0), direction2=(0.0, 1.0, 0.0), instanceList=('Part-1-1',
     ↪  ), number1=no_stiffeners,
    number2=1, spacing1=plate_width, spacing2=73.5)
if (no_stiffeners == 3):

     ↪  mdb.models['Model-1'].rootAssembly.InstanceFromBooleanMerge(domain=MESH,

         ↪  instances=(mdb.models['Model-1'].rootAssembly.instances['Part-1-1'],

         ↪  mdb.models['Model-1'].rootAssembly.instances['Part-1-1-lin-2-1'],
         ↪  mdb.models['Model-1'].rootAssembly.instances['Part-1-1-lin-3-1'],
         ↪  mdb.models['Model-1'].rootAssembly.instances['Part-4-1']),
        mergeNodes=BOUNDARY_ONLY, name='Part-3',
         ↪  nodeMergingTolerance=1e-06,
        originalInstances=DELETE)
elif (no_stiffeners == 4):

     ↪  mdb.models['Model-1'].rootAssembly.InstanceFromBooleanMerge(domain=MESH,

         ↪  instances=(mdb.models['Model-1'].rootAssembly.instances['Part-1-1'],

         ↪  mdb.models['Model-1'].rootAssembly.instances['Part-1-1-lin-2-1'],
         ↪  mdb.models['Model-1'].rootAssembly.instances['Part-1-1-lin-3-1'],
         ↪  mdb.models['Model-1'].rootAssembly.instances['Part-1-1-lin-4-1'],
         ↪  mdb.models['Model-1'].rootAssembly.instances['Part-4-1']),
        mergeNodes=BOUNDARY_ONLY, name='Part-3',
         ↪  nodeMergingTolerance=1e-06,
        originalInstances=DELETE)
elif (no_stiffeners == 2):

     ↪  mdb.models['Model-1'].rootAssembly.InstanceFromBooleanMerge(domain=MESH,

         ↪  instances=(mdb.models['Model-1'].rootAssembly.instances['Part-1-1'],

         ↪  mdb.models['Model-1'].rootAssembly.instances['Part-1-1-lin-2-1'],
         ↪  mdb.models['Model-1'].rootAssembly.instances['Part-4-1']),
        mergeNodes=BOUNDARY_ONLY, name='Part-3',
         ↪  nodeMergingTolerance=1e-06,
        originalInstances=DELETE)
elif (no_stiffeners == 5):

     ↪   mdb.models['Model-1'].rootAssembly.InstanceFromBooleanMerge(
        domain=MESH,
        instances=(mdb.models['Model-1'].rootAssembly.
        instances['Part-1-1'],
        mdb.models['Model-1'].rootAssembly.
        instances['Part-1-1-lin-2-1'],
        mdb.models['Model-1'].rootAssembly.
        instances['Part-1-1-lin-3-1'],
        mdb.models['Model-1'].rootAssembly.
        instances['Part-1-1-lin-4-1'],
        mdb.models['Model-1'].rootAssembly.
        instances['Part-1-1-lin-5-1'],
        mdb.models['Model-1'].rootAssembly.
        instances['Part-4-1']),
        mergeNodes=BOUNDARY_ONLY, name='Part-3',
         ↪  nodeMergingTolerance=1e-06,
        originalInstances=DELETE)
elif (no_stiffeners == 1):
    mdb.models['Model-1'].rootAssembly.InstanceFromBooleanMerge
        (domain=MESH,
        instances=(mdb.models['Model-1'].rootAssembly.
        instances['Part-1-1'],
        mdb.models['Model-1'].rootAssembly.
        instances['Part-4-1']),
        mergeNodes=BOUNDARY_ONLY, name='Part-3',
         ↪  nodeMergingTolerance=1e-06,
        originalInstances=DELETE)
elif (no_stiffeners == 6):

     ↪   mdb.models['Model-1'].rootAssembly.InstanceFromBooleanMerge(domain=
        MESH,instances=(mdb.models['Model-1'].rootAssembly.
        instances['Part-1-1'],
        mdb.models['Model-1'].rootAssembly.
        instances['Part-1-1-lin-2-1'],
         ↪  mdb.models['Model-1'].rootAssembly.
        instances['Part-1-1-lin-3-1'],
         ↪  mdb.models['Model-1'].rootAssembly.
        instances['Part-1-1-lin-4-1'],
         ↪  mdb.models['Model-1'].rootAssembly.
        instances['Part-1-1-lin-5-1'],
         ↪  mdb.models['Model-1'].rootAssembly.
        instances['Part-1-1-lin-6-1'],
         ↪  mdb.models['Model-1'].rootAssembly.
        instances['Part-4-1']),
        mergeNodes=BOUNDARY_ONLY, name='Part-3',
         ↪  nodeMergingTolerance=1e-06,
        originalInstances=DELETE)
mdb.models['Model-1'].rootAssembly.Instance(dependent=ON,
 ↪  name='Part-2-1',
    part=mdb.models['Model-1'].parts['Part-2'])

 ↪  mdb.models['Model-1'].rootAssembly.translate(instanceList=('Part-2-1',
 ↪  ), vector=
    (0.0, e_c, -bearing_width))
mdb.models['Model-1'].rootAssembly.Instance(dependent=ON,
 ↪  name='Part-2-2',
    part=mdb.models['Model-1'].parts['Part-2'])

 ↪  mdb.models['Model-1'].rootAssembly.translate(instanceList=('Part-2-2',
 ↪  ), vector=
    (0.0, e_c, bearing_width+unit_length))



# Create steps
mdb.models['Model-1'].ImplicitDynamicsStep(name='Step-1',
 ↪  nlgeom=ON, previous=
    'Initial',alpha=DEFAULT, amplitude=RAMP,
     ↪  application=QUASI_STATIC, initialConditions=OFF,
     ↪  nohaf=OFF, maxInc=1,
                                        maxNumInc=100,
                                         ↪  initialInc=0.0001,
                                         ↪  minInc=1e-15,
                                         ↪  timePeriod=75.0)




# Amplitude assignment
mdb.models['Model-1'].TabularAmplitude(data=((0.0, 0.0),
 ↪  (200.0, 10.0)),
    name='disp_amp', smooth=SOLVER_DEFAULT, timeSpan=STEP)

# BC assignment
longitudinal_BC1 =
 ↪  mdb.models['Model-1'].rootAssembly.Set(name='longitudinal_BC1',
 ↪  nodes=

     ↪  mdb.models['Model-1'].rootAssembly.instances['Part-3-1'].nodes.
    getByBoundingBox(0, 0, 0, 0+mesh_size/2, 0, unit_length))
```

```
longitudinal_BC2 =
↪   mdb.models['Model-1'].rootAssembly.Set(name='longitudinal_BC2',
↪   nodes=

    ↪   mdb.models['Model-1'].rootAssembly.instances['Part-3-1'].nodes.
    getByBoundingBox(no_stiffeners*plate_width-mesh_size/2, 0,
    ↪   0,
                    no_stiffeners*plate_width+mesh_size, 0,
                    ↪   unit_length))
transverse_BC1 =
↪   mdb.models['Model-1'].rootAssembly.Set(name='transverse_BC1',
↪   nodes=

    ↪   mdb.models['Model-1'].rootAssembly.instances['Part-3-1'].nodes.
    getByBoundingBox(-plate_half_width, 0, 0,
                    no_stiffeners*plate_width+mesh_size,
                    ↪   web_height_ctc+1, 0))
transverse_BC2 =
↪   mdb.models['Model-1'].rootAssembly.Set(name='transverse_BC2',
↪   nodes=

    ↪   mdb.models['Model-1'].rootAssembly.instances['Part-3-1'].nodes.
    getByBoundingBox(-plate_half_width, 0, unit_length,
                    no_stiffeners*plate_width+mesh_size,
                    ↪   web_height_ctc+1, unit_length))

mdb.models['Model-1'].DisplacementBC(amplitude=UNSET,
↪   createStepName='Initial',
                            distributionType=UNIFORM,
                            ↪   fieldName='', localCsys=None,
                            name=loading_direction + '_BC1',

                            ↪   region=mdb.models['Model-1'].rootAssembly.
                            sets[loading_direction + '_BC1'],
                            u1=SET, u2=UNSET, u3=UNSET,
                            ↪   ur1=UNSET, ur2=UNSET,
                            ↪   ur3=UNSET)
mdb.models['Model-1'].DisplacementBC(amplitude=UNSET,
↪   createStepName='Initial',
                            distributionType=UNIFORM,
                            ↪   fieldName='', localCsys=None,
                            name=loading_direction + '_BC2',

                            ↪   region=mdb.models['Model-1'].rootAssembly.
                            sets[loading_direction + '_BC2'],
                            u1=SET, u2=UNSET, u3=UNSET,
                            ↪   ur1=UNSET, ur2=UNSET,
                            ↪   ur3=UNSET)
```

```
mdb.models['Model-1'].ConcentratedForce(cf3=-100.0,
 ↪  createStepName='Step-1',
    distributionType=UNIFORM, field='', localCsys=None,
     ↪  name='Load-1', region=
    mdb.models['Model-1'].rootAssembly.sets['transverse_BC1'])


mdb.models['Model-1'].rootAssembly.Set(edges=

     ↪  mdb.models['Model-1'].rootAssembly.instances['Part-2-1'].edges.
    getSequenceFromMask(
    ('[#f ]', ), ), name='moving_beam')
mdb.models['Model-1'].rootAssembly.Set(edges=

     ↪  mdb.models['Model-1'].rootAssembly.instances['Part-2-2'].edges.
    getSequenceFromMask(
    ('[#f ]', ), ), name='fixed_beam')

mdb.models['Model-1'].DisplacementBC(amplitude='disp_amp',
 ↪  createStepName='Step-1',
                                    distributionType=UNIFORM,
                                     ↪  fieldName='',
                                     ↪  localCsys=None,
                                    name='moving_beam',

                                     ↪  region=mdb.models['Model-1'].rootAssembly
                                    .sets['moving_beam'],
                                    u1=SET, u2=SET, u3=UNSET,
                                     ↪  ur1=SET, ur2=SET,
                                     ↪  ur3=UNSET)
mdb.models['Model-1'].DisplacementBC(amplitude=UNSET,
 ↪  createStepName='Initial',
                                    distributionType=UNIFORM,
                                     ↪  fieldName='',
                                     ↪  localCsys=None,
                                    name='fixed_beam',

                                     ↪  region=mdb.models['Model-1'].rootAssembly
                                    .sets['fixed_beam'],
                                    u1=SET, u2=SET, u3=SET,
                                     ↪  ur1=UNSET, ur2=SET,
                                     ↪  ur3=SET)

# Assign beam property
```

```
↪   mdb.models['Model-1'].parts['Part-2'].SectionAssignment(offset=0.0,
↪   offsetField=
    '', offsetType=MIDDLE_SURFACE,
     ↪  region=mdb.models['Model-1'].parts['Part-2'].sets['beam']
    , sectionName='beam', thicknessAssignment=FROM_SECTION)

↪   mdb.models['Model-1'].parts['Part-2'].assignBeamSectionOrientation
    (method=
    N1_COSINES, n1=(0.0, 0.0, -1.0), region=
    mdb.models['Model-1'].parts['Part-2'].sets['beam'])

# Rigid beam assignments
mdb.models['Model-1'].rootAssembly.ReferencePoint(point=
    mdb.models['Model-1'].rootAssembly.instances['Part-2-1'].
    InterestingPoint(

     ↪  mdb.models['Model-1'].rootAssembly.instances['Part-2-1'].edges[0],
     ↪  MIDDLE))
mdb.models['Model-1'].RigidBody(bodyRegion=
    mdb.models['Model-1'].rootAssembly.sets['moving_beam'],
     ↪  name='moving_beam',
    refPointRegion=Region(referencePoints=(
    mdb.models['Model-1'].rootAssembly.referencePoints.findAt
    ((plate_width*no_stiffeners/2, e_c, -bearing_width)), )))

mdb.models['Model-1'].rootAssembly.ReferencePoint(point=
    mdb.models['Model-1'].rootAssembly.instances['Part-2-2'].
    InterestingPoint(

     ↪  mdb.models['Model-1'].rootAssembly.instances['Part-2-2'].edges[0],
     ↪  MIDDLE))
mdb.models['Model-1'].RigidBody(bodyRegion=
    mdb.models['Model-1'].rootAssembly.sets['fixed_beam'],
     ↪  name='fixed_beam',
    refPointRegion=Region(referencePoints=(
    mdb.models['Model-1'].rootAssembly.referencePoints.
    findAt((plate_width*no_stiffeners/2, e_c,
    bearing_width+unit_length)), )))

# Constraints assignment
mdb.models['Model-1'].rootAssembly.Set(name='master_nodes_1',
↪   nodes=

     ↪  mdb.models['Model-1'].rootAssembly.instances['Part-2-1'].nodes.
        getByBoundingBox(
        plate_width*no_stiffeners/2-mesh_size/2, e_c,
         ↪  -bearing_width,
        plate_width*no_stiffeners/2+1, e_c, -bearing_width))

 ↪   mdb.models['Model-1'].MultipointConstraint(controlPoint=mdb.models['Model-1']
                csys=None, mpcType=BEAM_MPC,
                 ↪   name='Constraint-1', surface=

                 ↪   mdb.models['Model-1'].rootAssembly.sets[unloading_direction
                 ↪   + '_BC1'],
                userMode=DOF_MODE_MPC,
                userType=0)

mdb.models['Model-1'].rootAssembly.Set(name='master_nodes_2',
 ↪   nodes=

     ↪   mdb.models['Model-1'].rootAssembly.instances['Part-2-2'].nodes.
        getByBoundingBox(
        plate_width*no_stiffeners/2-mesh_size/2, e_c,
         ↪   bearing_width+unit_length,
        plate_width*no_stiffeners/2+1, e_c,
         ↪   bearing_width+unit_length))

 ↪   mdb.models['Model-1'].MultipointConstraint(controlPoint=mdb.models['Model-1']
                csys=None, mpcType=BEAM_MPC,
                 ↪   name='Constraint-2', surface=

                 ↪   mdb.models['Model-1'].rootAssembly.sets[unloading_direction
                 ↪   + '_BC2'],
                userMode=DOF_MODE_MPC,
                userType=0)

# Output setups


 ↪   mdb.models['Model-1'].HistoryOutputRequest(createStepName='Step-1',
 ↪   name=
    'H-Output-1', rebar=EXCLUDE, frequency=1, region=
    mdb.models['Model-1'].rootAssembly.sets['fixed_beam'],
     ↪   sectionPoints=DEFAULT
    , variables=('CF1', 'CF2', 'CF3', 'CM1', 'CM2', 'CM3'))

 ↪   mdb.models['Model-1'].HistoryOutputRequest(createStepName='Step-1',
 ↪   name=
    'H-Output-2', rebar=EXCLUDE, frequency=1, region=
    mdb.models['Model-1'].rootAssembly.sets['moving_beam'],
     ↪   sectionPoints=
    DEFAULT, variables=('U1', 'U2', 'U3', 'UR1', 'UR2', 'UR3'))

# Job creation
```

```
    mdb.Job(atTime=None, contactPrint=OFF, description='',
     ↪  echoPrint=OFF,
        explicitPrecision=SINGLE, getMemoryFromAnalysis=True,
         ↪  historyPrint=OFF,
        memory=90, memoryUnits=PERCENTAGE, model='Model-1',
         ↪  modelPrint=ON,
        multiprocessingMode=DEFAULT, name=jname,
         ↪  nodalOutputPrecision=FULL,
        numCpus=4, numDomains=4, numGPUs=0, queue=None,
         ↪  resultsFormat=ODB, scratch=
        '', type=ANALYSIS, userSubroutine='', waitHours=0,
         ↪  waitMinutes=0)

    # Input file write
    mdb.jobs[jname].writeInput(consistencyChecking=OFF)
    mdb.jobs[jname].submit()
    mdb.close()


jnamelist = []
for no in no_stiffeners_gen:
    nosti = no
    for y_p in yld_plate_gen:
        yieldp = y_p
        for y_s in yld_stiffener_gen:
            yields = y_s
            for fw in flange_width_gen:
                flangew = fw
                for t in plate_thickness_gen:
                    thick = t
                    for sd in plate_width_gen:
                        platew = sd
                        for st in stiff_thickness_gen:
                            stithi = st
                            jname =
                             ↪  'Testing'+str(no_stiffeners_gen.index(no))+'_yp

                             ↪  #jnamelist.append('t_'+str(plate_thickness_gen.

                             ↪  #jnamelist.append('_t_'+str(plate_thickness_gen
                            generate_jobs(jname, yieldp, yields,
                             ↪  thick, flangew, nosti, platew,
                             ↪  stithi)
```

A2 - Code for Testing Model Outside of Training Range

```
from abaqus import *
from abaqusConstants import *
from caeModules import *
import visualization
#from viewerModules import *
from driverUtils import executeOnCaeStartup
import odbAccess
import time
import csv
from WorkingInputGen import *

start = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
import regionToolset
import os

def extractor(jname, yld_plate, yld_stiffener, plate_thickness,
 ↪  flange_width, no_stiffeners, plate_width, web_thickness,
 ↪  mesh_size):

    executeOnCaeStartup()
    #session.reset()
    o2 = session.openOdb(name=jname+'.odb')
    session.viewports['Viewport: 1'].setValues(displayedObject=o2)
    session.viewports['Viewport: 1'].makeCurrent()
    odb = session.openOdb(jname+'.odb')

    path = ''

    for xyData in session.xyDataObjects.keys():
        del session.xyDataObjects[xyData]

    session.xyDataListFromField(odb=odb, outputPosition=NODAL,
     ↪  variable=(('RF',
        NODAL, ((COMPONENT, 'RF3'), )), ), nodePick=(('ASSEMBLY',
         ↪  1, ('[#2 ]', )),
        ), )

    session.xyDataListFromField(odb=odb, outputPosition=NODAL,
     ↪  variable=(('U',
        NODAL, ((COMPONENT, 'U3'), )), ), nodePick=(('ASSEMBLY', 1,
         ↪  ('[#1 ]', )),
        ), )
    u_val = session.xyDataObjects['U:U3 PI: ASSEMBLY N: 1']
    rf_val = session.xyDataObjects['RF:RF3 PI: ASSEMBLY N: 2']
```

```
#xy3 = combine(xy1, -xy2/1000)

file = open(jname+"rfpost.csv", "w")
for i in range(len(rf_val)):
    file.write(str(rf_val[i]).replace('(', '').replace(')', '')
     ↪  +"\n")

file.close()


file = open(jname+"upost.csv", "w")
for i in range(len(u_val)):
    file.write(str(u_val[i]).replace('(', '').replace(')', '')
     ↪  +"\n")

file.close()

input_file = jname+'upost.csv'
output_file = jname+'output.csv'

right_values_u = jname+'right_values_u'
right_values_u = []

with open(input_file, 'r') as infile:
    reader = csv.reader(infile, delimiter = ',')
    for row in reader:
        right_values_u.append(float(row[1]))#

with open(output_file, 'w') as outfile:
    writer = csv.writer(outfile)
    for value in right_values_u:
        writer.writerow([value])

input_file_rf = jname+'rfpost.csv'
output_file_rf = jname+'outputrf.csv'

right_values_rf = jname+'right_values_rf'
right_values_rf = []

with open(input_file_rf, 'r') as infile:
    reader = csv.reader(infile, delimiter = ',')
    for row in reader:
        right_values_rf.append(float(row[1]))#

with open(output_file_rf, 'w') as outfile:
    writer = csv.writer(outfile)
    for value in right_values_rf:
        writer.writerow([value])
```

```
#Create a output file and write the obtained data
text = open(jname + ".csv", "w")
text.write("%60s" % (jname))
text.write("\n")
text.write('%8s%4s%17s%2s%17s%5s%12s%10s' % (
" ", "Disp", " ", "RF", " ", "Slope", " ", "Slope Rate"))
text.write("\n")
#Calculate the buckling load from the slope change
#of the force-displacement data
slope = 0.0
slope_list = [0.0]
slope_rate_list = [0.0, 0.0]
buckle_found = False
buckle_load = 0.0
for i in range(0, len(rf_val)):
    if i > 0:
        slope = (right_values_rf[i] - right_values_rf[i - 1]) /
        ↪  (right_values_u[i] - right_values_u[i - 1])
        slope_list.append(slope)
    if i > 1:
        slope_rate = abs(slope_list[i] - slope_list[i - 1]) /
        ↪  slope_list[i]
        slope_rate_list.append(slope_rate)
        if slope_rate > 0.1 and buckle_found == False:
            buckle_load = right_values_rf[i]
            buckle_found = True
    text.write('%20.6f%20.2f%20.2f%20.2f' % (right_values_u[i],
    ↪  right_values_rf[i], slope, slope_rate_list[i]))
    text.write("\n")

slope_list = slope_list[:-1]
slope_rate_list = slope_rate_list[:-1]
text.write("\n")
text.write("%20s%20.2f" % ("Buckle Load :", abs(buckle_load)))
text.write("\n")
text.close()
process = open(path + jname + ".pro", "a")
process.write("Output file is created.\n")
end = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
process.write(end)
process.close()

resultfile = open("results.csv", "a")
```

```
     ↪  resultfile.write('%20.6f%20.2f%20.2f%20.2f%20.2f%20.2f%20.2f%20.2f%20.2
     ↪  % (yld_plate, yld_stiffener, plate_thickness, flange_width,
     ↪  no_stiffeners, plate_width, web_thickness, mesh_size,
     ↪  abs(buckle_load)))
    resultfile.write("\n")
    resultfile.close()



    mdb.close()



jnamelist = []




for no in no_stiffeners_gen:
    nosti = no
    for y_p in yld_plate_gen:
        yieldp = y_p
        for y_s in yld_stiffener_gen:
            yields = y_s
            for fw in flange_width_gen:
                flangew = fw
                for t in plate_thickness_gen:
                    thick = t
                    for sd in plate_width_gen:
                        platew = sd
                        for st in stiff_thickness_gen:
                            stithi = st
                            for ms in mesh_size_gen:
                                mesi = ms
                                jname =
                                 ↪  'ConvTest'+str(mesh_size_gen.index(ms))+'_m
                                extractor(jname, yieldp, yields,
                                 ↪  thick, flangew, nosti, platew,
                                 ↪  stithi, mesi)
```

A3 - Input File

```
# Geometry of L-bar unit
unit_length = 750
no_stiffeners_gen = [3]
#no_stiffeners_gen = [1, 2, 3, 4, 5, 6]

plate_width_gen = [260]
#plate_width_gen = [240, 260, 320]
web_height = 40
mid_lines_large = 118
#flange_width_gen = [30, 35, 42]
flange_width_gen = [40]
#flange_width_gen = [30, 40]
#web_thickness = 5
#stiff_thickness_gen = [3.5, 4, 4.5]
stiff_thickness_gen = [3]
#flange_thickness = 5
plate_thickness_gen = [3]
#plate_thickness_gen = [2, 3, 4]
e_c = 20
bearing_width = 10
trans_thickness = 5
trans_height = web_height+20


# Material property
density = 7.85e-9
E_modulus = 210000
poison_ratio = 0.3

#yld_plate_gen = [190, 245, 355]
yld_plate_gen = [245]
#yld_plate_gen = [280]
n_plate = 35.6
offset_plate = 0.438

yld_stiffener_gen = [235]
#yld_stiffener_gen = [235, 355, 400]
n_stiffener = 23.6
offset_stiffener = 0.452


# Mesh

mesh_size = 10
```

```
# Load

loading_direction = 'longitudinal'
unloading_direction = 'transverse'
```

B - NEURAL NETWORK CODE

B1 - ANN Model Generator

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import regularizers
from tensorflow.keras.layers import Dense, Dropout
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Load data
data = pd.read_csv('resultsinside.csv')
X = data[['PlateStrength', 'StiffStrength', 'Thickness',
↪  'FlangeWidth', 'Stiffeners', 'PlateWidth', 'StiffThick']]
y = data['BucklingLoad']

#Splitting data into testing and validation sets
X_temp, X_test, y_temp, y_test = train_test_split(X, y,
↪  test_size=0.1, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp,
↪  test_size=0.15, random_state=42)

print("Shape of x_train:",X_train.shape)
print("Shape of x_val:",X_val.shape)
print("Shape of x_test:",X_test.shape)
print("Shape of y_train:",y_train.shape)
print("Shape of y_val:",y_val.shape)
print("Shape of y_test:",y_test.shape)

#Scaling the data
```

```
scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# Define and compile the model
model = tf.keras.Sequential([
    Dense(128, activation='relu', input_shape=(7,)),
    #Dropout(0.2),
    Dense(64, activation='relu',
    ↪  kernel_regularizer=regularizers.l2(0.001)),
    #Dropout(0.2),
    Dense(32, activation='relu'),
    ↪  #kernel_regularizer=regularizers.l2(0.001)),
    #Dense(32, activation='relu'),
    #Dropout(0.2),
    Dense(16, activation='relu'),
    Dense(1)
])

model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Train the model
history = model.fit(X_train_scaled, y_train,
↪  validation_data=(X_val_scaled, y_val), epochs=1200)

loss, mae = model.evaluate(X_test_scaled, y_test)
print(f"Mean Absolute Error on test data: {mae}")

predicted_loads = model.predict(X_test_scaled)

# Plots

plt.figure(figsize=(10, 6))

#Scatter plot of true vs. predicted values
plt.scatter(y_test, predicted_loads, alpha=0.6, edgecolors="w",
↪  linewidth=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
↪  'r')  # red line for perfect fit

plt.xlabel('True Buckling Loads')
plt.ylabel('Predicted Buckling Loads')
plt.title('True vs. Predicted Buckling Loads Joint Dataset')
plt.grid(True)
plt.tight_layout()
plt.show()
plt.figure(figsize=(10, 6))

# Plot training & validation loss values
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss over Epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.yscale('log')
plt.legend(loc='upper right')
plt.grid(True)
plt.tight_layout()
plt.show()

def plot_percentage_error_distribution(y_true, y_pred):
    # Calculate percentage errors
    percentage_errors = ((y_pred.flatten() - y_true) / y_true) *
     ↪  100
    plt.figure(figsize=(10, 6))
    plt.hist(percentage_errors, bins=50, edgecolor='black',
     ↪  color='skyblue')
    plt.xlabel('Percentage Error (%)')
    plt.ylabel('Frequency')
    plt.title('Error Distribution Joint Dataset')
    plt.grid(True)
    plt.tight_layout()
    plt.show()

plot_percentage_error_distribution(y_test, predicted_loads)


# Convert pandas Series to NumPy arrays (if they are Series)
predicted_loads_array = predicted_loads.values if
 ↪  isinstance(predicted_loads, pd.Series) else
 ↪  predicted_loads.flatten()
y_test_array = y_test.values if isinstance(y_test, pd.Series) else
 ↪  y_test.flatten()

# Calculate MAPE
mape = np.mean(np.abs((y_test_array - predicted_loads_array) /
 ↪  y_test_array)) * 100

print(f"Mean Absolute Percentage Error (MAPE): {mape:.2f}%")

# Calculate percentage differences
percentage_differences = np.abs((y_test_array -
 ↪  predicted_loads_array) / y_test_array) * 100
```

```
# Find the maximum percentage difference
max_percentage_difference = np.max(percentage_differences)

print(f"Maximum Percentage Difference:
↪  {max_percentage_difference:.2f}%")

model.save('newnewnewattempt.keras')



def calculate_permutation_importance(model, X_val_scaled, y_val,
↪  feature_names):
    # Calculate the baseline performance with the unpermuted data
    baseline_mse = mean_squared_error(y_val,
    ↪  model.predict(X_val_scaled))

    importance_scores = {}  # Corrected variable name

    # Iterate over each feature by index
    for i, feature in enumerate(feature_names):
        # Save a copy of the original feature column
        saved_column = X_val_scaled[:, i].copy()

        # Permute the feature column values
        X_val_scaled[:, i] = np.random.permutation(X_val_scaled[:,
        ↪  i])

        # Calculate new performance with the permuted data
        new_mse = mean_squared_error(y_val,
        ↪  model.predict(X_val_scaled))

        # Calculate the importance score as the difference in
        ↪  performance
        importance_score = new_mse - baseline_mse

        # Store the importance score in the dictionary
        importance_scores[feature] = importance_score

        # Restore the original feature column
        X_val_scaled[:, i] = saved_column

    return importance_scores

# Assume you have a list of feature names corresponding to the
↪  columns in your original DataFrame
```

```
feature_names = ['PlateStrength', 'StiffStrength', 'Thickness',
↪  'FlangeWidth', 'Stiffeners', 'PlateWidth', 'StiffThick']  #
↪  Replace with your actual feature names
importance_scores = calculate_permutation_importance(model,
↪  X_val_scaled, y_val, feature_names)

# Sort and display the feature importance
#sorted_importance_scores = sorted(importance_scores.items(),
↪  key=lambda x: x[1], reverse=True)
#for feature, importance in sorted_importance_scores:
#    print(f"{feature}: {importance}")


# Calculate the absolute errors between the predictions and the
↪  actual values
errors = np.abs(predicted_loads.flatten() - y_test.to_numpy())

# Find the indices of the 5 largest errors
largest_errors_indices = np.argsort(percentage_differences)[-40:]

# Extract the rows from X_test corresponding to these indices
largest_errors_rows = X_test.iloc[largest_errors_indices]


# Add the corresponding error values as a new column to the
↪  DataFrame
largest_errors_rows['Error'] =
↪  percentage_differences[largest_errors_indices]

# Display the specific values for the input parameters that
↪  resulted in the largest errors
print("Input parameters for the 40 largest errors:")
print(largest_errors_rows)




errors_0_3 = np.sum((percentage_differences > 0) &
↪  (percentage_differences <= 3))
errors_3_5 = np.sum((percentage_differences > 3) &
↪  (percentage_differences <= 5))
errors_over_5 = np.sum(percentage_differences > 5)

print(f"Number of errors between 0% and 3%: {errors_0_3}")
print(f"Number of errors between 3% and 5%: {errors_3_5}")
print(f"Number of errors over 5%: {errors_over_5}")
```

```
def scale_importance_scores(importance_scores):
    # Extract scores from the dictionary and convert to a numpy
    ↪  array
    scores_array = np.array(list(importance_scores.values()))

    # Compute the minimum and maximum values from the array
    min_score = np.min(scores_array)
    max_score = np.max(scores_array)

    # Avoid division by zero if all scores are the same
    if min_score == max_score:
        return {key: 1 for key in importance_scores}  # All scores
        ↪  set to 1

    # Perform min-max scaling to transform scores to the range 1 to
    ↪  100
    scaled_scores = 1 + 99 * (scores_array - min_score) /
    ↪  (max_score - min_score)

    # Return scaled scores in the same dictionary format
    return dict(zip(importance_scores.keys(), scaled_scores))

# Scale the importance scores
scaled_importance_scores =
↪  scale_importance_scores(importance_scores)

# Print the scaled scores
for feature, score in scaled_importance_scores.items():
    print(f"{feature}: {score:.2f}")



r_squared = r2_score(y_test, predicted_loads)

print("R^2 Score:", r_squared)
```

B2 - Code for Testing Model Outside of Training Range

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow import keras
from NNstart import scaler
from sklearn.metrics import mean_squared_error
import pandas as pd
from tensorflow import keras
from sklearn.metrics import r2_score
# from sklearn.preprocessing import StandardScaler  # Uncomment if
↪  you need to import StandardScaler

# Load and preprocess new data
data_path = 'NewResultsOutside.csv'
new_data = pd.read_csv(data_path)
x_new = new_data[['PlateStrength', 'StiffStrength', 'Thickness',
↪  'FlangeWidth', 'Stiffeners', 'PlateWidth', 'StiffThick']]
x_new_scaled = scaler.transform(x_new)  # Use the same scaler as
↪  you used in training

y = new_data['BucklingLoad']

# Load the trained model
model = keras.models.load_model('newnewnewattempt.keras')

# Make predictions
predictions = model.predict(x_new_scaled)
#print(predictions)


# Create a scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(y, predictions, alpha=0.5)
plt.xlabel('True Load')
plt.ylabel('Predicted Load')
plt.title('True Load vs. Predicted Load Joint Dataset (Outside
↪  Range)')
plt.grid(True)

# Add a perfect fit line (y = x)
x = np.linspace(min(y), max(y), 100)
```

```
plt.plot(x, x, color='red', linestyle='--', label='Perfect Fit')

plt.legend()
# Show the plot
plt.show()

# Convert pandas Series to NumPy arrays (if they are Series)
predicted_loads_array = predictions.values if
 ↪  isinstance(predictions, pd.Series) else predictions.flatten()
true_load_array = y.values if isinstance(y, pd.Series) else
 ↪  y.flatten()

# Calculate MAPE
mape = np.mean(np.abs((true_load_array - predicted_loads_array) /
 ↪  true_load_array)) * 100

print(f"Mean Absolute Percentage Error (MAPE): {mape:.2f}%")

# Calculate percentage differences
percentage_differences = np.abs((true_load_array -
 ↪  predicted_loads_array) / true_load_array) * 100

# Find the maximum percentage difference
max_percentage_difference = np.max(percentage_differences)

print(f"Maximum Percentage Difference:
 ↪  {max_percentage_difference:.2f}%")

def plot_percentage_error_distribution(y_true, y_pred):
    # Calculate percentage errors
    percentage_errors = ((y_pred.flatten() - y_true) / y_true) *
     ↪  100
    plt.figure(figsize=(10, 6))
    plt.hist(percentage_errors, bins=50, edgecolor='black',
     ↪  color='skyblue')
    plt.xlabel('Percentage Error (%)')
    plt.ylabel('Frequency')
    plt.title('Error Distribution Joint Dataset (Outside Range)')
    plt.grid(True)
    plt.tight_layout()
    plt.show()

plot_percentage_error_distribution(true_load_array,
 ↪  predicted_loads_array)
```

```
def calculate_permutation_importance(model, X_val_scaled, y_val,
↪  feature_names):
    # Calculate the baseline performance with the unpermuted data
    baseline_mse = mean_squared_error(y_val,
     ↪  model.predict(X_val_scaled))

    importance_scores = {}  # Corrected variable name

    # Iterate over each feature by index
    for i, feature in enumerate(feature_names):
        # Save a copy of the original feature column
        saved_column = X_val_scaled[:, i].copy()

        # Permute the feature column values
        X_val_scaled[:, i] = np.random.permutation(X_val_scaled[:,
         ↪  i])

        # Calculate new performance with the permuted data
        new_mse = mean_squared_error(y_val,
         ↪  model.predict(X_val_scaled))

        # Calculate the importance score as the difference in
         ↪  performance
        importance_score = new_mse - baseline_mse

        # Store the importance score in the dictionary
        importance_scores[feature] = importance_score

        # Restore the original feature column
        X_val_scaled[:, i] = saved_column

    return importance_scores

# Assume you have a list of feature names corresponding to the
↪  columns in your original DataFrame
feature_names = ['PlateStrength', 'StiffStrength', 'Thickness',
↪  'FlangeWidth', 'Stiffeners', 'PlateWidth', 'StiffThick']  #
↪  Replace with your actual feature names
importance_scores = calculate_permutation_importance(model,
↪  x_new_scaled, y, feature_names)

# Sort and display the feature importance
sorted_importance_scores = sorted(importance_scores.items(),
↪  key=lambda x: x[1], reverse=True)
for feature, importance in sorted_importance_scores:
    print(f"{feature}: {importance}")
```

```
# Calculate the absolute errors between the predictions and the
↪  actual values
errors = np.abs(predictions.flatten() - y.to_numpy())

# Find the indices of the 5 largest errors
largest_errors_indices = np.argsort(percentage_differences)[-200:]

# Extract the rows from X_test corresponding to these indices
largest_errors_rows = x_new.iloc[largest_errors_indices]

largest_errors_rows['Error'] =
↪  percentage_differences[largest_errors_indices]

# Display the specific values for the input parameters that
↪  resulted in the largest errors
print("Input parameters for the 40 largest errors:")
print(largest_errors_rows)

largest_errors_rows.to_csv('filename.csv')
#def get_largest_error_per_stiffener(group):
#    return group.loc[group['Error'].idxmax()]

# Group by the number of stiffeners and apply the function
#largest_error_per_stiffener =
↪  largest_errors_rows.groupby('Stiffeners').apply(get_largest_error_per_stiff
#print(largest_error_per_stiffener[['Stiffeners', 'Error']])


errors_0_5 = np.sum((percentage_differences > 0) &
↪  (percentage_differences <= 5))
errors_5_10 = np.sum((percentage_differences > 5) &
↪  (percentage_differences <= 10))
errors_over_10 = np.sum(percentage_differences > 10)

print(f"Number of errors between 0% and 5%: {errors_0_5}")
print(f"Number of errors between 5% and 10%: {errors_5_10}")
print(f"Number of errors over 10%: {errors_over_10}")


r_squared = r2_score(y, predictions)

print("R^2 Score:", r_squared)
```

C - SIDENOTE STATISTICS

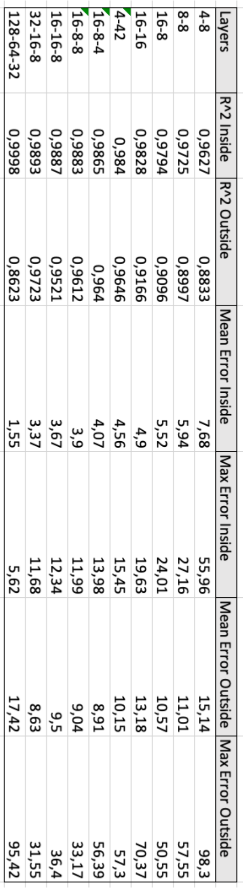
C1 - Error results outside training range

```
PlateStrength,StiffStrength,Thickness,FlangeWidth,Stiffeners,PlateWidth,StiffThick,Error
450.0,355.0,2.5,37.0,3.0,220.0,3.5,12.868063 450.0,190.0,3.5,30.0,3.0,300.0,3.5,12.894570
245.0,190.0,2.5,30.0,4.0,300.0,3.5,12.927531 245.0,355.0,3.5,32.0,2.0,220.0,4.25,13.02270
245.0,190.0,3.5,37.0,2.0,220.0,3.5,13.100294 450.0,355.0,3.5,37.0,3.0,340.0,3.5,13.111837
450.0,355.0,2.5,37.0,2.0,220.0,4.25,13.11353 450.0,355.0,2.5,37.0,2.0,300.0,4.25,13.13664
245.0,355.0,2.5,32.0,3.0,220.0,4.25,13.13666 190.0,355.0,2.5,37.0,3.0,340.0,3.5,13.208420
450.0,190.0,2.5,30.0,3.0,300.0,3.5,13.227354 190.0,355.0,3.5,37.0,2.0,340.0,4.25,13.26042
245.0,355.0,3.5,32.0,3.0,220.0,4.25,13.29192 450.0,190.0,3.5,30.0,3.0,340.0,4.25,13.32084
245.0,190.0,2.5,30.0,4.0,340.0,4.25,13.34422 245.0,190.0,2.5,30.0,4.0,220.0,3.5,13.346059
245.0,190.0,2.5,37.0,2.0,220.0,3.5,13.375014 450.0,355.0,2.5,37.0,3.0,300.0,3.5,13.429726
245.0,190.0,3.5,30.0,3.0,340.0,3.5,13.449367 245.0,190.0,3.5,30.0,4.0,340.0,3.5,13.570898
450.0,355.0,3.5,37.0,3.0,220.0,3.5,13.610148 190.0,355.0,2.5,32.0,3.0,340.0,3.5,13.627604
245.0,355.0,2.5,37.0,2.0,340.0,4.25,13.65823 450.0,190.0,2.5,30.0,4.0,300.0,4.25,13.66278
190.0,190.0,3.5,30.0,4.0,300.0,3.5,13.678762 190.0,355.0,3.5,32.0,3.0,340.0,3.5,13.752939
245.0,190.0,3.5,30.0,2.0,220.0,4.25,13.76864 245.0,190.0,2.5,30.0,3.0,220.0,3.5,13.776400
450.0,190.0,3.5,37.0,2.0,220.0,3.5,13.779919 190.0,190.0,3.5,30.0,4.0,340.0,4.25,13.82621
190.0,190.0,3.5,30.0,4.0,220.0,3.5,13.828193 190.0,355.0,3.5,37.0,3.0,340.0,3.5,13.901605
450.0,190.0,3.5,37.0,2.0,300.0,3.5,13.953764 450.0,355.0,3.5,37.0,2.0,220.0,4.25,13.96462
450.0,190.0,2.5,37.0,2.0,340.0,3.5,13.972668 245.0,355.0,2.5,37.0,3.0,340.0,3.5,13.995229
190.0,355.0,2.5,32.0,3.0,220.0,3.5,14.119405 190.0,190.0,3.5,30.0,4.0,340.0,3.5,14.176127
450.0,355.0,3.5,37.0,2.0,300.0,4.25,14.22813 450.0,355.0,3.5,37.0,2.0,340.0,3.5,14.254505
450.0,190.0,2.5,30.0,2.0,300.0,4.25,14.27743 450.0,190.0,2.5,37.0,2.0,220.0,3.5,14.279019
450.0,190.0,2.5,30.0,3.0,340.0,4.25,14.29998 450.0,355.0,3.5,37.0,3.0,300.0,3.5,14.315070
190.0,190.0,3.5,30.0,2.0,300.0,4.25,14.32443 190.0,190.0,2.5,30.0,3.0,300.0,3.5,14.338598
190.0,355.0,2.5,37.0,2.0,220.0,4.25,14.40525 190.0,355.0,2.5,37.0,3.0,220.0,3.5,14.438549
245.0,355.0,2.5,32.0,3.0,340.0,3.5,14.487386 190.0,190.0,2.5,30.0,4.0,340.0,4.25,14.49424
245.0,355.0,3.5,32.0,3.0,340.0,3.5,14.525939 450.0,190.0,3.5,30.0,4.0,300.0,3.5,14.570992
190.0,190.0,2.5,30.0,4.0,300.0,3.5,14.586462 450.0,190.0,3.5,30.0,4.0,220.0,4.25,14.64897
450.0,355.0,3.5,37.0,2.0,220.0,3.5,14.652067 190.0,355.0,3.5,32.0,3.0,220.0,3.5,14.692511
245.0,190.0,2.5,30.0,2.0,220.0,4.25,14.70801 245.0,355.0,3.5,37.0,2.0,340.0,4.25,14.73442
450.0,190.0,2.5,30.0,4.0,220.0,4.25,14.74317 190.0,190.0,3.5,37.0,2.0,340.0,3.5,14.904668
245.0,355.0,3.5,37.0,3.0,340.0,3.5,14.939742 190.0,190.0,3.5,30.0,3.0,220.0,3.5,15.010683
```

```
450.0,190.0,2.5,30.0,4.0,300.0,3.5,15.029635 450.0,190.0,3.5,30.0,2.0,300.0,4.25,15.04723
245.0,190.0,2.5,30.0,4.0,340.0,3.5,15.072021 245.0,355.0,2.5,37.0,2.0,220.0,4.25,15.08971
450.0,355.0,2.5,37.0,2.0,340.0,3.5,15.160128 245.0,355.0,2.5,37.0,3.0,220.0,3.5,15.186011
190.0,190.0,2.5,30.0,4.0,220.0,3.5,15.204658 450.0,355.0,2.5,37.0,2.0,220.0,3.5,15.214054
190.0,190.0,3.5,37.0,2.0,220.0,3.5,15.315788 190.0,190.0,3.5,30.0,3.0,340.0,3.5,15.336356
245.0,190.0,2.5,32.0,2.0,340.0,3.5,15.370502 245.0,190.0,2.5,30.0,3.0,340.0,3.5,15.411852
450.0,190.0,3.5,30.0,3.0,220.0,3.5,15.413038 190.0,190.0,2.5,30.0,2.0,300.0,4.25,15.41971
190.0,355.0,2.5,37.0,3.0,300.0,3.5,15.433411 450.0,190.0,3.5,37.0,2.0,340.0,3.5,15.480868
190.0,355.0,2.5,32.0,3.0,300.0,3.5,15.546658 190.0,355.0,3.5,32.0,3.0,300.0,3.5,15.580227
190.0,355.0,2.5,37.0,2.0,300.0,4.25,15.59189 450.0,190.0,3.5,30.0,4.0,340.0,4.25,15.63750
190.0,190.0,2.5,30.0,4.0,340.0,3.5,15.663474 245.0,190.0,2.5,32.0,2.0,220.0,3.5,15.687944
190.0,190.0,2.5,37.0,2.0,220.0,3.5,15.694088 450.0,190.0,3.5,30.0,4.0,220.0,3.5,15.705370
190.0,355.0,3.5,37.0,3.0,220.0,3.5,15.738169 245.0,190.0,3.5,30.0,2.0,340.0,4.25,15.79880
245.0,355.0,2.5,37.0,3.0,300.0,3.5,15.827424 190.0,190.0,2.5,37.0,2.0,340.0,3.5,15.832457
245.0,190.0,3.5,30.0,5.0,220.0,4.25,15.83461 245.0,355.0,2.5,32.0,3.0,220.0,3.5,15.852228
245.0,190.0,3.5,30.0,5.0,300.0,4.25,15.90099 245.0,355.0,2.5,37.0,2.0,300.0,4.25,15.95229
450.0,190.0,2.5,30.0,3.0,220.0,3.5,16.101884 190.0,355.0,3.5,37.0,3.0,300.0,3.5,16.141013
245.0,190.0,2.5,30.0,5.0,220.0,4.25,16.14439 190.0,355.0,3.5,37.0,2.0,220.0,4.25,16.26766
450.0,190.0,2.5,30.0,4.0,340.0,4.25,16.29814 245.0,355.0,3.5,32.0,3.0,300.0,3.5,16.349899
190.0,355.0,3.5,37.0,2.0,300.0,4.25,16.37775 245.0,355.0,2.5,32.0,3.0,300.0,3.5,16.409944
450.0,190.0,2.5,30.0,4.0,220.0,3.5,16.416164 245.0,355.0,3.5,37.0,3.0,300.0,3.5,16.471885
450.0,190.0,3.5,30.0,5.0,300.0,4.25,16.54206 245.0,190.0,3.5,32.0,2.0,220.0,3.5,16.544922
190.0,190.0,2.5,30.0,3.0,220.0,3.5,16.568124 245.0,355.0,3.5,37.0,3.0,220.0,3.5,16.588408
245.0,190.0,2.5,30.0,5.0,300.0,4.25,16.60435 245.0,355.0,3.5,32.0,3.0,220.0,3.5,16.655700
245.0,355.0,3.5,37.0,2.0,300.0,4.25,16.74463 450.0,190.0,2.5,32.0,2.0,220.0,3.5,16.807110
245.0,355.0,3.5,37.0,2.0,220.0,4.25,16.96665 450.0,355.0,3.5,37.0,2.0,300.0,3.5,17.231917
190.0,190.0,2.5,30.0,3.0,340.0,3.5,17.302568 245.0,190.0,3.5,32.0,2.0,340.0,3.5,17.313668
450.0,190.0,3.5,30.0,4.0,340.0,3.5,17.316300 450.0,190.0,3.5,30.0,5.0,340.0,4.25,17.42370
190.0,190.0,3.5,30.0,5.0,300.0,4.25,17.46163 245.0,190.0,3.5,30.0,5.0,300.0,3.5,17.496346
450.0,190.0,2.5,30.0,5.0,300.0,4.25,17.60723 190.0,190.0,3.5,32.0,2.0,300.0,3.5,17.681918
190.0,190.0,3.5,30.0,5.0,220.0,4.25,17.70665 450.0,190.0,3.5,30.0,5.0,220.0,4.25,17.71259
450.0,190.0,3.5,30.0,2.0,220.0,4.25,17.72340 450.0,190.0,2.5,30.0,2.0,220.0,4.25,17.81140
190.0,190.0,3.5,30.0,2.0,220.0,4.25,17.81427 190.0,190.0,2.5,32.0,2.0,300.0,3.5,17.843446
245.0,190.0,2.5,30.0,2.0,340.0,4.25,17.85185 245.0,190.0,3.5,30.0,5.0,220.0,3.5,17.858464
450.0,190.0,2.5,32.0,2.0,340.0,3.5,17.923767 245.0,190.0,2.5,30.0,5.0,220.0,3.5,17.987086
190.0,190.0,2.5,30.0,5.0,220.0,4.25,17.99970 450.0,355.0,2.5,37.0,2.0,300.0,3.5,18.052019
190.0,190.0,2.5,30.0,5.0,300.0,4.25,18.15915 450.0,190.0,2.5,30.0,5.0,220.0,4.25,18.16877
450.0,190.0,3.5,30.0,3.0,340.0,3.5,18.181907 245.0,190.0,2.5,30.0,5.0,300.0,3.5,18.185331
245.0,190.0,3.5,30.0,5.0,340.0,4.25,18.28433 190.0,190.0,3.5,30.0,5.0,300.0,3.5,18.306486
450.0,190.0,2.5,30.0,5.0,340.0,4.25,18.34327 190.0,190.0,3.5,30.0,2.0,340.0,4.25,18.44473
450.0,190.0,2.5,30.0,4.0,340.0,3.5,18.519084 450.0,190.0,3.5,32.0,2.0,220.0,3.5,18.536490
190.0,190.0,2.5,30.0,2.0,220.0,4.25,18.62992 190.0,190.0,3.5,30.0,5.0,220.0,3.5,18.704359
190.0,190.0,2.5,30.0,5.0,300.0,3.5,19.036296 190.0,190.0,2.5,30.0,5.0,220.0,3.5,19.190277
245.0,190.0,2.5,30.0,5.0,340.0,4.25,19.19692 450.0,190.0,3.5,32.0,2.0,340.0,3.5,19.226384
190.0,355.0,3.5,37.0,2.0,340.0,3.5,19.358178 190.0,190.0,3.5,30.0,5.0,340.0,4.25,19.45806
450.0,190.0,2.5,30.0,3.0,340.0,3.5,19.464093 190.0,190.0,2.5,32.0,2.0,340.0,3.5,19.554937
190.0,190.0,3.5,30.0,5.0,340.0,3.5,19.578527 245.0,190.0,3.5,30.0,5.0,340.0,3.5,19.627770
190.0,355.0,2.5,37.0,2.0,340.0,3.5,19.730174 190.0,355.0,2.5,37.0,2.0,220.0,3.5,19.870329
```

```
450.0,190.0,3.5,30.0,5.0,300.0,3.5,19.873524 190.0,355.0,3.5,37.0,2.0,220.0,3.5,19.972042
190.0,190.0,3.5,32.0,2.0,340.0,3.5,20.199873 190.0,190.0,2.5,30.0,2.0,340.0,4.25,20.26253
190.0,190.0,2.5,30.0,5.0,340.0,4.25,20.36719 190.0,190.0,2.5,32.0,2.0,220.0,3.5,20.449358
245.0,190.0,2.5,30.0,5.0,340.0,3.5,20.573025 190.0,190.0,2.5,30.0,5.0,340.0,3.5,20.575218
245.0,190.0,3.5,30.0,2.0,300.0,3.5,20.621132 245.0,355.0,3.5,37.0,2.0,340.0,3.5,20.721992
450.0,190.0,2.5,30.0,5.0,300.0,3.5,20.783300 245.0,355.0,2.5,37.0,2.0,340.0,3.5,21.043668
190.0,190.0,3.5,32.0,2.0,220.0,3.5,21.349096 245.0,355.0,2.5,37.0,2.0,220.0,3.5,21.802937
450.0,190.0,3.5,30.0,5.0,340.0,3.5,22.090855 245.0,355.0,3.5,37.0,2.0,220.0,3.5,22.273726
450.0,190.0,3.5,30.0,5.0,220.0,3.5,22.378139 450.0,190.0,3.5,30.0,2.0,300.0,3.5,22.632625
450.0,190.0,2.5,30.0,5.0,220.0,3.5,22.680443 245.0,190.0,2.5,30.0,2.0,300.0,3.5,22.727434
450.0,190.0,2.5,30.0,5.0,340.0,3.5,22.808638 450.0,190.0,3.5,30.0,2.0,340.0,4.25,23.24471
190.0,355.0,2.5,37.0,2.0,300.0,3.5,23.665662 450.0,190.0,2.5,30.0,2.0,340.0,4.25,23.78866
190.0,355.0,3.5,37.0,2.0,300.0,3.5,24.129039 450.0,190.0,2.5,30.0,2.0,300.0,3.5,24.151233
245.0,355.0,3.5,37.0,2.0,300.0,3.5,24.535685 245.0,355.0,2.5,37.0,2.0,300.0,3.5,24.761456
190.0,190.0,3.5,30.0,2.0,300.0,3.5,25.234834 245.0,190.0,3.5,30.0,2.0,340.0,3.5,25.258165
245.0,190.0,3.5,30.0,2.0,220.0,3.5,25.754294 450.0,190.0,3.5,30.0,2.0,220.0,3.5,26.317654
245.0,190.0,2.5,30.0,2.0,340.0,3.5,27.111447 450.0,190.0,2.5,30.0,2.0,220.0,3.5,27.868489
190.0,190.0,3.5,30.0,2.0,340.0,3.5,28.418724 245.0,190.0,2.5,30.0,2.0,220.0,3.5,28.462968
190.0,190.0,2.5,30.0,2.0,300.0,3.5,28.928398 190.0,190.0,2.5,30.0,2.0,340.0,3.5,30.838319
190.0,190.0,3.5,30.0,2.0,220.0,3.5,31.036074 450.0,190.0,3.5,30.0,2.0,340.0,3.5,31.176898
450.0,190.0,2.5,30.0,2.0,340.0,3.5,32.486952 190.0,190.0,2.5,30.0,2.0,220.0,3.5,33.708223
```

C2 - Testing Network Configurations



Layers	R^2 Inside	R^2 Outside	Mean Error Inside	Max Error Inside	Mean Error Outside	Max Error Outside
4-8	0,9627	0,8833	7,68	55,96	15,14	98,3
8-8	0,9725	0,8997	5,94	27,16	11,01	57,55
16-8	0,9794	0,9096	5,52	24,01	10,57	50,55
16-16	0,9828	0,9166	4,9	19,63	13,18	70,37
4-42	0,984	0,9646	4,56	15,45	10,15	57,3
16-8-4	0,9865	0,964	4,07	13,98	8,91	56,39
16-8-8	0,9883	0,9612	3,9	11,99	9,04	33,17
16-16-8	0,9887	0,9521	3,67	12,34	9,5	36,4
32-16-8	0,9893	0,9723	3,37	11,68	8,63	31,55
128-64-32	0,9998	0,8623	1,55	5,62	17,42	95,42

Figure B.1: Network Configurations