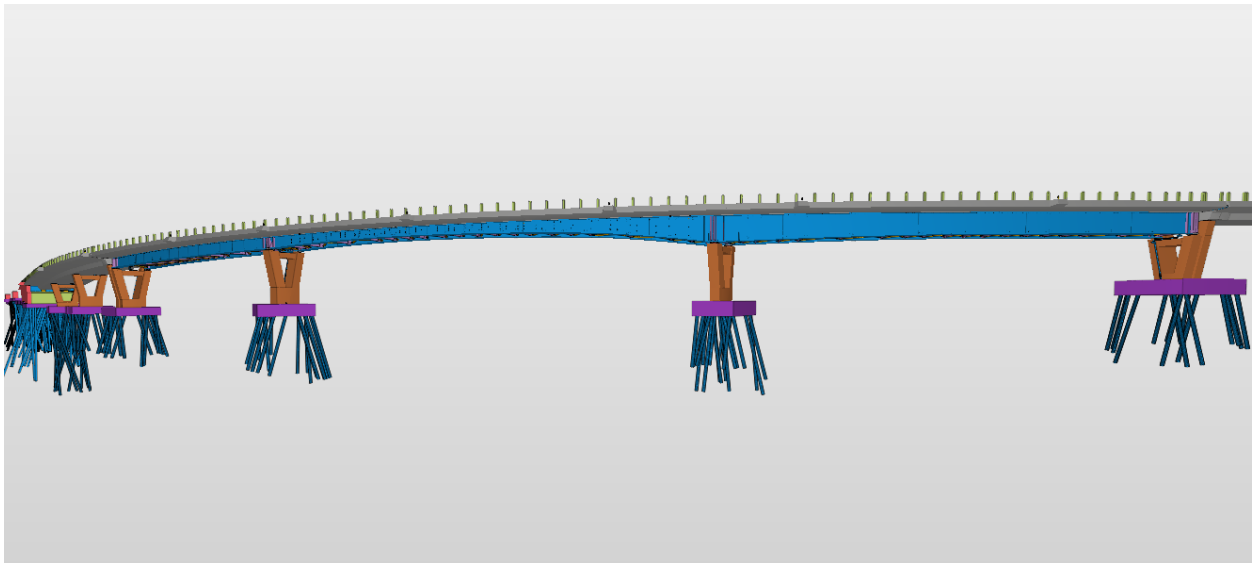
CHALMERS
UNIVERSITY OF TECHNOLOGY



# Rational Method for FE-Model Updating: Comparison Between Conventional and AI Methods

## A case-study of Tvärbanan, stage Solvalla

Master's thesis in Master Program Structural Engineering and Building Technology

KIM ANDERSSON BULL
DAVID BYRÉN CLAESSON

Department of Architecture and Civil Engineering

CHALMERS UNIVERSITY OF TECHNOLOGY
Master's thesis ACEX30
Gothenburg, Sweden 2024

Master's thesis ACEX30

# Rational Method for FE-Model Updating: Comparison Between Conventional and AI Methods

A case-study of Tvärbanan, stage Solvalla

KIM ANDERSSON BULL
DAVID BYRÉN CLAESSON


CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Architecture and Civil Engineering
*Division of Structural Engineering*
Concrete Structure Group
Chalmers University of Technology
Gothenburg, Sweden 2024

Rational Method for FE-Model Updating: Comparison Between Conventional and AI Methods
A Case-study of Tvärbanan, stage Solvalla

KIM ANDERSSON BULL
DAVID BYRÉN CLAESSON



Supervisors: Christoffer Svedholm, ELU
Erik Flink, ELU
Anna Teike, Trafikverket

Examiner: Carlos Gil Berrocal, Department of Architecture and Civil Engineering

Department of Architecture and Civil Engineering
Division of Structural Engineering
Concrete Structure Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Model of the relevant bridge segments used in the Case-study, for visualisation of boundary limitation.

Department of Architecture and Civil Engineering
Gothenburg, Sweden 2024

Rational Method for FE-Model Updating: Comparison Between Conventional and AI Methods
A case-study of Tvärbanan, stage Solvalla
KIM ANDERSSON BULL
DAVID BYRÉN CLAESSON
Department of Architecture and Civil Engineering
Chalmers University of Technology

# Abstract

The aim of this thesis is to evaluate and develop alternative and rational methods for model updating. Ordinary and already established optimisation methods (Nelder-Mead simplex, pattern search, genetic algorithm and particle swarm) were compared against the contemporary and emerging principles of neural networks and AI. Their performance was compared using three different beam test cases, with increased complexity, with intention of solving unknown variables based on the corresponding "true" deflection. Through the test cases, each method was tested and evaluated based on accuracy, computational demand and scalability against complexity.

The results indicated that the traditional and already established methods performed best, yielding high accuracy for low computational demand. Therefore, the developed optimisation methods based on neural networks and use of the ChatGPT AI could not compete.

However, findings indicate that neural networks methods and AI have future potential in other areas then initially assumed. The conclusion of this thesis is that neural network methods could be useful, particularly when implemented as a tool rather than a replacement for model updating. Neural networks with the capacity to predict deflections based on input variables could be simultaneously trained based on information gained trough a traditional model updating routine. Once trained, re-evaluation of model behaviour could be produced without a FEM-model, requiring just a few seconds, providing improved abilities in engineering judgements.

Keywords: Model updating, Optimization, Nelder-Mead simplex, Pattern search, Genetic alghoritm, Particle swarm, Neural Network, PINN, ChatGPT, Structural Engineering.

# Acknowledgements

We would like to express our deepest gratitude towards extraordinary work and effort our supervisors from ELU, Christoffer Svedholm and Erik Flink, have contributed with. With guidance in programming, modelling and optimising, this thesis would not be possible without their help.

Additionally, our examiner Carlos Gil Berrocal from Chalmers deserves a warm thanks for being an essential role through the thesis. In contributing through guidance and essential feedback, your help and dedication have been greatly appreciated.

Futher, we would like to thank Anna Teike from Trafikverket and Magnus Ekh from Chalmers for their contributions. Anna Teike's dedication in providing case-study measurements was of huge importance. Magnus Ekh provided extra sessions and help within neural networks, his guidance and support to us have been very helpful throughout this thesis.

Lastly, we would like to thank ELU and all their employees which provided with a peacefully and nice environment in which this thesis were written.

Kim Andersson Bull & David Byrén Claesson, Gothenburg, June 2024

# Contents

# List of Figures

# List of Tables

# List of Acronyms

| | |
|---|---|
| **AGI** | Artificial General Intelligence |
| **AI** | Artificial Intelligence |
| **API** | Application Programming Interface |
| **BFGS** | Broyden Fletcher Goldfarb Shanno |
| **COT** | Chain Of Thought |
| **DOF** | Degree Of Freedom |
| **DQ** | Data Quality |
| **ELM** | Extreme Learning Machines |
| **ELU** | Exponential Linear Unit |
| **FBPINN** | Finite Based Physics Informed Neural Network |
| **FDM** | Finite Difference Method |
| **FEA** | Finite Element Analysis |
| **FEM** | Finite Element Method |
| **FEM-NN** | Finite Element Method Neural Network hybrid |
| **FF** | Forward Feed |
| **FFR** | Forward Feed Reversed |
| **GA** | Genetic Algorithm |
| **GD** | Gradient Descent |
| **GELU** | Gaussian Error Linear Unit |
| **HGA** | Hybrid Genetic Algorithm |
| **L-BFGS** | Limited memory Broyden Fletcher Goldfarb Shanno |
| **LASSO** | Least Absolute Shrinkage and Selection Operator |
| **LLM** | Large Language Models |
| **MAPE** | Mean Absolute Percentage Error |
| **MEMS** | Micro-Electro-Mechanical Systems |
| **NLP** | Natural Language Processing |
| **NN** | Neural Network |
| **ODE** | Ordinary Differential Equation |
| **PDE** | Partial Differential Equation |
| **PINN** | Physics Informed Neural Network |
| **PL** | Piecewise Linear |
| **PReLU** | Parametric Rectified Linear Unit |
| **PS** | Pattern Search |
| **PSO** | Particle Swarm Optimization |
| **ReLU** | Rectified Linear Unit |
| **RT** | Run Time |

| | |
|---|---|
| **Std Dev** | Standard Deviation |
| **SELU** | Scaled Exponential Linear Unit |
| **SGD** | Stochastic Gradient Decent |
| **SHM** | Structural Health Monitoring |
| **SiL** | Sigmoid Shrinkage |
| **SiLU** | Sigmoid Linear Unit |
| **SLP** | Single Layer Perceptron |
| **SLS** | Service Limit State |
| **SMHT** | Sobolev Modified Hyperbolic Tangent |
| **SVM** | Support Vector Machines |
| **TD** | Training Data time |
| **TanH** | Hyperbolic Tangent |
| **TM** | Training Model time |
| **ULS** | Ultimate Limit State |

# 1

# Introduction

Different types of engineering measurement are broadly used to observe and monitor the behaviour of structures, both for new constructions and for existing constructions such as in structural health monitoring (SHM). It is important and increasingly more common to monitor buildings or structures during the construction phase to ensure that what is constructed is built according to the plans. After the construction phase, buildings or structures can be continuously monitored to ensure and verify that the performance is aligned with expectations. By monitoring during the construction phase and throughout the service life, potential deviations in behaviour or measured values can be detected and addressed (Enckell, 2006).

The commonly used basis regarding observational methods was introduced as early as 1969 and is still valid to proactively limit risks in construction (Peck, 1969). Primarily, the most unfavourable conditions and their effects on structural behaviour should be estimated, followed by the decision on the limit value and possible actions if exceeded. The limit value often coexists with warning values close to the limit value and acts as an indication of unforeseen behaviour and required review of calculations. The limit value is often set to a measurement tolerance where the serviceability limit state, is most likely exceeded (Devriendt, 2012).

If observations indicate deviant behaviour, revaluation and verifications are often necessary. This is commonly performed through model updating, of which models are updated to match the observed behaviour. The outcome of a successful model updating is reduced uncertainties compared to dimensioning assumptions, allowing the model to represent reality in a larger degree. However, to reduce uncertainties to a satisfactory degree, careful considerations must be taken. Not only does it put requirements on a sufficient model to begin with, the measured data has requirements on accuracy and correctness.

Consequently, the necessary reevaluation to verify structures is often both a costly and time-consuming process which is required promptly. Most reevaluations of complicated structures are based on finite element method (FEM) models. Therefore, recalculations need to be performed using an updated model that correlates with the measurement data. With a stream of new input data, continuous iterative recalculations may be necessary and can be done in many different ways.

As of today, no general or standard method for continuous model updating is stated within regulations and guidelines. One of the reasons might be that observation methods often produce vast amounts of data over time, which are difficult to evaluate effectively. Using a rational method for such evaluations continuously throughout the lifespan of structures would be of great benefit.

For this thesis, a segment of the ongoing project of *Tvärbanan* in Stockholm was used as a case-study. To accommodate the growing population, accessibility and transportation are essential infrastructure elements in need of development. The current light rail transit, Tvärbanan, is a 20 km long infrastructure that connects major parts of Stockholm to the airport, metro, train and more (Wikipedia, 2024c). Currently, Tvärbanan is being extended to respond to the increased user demands put on infrastructural services. The project involves 8 km of additional railway that connects the Bromma Airport to Helenelund. Located in the proposed path, in Solvalla, Tvärbanan passes over railroads and a highway via a bridge. Therefore, the future behaviour of the bridge must be predicted with high precision, as the existing infrastructure imposes fixed deflection limitations. Site tests of various static loadings have been conducted in an unfinished state to facilitate a precise model updating. The desired outcome was to achieve a sufficiently updated FEM model which could allow for accurate predict the finished deflection.

## 1.1 Aim and objectives

The aim of this thesis is to evaluate the possibility and benefits of using rational methods for model updating based on measured data. Benchmarking and comparisons between conventional and more recent principles of neural networks are of interest for evaluating future applications. For the case-study, the goal is to apply and asses the robustness and usability of proposed optimisation routines to verify their ability to adequately correct the parameters of the FEM model. Through this, global deformations and behaviour are to be evaluated for future load cases.

Set objectives to achieve are:

- Establish appropriate test cases and evaluate different automated updating methods.
- Investigate and compare the capabilities of AI updating in relation to traditional methods.
- Establish an appropriate model for the bridge segment and test the developed updating routine.

## 1.2 Method

The project began with a literature review which laid the foundation for continued work. Areas of interest were methods of model updating, optimisation methods and algorithms, machine learning and deep learning using neural networks, as well as methods to manage observational data. In parallel, FEM models for test cases were established and laid the basis for benchmarking and the development of model updating routines. Subsequently, through the case-study, the developed update routines were applied and evaluated in terms of efficiency and applicability. After the model was calibrated, the possibility of predicting or estimating future displacements was investigated. Through this, the possibility of conducting an automated method of model updating using various techniques was evaluated, for example, with the assistance of machine learning and neural networks, to allow for more rational evaluation over time.

## 1.3 Limitations

For the analysis performed, the relative performance between different principles was of interest. Thus, reservations must be made for the likelihood that optimal settings or configurations are not presented or used. Additionally, recorded durations of different optimisers are highly dependent on hardware and code implementation, thus these do not serve a purpose beyond comparisons within this thesis.

The study does not explicitly cover any model updating regarding nonlinearity. In the case-study, the global behaviour was simplified with the assumption that the bridge behaves linearly. Additionally, since the update was performed in response to a test loading, instantaneous deflections are considered. Any potential creep and shrinkage was excluded. This approach significantly reduced both the size of the simulations and the computational time required but was still considered realistic in the context of global and instantaneous behaviour. Thus, any nonlinear behaviour captured relies on a routine for updating through the linear model.

In addition, regarding the case-study, only a limited bridge segment was considered, see Figure in the Appendix A.1. Additional segments, foundations or pilings were not studied.

## 1.4 Social, ethical and ecological aspects

The rationalisation of the model update, along with the management and monitoring of the constructions, could produce significant and beneficial effects. Initially, such a method could transform the daily operations of civil engineers by facilitating more efficient monitoring practices. This would decrease the need for manual labour and enable a more effective method for evaluating structural behaviours, leading to gains in daily efficiency and consequently a reduction in time and financial expenditures. Subsequently, there is social and ethical gain in the form of increased security and

trust. With the ability to more rationally monitor existing structures over time, society would likely enhance the trust and sense of safety of existing infrastructure. Furthermore, social benefits could include the creation of new employment opportunities, since companies could establish dedicated roles for the oversight of monitoring measurements and equipment.

Finally, the rationalisation of such processes could also result in ecological benefits. Continuous monitoring allows for early detection and intervention before failures occur, potentially decreasing the likelihood of damage and the need for expensive reconstructions. This approach would minimise the demand for additional materials. In addition, the estimated service life of constructions could be reassessed as an alternative to traditional evaluation methods, especially if monitoring persists post-construction. The data derived from updated monitoring could refine calculation models to act as a digital twin, signalling unexpected behaviours and possibly forecasting the capacities of the structure. Therefore, a potential outcome is a more efficient and optimised use of existing infrastructures, aligned with the goals of sustainable development.

## 1.5 Outline of thesis

**Chapter** 2 and **Chapter** 3 presents the essential and relevant information gathered through literature review.

**Chapter** 2 addresses the general concept of monitoring, structural health management and data management.

**Chapter** 3 examines optimisation methods with conventional and more experimental methods with neural networks, followed by an overview of the Finite Element Method (FEM) principals and methods of model updating.

**Chapter** 4 covers how the findings of previous chapters were synthesised and applied to develop an optimal model updating routine. This was mainly done through constructed test cases. Furthermore, it details the adaptation of the case-study into an FEM model, including both modeling simplifications and methodologies of updating.

**Chapter** 5 addresses the outcomes and results of the analysis performed. The performance of the resulting updating routine is presented as well as briefly discussed. This for both test cases and applied case-study.

**Chapter** 6 concludes the thesis, summarising the research findings and implications.

# 2
# Monitoring systems

If rationalisation of automated model updating is to be successful, the monitoring must be performed and interpreted appropriately. To predict structural behaviour accurately, the true behaviour must be captured from measurements and observations. Observation data can exhibit deviations or faulty measurements that can be interpreted as noise. If data are too noisy, inaccurate, or faulty, the updated model will not be able to capture the correct behaviour, thus not predicting the optimal results. Measurement performance must be carefully considered and controlled to capture the relevant data. The selection of which data to capture is not always straightforward. Differences in equipment, their abilities and limitations are thus highly relevant to ensure that reliable data is captured.

Moreover, in addition to appropriate methods of measurement, a vast amount of data needs to be both stored and easy to grasp. If the data are clustered, chaotic or of low quality, they will risk losing usability and can be counterproductive. The selection of a robust method to ensure adequate management and quality of the data is therefore of high importance.

This section will thereby present some of the more common measuring techniques and practices within structural engineering.

## 2.1 Monitoring of mechanical parameters

Essential to structural health monitoring is continuous, robust, and long-term measurement of structural behaviour (Reschetiuk et al., 2012). By ensuring sufficient points of measurement, choice of relevant parameter to assess and appropriate locations, a satisfactory quality of data can be achieved. To strengthen data quality, a common practice is the cross-evaluation of different measured parameters to obtain a broad overview. The choice of how and what to be measured often depends on the type of structure, environment and expected behaviour. The importance of carefully evaluating and choosing appropriate methods in the early stages may be crucial for future health monitoring.

### 2.1.1 Displacement

An often central monitoring parameter is relative displacement, namely changes in the X, Y, and Z directions. The different methods are often divided into point-based or surface-based methods. These methods exhibit significant differences in the degree of manual labour, cost and application.

Point-based methods have, as the name indicates, specifically defined points/nodes that provide reference markers. These markers are then related to a fixed survey marker to evaluate relative motion (Reschetiuk et al., 2012). Point-based methods offer high accuracy, up to 0.2 mm/km, and are applicable in tight spaces. However, this comes at the cost of capturing data only for discrete nodes. The method of capturing the data may also differ. Typically, these measurements may be obtained by manual procedures through levelling. Manual control and measuring for each node must be performed to evaluate relative displacement against other reference points, for principle see Figure 2.1. Although this yields the most accurate readings, it is limited to only vertical displacements and suffers from being time-consuming and difficult to automate. An efficient alternative method is to utilise robotic total stations. This method is based on the principle of measuring lengths and angles between fixed reference points. By relating reference points to placed prisms or targets, the relative positions can be mapped. The measured data are then related to other points and are automatically updated via the total station. The simplified principle can be summarised in Figure 2.1. This method significantly improves efficiency and reduces the manual labour required to map larger areas. Displacements are measured in three dimensions, producing both vertical and horizontal displacements. However, this method exhibits a loss of accuracy and increased costs compared to the levelling method.

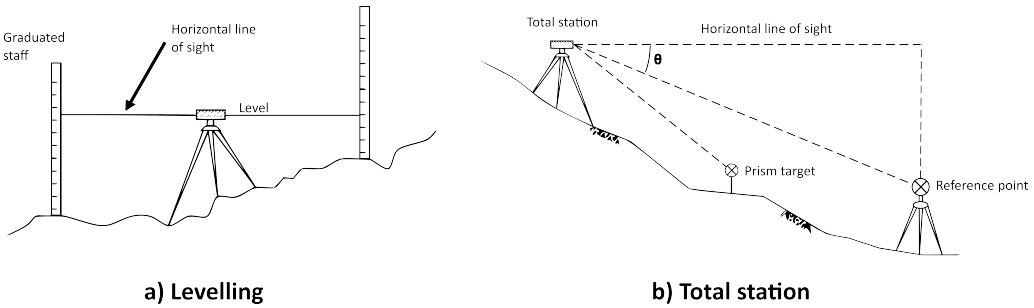



**Figure 2.1:** Principles of displacement measuring, for levelling and total station.

Unlike point-based methods, surface measurement systems allow comprehensive mapping with much higher resolution to be performed in short time periods (Reschetiuk et al., 2012). This results in more data points, which allows for a more complete evaluation of local displacements rather than just the global overview. Compared to point-based measurements, surface systems achieve less accuracy, usually with an uncertainty of around 5 - 20 mm. Such measurements could be performed using photogrammetry or laser scanning. With photogrammetry, physical structures can be interpreted using ordinary images and videos. This technology is based on the stitching of multiple images together and connecting them to known reference points, thereby creating a point cloud with relative measurement data for each point. This type of measurement is rather easy to perform and relatively cost-effective, but depends on sufficient lighting. Using laser scanning, the dependency on light can be avoided. However, the accuracy of this method depends instead on the reflectance of the target surfaces and is slightly more costly.

### 2.1.2 Strain

Strain gauges are able to capture deformation along their own axis. The most common types of gauges in civil engineering are often resistance-based, vibrating wire, or optical wire, each with different characteristics and thus different applicabilities (Yehia et al., 2008). The oldest and most common type is the resistance-based strain gauge (Enckell, 2006). It utilises the principle of geometric dependence with respect to the resistance of a conductor. An elongation of the conductor results in an increased length and a corresponding decrease in cross-section, consequently leading to an increased resistance. By measuring the change in voltage through the conductor as a result of the resistance change, the corresponding strain can be derived. This method is inexpensive, precise and often self compensating against temperature changes, however is tedious to install and to monitor automatically.

Another option is the vibrating wire strain gauge. This strain gauge is easy to install and automate, while still being inexpensive and accurate. The vibrating wire strain gauge is often preferred for long-term monitoring (Yehia et al., 2008). It consists of an encapsulated pre-tensioned steel wire anchored between two steel plates. When the gauge is strained, the tension on the wire increases. By excitation of the steel wire through an electromagnetic coil, the eigenfrequency of the wire in its tensioned state can be related to that of the untentioned state, allowing for elongation and corresponding strain being estimated.

Lastly, as a relatively new technology, fibre-optic sensors are on the rise within civil engineering (Yehia et al., 2008). Among different principles, a more commonly used method consist of a fibre-optic core with engravings. As light passes through these engravings, different wavelengths can be observed due to refraction. In addition to other measurements, such as temperature, strain can be measured. Although this method is insensitive to magnetic interference, it is relatively new within applied structural engineering for long-term monitoring and thereby lacks guidelines for such. Moreover, it is expensive and fragile, making it less common at present.

### 2.1.3 Acceleration

Acceleration is measured through accelerometers, being one of the more common measuring equipment within structural engineering. Not only does it capture the dynamic response in acceleration, many sensors allow for integration to retrieve velocity and displacement responses (Hanly, n.d.). The equipment are often divided into the categories of AC and DC equipment, which distinct characteristics.

Starting by the AC type, it only measures change in acceleration signal (Hanly, n.d.). Thus, it excludes steady states and only relate relative change, rendering it unfit to measure static events. Further, its internal RC time constant results in a delay, often undershooting signals. Consequently, small signal amplitude deviations may arise. While being of insignificant size for the acceleration signal, it posts major errors for numerical integration. Thus, AC sensors are not suitable for use in cases of integration towards velocity and displacement signals. However, the AC sensor have exceptional capabilities in capturing higher frequency events while providing a wide frequency response. Further, it also inherits a high signal to noise ratio.

In contrast, the DC sensor captures static events, capturing sustained accelerations and gravity (Hanly, n.d.). Thus, though DC equipment, velocity and displacement signals can be derived through numerical integration. The DC based sensors also inherits a better directional distinction, allowing for multi directional capturing of acceleration. However, compered to the AC type, DC based sensors are much more sensitive against environmental factors such as temperature, and thus are more likely to drift over time. Thereby, DC sensors are less appropriate in terms of long-term monitoring.

One of the more common types of accelerometers is piezoelectric accelerometers (Enckell, 2006). Their principle consists of utilising the ability of crystals generating voltage when put under mechanical stress. The technology consists of an encapsulated seismic mass which is mounted trough a piezoelectric element. Through seismic excitation, the stress applied on the crystal through the inertia created by the mass produces an electrical signal. This type of sensor if often preferred to their ability to produce a wide frequency response, being easily installed and providing a good sensitivity. However, being of AC type, the method only is appropriate for measuring of seismic events of higher frequency.

Another often preferred equipment is the piezoresistive accelerometer (Enckell, 2006). It works on the principle of measuring electrical change in a material due to strain. A seismic mass, either being gas or fluid damped, is attached to a flexural element, of which strain can be directly measured. For description of the strain gauge, see chapter 2.1.2. While being of DC type, allowing for measuring of static and low frequencies, the piezoresistive accelerometer suffers from significantly less accurate readings. Further, the equipment is far more sensitive to temperature, requiring temperature compensating measures to be taken.

Lastly, and as a increasingly more common equipment is the capacitive MEMS accelerometer. MEMS, standing for Micro-Electro-Mechanical Systems, make highly dense and small equipment (Wikipedia, 2024b). Normal span of component size vary from 1-100 micrometres, resulting in equipment in size of 20 micrometre towards a millimetre in total. Trough improved manufacturing techniques, the cost of a MEMS sensor is significantly lower compared to the other equipment discussed, mostly due to major decreases in material use (TE Connectivity, 2017). Further, being of DC type, they allow for capturing of static signals. However, the equipment often suffers from poor signal to noise performance and poor dynamic ranges. Commonly, this type of accelerometer is used in consumer products, such as phones and airbags, whoever is within the rise for the structural applications.

## 2.2 Monitoring of physical parameters

In addition to the mechanical parameters, physical parameters and environmental measurements play an essential role in structural monitoring. This information not only aids in modelling and monitoring but can also be used to calibrate measuring equipment, resulting in more precise predictions of future behaviours. Commonly monitored parameters in civil engineering, for instance, include temperature.

### 2.2.1 Temperature

Observing temperature is both straightforward and cost-effective (Enckell, 2006). In addition, the benefit of measuring the temperature is often of great importance. Temperature can play a crucial role in crack control during casting, early hardening stages and strength development, while it can also be used to compensate for temperature effects in other measurements. For instance, the vibrating wire strain gauge mentioned in Chapter 2.1.2 must be corrected for temperature fluctuations, as the tension in the wire is affected by changes in temperature.

The most common method for temperature monitoring is through thermocouples. This principle utilises the fact that conductors subjected to a temperature gradient generate voltage (Enckell, 2006). By joining two dissimilar metals, the generated voltage can be monitored and correlated with the corresponding temperature gradient. In addition, resistance thermometers are also frequently used in structural health monitoring. This method is based on the principle that a change in temperature alters the resistance of a conductor. The resulting voltage change can then be correlated with a specific temperature. Resistance thermometers offer improvements over thermocouples in terms of stability, accuracy, and repeatability, while thermocouples exceed in ease of installation and cost efficiency.

## 2.3 Data management

Data quality (DQ) is a recurring term in the field of data management (Ehrlinger & Wöß, 2022). DQ is a measure of the condition and validation of the data. This condition is assessed through various dimensions and criteria, which vary depending on the author and the application. However, the generally agreed-upon criteria include accuracy, reliability, relevance, timeliness, and completeness. Nevertheless, high data quality is of no use if the data is not easily accessible and comprehensible. Therefore, an appropriate and systematic organisation of the data is essential to avoid the risk of data loss due to inaccessibility.

The importance and benefits of achieving high data quality are not limited to increased work efficiency and economic profit, they also establish an essential building block for the integration of machine learning (Ehrlinger & Wöß, 2022). Implementing such tools would be highly beneficial, as manually evaluating and using the data could be overwhelming due to the shear quantity. To efficiently and accurately integrate machine learning, high-quality data sets are required. The process of achieving sufficient DQ depends on the type of data and its intended use. In the context of structural observational methods, multiple techniques can be employed and combined. Typically, the criteria for DQ can be achieved either by methods of how the measurements are acquired or by implementation of data cleansing.

### 2.3.1 Improving data quality

Two important questions to ask when measuring structural behaviour are *what* and *where*. The term *what* refers to the specific characteristic value that is to be measured. Even in the initial phase of measurement, the importance of predicting critical events should not be overlooked. By foreseeing such events, expectations on behaviour and critical movements could indicate *what* to measure and observe. This leads to the second term, *where*.

In addition to predictions of critical events, the sources and locations of the anticipated behaviour should be considered to ensure that the desired data are captured accurately. Expectations of behaviour for monitored structural elements can indicate *where* within the structure certain measurements would be most advantageous. Following these principles, a better DQ can be achieved through a more thoroughly performed measurement process (Ehrlinger & Wöß, 2022).

Additionally, achieving a balance in the number of measurement points could also be important (Khoa & Takayama, 2018). Naturally, fewer measurement points put higher requirements for accuracy and error tolerance per measured point. Thus, by increasing the number of measurement points, improvements in DQ can be achieved. More points will produce a more complete and coherent data set, where points of deviation have less effect. However, a decrease in general efficiency could occur if too many measurement points are used, along with an obvious increase in resource usage.

Regardless of how carefully measurements are performed, errors and measurement faults are likely to occur. This is in part due to errors in the hardware and disturbances during measurement and data handling. Such errors should be noted, corrected or ignored to avoid triggering false alarms and to maintain high data quality (Ehrlinger & Wöß, 2022). To overcome obvious errors, data cleansing can be utilised. As a result, collected data are corrected and cleared of obvious anomalies. This process involves comparing such anomalies to adjacent values to assess the realism of these values. Furthermore, cross-evaluation among other measured quantities can be important for identifying reasonable causes and sources of error. However, it is always recommended to mark anomalies to allow a person to evaluate them and make the final decision.

# 3
# Model updating

The concept of model updating is far from new, with its origin and pioneering steps toward FEM implementation dating from the late 20$^{th}$ century (Behmanesh et al., 2015). Since then, countless approaches have been developed with the aim of creating faster and more precise models.

In the following chapter, different approaches and techniques of model updating and optimising will be presented. From the pioneering stages of optimising to the unproven and currently emerging principles, a wide range of topics will be covered. Firstly, more conventional methods, often preferred and used within the field of structural engineering, are presented and discussed. The section will mainly cover four of the more common optimisers, their principles strengths and weaknesses. Following, the more contemporary area of neural network will be discussed and presented. Their underlying principles, different methodologies and application in terms of optimisation will be presented. Moreover, the pre-trained neural network of ChatGPT, created by OpenAI, is presented. Its abilities and relevant characteristics in terms of model updating applications will be discussed. Lastly, as an important part of FEM model updating, the most critical aspects of FEM modelling will be presented.

## 3.1 Optimisation

Regardless of the method, the basis and principles of model updating and optimisation are relatively simple. An objective function, which consists of one or multiple model parameters, serves as the foundation for the update process. In terms of model updating, the objective function often contains a residual value to compare measured or observed data against modeled data. For each iteration, the current values of the objective functions are evaluated and are often referred to as the cost. The process of updating model parameters, remodelling and evaluating the cost continues until the cost is deemed low enough and thus satisfactorily minimised.

The models are often categorised according to their underlying algorithms and characteristics. The first categorisation is deterministic or stochastic models (Wan & Ren, 2016). Deterministic models assume the existence of only one precise solution for each data set. Thus, recalculation with the same data set yields the exact same solution. In contrast, stochastic methods introduce some randomness, which allows for considerations of measurement variability and probability density functions. The

slight randomness can contribute to an increased ability to climb out of local minima in the hope of finding a global minimum, but this ability often comes at the cost of a slight increase in computation time.

Heuristic algorithms represent a category of methods designed to search for approximate or near-optimal solutions. They aim to reduce computational times at the expense of accuracy and completeness, proving effective for problems lacking an exact solution and for those that would otherwise be computationally intensive (Singh, 2020). The heuristic methods can be divided into two categories, construction methods (greedy algorithms) and the local search methods. The greedy algorithm aims to find the optimal solution for the whole system in phases, attempting in each iteration to explore the nearest unexplored values to find the overall optimal solution. Local-search methods, on the other hand, start with an initial guess, and by exploring neighboring solutions, iteratively refine the value to achieve the most optimal solution.

Moreover, the distinction between gradient and gradient-free methods is often made where the gradient refers to the derivative of the objective function (Majid Solat Yavari et al., 2016). Through a gradient-based method, the next iterative step is based upon the gradient towards the steepest descent. While this often results in faster convergence towards the minimum, there is a risk of entrapment within a local minimum. However, if the objective function is based on nonlinear, discontinuous, or discrete data, gradients and derivatives cannot be calculated. For such cases, a gradient-free method might be more appropriate. Through gradient-free methods, the next step is evaluated only through the current and past values of the objective function. Although such methods tend to converge slower, they are often less computationally intensive and easier to implement. There is also a gain, as gradient-free methods tend to avoid local minimum entrapment more efficiently.

Lastly, there is often a distinguishing between local and global search algorithms (Khouri Chalouhi, 2022). Local search methods evaluate their current cost based on certain parameters and step towards the direction of the lowest cost in their vicinity. In contrast, global search methods often incorporate a phase of exploration, in which general areas are randomly explored. Consequently, this approach drastically increases the chances of avoiding local minimas.

The optimal choice of method and algorithm depends heavily on the intended scenario and the desired outcome. For certain cases, combining multiple methods might work to combine the strengths of different methods, while other methods might be outright inappropriate. Thus, one must carefully choose the appropriate method and set objective functions that efficiently and accurately optimise towards the pursued outcome. A general scheme of different methods and categorisations is illustrated in Figure 3.1.

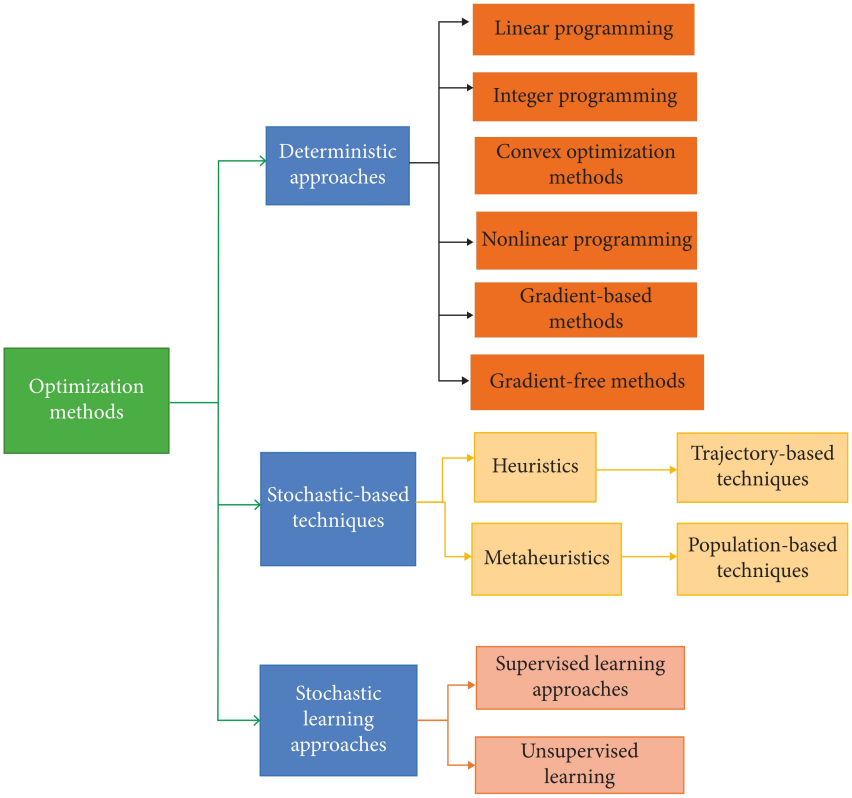



**Figure 3.1:** Categorisation of updating/optimising techniques (Mashwani et al., 2021).

## 3.2 Conventional optimisation

Within the field of structural engineering, the updating or optimisation of models is commonly used. Most often, model updating is used when structural behaviour is observed and deviates from modelling. By updating and optimising the model in an iterative way, the model parameters can be adjusted until an appropriate correlation is achieved. Consequently, the pursuit of global minima frequently constitutes the objective, as achieving such minima yields solutions that are both optimal and efficient (Khouri Chalouhi, 2022). Furthermore, the data obtained from the observations, which are characterised by their discrete and discontinuous nature, often delineate behavioural patterns through nonlinear relationships. Given these complexities, the application of gradient-free methods becomes a preferable approach.

During the last decades, a vast amount of research has been preformed regarding both the benchmarking and development of optimisation algorithms within the field of structural engineering. Throughout such research, it is evident that more efficient and accurate methods are developed periodically. However, it appears that the industry conventionally relies on older methodologies that are both proven and comparatively efficient.

### 3.2.1 Nelder-Mead Simplex

The Nelder-Mead method is a non-gradient, deterministic, local search method that was first introduced in 1965 by John Nelder and Roger Mead (Chang, 2012). Its one of the most prominent method for nonlinear optimisation that does not require derivatives (Ali et al., 2013). The Nelder-Mead method has an inherent resemblance to how the earlier developed Simplex method finds an optimal solution. The Simplex method was developed by George Dantzig in 1947 and the method is categorised as linear programming or linear optimisation and represents a fundamental technique in linear optimisation (Wright, 2022). Fundamentally, the objective of such a method is to minimise or maximise an outcome, such as achieving maximum profit at the lowest cost. These functions operate on simplicial corners that are formed from constraints, these constraints delineate the corners of a geometrical polytope (Eiselt & Sandblom, 2007). The Nelder-Mead method is based on the same procedure as the simplex method, which leverages the geometrical vertices of the function to find the maximum or minimum for a linear system. The optimiser works by initiating three values ($x_1$, $x_2$ and $x_3$) that collectively form a triangle or a simplex. This formation is complemented by the central point ($x_o$) of the three vertices. Each vertex is evaluated and arranged in a vector with the best approximation in the first order and the worst approximation at the end. The least favourable vertex is subsequently discarded and supplanted by a new estimation. During each iteration, the new guess is estimated using four different techniques, reflection, expansion, contraction, or shrinkage, as illustrated in Figure 3.2.

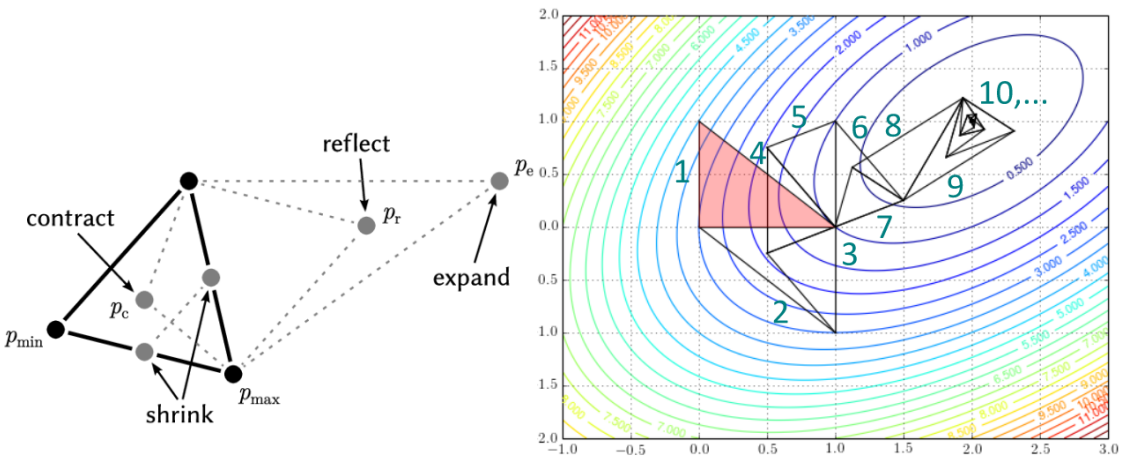



**Figure 3.2:** Updating procedure of the Nelder-Mead method (Yu Cheng & Mailund, 2015).

The reflection procedure involves inverting the triangle from the position of the worst guess to the opposite side of the triangle, see Equation 3.1. The reflected value ($x_r$) is retained if the approximation is better than the second worst guess, but does not exceed the most favourable value.

$$x_r = x_o + \alpha(x_o - x_n) \qquad with \quad \alpha > 0 \tag{3.1}$$

The expand procedure is used if the outcome derived from the reflection process yields the most favourable value. Then instead of merely moving an inverted step size of magnitude in that direction, an additional expanded step ($x_e$) is explored in that direction to evaluate that trajectory, see Equation 3.2. The worst guess is substituted with this expanded value if this approximation is better than the second-best and better than the reflected value.

$$x_e = x_o + \gamma(x_r - x_o) \qquad with \quad \gamma > 1 \tag{3.2}$$

The contraction procedure is executed through two distinct approaches, external ($x_{c,ext}$) or internal ($x_{c,int}$) relative to the initial simplex. If the reflected value does not result in an improved approximation but still exceeds the average centre point ($x_o$), contraction can be conducted outside ($x_{c,ext}$) the simplex. This involves selecting a point that lies between the centre, mean point and the reflected value, as illustrated in Equation 3.3. In contrast, in scenarios where the reflected value does not result in an improved approximation and is still worse than the centre ($x_o$) while better than the worst ($x_n$), an inward contraction is performed within the simplex ($x_{c,int}$).

$$\begin{aligned} x_{c,int} &= x_o + \rho(x_n - x_o) \qquad with \quad 0 > \rho > 1 \\ x_{c,ext} &= x_o + \rho(x_r - x_o) \qquad with \quad 0 > \rho > 1 \end{aligned} \tag{3.3}$$

The shrinkage procedure is invoked when none of the previous attempted strategies, reflection, expansion, or contraction have resulted in any improvements. The shrinkage operation adjusts all the vertex values except for the one corresponding to the best approximated value ($x_1$), as seen in Equation 3.4.

$$x_s = x_1 + \sigma(x_n - x_1) \qquad with \quad 0 > \sigma > 1 \tag{3.4}$$

Standard values for $\alpha = 1$, $\gamma = 2$, $\rho = 0.5$ and $\sigma = 0.5$ (Yu Cheng & Mailund, 2015).

### 3.2.2 Pattern Search

The foundational principles of the Pattern Search (PS) method were first presented in 1961 by Robert Hooke and T.A. Jeeves (Lewis et al., 2000). The proposed method is characterised as a non-gradient type and can be implemented with both stochastic and deterministic character. Despite its inception over six decades ago, it still remains favoured across various disciplines due to its well established efficacy. Although the method is simplistic, it rivals other more sophisticated methods. It has further been observed that the PS method even outperforms more sophisticated algorithms due to its straightforwardness, by circumventing typical complications that challenge alternative approaches.

The fundamental principle of the pattern search method operates on an iterative process of trials and reevaluation, similar to a trial-and-error method (Khouri Chalouhi, 2022). The procedure is visually shown in Figure 3.3 below. Optimisation commences with an arbitrary initial solution $X_0$ from which the cost of the objective function is determined. Subsequently, a change in the variable is evaluated one direction or parameter at a time through a technique known as Polling. In each trial, the current cost of the objective function is evaluated and compared with the costs at neighbouring positions, determined at a preset step size from the initial point. These incremental trial steps are termed exploratory moves. For instance, $\pm\Delta$d are independently examined along the directions of $d_1$ and $d_2$. Should any trial result in an improved solution, it constitutes a successful poll which then guides the subsequent pattern move. For visualisation, see Figure 3.3, considering the origin of $X_0$, the exploratory move of $+\Delta$d was successful in both directions. Conversely, if the adjustment of $\pm\Delta$d fails to produce an improved solution in any direction, the poll is considered unsuccessful, prompting a reduction in the step size of $\Delta$d on the presumption that the optimal solution is in close proximity to the current position. This cycle persists until predefined termination criteria are satisfied.

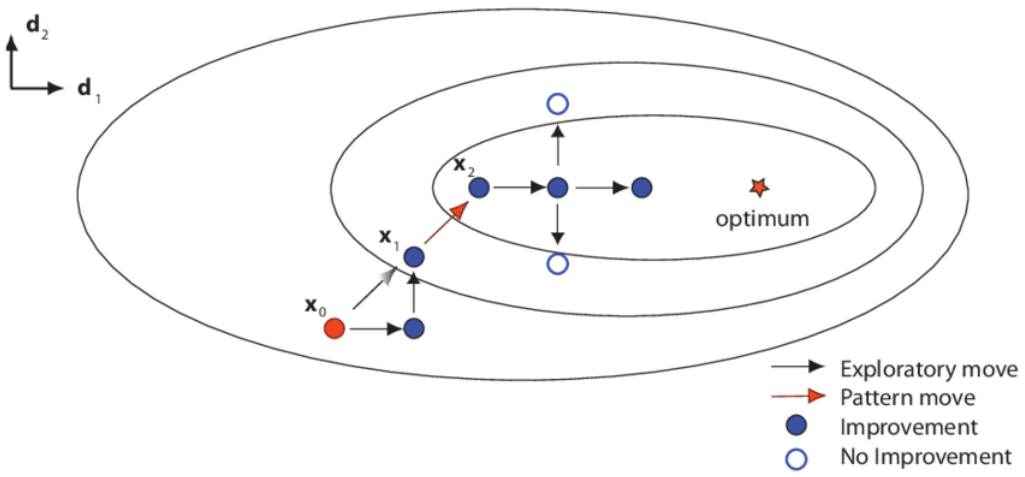



**Figure 3.3:** Updating procedure for pattern search algorithm (Kramer et al., 2011).

### 3.2.3 Genetic Algorithm

The genetic algorithm (GA) is a stochastic, gradient-free, global search method that is frequently used and often preferred within the domain of structural engineering (Ereiz et al., 2022). Its success is largely based on the simplicity of its implementation. It originates from Darwin's theory of evolution and the concept of natural selection with survival of the fittest. Within a population, individuals that exhibit advantageous traits are more likely to reproduce, thus passing on these traits to subsequent generations. The principle of genetic algorithm operates on a similar premise. It starts with a set of different solutions that represent the population. Each solution or individual is evaluated and assigned a fitness score based on its cost. Solutions with a better fitness score are more likely to reproduce, which enhances the probability that superior traits will be passed on to future generations through a selection process favouring those with more desirable traits. Subsequently, the crossover process occurs among the better performing solutions, generating offspring that inherit a combination of the characteristics of their parents. Mutation also plays a crucial role in this process. For each generation, a specified percentage of individuals undergo a mutation. With the introduced random characteristics, the population diversity is enhanced and the algorithm's local search capabilities are improved.

These steps constitute a single iteration that is repeated until either a sufficiently low error is achieved, below a predetermined threshold, or the maximum number of iterations is reached. The general concept is visualised in Figure 3.4.

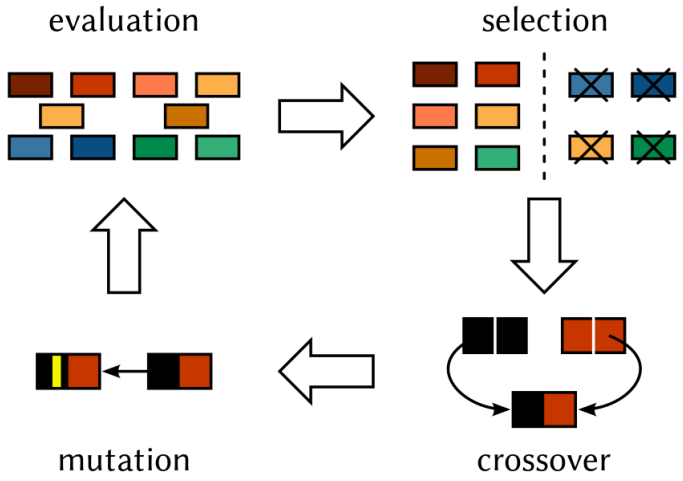



**Figure 3.4:** Updating procedure for genetic algorithm (Cheng Yu, n.d.).

The algorithm for genetic optimisation may vary depending on application and preference, however, common basis are presented below. Initially, the objective function must be stated and the cost is evaluated for each individual or chromosome. The objective function is tailored to the problem at hand and the desired result, with the cost associated with each chromosome determining its fitness. The method of evaluating fitness presented below is often referred to as the roulette wheel method (Kostadinov, 2019). The procedure involves calculating the probability of fitness of each chromosome, using Equation 3.5 where $F_i$ represents the fitness of each chromosome. This equation is designed to assign a higher probability for larger values of $F_i$, under the premise that substantial $F_i$ values should be indicative of lower cost values. Consequently, $F_i = cost^{-1}$ can be set in minimisation scenarios.

$$FP_i = \frac{F_i}{\sum F_i} \tag{3.5}$$

Once the fitness probability is determined, a cumulative summation is established by Equation 3.6. Resulting in cumulative probabilities that range from zero to one. Subsequently, random values within this interval are generated to select chromosomes for crossbreeding. Chromosomes that exhibit a higher probability of fitness are more likely to be selected, thereby enhancing the probability that their advantageous traits will be propagated.

$$C_i = \sum_{n=1}^{N_Pop} FP_i \tag{3.6}$$

The different methods to execute crossbreeding and mutation can vary significantly. Regarding crossbreeding, one of the simplest strategies involves selecting random traits between parents (Hermawanto, 2013). Conversely, more sophisticated methods that involve interpolating, calculating, or combining the traits of the parents have also been studied with varying results. Mutations can in a similar manner be implemented in various ways, where the more simple method of mutation could be implemented by arbitrarily adjusting certain traits.

Genetic algorithms are a widely adopted strategy for optimisation, particularly noted for their capacity to navigate discrete data characterised by unknown and intricate relationships. (Ereiz et al., 2022). The drawbacks of the GA optimisation procedure are that the method is computationally heavy. With large systems and multi-objective optimisation tasks potentially result in poor performance in terms of computational time. Comparative studies have highlighted GA's inefficiency relative to other optimisation methodologies, prompting the development of various adaptations. Among these, the Hybrid Genetic Algorithm (HGA) that combines the genetic algorithm with Nelder-Mead's simplex method, achieving enhanced efficiency.

### 3.2.4 Particle Swarm

The principle of the Particle Swarm Optimiser algorithm (PSO) was developed by James Kennedy and Russell C. Eberhart, who jointly formalised the concept (Ereiz et al., 2022). It is a global, gradient-free, stochastic optimiser that mimics collective behaviours from animals and insects. A group of individuals improves their search efficiently by communicating and collaborating on their search activity. This concept is often analogous to birds and their methods of searching for feeding grounds. By each individual sharing their findings, the group as a whole can concentrate its efforts towards the most promising locations. At its core, the particle swarm algorithm closely resembles this. A set of random solutions, representing individuals, is scattered throughout the solution space, each assigned an initial velocity and direction. With each iteration, the individuals move in their corresponding direction. The cost of the objective function is then assessed for each individual in their new positions. Should an individual reach a new personal best, this achievement is stored individually. If the new position also corresponds to the best position globally, it is recorded individually and communicated to all individuals in the group. In subsequent iterations, each individual adjusts their velocity and direction towards a direction that compromises personal and global best results with the aim of a good balance between the both.

During each iteration, the updated velocity for an individual denoted as $i$ is obtained by Equation 3.7. The term $w$, commonly referred to as the inertia weight constant, has a value in the range of zero to one, which decides the degree of reluctance to change velocity and direction (Tam, 2021). The following terms, $c_1$ and $c_2$, represent the cognitive and social coefficients correspondingly, both of which have a value between zero and one. The relationship between $c_1$ and $c_2$ facilitates a balance between an individual's preference for its own optimal path and the collectively acknowledged optimal direction. Furthermore, $r_1$ and $r_2$ introduce randomised values within the same range that ensure variation and diversity between the weights, thus enhancing the algorithm's ability to circumvent local minima.

$$V^i(t+1) = wV^i(t) + c_1 r_1(X^i_{Pbest} - X^i(t)) + c_2 r_2(X^i_{Gbest} - X^i(t)) \tag{3.7}$$

With the newly updated velocity and direction, the positions of the individuals are recalculated using Equation 3.8. The term $X^i$ corresponds to a coordinate within the solution space, consisting of both $x_1$ and $x_2$. This coordinate is updated by summing it with the product of velocity in the specified direction, facilitating movement within the search space towards potential solutions.

$$X^i(t+1) = X^i(t) + V^i(t+1) \tag{3.8}$$

An visual representation of the particle swarm principles can be seen in Figure 3.5. The solution space are defined by the Equation $f(x_1, x_2) = (1 - \frac{x_1}{2} + x_1^5 + x_2^5)e^{-x_1^2 - x_2^2}$. The function was purposely selected for its inclusion of both local and global minima that are closely aligned in terms of value and spatial position. This

function effectively demonstrates the algorithm's capability to navigate past local minima and can be seen in Figure 3.5. In the illustrations in Figure 3.5, the direction and magnitude of the arrow represent each particles current direction and velocity. The gray dots correspond to each particle's personal best achieved so far. The red star marks the position of the current global best identified by the swarm. A white cross indicates the absolute minimum of the function, which remains undisclosed to the optimiser. For iteration 01, the particles are observed to move in random directions, with the current global best situated near a local minimum. By iteration 03, most particles have changed direction as they align towards the reported best, however a new global best has been recorded near the global minimum. For iteration 07, there is a discernible adjustment in the particles' trajectory from the former global best towards the latest one. Finally, by iteration 30, it is evident that the majority of the particles have converged in the vicinity of the actual global minimum, illustrating the PSO's effectiveness in honing in on the optimal solution over time.

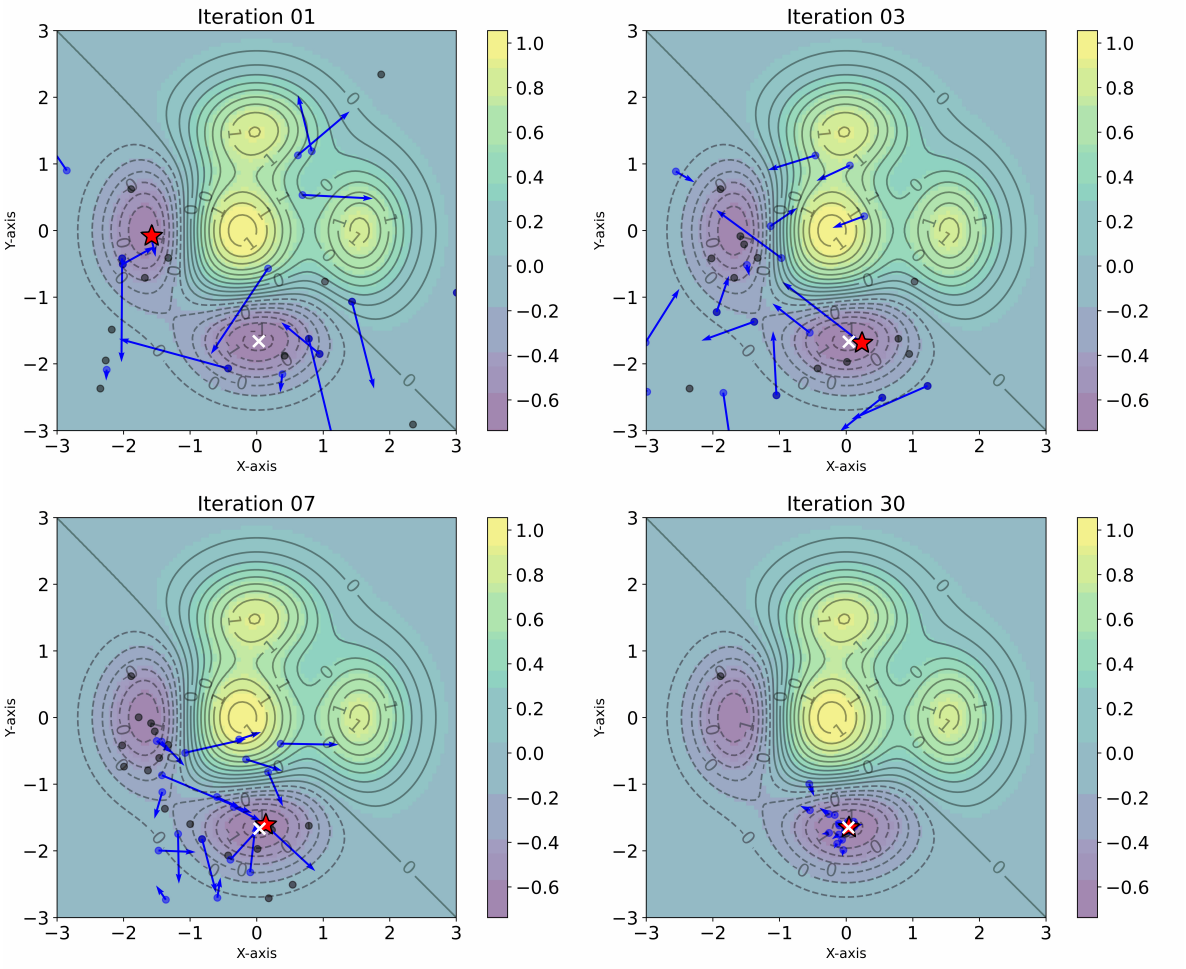



**Figure 3.5:** Updating procedure for particle swarm algorithm. Example iterations.

## 3.3 Artificial Intelligence optimisation

Artificial Intelligence (AI) has gained considerable interest and popularity in recent decades (Anyoha, 2017). Beginning in the 1950s with pioneers such as Alan Turing, who deciphered the German Enigma cipher and formulated the concept of artificial intelligence, known as the Turing test (Copeland, 2024). Significant milestones include the creation of the first self-driving car, Navlab, in 1986, IBM's Deep Blue defeating world chess champion Gary Kasparov in 1997 and the development of the speech recognition software Dragon Systems in the same year (Anyoha, 2017). Artificial intelligence can be categorised into three main categories: weak, strong, and super AI. Currently, there are only weak or narrow AI that exists, defined as computer-driven intelligence that performs specific tasks through training. Strong or General Artificial Intelligence (AGI), which remains theoretical, would possess the capability to learn new tasks through the experience of previously acquired skills. Lastly, super-AI, also theoretical, represents a future in which computer intelligence could surpass human cognitive abilities in learning, thinking, and reasoning (IBM Data and AI Team, 2023).

Artificial intelligence or AI is a broad definition of several subcategories, including for example (Chowdhary, 2020):

- Machine learning, including decision trees, rule learning, and instance-based learning.
- Speech Processing and Natural Language Processing (NLP).
- Computational Neuroscience and Evolutionary Computation, featuring evolutionary programming, neural networks, and classification algorithms.
- Machine Vision and Robotics, focusing on image recognition and autonomous exploration.
- Support Vector Machines (SVM) for pattern recognition.
- Statistical Machine Learning, applied in data mining and learning from data, along with the Naive Bayes classifier.
- Deep Learning and Reinforcement Learning.

### 3.3.1 Neural Network

Neural Networks (NN) draw inspiration from the structural and functional aspects of the human brain. They can be implemented in a multilayer configuration, often designed as a series of interconnected layers that extend deep, where each neurone functions as an individual computational unit (Marwala, 2010). These neurones, inspired by their biological counterparts, use forward and backward propagation techniques, utilising a form of gradient descent optimisation method alongside an error-based cost function.

The process of forward propagation and evaluation is illustrated in Figure 3.6 and 3.7, which can be briefly described as follows: A neural network is structured into layers consisting of an input layer at the beginning, several hidden layers in the middle, and an output layer at the end. The input data are separated into validation data and training data and the training data is passed through the neural network in iterations or epochs. For each forward pass through the neural network, the model's biases and weights are used to approximate the output. The model predictions are compared with the validation data, the separated data that were not part of the training data. The size of the error then determines how much each weights and biases should be adjusted in the network. Once the difference between the validation data and approximated equivalent is small enough, convergence is met and training is complete.

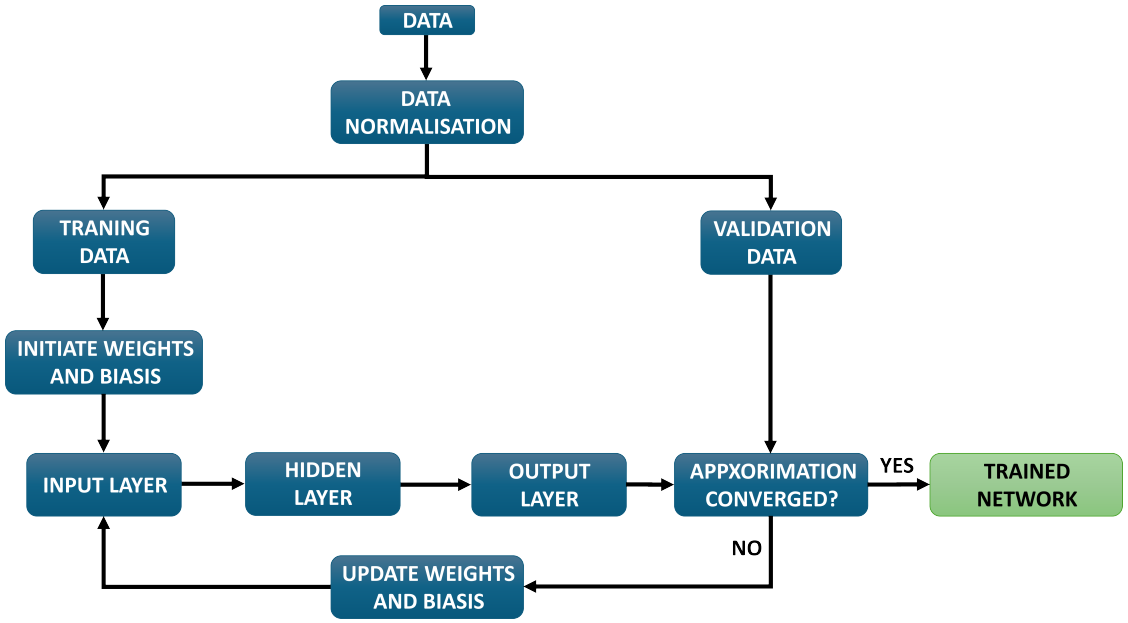



**Figure 3.6:** Illustrated workflow of the neural network training process.

The neural network first layer, also known as the input layer, is shown in Equation 3.9. The input is often firstly subjected to a scaling or normalisation process. The input layers typically consist of a vector $a^{(k=1)}$ with $k$ number of rows, see Equation 3.9.

$$x_{(k)} = a^{(k)} \in \mathbb{R} \tag{3.9}$$

The next step is when the information $a^{(k+1)}$ gets fed through the hidden layers, which can vary in both width and length. A conventional method involves compiling the values for each layer into matrices. The calculation process for each neurone starts with the input from the previous step, either from the input layer or a previous hidden layer, which is multiplied with the neurones weights $w^{(k+1)}$ by the standard execution of matrix multiplication, where it is important to ensure that the rows and columns match. An example of these weights and bias calculations can be seen in Equation 3.10.

$$z^{(2)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} & W_{14}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} & W_{24}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \end{bmatrix} \tag{3.10}$$

Followed by combining the neurones bias $b^{(k)}$ to obtain a partial result $z^{(k+2)}$, as seen in Equation 3.11. The last value $a^{(k+2)}$ are the output result from the neuron and serves as input for next layer as seen in Equation 3.12.

$$z^{(k+2)} = w^{(k+1)} \cdot a^{(k+1)} + b^{(k+1)} \tag{3.11}$$

$$a^{(k+2)} = f(z^{(k+2)}) \tag{3.12}$$

Illustrated below in Figure 3.7, is an example of a four layer neural network, its neurones with their weights, biases and activation function. The final output value $s$, represents the predictions of the neural network from the input data $x$.

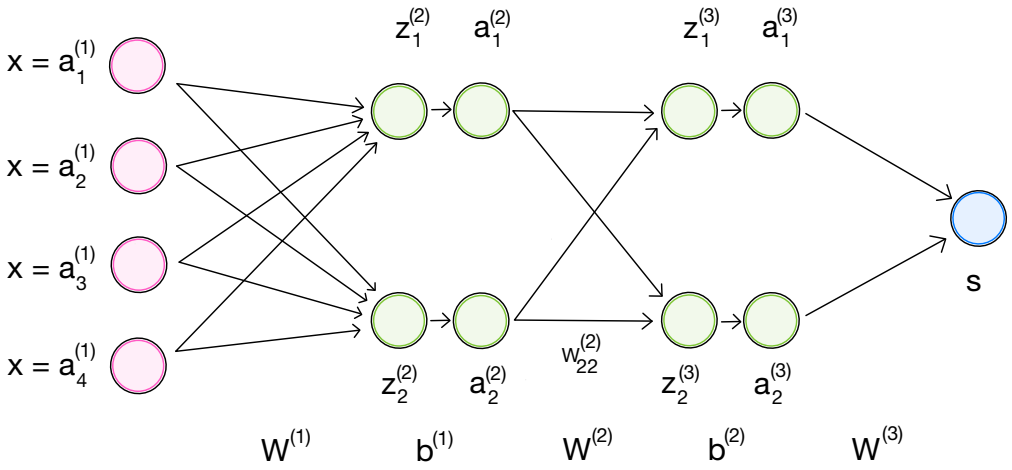



**Figure 3.7:** Illustration of a four layer neural network with input (purple), hidden (green) and output layer (blue).

This output is compared against an expected value $y$, derived from the training input data $x$, which acts as the target response. The discrepancy between the predicted output $s$ and the target $y$ is assessed using either a cost or a loss function, with the appropriate choice depending on the nature of the problem. The cost function computes the average error throughout the training dataset, whereas the loss function is determined for a single training data set. The outcome of either the cost or loss function provides a consolidated measure of the performance of the neural network to approximate the target value, as shown in 3.13 (Singh, 2010).

For regression problems, the Mean Squared Error (MSE) may serve as the loss function. In contrast, for classification tasks, either a binary classification loss or a more sophisticated cross-entropy loss for multi-class outputs may be used.

$$C = cost(x, y) \tag{3.13}$$

Through the application of the cost function, new approximations are derived by adjusting weights and biases by backpropagation (IBM, n.d.). The weights correspond to how strong the connection is between nodes and the direction, if its positive or negative. Stronger connections have a more significant impact on output parameters and enhance the learning process within the neural network.

Biases are a constant value added in the feedforward process in the neural network. They enable the activation function of a neurone to adjust its output towards either a positive or negative direction. Together with the weights that can change the steepness of the function, the biases can offset the whole function to manipulate the output value. Biases play a crucial role in determining the activation threshold of neurones within the network, effectively pushing the output values to the extremes of the activation functions.

Backpropagation is fundamental to neural networks, training them by computing gradients using the chain rule (Kostadinov, 2019). The gradient of the cost function with respect to the input value x, indicates the direction of change required for $C$ to minimise the cost. With each epoch or loop, the weight and bias are updated according to Equation 3.14 with the learning rate $\epsilon$.

$$w = w - \epsilon \cdot \frac{\partial C}{\partial w} \qquad b = b - \epsilon \cdot \frac{\partial C}{\partial b} \tag{3.14}$$

## Activation functions

Activation functions determine whether a neurone should be activated or not and in the case of nonlinear functions, to what extent (Buhl, 2023). The choice of activation function depends on the nature of the problem, the context, the constraints and the complexity. The activation function are used to transform the input values where the desired outcome depends on the nature of the problem. Common activation functions can be seen in Figure 3.8 and is described in more detail below.

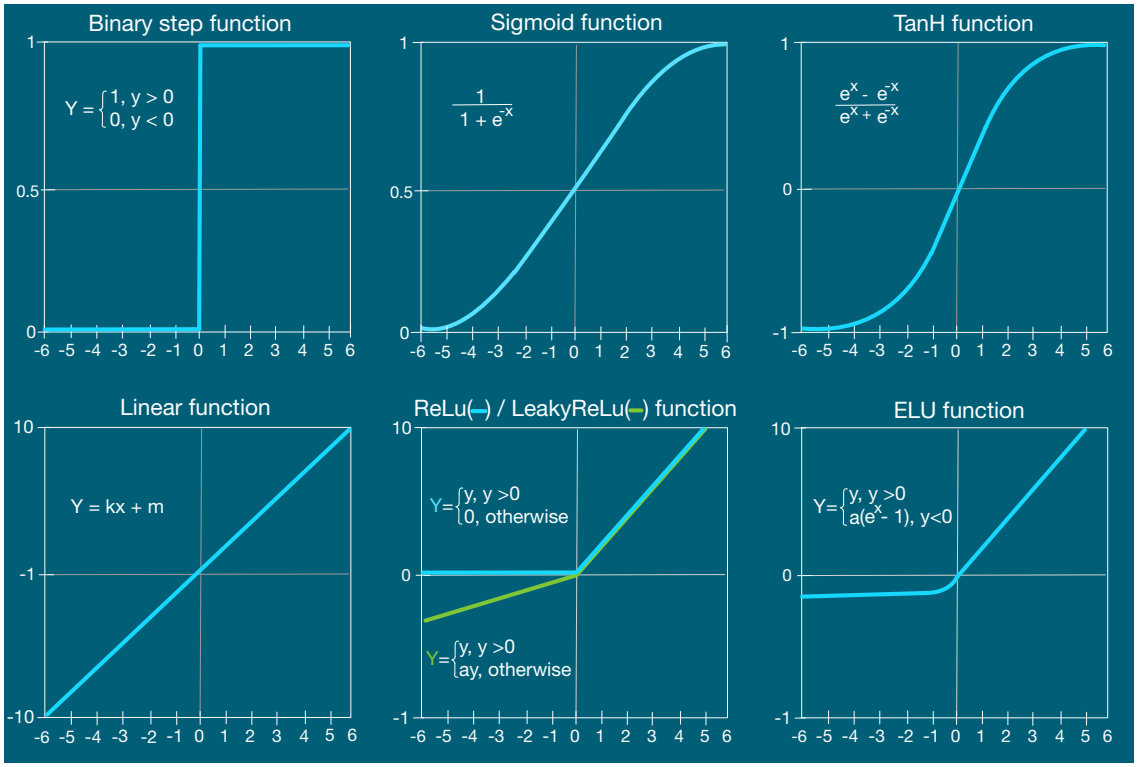



**Figure 3.8:** Samples of different activation functions in neural networks.

Linear activation functions are fast and robust, while nonlinear functions enable neurones to comprehend more intricate relationships within the input data. The derivative contains variables and enables the neural network to adjust the parameters in the backpropagation to better fit the expected outputs (Buhl, 2023).

### Binary Step

The simplest form of an activation function is the binary step function, as illustrated in Figure 3.8. This function assigns a value of either zero or one, based on a predefined threshold (on/off), determined by whether the input exceeds or falls short of this limit. While suitable for categorisation, the step function encounters difficulties in differentiating between multiple classes because of the simplicity of the function. The step function can be used to build the simplest feedforward neural network, Single-Layer Perception (SLP) (C. C. Aggarwal, 2018). The single-layer perceptron acts as a linear classifier, constrained to binary classification tasks involving linearly separable data sets.

### Identity

A linear function gives a linear relationship to the input and thus has a constant derivative, resulting in linearly dependent weights or biases (Buhl, 2023). Incorporating linear functions into deeper layers does not enhance the model's learning capacity, as the output of each layer remains linearly dependent on its predecessor. However, linear functions can be effectively integrated into deeper layers when combined with nonlinear functions. They are applicable to both categorisation and regression problems. Beyond the standard linear or identity function, there exist variants such as the Piece wise Linear (PL) function. This function utilises an affine function for linear interpolation within a specified range, assigning values above this range as one and those below as zero.

### Sigmoid

The Sigmoid activation function $[1/(1+e^{-x})]$ also known as the logistic, see Figure 3.8, is nonlinear in nature. The function smoothly transitions between the upper and lower limits of one and zero, respectively, as the values on the x-axis increase, the function approaches the limits on the y-axis. It can be used for classification or probability problems. However, its derivative can lead to issues with vanishing gradients for large input values, potentially decelerating or halting the learning process.

### Hyperbolic Tangent

The Tanh activation function or hyperbolic tangent $[tanh(\mathrm{x})^{-1} = (2/1 + e^{-2x}) - 1]$, see Figure 3.8, is highly favoured in neural networks and can be regarded as a scaled version of the Sigmoid function. This nonlinear function ranges between -1 to 1, featuring a steeper transition compared to the Sigmoid. The output values are zero-centric and simplify the scaling between positive and negative values. Although its gradients are steeper, similar to the Sigmoid function, it is also susceptible to the issue of vanishing gradients.

### Rectified Linear Unit

The ReLU (Rectified Linear Unit) activation function $[\mathrm{x}^{+} = max(0, x) = (x+|x|)/x]$, see Figure 3.8, is a piece-wise function with a constant linear slope that advances to infinity for all y-values above zero. The breaking point for non-zero values, makes the function inherit nonlinear qualities. ReLU are generally less computationally expensive compared to other nonlinear functions. While ReLU is not differentiable at all points, gradients can still be calculated. For input values below zero, the neurons outputs zero. The ReLU function has the advantage of being sparse for non-activated neurones in the sequence, which can result in faster training. However, it is susceptible to the "Dying ReLU" problem, where neurones become inactive and fail to recover from zero or negative input, halting learning in parts of the network. Additionally, without an upper bound, ReLU activations risk "exploding" for large input values. A variant of ReLU, known as Leaky ReLU, addresses the Dying-ReLU problem by assigning a small, non-zero gradient (typically 0.01) for values below zero, instead of a flat zero. This ensures that derivatives remain non-zero for negative inputs.

### Exponential Linear Unit

The ELU (Exponential Linear Unit) activation function is denoted as $[f(x, a) = \mathrm{x}$ for $x \geq 0$ otherwise $= a(e^{x}$ - 1)], seen Figure in 3.8 (Franco, 2024). The idea of this activation variant are to compensate for a noise problem that risk to submerge when using leaky ReLU (and PReLU). When introducing non-zero values with a small gradient below zero, there is a risk of an unbalance in the system submerged for very large negative values. Noise sensitivity is reduced by introducing a negative logarithmic value for values below zero instead of a straight line. Although ELU is not differentiable across its entire domain, it enables gradient computation without being susceptible to the Dying-ReLU issue. SELU (Scaled Exponential Linear Unit), an adaptation of ELU, refines the normalisation concept in ELU to control the gradient more efficiently and removes the risk of vanishing gradient and Dying-ReLU problem to improve learning rate.

Additional examples of activation functions include: Softplus, Maxout, Parametric Rectified Linear Unit (PReLU), Sobolev Modified Hyperbolic Tangent (SMHT), Gaussian Error Linear Unit (GELU), Sigmoid Linear Unit (SiLU), Sigmoid Shrinkage (SiL) and Smooth Non-Monotonic (Swish) function (Wikipedia, 2024a).

## Convergence in neural networks

Neural networks converge towards an optimal solution using the same procedure as for regular optimisation problems. The principle of deriving appropriate weights and biases comes from minimising the loss function. Adjustments and optimisation of the weights and bias, often derived through gradients and backpropagation, enable the network to adapt and converge towards optimal values. This adaptation and convergence can be facilitated by various optimisation methods.

Commonly used optimisation methods for neural networks use first- or second-order derivatives, with a fixed step size or combined with momentum. To locate the minimum of the loss function, Gradient Descent (GD) or a variation of this method is often utilised (N. Aggarwal, 2023).

The gradient decent uses the gradient to stepwise converge to a minimum cost. It purely relies on first-order derivatives, which indicate the direction towards the lowest point (Hassan et al., 2023). The method is of deterministic character and thus seeks the single most optimal solution. During backpropagation, the process of updating weights and biases involves assessing the gradient across the entire training set for each iteration. The weights (w) are subtracted with the gradient of the loss function (L) with a fixed learning rate ($\epsilon$) as seen in Equation 3.15.

$$w = w - \epsilon \cdot \nabla_w L(w) \tag{3.15}$$

Although this methodology produces an optimal solution, it comes with a significant computational cost for large data sets.

Stochastic Gradient Descent (SGD) is a variant of GD designed to lessen computational demands by evaluating either a single example or a small batch of data for each iteration (Roy, 2023). Similar to gradient descent, the weights are updated with a fixed learning rate for corresponding gradient to the selected batch, see Equation 3.16.

$$w = w - \epsilon \cdot \nabla_w L(x_i, y_i, W) \tag{3.16}$$

The selection of data sets for each iteration is done randomly, and thus a stochastic approach is used. As a result, the gradients computed in each iteration serve as approximate gradients rather than precise representations of the true gradient. Consequently, additional iterations may be required since SGD typically follows a more erratic path to the minimum compared to GD, as illustrated in Figure 3.9. The characteristic of an erratic trajectory of SGD proves advantageous to address objective functions made up of multiple functions (Kingma & Ba, 2015). The reduction in computational demands compensates for the increase in the number of iterations, which frequently enables SGD to surpass GD in performance. The erratic behaviour further improves the ability to escape shallow local minimums, as it often overshoots due to the stochastic variations. Nonetheless, careful consideration

of the learning rate is essential, as SGD may oscillate around the minimum rather than converging directly. However, this could be avoided by decreasing the learning rate, while a too low learning rate could result in an inefficient solver.

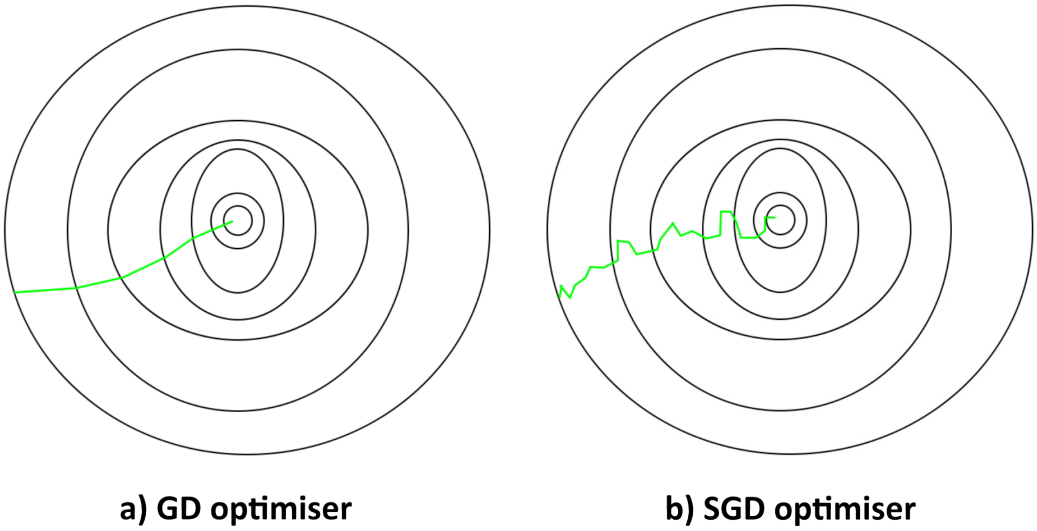



**Figure 3.9:** Principle of GD and SGD optimisation.

The development of a strategy to overcome the problems related to learning rates with respect to SGD has resulted in the Adaptive Moment Estimation, known as Adam (Agarwal, 2023). Like SGD and GD, Adam operates on a first-order derivative basis but differentiates itself by assigning a unique learning rate to each parameter, unlike SGD, which applies a single fixed learning rate universally. Adam incorporates the concept of *momentum* from physics to facilitate convergence. The notion of momentum in gradient descent methods can be compared to an object with mass, sliding across an uneven surface, propelled by velocity and opposed by friction. Provided that there is a consistent descent, the object gains momentum, leading to an increment in step size. When the direction of the steepest direction changes, the sliding object changes direction, but with a delay. The velocity of the object decays at each time step, influenced by a hypothetical friction coefficient (Hassan et al., 2023). The moment in GD methods is used to determine the velocity and can be of first or second order, where first orders are the sum of the gradients and second orders are the squared sum of the gradients (Jiang, 2020). Adam can be viewed as a combination of two other optimisation methods, Adagrad and RMSprop (Kingma & Ba, 2015). Adagrad introduces an adaptive learning rate. It adjusts the learning rate by making smaller updates to parameters associated with frequently occurring features and larger updates for less common features. This method also enables the average gradient to be adjusted for sparse features in the training. The weights are updated considering both squared and root of the gradients, where the gradients are summed from the first iteration. Adagrad uses the second moment and, like other gradient-squared-based methods, helps the optimiser to escape a local saddle point. Adagrad accumulates the gradients for each iteration, which may slow or impede convergence due to the incremental increase in sum with each iteration (Hassan

et al., 2023). RMSprop, while also using the second moment as Adagrad, enhances the convergence by accumulating the gradients from the previous step with a decayed sum of the gradient. This method prioritises the evaluation of more recent gradients, effectively diminishing the influence of initial gradients. RMSprop evaluates the steepness of the function and for solutions with great steepness/gradients, smaller steps are taken. Similarly, low gradients result in larger steps. This strategy is designed to prevent overshooting the minimum with large gradients and to enhance the convergence speed with smaller gradients. Adam uses both the first and second moments (Hassan et al., 2023). Similar to Adagrad and RMSprop, it uses the second moment with previously squared gradients ($v_t$). Additionally, Adam computes an exponential average of the past gradients, representing the first moment ($m_t$), and incorporates a decay mechanism similar to RMSprop, as outlined in Equation 3.17.

$$\begin{aligned} m_{t+1} &= \beta_1 m_t + (1-\beta_1)\nabla_w L(x, y, w_t) \\ v_{t+1} &= \beta_2 v_t + (1-\beta_2)(\nabla_w L(x, y, w_t))^2 \\ \hat{m}_{t+1} &= \frac{m_{t+1}}{1-\beta_1^{t+1}} \\ \hat{v}_{t+1} &= \frac{v_{t+1}}{1-\beta_2^{t+1}} \\ w_{t+1} &= w_t - \epsilon \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}} + \eta} \end{aligned} \tag{3.17}$$

The $\beta$ parameters serve as adjustable hyperparameters that represent decay or friction, generally set at a small value close to one. The parameter $\eta$ acts as a stabilising constant added to the denominator to ensure numerical stability. The $\hat{m}$ and $\hat{v}$ are initial values to avoid having the first derivative as zero, which would result in a very high initial velocity.

Studies shows that Adam generally outperforms SGD with faster convergence, which is one of many reasons making it one of the most preferred optimisers currently. This enhanced convergence rate is facilitated by the ability to adjust parameters on an individual basis. The combination of speed, low memory usage, and straightforward application has made Adam a widely preferred optimisation method for NN (Bock & Weis, 2019).

Two notable variants of Adam include NAdam and AdaMax. NAdam, or Nesterov-accelerated Adaptive Moment Estimation, incorporates Nesterov momentum (NAG) in its momentum calculations. NAG computes the anticipated momentum, allowing the optimiser to predict the next position and adjust accordingly. This technique refines Adam's adaptive learning rate and proves particularly beneficial in navigating noisy regions (Brownlee, 2021). AdaMax automatically adjusts the learning rate for

each individual parameter during training. Unlike Adam, which scales the weight adjustments inversely proportional to the $L^2$ norm, AdaMax applies the infinity norm (Max) of past gradients, $L^\infty$. This approach is especially advantageous for large models characterised by sparse gradients (Das, 2023).

The Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm serves as an alternative to stochastic gradient descent. It is based on the principle of evaluating the inverse of the Hessian matrix, thus a second-order derivative, consequently determining an optimal direction and step size for subsequent iterations (Nocedal & Wright, 1996). Methods based on second-order derivatives, such as Newton's methods, result in significant computational demands. The BFGS approximates the Hessian rather than deriving it and thereby reduces computational demands. This approach classifies it as a quasi-Newton method. An example of pseudo-code for the BFGS algorithm is illustrated in Equation 3.18.

**Pseudo-code for algorithm BFGS**

1. Initialise $x = x_0$, Initialise $B_0$ as identity matrix, and set $k = 0$
   while not converged:
2. Obtain descent direction $d_k = -B_k^{-1}\nabla f(x_k)$
3. Perform line search to find step size, set $\alpha_k = 1$
4. Calculate the step $s_k = d_k \cdot \alpha_k$
5. Update the design $x_{k+1} = x_k + s_k$
   if $|x_{k+1} - x_k| < \text{tol}$ or $|\nabla f(x_{k+1})| < \text{tol}$ : then break
6. Obtain the variation in the gradient $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
7. Update the Hessian approximation $B_{k+1} = B_k + \dfrac{y_k y_k^T}{y_k^T s_k} - \dfrac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}$
8. Increase the iterator $k = k + 1$ and go to step 2

   return $x$

(3.18)

Despite Hessian approximations, iterations within large systems remain computationally intensive. Consequently, the limited memory BFGS (L-BFGS) is an enhancement of BFGS that aims to reduce computational burden. Unlike traditional BFGS, which derives its approximation from the entirety of prior Hessians, L-BFGS utilises only a selection of recent Hessians to evaluate curvature. Furthermore, it simplifies the Hessian into a condensed matrix form. L-BFGS significantly enhances computational efficiency in vast systems with numerous variables. Nevertheless, it operates at a slower pace compared to Stochastic Gradient Descent (SGD). This trade-off in speed is compensated for by a significant increase in accuracy.

### Convergence problems

Even with a good optimisation technique, the training of the network can suffer from convergence problems due to overfitting or underfitting of the model. Overfitting can occur when the model fits the predictions well in accordance to the training data but has difficulty giving good approximations for new input data (MathWorks, n.d.). Underfitting is the opposite and results in poor performance for both the approximation of the training data and good prediction from new data, see Figure 3.10.

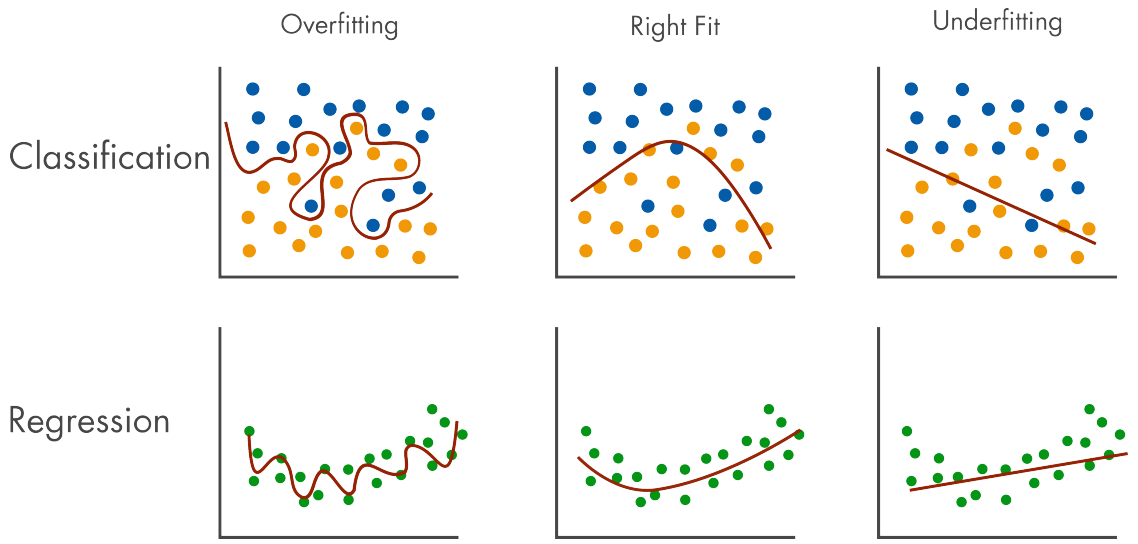



**Figure 3.10:** Overfitting, underfitting and a good fit for classification and regression problems.

Different techniques can be used to avoid the model fitting problem, where some commonly used techniques are validation and regularisation. Validation consists of comparing the error of prediction during the training towards a validation set to verify that the model is improving. Cross-validation is another algorithm that makes predictions on test data on which the network has not been trained and is especially useful when estimating performance with limited test data (Brownlee, 2023). The error can then be computed once for the entire test data. To increase the confidence that the prediction of the models was not a coincidence, a technique called k-fold cross-validation can be used. With k number of subsets, the test data are divided and evaluated. Often between k = 5 - 10, resulting in a five- or ten-fold cross-validation.

Regularisation can be implemented during the training of the model to avoid too complex or too rough by applying a penalty (Murel & Kavlakoglu, 2023). Within machine learning, regularisation techniques are often LASSO (L1 norm), Rigde (L2 norm) or Elastic net. LASSO-regularisation, (Least Absolute Shrinkage and Selection Operator) penalises large coefficient values in the loss function for linear regression by computing the sum of the absolute error and adds this penalty term in the loss function. Feature with less contribution shrinks and therefore, reduce features of less importance in the model. Rigde-regulatisation, are similar to LASSO, but

sums the squares of the error coefficients, and unlike LASSO, are better at handling larger numbers while aiming to shrink the model. Ridge can improve the stability of the model and reduce the risk of collinearity and multicollinearity. Collinearity is a statistical phenomenon in which independent variables become linearly dependent on each other, which yields a prediction of low reliance and makes it difficult to distinguish between importance among variables. The elastic net essentially combines the regularisation of LASSO (L1) and Ridge (L2), by inserting both into the loss function and thereby addressing both the multicollinearity and feature selection.

## Architecture of the neural network

The neural network can be used to solve a wide range of problems, including pattern recognition, prediction and adaptation, regression, classification and optimisation problem, among others. This field is constantly evolving, with ongoing research that introduces new methods and techniques. When it comes to the architecture of the network, it may play an essential role in how well the neural network will perform. The following section below presents some examples of different types of neural network structures, also see Figure 3.11:

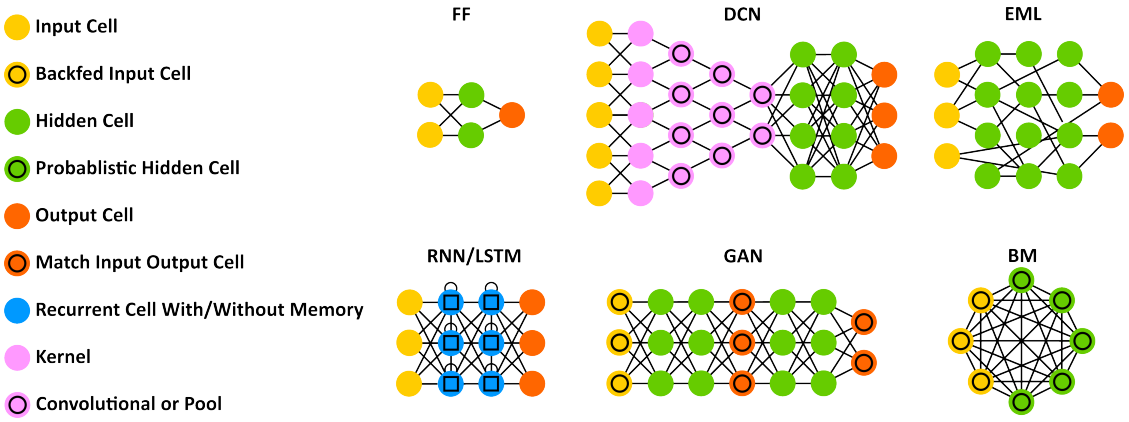



**Figure 3.11:** Visualisation of different neural network architectures.

Feedforward network (FFNN) are a network architecture that only uses forward propagation to process the input data through functions without backpropagation (Markowska-Kaczmar & Kosturek, 2021). An example of such structures is Extreme Learning Machines (ELMs), which assign random values to weights and identify the configuration with minimal error via the least-squares fit method. Unlike traditional approaches, these networks do not use gradient-based optimisation methods but instead use the Moore-Penrose generalised inverse method to determine the weights of the network (Erdem, 2020). The absence of backpropagation makes this type of network fast and can be more effective than networks based on backpropagation for smaller datasets.

Another type of network involves pairing two neural networks with distinct objectives. A notable example is Generative Adversarial Networks (GANs), where the first network generates content, while the second evaluates its quality (Van Veen, 2016). This can be used to create an internal competition inside the network, where for example, the first creates an image, and the next tries to find errors in it.

Some structures combines activation functions in the network to perform different task along the training progress by utilising unique strengths of different functions, applied in for instance Convolutional Neural Networks (CNNs) or Deep Convolutional Neural Networks (DCNNs). CNN uses some layers with activation functions better fitted for classification at the beginning, others for pooling layers to reduce the complexity, and finally a Fully Connected (FC) at the end for the output. This network design excels at handling spatial data, such as images.

Another architectural approach is to allow neurones to train in loops, where the output from one neurone can serve as its input in a subsequent iteration. This enables neurones to maintain a memory state and learn from previous iterations. Recurrent Neural Networks (RNN) are one type of network that uses this technique, which is especially good for interpreting sequential data to predict words in sequence and in language recognition (Ranghar, 2023).

Other architectures aim to incorporate a variety of memory types within the hidden layers, such as Short- and Long-Term Memory Networks (LSTM) by equipping neurones with a memory cell. The memory cell has three gates, input, output, and forget which regulate the flow of information and also allow for the discarding of data that do not contribute to learning. This capability enables the network to learn complex sequences and can even be applied to tasks such as composing music.

Other architecture can be circle shapes like Markov chains (MC or DTMC), Hopfield network (HN) or Boltzmann machines (BM).

### 3.3.2 Physics Informed Neural Network

The Physics-Informed Neural Network (PINN) is an extension of the neural network that has recently grown in popularity. The principle and foundation of PINN closely resemble that of conventional NN, with a distinct difference in the formulation of the loss function (Moseley, 2021). The ordinary NN tries to minimise the cost function based on the difference between predicted and true values, and thereby evaluates patterns. Consequently, neural networks are often adequate for interpolation within the limitations of trained data. However, their ability to extrapolate beyond these bounds is often limited. The physics-informed neural network aims to inform the network regarding the relevant physical relationships for the problem in which the network is trained (Moseley, 2021). PINN can be used to solve an approximation for differential equations (ODE or PDE) in its general form, see Equation 3.19.

$$\begin{aligned} D[u(x,t);\lambda] &= f(x,t) \qquad \forall(x) \in \Omega, \quad \forall(t) \in T \\ B_k[u(x,t)] &= g_k(x,t) \qquad\quad x \in \Gamma_k \subset \partial\Omega, \quad t \in \Gamma_k \subset \partial T \end{aligned} \tag{3.19}$$

Where $D$ denotes the differential operator and $B_k$ with $k = 1, 2, \ldots, n_b$ are the boundary operators for the set of boundary conditions. $u \in \mathbb{R}^d$ are the approximate solution in the Cartesian space for the PDE in d-dimensions for $d \geq 1$. $f(x)$ are the forcing function and $x$ is a d-dimensional input vector in the $\Omega$ domain. $\Omega$ denotes the spatial domain, T denotes the temporal domain where $\partial\Omega$ and $\partial T$ denote the boundary of space and time, where $\Gamma_k$ are the k-boundary of the domain. $\lambda$ are an optional model parameter of the differential operator.

This general form of equation can describe many types of physical systems regarding time and space. For a structural engineering problem, a common equation is the dimensional Euler-Bernoulli equation, with the following Equation, see 3.20, which includes Dirichlet and Neumann boundary conditions for a fixed-free cantilever beam.

$$\frac{\partial^2 u}{\partial x^2} \cdot \left( EI \cdot \frac{\partial^2 u}{\partial x^2} \right) = q(x) \tag{3.20}$$

**Fixed end** ($at\ x = 0$)**:** $\quad u\Big|_{x=0} = 0 \quad ; \quad \dfrac{\partial u}{\partial x}\Big|_{x=0} = 0$

**Free end** ($at\ x = L$)**:** $\quad \dfrac{\partial^2 u}{\partial x^2}\Big|_{x=L} = 0 \quad ; \quad \dfrac{\partial^3 u}{\partial x^3}\Big|_{x=L} = 0$

By introducing the physical relationship to the neural network, the aim is to produce predictions closer to the range within the physical boundaries. This knowledge of physical behaviour can also enhance the network's ability to extrapolate beyond the range covered by the training data.

Physics-informed neural network is achieved by extending the loss function by the underlying physics behind studied behaviour. The result is, likewise for ordinary NN, an approximation towards measured data. However, the integration of physical principles not only enables more precise predictions and faster convergence but also improves the predictions beyond the constraints of the training data. This characteristic of PINNs results in improved predictions, even in sparse data scenarios, while simultaneously reducing computational demands (H. Chen et al., 2022). An general visualisation of the workflow for a PINN framework with beam application can be seen in Figure 3.12.

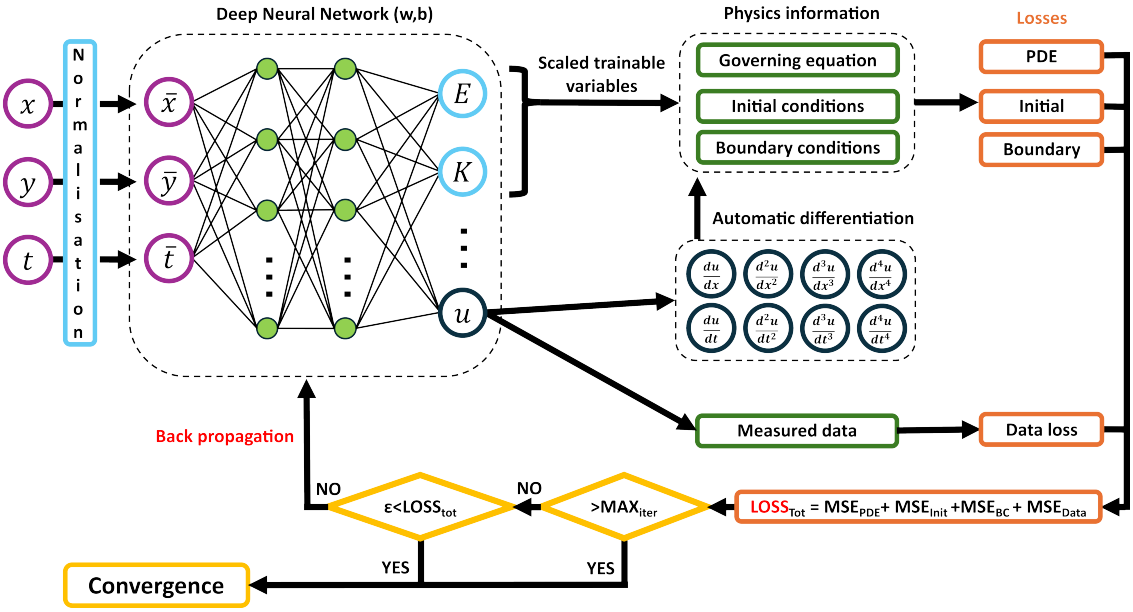



**Figure 3.12:** Visualisation of PINN workflow.

Most often, the loss function in a neural network focusses solely on the discrepancy between the approximated and observed data, often referred to as data loss. This discrepancy can be computed through either the mean absolute error (MAE or L1), the mean absolute percentage error (MAPE) or the mean squared error (MSE or L2) as seen in Equation 3.21. The loss function can be extended by a physics loss, see Equation 3.22, and a boundary loss, see Equation 3.23, resulting in a combined total loss function as seen in 3.24.

$$L_{data} = \sum \left(u_{true} - u_{pred}\right)^2 \tag{3.21}$$

$$L_{physics} = \sum \left(EI \cdot \frac{\partial^4 u_{pred}}{\partial x^4} - q(x)\right)^2 \tag{3.22}$$

$$L_{boundary} = \sum \left(u_{pred}\Big|_{x=0} + \frac{\partial u_{pred}}{\partial u_{pred}}\Big|_{x=0} + \frac{\partial^2 u_{pred}}{\partial u_{pred}^2}\Big|_{x=L} + \frac{\partial^3 u_{pred}}{\partial u_{pred}^3}\Big|_{x=L}\right)^2 \tag{3.23}$$

$$L_{tot} = \sum \left(L_{data} + L_{physics} + L_{boundary}\right) \tag{3.24}$$

For this example, the physics loss is exemplified through Euler-Bernoulli's beam theory. The physics loss can be modelled through any appropriate partial or ordinary differential equations (PDE or ODE) that correspond to the analysed behaviour. Note that the loss function for PINN does not explicitly need the data loss component. Therefore, test data can be omitted, allowing the evaluation to focus solely on the physics loss. The loss for the PDE or ODE aims at minimising the residual of the equation by rearranging all components to one side, resulting in an equation of the form that equals zero.

For every derivative of the PDE or ODE, known boundary conditions or initial conditions can be specified for the respective derivative to guarantee a unique solution. By defining physical boundaries through partial differential equations, the training time is reduced compared to NN (H. Chen et al., 2022). The complexity and nature of the PDEs used influence both the training performance and the accuracy of the output. Similarly to humans, PINN showed a decrease in performance when provided with more intricate physics. Thus, it could be beneficial to reduce the complexity of given physics to its most reduced form.

In scenarios involving large multidimensional problems or complex physics, PINNs have been observed to significantly decrease accuracy and efficiency (H. Chen et al., 2022). In certain instances, PINNs have been outperformed by traditional computational methods such as the Finite Element Method (FEM) and the Finite Difference Method (FDM) in terms of both accuracy and computational efficiency. To overcome this, a variety of modifications and alternative approaches have been introduced to make PINN viable.

One such variation is the Finite Element Model Neural Network Hybrid (FEM-NN), which makes use of the already established benefits of the Finite Element Method to enhance the performance of PINNs (Meethal et al., 2023). FEM-NN makes use of an FEM solver together with an FEM expression in the loss function to get the physical information to aid the network learn complex structures on a larger scale.

Another approach aimed at enhancing PINNs is the sequence-to-sequence training scheme. This method involves dividing a problem into several subdomains to train the network on one subdomain at a time, a process commonly known as domain decomposition. Using the earlier example of the Euler-Bernoulli beam, PINNs can train on sublengths of the beam, denoted as $\Delta$L. Once successfully trained and convergence is reached for one subdomain, the next subdomain is trained with previous results as boundaries. Sequence-to-sequence training not only shows drastic improvements in computational time, but also shows better accuracy towards final solutions.

Although domain decomposition methods frequently outperform traditional PINNs, they may encounter difficulties in accurately defining boundaries. Typically, the loss function only weakly enforces boundary constraints, and thus strictly only considering the very interfaces. Consequently, discontinuities in solutions might appear over interfaces of subdomains, resulting in loss of accuracy.

To address challenges related to scaling of problems, handling multidimensional data and ensuring continuity across interfaces, the Finite Basis Physics-Informed Neural Network (FBPINN) was introduced (Moseley et al., 2023). It follows the principle of domain decomposition but introduces domain overlapping. This strategy involves intentionally overlapping the training domains to ensure continuity between the interfaces. An illustrative scheme of the training strategy is shown in Figure 3.13. Thick lines denote domains under active training, thin lines indicate domains awaiting training, and dotted lines represent already trained domains. The coloured areas highlight the overlapping training domains, visually demonstrating the intersection between models.

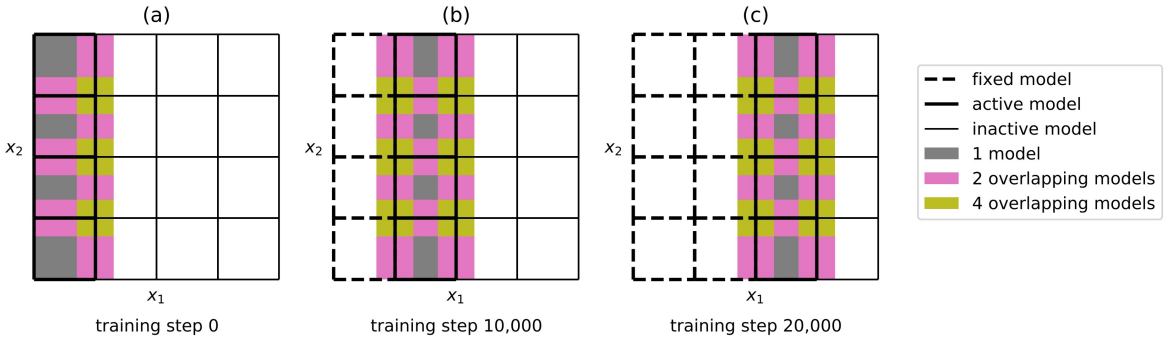



**Figure 3.13:** Visualisation of FBPINN traning procedure (Moseley et al., 2023).

FBPINN does not only produce more accurate results compared to normal domain decomposition, it also does so along with faster convergence and trough less computational demand (Moseley et al., 2023). Additionally, FBPINN offers flexible training schedules, enhancing both convergence and accuracy. Through flexible training, complete control of the training order and origin is gained. Through this adaptability, complete control over the order and starting point of training is achieved, enabling solutions to originate from and be fully informed of boundary conditions. This contrasts ordinary PINN which may fixate on solutions, located away from the boundaries, that are inconsistent of the boundaries. This could occur as PINN tries to fit solutions within the domain before truly learning the boundary constraints. FBPINNs flexible learning can thus be utilised to force solutions to originate from constraints and then proceed stepwise, thereby ensuring consistency.

A fundamental limitation of PINN is the narrow application of trained network. Although they are capable of extrapolating results beyond their training boundaries, their utility is confined to the physics on which they have been trained on. Thus, a trained PINN becomes restricted to solving only problems of similar type. This could be problematic as each network needs to be specifically built and trained for each specific application. Furthermore, the computational resources and time required to train PINN-based methods often exceed those required for conventional FEM applications. As a result, PINNs are not currently a viable replacement for FEM software. However, the utilisation and benefits of PINN can be seen for cases where identical structures need to be evaluated multiple times. Once intensive training is performed, highly effective solutions could be established for continued evaluation. In particular, applications in long-term monitoring could prove to be more efficiently reevaluated through PINN.

### 3.3.3 Large Language Models & ChatGPT

During recent advances in artificial intelligence, Large Language Models (LLMs) have gained significant attention and achieved remarkable success, with ChatGPT by OpenAI being one of the more noticeable. With an release date of 30$^{th}$ of November 2022, a user count of more than a million was reached during the initial 5 days, with current numbers of users exceeding 1.6 billion as of January 2024 (Franco, 2024).
The broad utility of ChatGPT comes from the extensive and diverse dataset on which it was trained, allowing the model to successfully execute a wide range of tasks. A notable advantage introduced with the development of OpenAI's model is its capability to adhere to specific instructions, allowing for highly customised usage.

The most basic large language models generally tries to predict subsequent words based on the input sequence provided (Fulford & Ng, n.d.-b).
Thereby, LLMs generate the most probable word sequence based on the training data on which it is built upon. However, this might not always lead to the intended outcome, as the most probable words may deviate from the desired response. For instance, an input question of "What is the capital of France?" could be answered by the LLM through further questions such as "What is France's population?". Although this might not be the desired response to the question, it could be a plausible response since the LLM might have been trained on geographical exams, consequently seeing a pattern of such sequencing questions. ChatGPT is considered an instruction tuned LLM which means that the network is trained to consider user specific instructions. Therefore, unlike simpler LLMs, ChatGPT can produce results that are more closely aligned with the user's expectations.

#### Architecture of OpenAIs ChatGPT

The basis of ChatGPT is an extension of the ordinary neural network architecture presented in Chapter 3.3.1. The key distinctions lie in how the model processes the input text and determines the significance of various parts of the input. This differentiation is achieved through a series of transformer blocks, giving rise to what is known as the transformer architecture. A more detailed description of the whole procedure is presented below in general terms, discussing how an input is transformed, interpreted and results in a response.

Initially, a textual input, often referred to as a prompt, is provided to the model. Thereby, the text is segmented into tokens containing parts and subparts of prompts differentiating more or less common followed characters. Common terms like "new" appear frequently enough to be treated as a single token. Less common terms, such as "lollipops," are broken down into several tokens, each representing a more frequently occurring sequence of characters, e.g., "l-oll-ip-ops" (Fulford & Ng, n.d.-a).
Next, each token is converted into numerical values that will be processed by the neural network. This is performed via a process refereed to as embedding. This conversion, known as embedding, transforms each specific sequence of tokens into a high-dimensional numeric vector. This vector captures the semantic and contextual context (Hutanu, 2023).

Even though the text prompt has been translated to its numeric representation, the determination of more relevant parts from the prompt must be decided and incorporated. This process is done through a series of transformer blocks. For each transformer block, there are two main processes, the self-attention mechanism and the subsequent forward neural network. Initially, the self-attention mechanism assigns weights to the vectorised embedded input. As a result, tokens considered crucial in achieving the desired output are assigned higher weights, increasing the chances of appropriate responses. Moreover, the weighted data enter the forward neural network. This step aligns more closely with the ordinary principles of NN. The output from self-attention is linearly transformed into a higher-dimensional space. This is an important step to ensure more diverse information extraction from embedded data, allowing for more complex pattern learning. After such a transformation, the data pass through a nonlinear activation function, often of ReLU type (Harsoor, 2023). Nonlinearity further allows for much more intricate relations to be learnt. After which, the data goes through a final linear transformation, transforming the data back to the original dimensional space. The output from the very last transformer block, and thus the very last forward neural network, finally passes a Softmax function. This function calculates a probability score across the vocabulary, thereby determining the likelihood of each possible sentence output.

**Features and applications through OpenAI**

OpenAI offers a variety of functionalities and features for different applications, each with different levels of customisation, trainability and complexity. Currently, OpenAI provides features such as *ChatGPT*,*API*, *AI assistants* and *finetuning* (OpenAI, n.d.-b).

One of the most well known features is the ChatGPT, which is a web browser AI. As of when this thesis was conducted, ChatGPT operates on either the free ChatGPT 3.5 engine or the more sophisticated ChatGPT 4, both of which instructed quires can be answered with conversational memory. Regardless of paid subscription or not, ChatGPT has limitations on input requests and memory (OpenAI, n.d.-c).
The input request limitation is defined by either the number of requests per minute or the number of tokens per minute, ranging from 3,000 to 10,000 requests or 40,000 to 2,000,000 tokens, depending on the engine and subscription plan. Regarding memory, ChatGPT uses a principled referred to as *context window*. This mechanism operates within a confined window size, keeping current interactions. Following each prompt and response, a summary of the most important interaction elements is stored within the context window. Exceeding the window's capacity leads to the oldest memories being discarded. However, with a current window size ranging from 4,096 to 128,000 tokens, depending on the selected model, a rather large conversational memory is provided (OpenAI, n.d.-a).
This capability allows ChatGPT not only to comprehend and execute complex instructions, providing detailed and sophisticated responses, but also to effectively revisit earlier parts of the conversation.

For development or application purposes, the web browser driven ChatGPT may lack convenience and offer limited customisability beyond prompt instructions. To address this, the API key feature is provided. API, standing or Application Programming Interface, enables the use of ChatGPT's capabilities through coding, offering a much higher degree of freedom. Although the API operates on the same ChatGPT engine and responds similarly to prompts, it offers wider options and applications. Regarding customisability, randomness in responses can be tuned, rules can be stated to the AI without being explicitly mentioned in prompts, tokens can be strictly limited for both prompts and responses and much more. However, a notable difference from web browser-driven ChatGPT is the absence of memory. Each prompt initiates a new unlabelled thread that is completely independent of the others. Thus, users must construct a memory function that is appropriate to its intended applications, allowing for more tailored memory management.

As compromise to the methods of web driven ChatGPT and API, the AI assistant is another feature. It combines the simplicity, completeness and flexibility of conventional ChatGPT with the customisability and code capabilities of the API. Assistants can be created either through the web or via code. Once created, specified rules can be stated and enforced throughout the use of the assistant, eliminating the need for repetitive prompt instructions. With a specific assistant ID assigned, each interaction and conversation can be both kept and stored, thus memory is pre-integrated. Additionally, pre-made add-ons are available, enhancing the AI's capabilities, such as interpreting and generating code or processing and understanding information from uploaded files.

If neither of above-mentioned features yield satisfying results or quality in responses, the method of fine tuning is recommended. Fine-tuning allows the AI to be trained on specific examples of input and expected output, tailoring the AI to particular tasks and outcomes. This differentiates compared to all other methods, as ChatGPT is pre-trained, thus, do not learn even if concepts are explicitly explained. Consequently, fine-tuning is the only customer-focused method to train the AI. Through this method, a better response quality could be expected with more concise prompts. Therefore, cost savings could be an outcome as each prompt costs less tokens. The method of training corresponds to a fabricated conversation in which the desired responses and prompts are stated. Through a set of such conversations, the AI learns patterns of desired responses and adapts thereafter.

### Prompt engineering

Efficient utilisation of large language models is significantly dependent on prompt engineering and is crucial to eliciting the best responses from these models (Machine Learning Mastery, 2023). Due to the prevailing popularity of ChatGPT, the fields of prompt engineering are fairly new however its significance cannot be understated. Given that ChatGPT operates as an instruction-driven AI, providing concise, clear, and comprehensive instructions is critical to obtaining the desired results. Employing straightforward techniques and strategies in prompt engineering can enhance response speed and quality, optimising the model's performance.

To ensure quality responses, it is important to provide the AI with sufficient context and instructions (Ekin, 2023). The prompts should be explicit enough to provide a clear description of the desired intention, outcome, and limitations while also avoiding ambiguity. Therefore, even though concise prompts are beneficial, over-simplification should be avoided. A promising approach to achieving reliable and high-quality responses is to divide the prompt into four distinct parts containing *instructions*, *context*, *input* and *output* (Giray, 2023). Through this method, the prompt should contain clearly stated instructions outlining the expected task to perform, followed by an explanation of the problem's context. Furthermore, it may also be relevant to clarify the structure of the input data followed by the desired output structure.

Regardless of the prompt's structure, ChatGPT's response quality seems to vary with different prompts. A plausible reason for this variation is overcomplicated or contradictory instructions, which hinder the generation of robust and quality responses (Fulford & Ng, n.d.-b). This can be analogous to the humans and our method of reasoning. Longer and more complex tasks often become more manageable when broken down into sequences of subtasks. Similarly, ChatGPT may exhibit improved response quality when dealing with subtask reasoning, often achieved through prompt chaining. By chaining subsequent prompts, the AI is presented with each subtask one at a time, thereby dividing individual and comprehensive prompts into multiple interlinked prompts.

Even with clear, well-structured, and chained prompts, ChatGPT might still yield responses that are overly simplistic or naive. (Fulford & Ng, n.d.-b). ChatGPT may generate responses that are incorrect or even illogical, often due to a hasty response formulation, which may lack logical reasoning. The human analogy is to be asked to answer a specific question quickly. Being rushed to answer a question can compromise the analytical depth of the response, thus risking accuracy in gain of speed. Similarly, the tendency towards hasty, illogical responses can be mitigated by adopting a more thorough and analytical approach, achievable through chain-of-thought reasoning. This method involves promoting deeper analytical thinking by outlining clear subgoals. For instance, the AI can be explicitly instructed to provide specific subresponses, encouraging stepwise reasoning. By enforcing more effort and time in the reasoning process, improved analytical reasoning is a likely outcome, a finding supported by research.

### Limitations of OpenAI

With the significant success of OpenAI's models, unexpected behaviours and limitations have been identified. Among these are inconsistency, violation of instructions, and numerical difficulties. Recognising these limitations is crucial to effectively avoiding and managing them. This awareness is also important in the context of implementing and automating AI, as automation routines can be disrupted or break.

A commonly reported phenomenon is the fluctuation in accuracy and success of ChatGPT 3.5 and 4 over time (L. Chen et al., 2023) . Recent research has observed significant variances in performance across a span of merely three months. Both ChatGPT AI engines exhibited mixed results, with performances for various tasks both improving and declining, pointing to unstable performance over time. This inconsistency was recorded across a range of tasks and applications, which included adherence to instructions, the ability to induce chain-of-thought reasoning, and the overall success rate of the tasks. The causes behind these substantial performance shifts over time remain unclear, further complicating efforts to address and overcome such unpredictable behaviour.

The study also emphasised the critical issue of failing to adhere to instructions (L. Chen et al., 2023). In the process of evaluating ChatGPT's performance on extensive sets of test prompts, the ChatGPT API was utilised to facilitate the collection of large amounts of data more efficiently. However, adherence to specified instructions regarding response outputs and formatting was crucial for automated retrieval of outputs. Thus, any deviation from the formatting instructions led to code failures and loss of responses.

Furthermore, studies have highlighted the numerical difficulties of ChatGPT. It has been observed that ChatGPT not only struggles with mathematical reasoning, but also tends to produce numerical values based on favoured numbers (Azaria, 2022). An investigation into the response to the prompt "How much is pi * 4.26 with 200 digits?" revealed not only highly inaccurate solutions, often exhibiting errors from the first decimal place, but also suggested a probabilistic rather than a numerical analytical approach. Among the decimals provided, there was a noticeable preference for certain numbers and sequences. The digit 7 appeared more frequently, whereas the digit 1 was the least common. This finding aligns with a survey that asked humans to choose their favourite number between 1 and 9, where 7 was the most favoured and 1 the least. Additionally, the 200 decimals included patterns like "123456789". Such tendencies indicate that ChatGPT lacks the capacity to perform numerical analysis and instead generates outputs based on patterns and more frequently occurring numerical values.

## 3.4 FEM model basis

The Finite Element Method (FEM) is a widely adopted approach for addressing numerous structural engineering challenges. It is presumed that the reader possesses a foundational understanding of this method, which is succinctly outlined below.

The approximate solution to the problem is derived through three steps: Preprocessing, FE calculations and post-processing. During pre-processing, decisions are made about input values for material parameters, extent of the model, boundary conditions, internal constraints, interactions, level of details in the model and loads applied to the structure. The geometry is converted into a mesh with discrete sizes and element types, *finite elements*. The FE calculations then solve the equation system established during preprocessing using an FE solver. The post-processing stage involves analysing the resulting outputs for visualisation, further analysis, and comparison. Before the model can be utilised in Finite Element Analysis (FEA) to predict behaviours, the approximate FE model should be validated, for instance, through hand calculations or comparison with known measured values.

Finite Element Analysis (FEA) can be executed in both linear and nonlinear formats. Linear analysis effectively captures complex geometries and 3D effects at the ultimate limit state (ULS) for conditions both before and after cracking. In linear FEA, the stress distribution does not correspond to actual sectional moments and forces, potentially resulting in unrealistic stress concentrations and large stresses due to restrained deformations. Nonlinear analysis, on the other hand, offers a more precise representation of nonlinear material properties, geometrical conditions, and boundary conditions. This enables the analysis to more accurately model the serviceability limit state (SLS) for cracked structures. Nonlinear FEA ensures that response and structural behaviour align more closely with actual conditions, allowing for more accurate predictions of capacities, responses, and stress distributions.

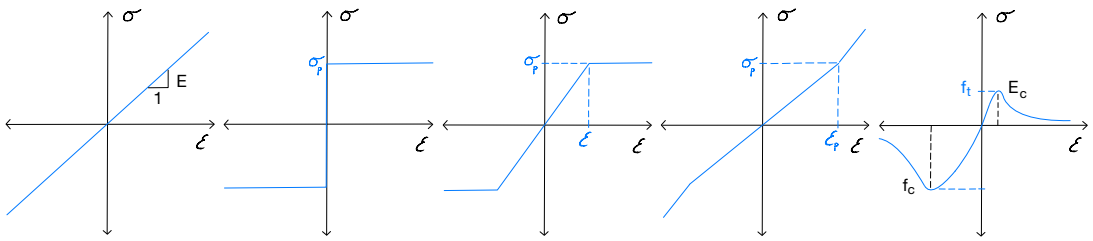



**Figure 3.14:** Stress-strain relationship, linear elastic, perfectly plastic, elastic-perfectly plastic, plasticity hardening and concrete softening

The elements used in Finite Element Analysis (FEA) are classified into three distinct types: One-dimensional line elements, two-dimensional surface elements, and three-dimensional solid elements. In Additional, these elements are categorised into one of three principal groups: continuum (subsurface), structural, and special-purpose elements. These elements may be either first-order, characterised by a linear displace-

ment distribution and constant stress distribution, or second-order, which features a second-order displacement distribution and a linear stress distribution. For a visual representation, see Figure 3.15. Response calculations within FEA are conducted at the element nodes and via integration schemes between these nodes.

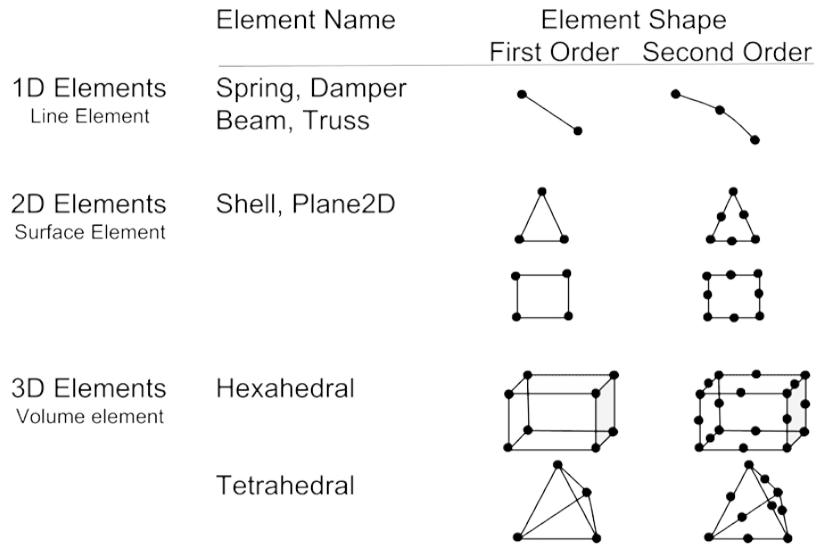



**Figure 3.15:** Example of element types of first and second order

Continuum elements possess only translational degrees of freedom (DOFs) at each node, making them suitable for examining stresses and strains. These elements can function as 1D elements, trusses, or cables. For 2D applications, they are commonly used in analyses involving plane stress and plane strain (or axisymmetric circular) scenarios. In the case of 3D applications, elements may assume shapes such as bricks, prisms, or tetrahedrons. For the purpose of performing numerical integration for calculations between nodes, methods such as the Gauss or Clenshaw-Curtis schemes can be utilised.

Structural elements are characterised by having both translational and rotational degrees of freedom (DOFs) at each node, and they can be modelled as either one-dimensional or two-dimensional elements. The simplest line elements (1D), such as truss or bar elements, only have axial DOFs ($T_x$, $T_y$, $T_z$) (CAE Assistant, 2022). More complex line elements (1D), such as beams, have three translational and three rotational DOFs ($T_x$, $T_y$, $T_z$ and $R_x$, $R_y$, $R_z$) at each node. Two-dimensional surface elements can be modelled as simpler membrane (2T and 1R) and plate (1T and 2R) elements or as more complex shell elements. The shell elements, which combine the characteristics of the membrane and the plate elements, have three translational and three rotational DOFs ($T_x$, $T_y$, $T_z$ and $R_x$, $R_y$, $R_z$). The inclusion of these additional DOFs enables analysis of sectional forces and moments. The necessity for a greater number of integration points for calculations between nodes leads to the use of analytical methods such as Simpson's rule or Newton-Cotes for analysis.

Furthermore, special-purpose elements are utilised to model specific scenarios within a structure, such as discontinuities or interactions with components like springs, interfaces, and gaps, or contact elements.

### 3.4.1 Beam element

Beams are line elements that possess six degrees of freedom (DOFs), three translational and three rotational, and are not only restricted to capture axial forces but can also simulate bending failure (Wikipedia, 2023). This versatility allows beam elements to address more complex engineering challenges, a capability that simpler elements, such as bar elements lack. The derivations in most common Finite Element Analysis (FEA) use the Timoshenko beam theory to account for the shear force, unlike the Euler-Bernoulli beam theory (Baba, 2020). However, beam elements cannot accurately describe failures due to shear stress. Furthermore, having more degrees of freedom in each node renders them more computationally intensive. Nevertheless, they are still less computationally demanding than solid continuum elements because the integration occurs between points, rather than within a parent domain through Gauss point integration.

### 3.4.2 Shell element

Shells, as two-dimensional surface elements, are capable of caring loads in any direction and possess six degrees of freedom (three translational and three rotational). This enables them to not only sustain axial force but also accommodate in-plane shear, bending, and twisting to some extent. This versatility allows for the modelling of curved geometries, unlike beam elements (Neto et al., 2015). Shell elements follow the classical plate theory with derivations based on the kinematic assumptions of Naghdi and Reissner-Mindlin for moderately thick plates (Chinosi et al., 1998). Similarly to beam elements, shell elements also face limitations in accurately representing out-of-plane shear failure (Charlotte et al., 2023).

### 3.4.3 Solid element

Solid elements can be represented as both two- and three-dimensional, where 2D triangles are the simplest, followed by 2D rectangular and quadrilateral elements (Neto et al., 2015). Solid 2D elements have two translational DOFs in each node ($T_x$ and $T_y$) while 3D elements have three ($T_x$, $T_y$ and $T_z$). From displacement, strain and stresses are derived using isoparametetric mapping and shape functions calculated according to Euler-Lagrange equations (Ferreira & Fantuzzi, 2009). Solid elements can be used to analyse bending and shear, where bending generally requires more than one layer and second-order elements (Broo et al., 2008). Quadrilateral with non-parallel edges could be problematic, since the Gaussian integration scheme requires mapping of the general quadrilateral element into a normalised rectangular geometry (Neto et al., 2015).

### 3.4.4 Convergence

For iterative and nonlinear problems, the Finite Element Method (FEM) does not always converge to an exact solution, making it necessary to consider the convergence of the result. There are various aspects of convergence and criteria to determine when convergence has been achieved.

Convergence can be categorised into three groups: Completeness, Compatibility, and Stability (Felippa, 2001). Completeness refers to the finite element's ability to accurately represent the correct behaviour of the system being modelled. Compatibility concerns the ability of the shape function to accurately capture the correct displacement as the domain deforms. Finally, stability ensures that the system can reach a unique solution without zero-energy kinetic modes (excluding those of the rigid body) within the elements and prevents excessive deformation of the elements.

If both completeness and compatibility are achieved, the solution is considered consistent (Felippa, 2001). According to the Lax-Wendroff theorem, if consistency and stability have been reached, the solution is considered to have converged.

When addressing completeness, the interpretation of the geometry into an appropriate mesh size is an important factor and can be categorised into three types of mesh refinement methods: the H-method, the P-method, and the R-method (Kittur & Huston, 1990). The H-method relies on the principle of refining the mesh by increasing the number of elements, resulting in a smaller and finer mesh size through adjustments in the characteristic length of the lines. The P-method involves enhancing the polynomial order for interpolation, typically to the second order, to improve the solution accuracy. Meanwhile, the R-method focusses on redistributing the position of the nodes to balance the potential energy within the domain. Another important aspect of discretising the domain into a mesh is the presence of corners or sharp edges, either internal or external. These characteristics can lead to singularities, where stresses may theoretically approach infinity (A. Harish, 2023).

The compatibility conditions must be fulfilled to ensure a coherent relationship between elements within the deformed domain (Patnaik et al., 1991). It is crucial that the deflection and slope maintain unique values between neighbouring elements to accurately match the displacement, preventing gaps that could multiply and potentially absorb or release spurious energy (Venkatesh et al., 2018). When altering the mesh size or element type within the domain, special attention is required to maintain this compatibility.

The stability is ensured by maintaining equilibrium in the governing equations and preserving energy balance between the mechanical power of external and internal forces for each time- or load-step in the calculations (Oden & Fost, 1973). This equilibrium ensures that the system response remains consistent and predictable under varying conditions.

One aspect of convergence issues that can arise involves various locking phenomena, which are common when using low-order elements and result in excessively stiff deformation behaviour (A. B. Harish & Matikainen, 2023). This leads to an extensive amount of strain energy being stored within the element.

There are several types of locking phenomena, starting with shear locking. This occurs with low-order elements, classical Lagrange polynomial shapes that fail to accurately depict deformation during bending (Baier-Saip et al., 2020). Ideally, when an element bends, the line between nodes should curve to accurately reflect the deformation. However, because of the low order of the element, it is unable to accurately capture the correct curvature between the nodes.

Continuing, volumetric locking is another occurring phenomena seen in FEM modelling. This phenomenon can manifest on both first- and second-order elements during full integration, particularly when the material is nearly incompressible (with Poisson's ratio close to 0.5) (Coombs et al., 2018). When integration occurs on the element, an unrealistic relationship between stress and strain develops, due to the constraints each element imposes on the domain.

Lastly, hourglassing is a further present locking phenomena which could arise. This is a form of indeterminacy that can occur with 8-noded elements undergoing reduced integration, leading to overly flexible behaviour when stress and strain integration is focused solely at the midpoint (Bower, 2008). This situation results in zero strain, and consequently zero stress at the centre, allowing motion that can increase indefinitely with zero energy expenditure.

# 4
# Methodology of model updating

The vast variety of presented updating principle have all previously been studied, however seldomly in relation to each other, especially for structural applications. For that reason, an extensive study was intended to evaluate, test and develop methodologies, to then evaluate their strengths and weaknesses. Therefore, this chapter is intended to describe the methodology of testing, details of employed principles and important choices made.

In chronological order, this chapter addresses the base problem used, which ensures equal tests are evaluated for each model updating principle. Following, each category of updating will be presented, of which conventional, neural network and AI methods are addressed. Each category will describe their corresponding studies performed, important factors and all needed prerequisites for following results.

## 4.1 Method of benchmarking

Before deciding on an appropriate updating routine, different methods and principles must be compared using fair testing grounds. Optimisation methods were compared by evaluating test cases, with the aim of avoiding excessive complexity while ensuring model scalability. The selected and performed evaluations were limited to two-dimensional problems of similar size and considered only static loading, including one statically determined case and two undetermined cases. Three test cases with varying complexity and numbers of free variables were conducted to assess how well different methods performed with increased complexity. Scalability was assessed through a direct increase in complexity rather than an increase in the size of the model.

The chosen cases for benchmarking are shown below in Figure 4.1. Starting with Case A, a statically determined problem which represents the simplest form of a cantilever beam with a point load at the end. In the subsequent case, Case B, the cantilever beam is subjected to a uniformly distributed load and with a spring support at the end. This setup includes a new parameter, the spring, which slightly affects the complexity and makes the problem statically undetermined. Test Case B is more likely to produce a unique solution than Case A, which produces an infinite number of solutions for different combinations of load and stiffness, as the two variables are linearly dependent. Finally, Case C increases the complexity by adding an additional beam, thus creating two separate elements. These elements are

interlinked by a rotational spring and receive additional spring support for the new segment. Like Case B, this last test case is statically undetermined. The complexity increases not only due to newly introduced parameters but also due to the presence of a rotational discontinuity at the interface between the elements.

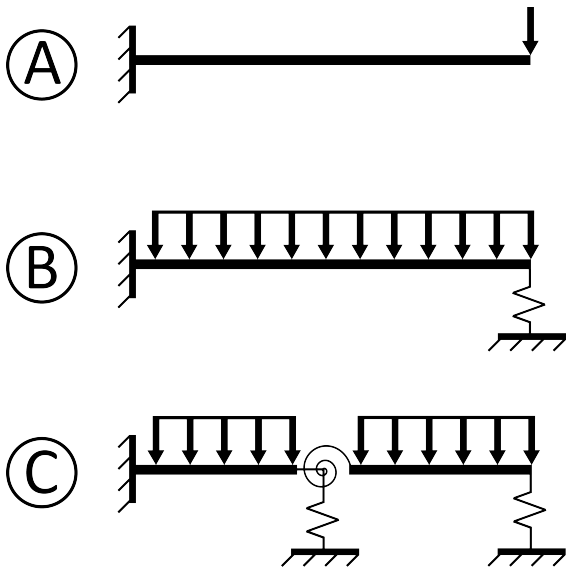



**Figure 4.1:** Test cases for benchmarks with varying complexity.

In benchmarking, various factors, including optimiser options, setups, and general hyperparameters, can greatly influence the outcome. It is crucial to acknowledge the likelihood that ideal settings were not utilised or discovered. The emphasis was on the relative performance of the different cases and the effects of various methods on convergence in each case.

For each case, the desired outcome was optimisation toward a "correct" solution. All the "correct" solutions for the chosen set of parameters were constructed using beam elements in the finite-element-based program Brigade2023, an extension of Abaqus. The choice of beam elements was based on the objective of obtaining as efficient a model as possible. Hence, computational effort was kept low for the models, resulting in differences caused almost solely by the different methods. For each test case, an interval of variables was provided within a range of typical values. Initial guess values within boundaries have also been provided for any optimiser that requires them.

The methods tested and evaluated consist of conventional optimising principles, neural networks, and OpenAI. A consistent methodology was used for each category of methods. Cases A, B, and C will be evaluated chronologically for each category. Additionally, for each case, at least one additional "side-track" study was performed that examines additional parameters, settings, or choices of interest that could affect the outcome.

### 4.1.1 Conventional methods

Primarily, the more common and conventional methods was evaluated, including the Nelder-Mead simplex, pattern search, genetic algorithm, and particle swarm optimisation. Although these methods have been tested and compared in numerous papers previously, a brief comparison was performed. This includes evaluating their convergence, scalability, and performance in response to changes in complexity. For this, accuracy in optimising was evaluated and related to achieved runtime (RT).

For all forthcoming conventional test cases, identical objective functions were used for cost evaluation. Therefore, a potential performance change was not attributed to differences in the cost function. The objective functions used considered deflections, sectional forces, and sectional moments. The addition of sectional forces and moments was chosen to avoid linearly dependent solutions. The objective function used can be seen in Equation 4.1. $U$, $SF$ and $SM$ correspond to deflections, sectional forces, and sectional moments, respectively. The index *true* denotes the "true" values obtained through the FE solution. Lastly, each quantity of $U$, $SF$ and $SM$ has been normalised through min-max normalisation against the corresponding true values, see Equation 4.2. Thus, this equation equalises any significant differences in units or size. By this equation, the lowest value of zero is reached only for a perfect solution.

$$OBJ = \frac{1}{n}\sum_{i=1}^{n}\left[(U_i - U_{i,true})^2 + (SF_i - SF_{i,true})^2 + (SM_i - SM_{i,true})^2\right] \tag{4.1}$$

$$X_{Scaled} = \frac{X - X_{min,true}}{X_{max,true} - X_{min,true}} \tag{4.2}$$

The optimising procedure was performed through Matlab, via the toolbox "Genetic Algorithm and Direct Search" (GADS). A MATLAB script was established that interacts with Brigade2023, changing and updating the parameters during the optimisation. For each case, settings such as tolerance were set to "auto", allowing each optimiser to run on recommended settings. For methods that require a population size, such as the genetic algorithm and particle swarm optimisation, three different population sizes are presented: a lower bound, an upper bound, and a compromise size, to better visualise the effect of such settings.

To conduct a fair benchmark, it was essential to evaluate similar convergence or stopping criteria throughout the study. The chosen approach allowed the Nelder-Mead simplex method to run until achieving a satisfactory tolerance level of $1 \times 10^{-4}$, with the runtime of all additional optimisers limited to this recorded duration. Each model was then stopped by either their representative convergence criterion or maximum runtime. It is important to note that the runtime limit was checked after each iteration. Consequently, if the limit has not been reached at initiation, the final runtime may exceed the time limit with the last iteration.

### Case A

Firstly, Case A, as the simplest problem, is a single-variable problem dependent solely on Young's modulus (E-modulus). The chosen "correct" value for the E-modulus is set to 35.6 GPa, based on the recorded deflection data for this variable. For the optimisers, an interval range of 30-40 GPa was set for the E-module. For the Nelder-Mead simplex and pattern search, an initial guess of 34 GPa was provided.

For the initial test Case A, a main study of performance was conducted, which involved evaluating the best achieved accuracy within the duration of the Nelder-Mead simplex recorded runtime. Furthermore, as a "side-track" study, repeatability was evaluated by assessing the variation for each optimiser over 10 identical runs. This study aimed to evaluate the robustness and reliability of the methods, especially considering the more stochastic nature of the genetic algorithm and particle swarm methods.

### Case B

Secondly, Test Case B increases complexity by introducing a distributed load and a spring support at the end of the beam. In this model, both the Young's modulus (E-modulus) and the spring support stiffness were treated as unknown parameters, making this case a multivariable problem. The change increased the complexity for the optimises, allowing for the evaluation of scalability. Similarly to Case A, the "correct" E-modulus was set to 35.6 GPa while the "correct" spring stiffness was set to 350 MN/m. As boundaries, 30-40 GPa and 10-1000 MN/m were set for E-modulus and spring stiffness respectively. For the Nelder-Mead simplex and pattern search methods, an initial guess of 34 GPa and 100 MN/m was provided.

Once again, for Case B, a performance study relating accuracy and runtime between the models for a set runtime was conducted. This study allowed for relative performance comparisons between the models, highlighting the best model for Case B. Furthermore, it provided a direct scalability comparison with the previous and simpler Case A. As motivation for the "side-track" study, it was noted that the methods of Nelder-Mead simplex and pattern search require an initial guess, unlike the genetic algorithm and particle swarm methods. Thus, to evaluate any potential effects of such a guess, both the Nelder-Mead simplex and pattern search were further studied with a forced initial guess at the lower bounds, upper bounds and an average between the two. Through this study, the benefits and effects of different initial guesses are evaluated. The importance of this study arises from the likelihood that an initial guess could post noticeable changes in performance.

### Case C

Lastly, Case C represents the most complex test case. The complexity arises from the introduction of additional unknown variables and discontinuities. In addition to the unknown Young's modulus (E-modulus) and a support spring stiffness (K) at the end of the beam, this case introduces an additional beam element with an intermediate spring support and a rotational spring between the beam elements. Thereby, effectively doubling the number of unknowns compared to Case B, as illustrated in Figure 4.1. Once again, the "correct" E-modulus was set to 35.6 GPa. The intermediate spring support (K1) had a stiffness of 250 MN/m, the end spring support (K2) had a stiffness of 120 MN/m and the rotational spring (KR) had a stiffness of 610 MNm/rad. The boundaries for the E-modulus were set between 30-40 GPa. For translation springs and the rotational spring, the boundaries were set to 10-1000 MN/m and 10-1000 MNm/rad, respectively.

The ordinary study relating accuracy and time was once again performed, allowing for direct comparisons between the models in addition to the scalability compared to the previous cases. The "side-track" study for the most complex cases involves a sensitivity analysis against the objective function. As the previously used objective function contained deflections, sectional forces, and sectional moments, a comprehensive objective function was considered, see Equation 4.1. This complete objective function was likely to be more coherent compared to the available data in applied cases within structural engineering. The complete objective function likely reduced or avoided local minima and general discontinuities in the objective function, which may have benefited certain methods. Thus, a reduced objective function was applied that only considered defections, see Equation 4.3.

$$\text{OBJ} = \frac{1}{n}\sum_{i=1}^{n}\left[(U_i - U_{i,\text{true}})^2\right] \tag{4.3}$$

The outcome of this study highlighted the sensitivity to less complete and coherent data, showing a performance change due to the objective function. This study was deemed important because the objective functions are tailored to each case, with varying levels of completeness for each application. This reduced objective function was set to resemble what could be used in real applications.

### 4.1.2 Neural Network method

In attempts to evaluate and relate neural networks with regard to performance, similar methods as presented in Chapter 4.1 were employed. Therefore, identical test cases of A, B, and C were evaluated. In the following analysis, two methods are evaluated based on the basic forms of neural networks. Firstly, a standard feedforward (FF) system was established, which estimated deflected behaviour based on input variables. Secondly, another feedforward system was established, but with a reversed solving order. Thus, deflections were considered known and used to acquire the unknown variables. For the continuation of this thesis, this feedforward system in reversed order is referred to as feedforward reversed (FFR).

Both directions of the neural network (NN), namely FF and FFR, are evaluated and compared to assess how the network scales in terms of direction. This may reveal differences in the learning process and highlight the more appropriate implementations of NN within structural engineering. Additionally, it must be noted that only the method of FFR was directly comparable to the other studies in this thesis because the variable predicted are the same and similar to the other optimisation methods in this study it determines unknown variable based on known deflections.

Throughout the analysis, the FF network was primarily used to test, evaluate, and examine differences in setups and performances. Since the FF was not directly comparable to other methods in this thesis, limits on runtime was disregarded. Once appropriate setups were established, an FFR network was developed and then compared to the other methods.

In the following studies, parameters of Mean Absolute Percentage Error (MAPE) with respect to total runtime (RT), composed of Training Data (TD) and Training of Model (TM), will be presented. All tests for FF and FFR have been conducted with a division of 80 percent training data and 20 percent test data of the total available data. Packages implemented in Python include TensorFlow and Keras API, with an early stopping criterion that only considered the best-trained model compared with test data and stopped further iterations if no improvement was made after 20 epochs. This ensured that models do not suffer from overfitting during the training process. If the stopping criterion was not reached, a maximum of 300 epochs was set, with a standard batch size setting of 32. For all test cases, Adam has been chosen as the optimiser and ReLU as the activation function.

For each test case study that follows, there are three main analyses per case. First, the architecture and size of the NN are evaluated for each test case. Next, for each case, the FFR performance is evaluated for the corresponding best size. Lastly, for each case, one additional study was performed to test different aspects and setups according to the literature review, see 3.3.1. This last study is considered the "side-track" study for that chapter.

### Case A

As previously described, Case A involves the simplest of cases. It sets a basis for performance, considering the least demanding system through a single-variable analysis. The only unknown variable consists of the E-modulus, which had boundaries set between 30-40 GPa.

To train the model, the previously constructed FEM model for Case A was used to create a training dataset. The acquired training data for Case A consisted of combinations of different locations of data point (x), corresponding displacement (U), Young´s modulus (E), height of the beam (H), width of the beam (B), corresponding moment of inertia (I) and point load at the free end (F). The beam was divided into 4 segments with 5 nodes (0.0m, 2.5m, 5.0m, 7.5m, 10.0m). For Case A, 5070 different combinations were made (a total of 25350 nodal outputs of data), where the variables of E, H, and F were combined within the intervals of 30-40 GPa, 0.4-1.2 m, and 500-1500 N, respectively. The entire training data (TD) required 61354.4 seconds (17.04 hours). The Span of trainable variables ranged between 137 (2L 8N) to $7.4 \times 10^6$ (8L 1024N).

#### FF - Feedforward

As a primary study, the appropriate architectural size of the neural network (NN) was of importance to verify the appropriate size of the model. Therefore, an extensive study of the effects of both the number and the size of hidden layers was conducted. For Case A, a more extensive study of architectural size was of importance and thus was performed compared to that of the following cases, B and C. This was deemed important at early stages, as no prior knowledge of appropriate network sizes nor their effect was known. Chosen intervals of 2 - 8 hidden layers, in combination with 8 - 1024 neurones each, were tested. The following "side-track" study was performed for the Feedforward (FF) network and consisted of an analysis comparing different optimiser options. The SGD, Adam, and RMSprop optimisers were tested for a range of different model sizes ranging from 2 - 8 layers with 32 - 512 neurones each. As different optimisers may perform differently depending on network size, this analysis studied both appropriate optimisers and their sensitivity to network sizes.

#### FFR - Feedforward reversed

With appropriate setups established for the network in terms of FF, the FFR model was established and evaluated. Through the FFR, the unknown variables were derived from inputs of known displacements. As the FFR model required deflections as inputs for each node of the beam, it naturally outputs an approximate variable value for each node. Consequently, for a single beam, five deflections were given as input, resulting in five separate approximations for the unknown, one per node. For this system, the effect of potentially excluding the first node at x=0 was tested

and evaluated, since the fixed node has no deflections and therefore has no physical correlation with the Young´s modulus (E).

The ability to approximate variables through the FFR model was tested and evaluated. In the same study, different batch sizes were evaluated to verify the potential impact. For various configurations, average errors for the predicted E-modulus are presented, both as mean and median values, along with the required training time.

### Case B

Moreover, Case B increases complexity and allowed further evaluation of scalability. With the introduction of an additional variable, increased complexity was expected to affect both FF and FFR studies. For Case B, once again, both the E-modulus and the spring stiffness were considered unknown. As with conventional methods, the E-modulus had boundaries of 30-40 GPa, while the spring stiffness varied between 10-1000 MN/m.

As previously, an FEM model of Case B formed the basis for creating training data. The total training set for Case B consisted of combinations of different locations for data points (x), corresponding displacement (U), uniformly distributed load (Q), Young´s modulus (E), height of the beam (H), width of the beam (B), corresponding moment of inertia (I) and support spring stiffness (K) at the end of the beam.

The beam was divided in to 4 segments with 5 nodes (0.0m, 2.5m, 5.0m, 7.5m, 10.0m). A training set of 11857 different unique combinations was made (total of 59285 nodal outputs of data). In which E, K, Q and H variables were combined within the interval of: 30-40GPa, 10-1000 MN/m, 50000-200000N and 0.4-1 meters respectively. The total collection of training data (TD) required 125018.2 seconds (34.72 hours). For the training process for Case B, a layer size variation between 4-16 was chosen for the studies. The addition of 16 layers, compared to Case A, increased the number of trainable parameters. The additions resulted in $248.705 \times 10^3$ (16L 128N), $988.929 \times 10^3$ (16L 256N), $3.944 \times 10^6$ (16L 512N) and $15.752 \times 10^6$ (16L 1024N) trainable parameters respectively.

#### FF - Feedforward

As change in complexity could impact the choice of appropriate network size, the preliminary study reassessed the model architecture. In contrast to the previous study, a less extensive study was conducted for Case B. A size variation between 4-16 hidden layers, with 128-1024 neurones each, was assessed. The range of hidden layers has now been increased compared to Case A, which consisted of 2-8 layers. This adjustment was made because the larger system was expected to perform well with a larger number of layers, thus shifting the range.

As a "side-track" study for Case B, the preliminary choice of using ReLU as the activation function was questioned and therefore evaluated. Different activation functions may also exhibit a dependency on the network size. Additionally, since the purpose of activation functions is to transform numerical values, strong dependencies on the size of input values may have a drastic impact. Consequently, activation functions were tested for both small and large architectural sizes, considering either Min-Max scaling or standard scaler methods for scaling inputs.

#### FFR - Feedforward reversed

With an understanding of the appropriate network sizes, as well as the verification of suitable optimisers and scaling methods, the FFR network was established and evaluated. The mean and median errors were evaluated for both the E-modulus and spring stiffness. Furthermore, the variation between mean and median results was of interest, prompting a focused study on plausible reasons for this variation. As the FFR approximates a variable for each node along the beam, multiple guesses were made for each variable. Consequently, the variation of guesses along the beam was studied and visualised in an attempt to understand the difference between the mean and median.

### Case C

Lastly, the complexity was further increased through Case C. The complexity arises from the introduction of two additional variables, potentially impacting both FF and FFR studies. In addition to the unknown E-modulus, the inclusion of the intermediate spring support (K1), the spring support at the end (K2), and the intermediate rotational spring (KR) resulted in four unknowns. As in the earlier study, the boundaries for the E-modulus were set between 30-40 GPa. For translation springs and rotational spring, the boundaries were set to 10-1000 MN/m and 10-1000 MNm/rad, respectively.

The FEM model created for Case C formed the basis for generating training data. The total training data for Case C consisted by combinations of location for data points (x), corresponding displacement (U), Young´s modulus (E), beam height (H), beam width (B) and the corresponding moment of inertia (I). In contrast to the other cases, uniformly distributed load in the first span (Q1), uniformly distributed load in the second span (Q2), spring stiffness of the first support (K1), spring stiffness of the second support (K2) and rotational spring stiffness for the intermediate spring (KR) were also introduced.

In contrast to cases of A and B, for Case C the beam was divided in to 8 segments with 9 nodes (0.0m, 2.5m, 5.0m, 7.5m, 10.0m, 12.5m, 15m, 17.5m, 20m). For Case C, 15435 different unique solutions laid the basis for the training data (total of 138915 nodal outputs of data). The variables E, K1, K2, KR, Q1, and Q2 were combined within the following intervals: 30-40 GPa, 10-1000 MN/m, 10-1000 MN/m, 10-1000 MN/m, 100000-180000 N, and 80000-160000 N, respectively. The entire training data collection (TD) took 198304.2 seconds to produce (55.04 hours). In contrast to previous studies, the upper limit of the number of hidden layers was increased to 32, resulting in an increased number of trainable parameters.

#### FF - Feedforward

For Case C, a less comprehensive architectural size study was conducted compared to the corresponding study for Case A. Similar to Case B, larger systems were studied, consisting of between 4 and 32 layers, with 128 - 1024 neurones each.

#### FFR - Feedforward reversed

The equivalent FFR, based on appropriate network sizes, was established for Case C. However, unlike previous FFR studies in which performance was related to a small range of batch sizes, the FFR study for Case C included a more comprehensive analysis, considering a larger range of batch sizes. This was done to push the limits and examine the breaking points rather than just to find the most accurate setting. Through such analysis, one could better understand the limits and consequences if exceeded, thus highlighting sensitivity and severity. This constitutes the "side-track" study, which was combined with the ordinary FFR analysis.

### 4.1.3 Physics Informed Neural Network method

As a further development of the neural network, the adaption of Physics Informed Neural Network (PINN) was implemented in attempts to achieve more accurate results. Similarly to previous studies in this report, the same test cases were used for evaluation and benchmarking as seen in Chapter 4.1. However, with the exception that the evaluation of the PINN was only conducted on test Cases A and B, the experiments were also limited by the goal of achieving convergence of the PINN models for a single test sample. This limitation was related both to the time available for the thesis work and to the purpose of implementing PINN in our study, as it serves as an extension of the above-mentioned NN. Thus, limited tests were considered in attempts to prove the concept rather than complete development.

Unlike the methodology for the ordinary Neural Network (NN), both feedforward and inverse problems were established through the same model with the PINN. This model was thereby directly comparable to the other methods investigated for optimisation in this report. The structure of the physics-informed neural network is similar to the regular network, and the main difference is the objective function or the loss function. The loss computations in the PINN are complemented by the loss of the residual of the physical differential equation and the loss from its corresponding boundary conditions. This method aimed to train the neural network not only to interpret patterns from the "true" deflections, but also to incorporate the underlying physics for the deflection. With the help of the Euler-Bernoulli beam equation for linear elastic deflection, along with the boundaries of the specific problem, the training was enhanced. The PINN models were built using the Python programming language with the integration of the PyTorch and Optuna packages.

For the following test case studies, two main analyses were performed for each test case, in which the performance for feedforward prediction and inverse solving was evaluated.

#### Case A

Test Case A, the first and simplest of the three in the report, was also the first to be analysed. As stated in Figure 4.1, Case A consists of a cantilever beam that is fixed at one end and free with an applied point load at the opposite end. The loss function only had a contribution from the boundary conditions and the residual of the differential equation. The potential loss from a comparison between the "true" solution and the predicted solution was not included in the total loss summation, thus, an unsupervised training was performed. The boundary conditions implemented in the PINN were as follows: No deflection ($u$) or rotation ($du/dx$) at the fixed end ($x = 0$). The moment ($d^2u/dx^2$) was also set at zero and the shear force ($d^3u/dx^3$) should be equal to the point load (P) applied at the free end ($x = L$), as seen in Equation 4.4. The resulting loss function for the PINN application for Case A can be seen in Equation 4.7, combined by the loss from Equations 4.5 and 4.6.

$$\begin{aligned} &\textbf{Fixed end } (x=0)\textbf{:} && u\Big|_{x=0} = 0 \quad ; \quad \frac{\partial u}{\partial x}\Big|_{x=0} = 0 \\ &\textbf{Free end with point load } (x=L)\textbf{:} && \frac{\partial^2 u}{\partial x^2}\Big|_{x=L} = 0 \quad ; \quad \frac{\partial^3 u}{\partial x^3}\Big|_{x=L} = P \end{aligned} \tag{4.4}$$

$$L_{\text{boundary}} = \sum \left( u_{\text{pred}}\Big|_{x=0} + \frac{\partial u_{\text{pred}}}{\partial u_{\text{pred}}}\Big|_{x=0} + \frac{\partial^2 u_{\text{pred}}}{\partial u_{\text{pred}}^2}\Big|_{x=L} + \frac{\partial^3 u_{\text{pred}}}{\partial u_{\text{pred}}^3}\Big|_{x=L} \right)^2 \tag{4.5}$$

$$L_{\text{differential equation}} = \sum \left( EI \cdot \frac{\partial^4 u_{\text{pred}}}{\partial x^4} \right)^2 \tag{4.6}$$

$$L_{\text{tot}} = \sum \left( L_{\text{differential equation}} + L_{\text{boundary}} \right) \tag{4.7}$$

**Feedforward prediction**

In this case, two different setups were carried out. The first experimental setup, implemented for Case A, aimed to investigate the use of dimensionless parameters during training. The second experiment aimed to use a PINN framework to predict displacement with input data from the test cases, similar to the previous setup but with true scaled units. Furthermore, the stability was investigated when noise was induced in the training data. In the training of the PINN, the predicted displacements from the model were used as input to compute the derivatives for the governing equation. The gradients were calculated using PyTorch's automatic differentiation engine, Autograd.

The dimensionless experiment was done with the argument that the neural network, with its weights and biases, uses small values in the activation functions, usually between 1 and -1, resulting in errors which tend to vanish or explode during the training process for large input values. Making the training process non-dimensionalised could avoid a more complicated implementation with parameters of different units and sizes. This approach could also potentially provide a higher robustness and tractability for the network and reduces the effort of fine-tuning the hyperparameters. The neural network has been trained with three parameters at 100 collocation points along the dimensionless beam (x = 0 to 1). The dimensionless parameters used in the training were the moment of inertia (I), the point load (P), and Young's modulus (E), except for the position of the beam (x).The network was composed of 2 hidden layers with 32 hidden neurones in each layer, giving it a total of 1217 trainable parameters. The Tanh activation function was used and mainly for two reasons. Firstly, it is one of the most commonly used activation functions for

PINN applications (Kapoor et al., 2024). Secondly, even if ReLu previously in the report was stated as the optimal choice, ReLu suffers from limitations in computing negative gradients, which made it unsuitable for computing the gradients for the Euler-Bernoulli beam differential equation. The optimiser for this test was chosen to L-BFGS, which is one of the most widely used optimisers in PINN studies (Kapoor et al., 2024), to minimise the loss function. To achieve a non-dimensionalised function, the following transformation was performed on the original equation as can be seen in Equation 4.8.

$$x = \frac{\bar{x}}{L}, \quad u = \frac{\bar{u}}{L}, \quad u_0 = \frac{\bar{u_0}}{L}, \quad P = \frac{\bar{P}L^3}{EI} \tag{4.8}$$

With the boundary conditions in Equation 4.5, displacement over the length of the beam with a point load at the edge can be described with help of a Dirac delta function to consider q(x), see Equation 4.9, the following dimensionless equation can be derived, see Equation 4.10.

$$EI \cdot \frac{\mathrm{d}^4 u}{\mathrm{d}x^4} \quad = P\delta(x - L) \tag{4.9}$$

$$u_{transformed} = P(x) \quad , \qquad x \in [0, 1] \tag{4.10}$$

This yields the same deflection at an arbitrary point with units, according to the derived equation from the given boundaries, see Equation 4.11.

$$u(x) = \frac{Px^2(3L - x)}{6EI} \tag{4.11}$$

The second setup was performed with three experiments for the PINN application for Case A, considering parameters with conserved units but with scaled dimensions through factors. For example, the modulus $E$ with a dimension of $30 \times 10^9$ Pa was scaled by $10^{-11}$ to obtain a value between 0 and 1. The neural network training were performed at five collocation points, the same positions on the beam as for the FEM solutions (x = 0.0, 2.5, 5.0, 7.5, and 10.0 m). This choice was made with the intention to ease the verification of the predicted deflection by the network. The network was composed of 2 hidden layers with 32 hidden neurones in each layer (1217 trainable parameters) with an Tanh activation function for all the layers except the last output layer. To evaluate different optimisers, the Adam optimiser was used in this experiment to minimise the loss function for the neural network, which has proven effective in previous tests for the thesis work. The choice of optimiser was based on the hypothesis that a potential benefit of using Adam instead of L-BFGS could be that, because Adam does not loop over each input batch until convergence has been reached. Therefore, it might be an beneficial optimiser when the training data are composed of combination sets with great variety, were searching for a

more general pattern are of interest. The network was trained on three parameters: Moment of inertia (I) and point load (P), with the corresponding positions studied along the beam (x). To evaluate how well the PINN performs even if the input variables are not perfect, or to simulate the effects of an imperfect equation, the input data for the training of the network were tested with and without the introduction of Gaussian noise. Three experiments were performed for the implementation of PINN in Case A. One experiment was carried out with a specific batch of input data for the moment of inertia (I) and a point load (P). Thereafter, two experiments were carried out with the introduction of Gaussian noise with a total amount of 10% and 20% noise. The results were compared for the predicted displacements, using test samples that were different from those on which the network had been trained upon.

**Inverse prediction**

The neural network training was carried out at five collocation points, the same positions as for the feedforward experiment, which are the same as the FEM solutions (x = 0.0, 2.5, 5.0, 7.5, and 10.0 m). The same network build was reused, similar to the feedforward experiment, composed of 2 hidden layers with 32 hidden neurones in each layer (1217 trainable parameters) with the Tanh activation function. The inverse experiment also used units with dimensions, like the second setup for the feedforward experiment. The conserved units for the input parameters were only scaled with a dimensional factor and the Adam optimiser was used to minimise the loss function for the network. Three variables in different batches were used to train the network: Moment of inertia (I) and point load (P), in addition to the studied locations along the beam (x). The same approach was used for the inverse as well as the feedforward experiments, with the predicted displacements from the model used as input to the four derivatives computed by PyTorch Autograd. To evaluate how well the PINN performs for the inverse problem under non-perfect conditions, potential variance in input variables or non-optimised equations were evaluated. The input data, as in the feedforward experiment, were tested both for the prediction and in the boundary loss equation with and without the introduction of Gaussian noise in three different experiments. One experiment was carried out with a specific batch of input data for the moment of inertia (I) and point load (P). Two experiments were carried out with the introduction of Gaussian noise with a total amount of 10% and 20% noise. The results were compared for both the predicted displacements and the E-modulus, using test samples that were different from those on which the network had been trained upon.

### Case B

Test Case B introduces a more complex problem, including both an indeterminate nonlinear problem and more unknown variables to solve for. The evaluation of Test Case B, does not only serve as a "proof-of-concept" for the PINN in general. But also to examine the scalability of the PINN for handling the combined complexity of nonlinear and indeterminate problems, and for deriving multiple variables, which the regular neural network proved insufficient for.

The loss function for the PINN implementation for test Case B consist of the loss from the boundary conditions, the loss from the residual of the forth derivative of the differential equation together with the data loss when comparing with the "true" solution.

$$\begin{aligned} &\textbf{Fixed end:} && u\Big|_{x=0} = 0 \quad ; && \frac{\partial u}{\partial x}\Big|_{x=0} = 0 \\ &\textbf{Free end with point load:} && \frac{\partial^2 u}{\partial x^2}\Big|_{x=L} = 0 \quad ; && \frac{\partial^3 u}{\partial x^3}\Big|_{x=L} = K \cdot u(x) \end{aligned} \tag{4.12}$$

$$L_{\text{boundary}} = \sum \left( u_{\text{pred}}\Big|_{x=0} + \frac{\partial u_{\text{pred}}}{\partial u_{\text{pred}}}\Big|_{x=0} + \frac{\partial^2 u_{\text{pred}}}{\partial u_{\text{pred}}^2}\Big|_{x=L} + \frac{\partial^3 u_{\text{pred}}}{\partial u_{\text{pred}}^3}\Big|_{x=L} \right)^2 \tag{4.13}$$

$$L_{\text{differential equation}} = \sum \left( EI \cdot \frac{\partial^4 u_{\text{pred}}}{\partial x^4} - q \right)^2 \tag{4.14}$$

$$L_{data} = \sum \left( u_{true} - u_{pred} \right)^2 \tag{4.15}$$

$$L_{\text{tot}} = \sum \left( L_{\text{boundary}} + L_{\text{differential equation}} + L_{\text{data}} \right) \tag{4.16}$$

#### Feedforward prediction

For test Case B, the experiment setup to predict displacement, using a PINN framework was similar to the experiment for test Case A. The training of the neural network was performed with conserved units transformed with a scaling factor and the predicted displacement was used to calculate the gradients. Furthermore, the collocation points used for the training process were carried out at the same five positions that the FE program had calculated the true solutions (x = 0.0, 2.5, 5.0, 7.5 and 10.0 m). The input parameters for the training of the network consisted of: Moment of inertia (I), Young´s modulus (E), uniformly distributed load (q) together

with the position of the beam (x). This setup, like previous PINN experiments, was also performed with 2 hidden layers with 32 neurones in each layer, making a total of 1217 trainable parameters. The Tanh activation function was chosen for all layers except the last output layer, together with the Adam optimiser. The same investigation of the sensitivity for non-perfect input variables was executed as for Case A, with an infusion of a total Gaussian noise between 10 - 20% to the training parameters. The resulting predictions for displacement trained on a test sample were then compared with the true solution of a sample that the network had not seen.

**Inverse prediction**

For test Case B, the experiment for the inverse problem aimed to predict the Young´s modulus (E) and the end support spring stiffness (K) for the beam. The training of the network was performed on the same five collocation points as for the previous feedforward prediction of displacements. The neural network used in this experiment had the same architecture as the previous one, consisting of 2 hidden layers with 32 neurones each, using Tanh as the activation function for all layers except the output layer and Adam as the optimiser. The inverse problem also used units with dimensions transformed with a scalar factor. The evaluation of the PINN implementation for test Case B was carried out through three experiments. The first experiment aimed to extract the Young's modulus (E), the second to extract the support spring stiffness (K), and the third to predict both the Young's modulus (E) and the support spring stiffness (K) for the beam. The predicted results were compared with the "true" solution obtained from the FE program.

### 4.1.4 OpenAI method

Similarly to previous studies conducted in this thesis, the three test cases presented in Figure 4.1 were used for evaluation regarding AI. In contrast to previous studies concerning conventional and neural network methods, the implementation of Artificial Intelligence (AI) in structural optimisation is less commonly addressed. Therefore, before conducting performance studies for specific test cases, it was crucial to develop robust methods to prompt and interact with AI. This was necessary to evaluate the differences and impacts of prompt engineering in the development of these robust methods. These principles were initially tested and developed with Case A, which was the simplest in terms of implementation. After establishing the appropriate methods, the studies progressed to more complex cases to validate their effectiveness and scalability.

OpenAI's ChatGPT was utilised, incorporating both the GPT-3.5 and GPT-4 engines. The analysis of ChatGPT was conducted before the update of May 13, 2024. Thus, any improvements to ChatGPT, as well as the new engine of ChatGPT-4o, were not of concern in the main study. However, limited sample tests of some studies were carried out through ChatGPT-4o, in verification purposes. The main studies were mainly conducted using the assistant feature due to the advantages presented in Chapter 3.3.3, which discusses the features of OpenAI. Consequently, an accessible user interface was provided through the web, alongside API access, while still including pre-defined memory functions. The assistant feature was selected as the most suitable approach to facilitate trial and error during the development phases. For each case, the assistant was provided with hidden rules, communicated once before each optimisation. This method ensured that rules and expectations were only explained once, reducing the prompt length for each iteration. Within these rules, four main components were clearly defined: *input*, *output*, *context* and *instructions*. Through description of input and output, ChatGPT was prepared based on the type of input data provided in addition to the expected formatting of output. As context, the general goal, complication and relevant information for each problem was explained. Lastly, through instructions, the boundaries, iteration step sizes and method of iteration were explained.

For the optimisation process, MATLAB and Brigade2023 were used once again. Through MATLAB, the results retrieved from Brigade2023 were processed and then provided to ChatGPT for evaluation and progression in iteration. Both the principles of providing direct results in terms of deflections and providing calculated objective function values were tested. In cases where an objective function value was used, an identical function evaluation and normalisation to that of the conventional methods were employed, see Equation 4.1 and Equation 4.2.

Similar to the methodology for the conventional methods, each case involves one primary study, evaluating accuracy compared to demand for different methods, followed by one "side-track" study per test case. In contrast to conventional methods, optimisation through AI has no pre-made methods or functions. Thus, emphasis was placed on the development and evaluation of different prompting concepts instead of direct implementations. In addition to accuracy and computational demand, rule adherence was also evaluated, as it could be an important factor in the automation of AI.

**Case A**

Initially, various methods were tested to prompt the OpenAI model. The primary Case A was the ideal candidate to facilitate the development of appropriate methods, as it was the least demanding and most evaluable case. Once again, Case A presents a single-variable problem, involving an unknown E-modulus. The boundaries of 30-40 GPa were still used.

The chosen methods involved providing the AI with the complete context of the problem, including geometry, load, and structure type. To provide this context, current displacements along the beam with their corresponding variable parameters were given as input. In parallel, the desired displacements for the "correct" solution were also provided. Additionally, a more concise prompt technique was further employed, where only the objective function value and the corresponding affecting variable value were given. Furthermore, chain-of-thought (COT) reasoning was integrated to evaluate potential benefits.

The first technique, which involved providing complete context, promotes numerical calculations. Although it could be demanding for the AI, the goal was to achieve a more direct and faster approach through calculations. The context and desired outcomes for the AI were explained through hidden rules, which were stated once before the optimisation started. The context provided included a brief explanation of the geometry of Case A, stating that it was a fixed-free cantilever beam with a point load at the end. Furthermore, input formatting was explained as that deflections along the beam at five locations (x = 0.0, 2.5, 5.0, 7.5 and 10.0 m) for the current E module, along with the desired "correct" deflections at the corresponding locations. The AI were then asked to output a singular value for the updated E-modulus.

The second method, with and without implementations of COT concepts, was evaluated as it could promote more analytical reasoning instead of calculation. Initially, hidden rules were communicated to the AI once again. Through the description of input and output, ChatGPT was prepared for the type of provided input data and the expected formatting of the output. For Case A, this consisted of an objective function value and the corresponding E-modulus, while the expected output was an updated E-modulus. Additionally, the context of the problem was provided. For this study, the context stated that a general optimisation based on objective function values and corresponding variable values was expected. Then, through instructions, the boundaries and explicit steps for the chain-of-thought reasoning were explained

when used. For the methods that incorporated COT reasoning, five distinct steps were outlined:

1. ChatGPT was instructed to list all previous objective function values along with the corresponding E-module value. Although this step increases inefficiency in terms of token usage, runtime and cost, it was considered necessary for promoting detailed memory.

2. The five most promising trials were requested. The AI was then forced to evaluate the data and determine a promising range.

3. The mean value for the E-modulus from the best attempts was requested, thereby promoting analytical reasoning once again.

4. An evaluation of the three most recent iterations was encouraged to prompt the AI to assess whether the current direction is logical, based on the most recent decisions.

5. From all previous subtasks, an updated E-modulus was requested.

Through these methods, the main study was conducted and the accuracy compared to the computational demand was evaluated for the different methods. Hence, the differences in performance between the methods can be studied to determine the most appropriate ones. Furthermore, as a "side-track" study, OpenAI's capabilities in numerical calculations were evaluated in an attempt to verify earlier statements from the literature study regarding poor numerical performance, see Chapter 3.3.3.

**Case B**

With the proven applicability of ChatGPT in Case A, the complexity was increased to assess scalability. Similarly to the studies of conventional and neural network methods, test Case B involves two unknown parameters, namely the E-modulus and the spring stiffness K. The E-modulus once again had boundaries of 30-40 GPa, while the spring stiffness had boundaries of 10-1000 MN/m.

The more concise prompt incorporating COT, developed for Case A, was utilised again, with the addition of the extra variable explained through the prompt. The methods used for Case B consisted of two variations in an attempt to improve efficiency and accuracy. The first method (B1) resembles the prompt developed for Case A, with extensions for the new variable. The second variation (B2) included a limit on the smallest step sizes and a threshold for the objective function value at which convergence could be considered. Additionally, whereas the previous COT steps required the AI to repeat every previous iteration to ensure better backtracking, this was now reduced to only the last 15 trials for B2. This adjustment aimed to produce a more accurate and less time-consuming iterative process.

Primarily, these methods were compared through the main study, evaluating accuracy against computational demands. Additionally, as a "side-track" study, the solution space for the objective function was plotted and the iterative steps from the AI were evaluated. This was done to visualise and understand the progression and behaviour of the AI. Through such analysis, appropriate modifications could be employed to address any obvious shortcomings in its progression.

### Case C

The intention was to perform a similar developments as for previous cases, however due to results presented below, further development for Case C was not conducted as no prompt technique could procure a robust or successful outcome for Case B.

## 4.2 Case-study

To verify and apply the proposed updating principles on a larger scale, a Case-study was used for implementation. Being an ongoing project during this thesis, the Solvalla bridge makes the perfect Case-study. Not only was it more representative than the test cases used for optimiser benchmarking, but it also magnified the scalability effects significantly. In the following chapter, the relevant project description and geological conditions will be described. Furthermore, the principles and methodology of the static test loading, in addition to structural complications, will be presented. Finally, the considerations and methodology of the implemented updating principles will be discussed.

### 4.2.1 Project description

In an attempt to expand accessibility and meet the growing demands on infrastructure, Stockholm has embraced an ambitious light rail transit project (Wikipedia, 2024c). Light rail transit is often considered a compromise between trains and trams. Unlike ordinary trams, they typically operate at higher speeds, have a higher capacity, and often run on separated tracks. Compared to metros and trains, they are generally much cheaper. With the current configuration, two lines connect Solna and Sickla, with a combined daily user count of about 100000 as of 2019.

Through an 8 km extension, the existing infrastructure is extended towards Helenelund, increasing accessibility and stretching through the three municipalities of Stockholm, Sundbyberg, and Sollentuna (Stockholms stad, 2024). The extension of Tvärbanan will provide direct connections to Bromma Airport, multiple metro stations, and train stations. Additionally, the extension is expected to be of high importance, as it covers some of the most significant development areas in Stockholm. Consequently, the project is considered an essential component of current development plans.

A global overview of Tvärbanan in its current state is visualised in Figure 4.2. The existing infrastructure is portrait by an orange line, while gray dotted lines represent the ongoing extension. The first stages of the extension towards Bromma Airport were completed and in use as of May 2021 (SL, n.d.).


**Figure 4.2:** Map overview of Tvärbanan, both existing and planned infrastructure (Brmf, 2020).

The bridge segment of interest for this study is a 160 meter section of a roughly 780 meter long bridge located near the horse racecourse of Solvalla (Svedholm et al., n.d.). For an illustration of the brigde of interest, see Figure A.1. This segment is of particular concern as it passes over a train railroad, the Mälaren Line (Mälarbanan), which enforces strict limitations on clearance height. Furthermore, the height of the rail top for the new Tvärbanan is defined and limited, resulting in a constrained beam height. In addition, the bridge segment of interest contains a significant curvature, creating additional complications. Consequently, a minimised section height is required at a critical point.

The bridge section studied is of composite character, consisting of two parallel steel girders in combination with a concrete slab. The steel girders were designed as cambered beams and dimensioned to proactively counteract deflection and sagging caused by permanent loads. The steel girders along the 160-meter segment cover three spans, stretching 41.3 m, 62.2 m, and 56.5 m, respectively, see Figure 4.3. The girders have a height variation between 1150 mm and 3000 mm, with the lowest bound due to the set height limitations. Furthermore, the two parallel girders are provided with lateral bracing spaced at intervals of 2.5 - 5.1 meters.

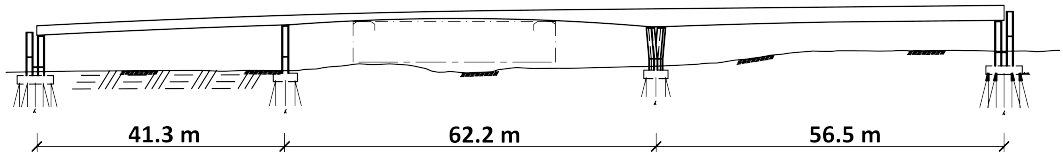



**Figure 4.3:** Side profile of studied bridge segment.

While the construction of the steel girders and the concrete slab is completed, further phases remain a concern. The deviation between models and real construction could be devastating if proved too large, especially considering the existing infrastructure beneath the bridge. Current FEM models must be updated to match the actual behaviour of the current construction phase to predict future deflections and margins with greater accuracy. The methodology involves testing the construction under static loading and measuring the deflection behaviour.

### 4.2.2 Static load test

The field measurement of the bridge was performed through a carefully considered test protocol. The loaded trucks were placed on the bridge, while the deflections were measured after a settling time and compared against the unloaded equivalent. In total, six different loading positions were considered for two different load combinations. A visualisation of all loading positions can be seen in Figure 4.4. The naming of the coordinates represents the span and track side, for instance, 56-S corresponds to the span between supports 5 and 6 and correlates to the southern track. The first load combination involves two parallel trucks placed in mid-spans, one span at a time, loading each girder equally. This case involves even loading, and three configurations were considered. This correlates to a combination of south and north positions together, for instance, 56-SN corresponds to the second span. The second load case considered skew loading. The loading consisted of a single truck placed over one girder at a time, for all three spans, for each girder. For skew loading, six unique configurations were considered. Through the two load combinations, nine different cases with corresponding deflections were considered, forming the basis for the correction of the FEM model.

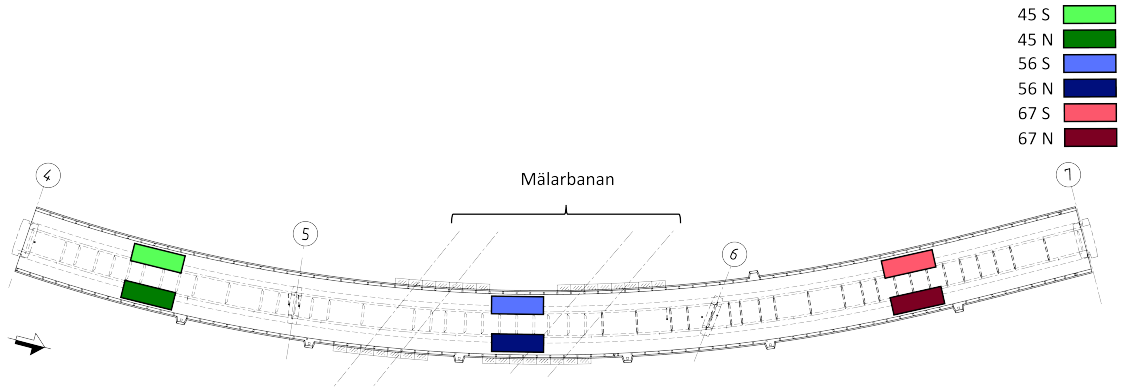



**Figure 4.4:** Overview of truck placements.

During the testing procedure, monitoring was performed to assess structural behaviour. The measured behaviour consisted of relative vertical displacements caused by static loads.

For each loading, deflections were measured approximately 10 minutes after load application to allow the construction to settle. Along the tested bridge segment, four rows of reference points were appointed for measurement: two along the edge beams and two parallel along each steel girder. For each loading configuration, measurements were taken along the entire line of the edge beams. To limit measurement time and costs, the measurements above the girders were chosen more selectively. In the span where the loading were applied, the number of measurement points were increased, excluding those blocked by the trucks. For the remainder of the spans, measurements above the girders were not recorded. This was deemed a sufficient compromise between time demand and data coverage.

Deflection measurements were performed using a digital level, model "Leica LS10." The instrument had a specified accuracy of 0.3 mm when using an Invar alloy staff and 1 mm when using a standard staff. Since a standard staff was used, 1 mm was assumed to be the error in reading. This method of levelling, as described in Chapter 2.1.1, features manual readings for each measurement point but produces the highest accuracy. This further strengthen the choice of this method, as it posts a reasonable ratio of accuracy and cost, compared to other principles discussed. For each measurement point, steel rods or bolts were drilled into the deck, creating temporary reference points. Based on the precision of this instrument, the loads imposed by the trucks were designed to yield a maximum displacement error of 5%.

No ground or material properties were measured or supplied for this analysis. Neither temperature, acceleration, nor strains were provided. For reference, the behaviour of the measured deflections for each load case can be seen in the Appendix A.6, where the measured behaviour is plotted. The measurements along the edge beams are discontinuous in some cases, as mishaps during measurement led to the absence of some points.

### 4.2.3 Loads and boundaries

The applied load during the testing consisted exclusively of the weight of the trucks. The trucks were Volvo trucks, of model FM 64R, resulting in three axles that transmit force. The trucks had a configuration of 4.1 meters separating the front axle and the first of the bogie axles, and an additional 1.37 meters between the two bogie axles themselves, see Figure 4.5. With a total load of approximately 27000 kg for the loaded truck, the load distribution between front and rear axles was approximately 33 to 67 percent. For each occurrence of loading, the truck was facing north.

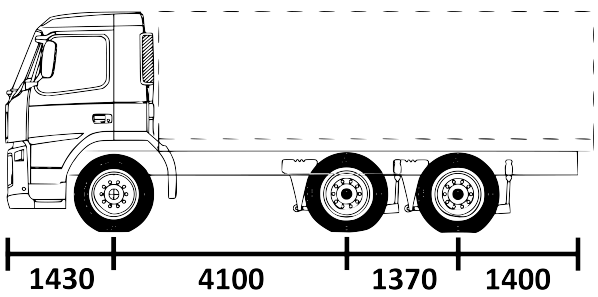



**Figure 4.5:** Truck dimensions.

The bridge segment rests on two bridge bearings per support. In support 4, the northern side has a fixed bearing, prohibiting movement, while the southern side uses one-way movable bearings. Along the northern side, the remaining bearings are one-way movable bearings, while the southern side consists of multi-directional bearings. For visualisation, the chosen bearings along the studied bridge segment can be seen in Figure 4.6. Each of the bearings had a minimum sustained SLS load of 4000 kN per bearing. This thereby refers to the least amount of vertical force necessary to mobilise intended bearing behaviour.

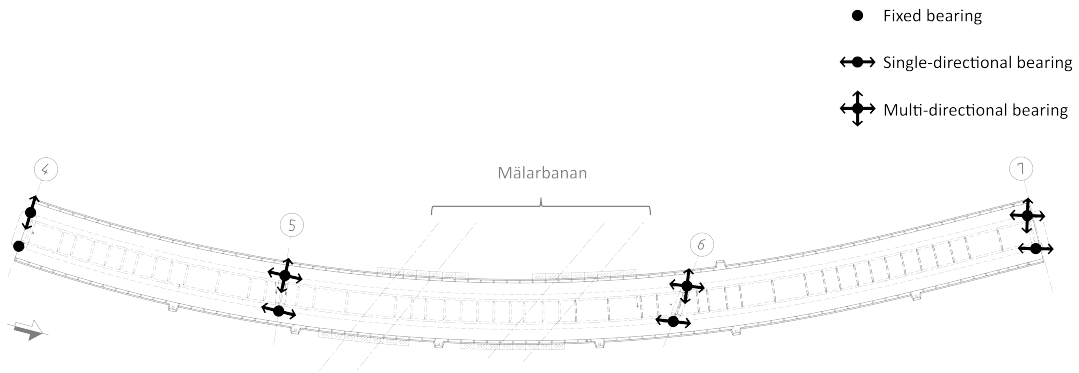



**Figure 4.6:** Overview of boundaries and bearing directions.

### 4.2.4 FEM model

During the design stages of the bridge, two distinct FEM models were established. A simplistic beam model, in which concrete and steel behaviour is modelled through beam elements, and a more detailed shell model were created. For the purposes of this thesis, as both models have been proven sufficiently accurate, combined with the computational benefits of beam models discussed in Chapter 3.4.1, the choice of the beam model was considered more appropriate within the scope and intention of the case-study.

The beam model of relevance is presented in Figure 4.7. It consists of four distinguishable parts: Supports 5 and 6 are shown in orange, the beams representing pure bending are shown in blue, the beams corresponding to pure torsion are shown in red, and the stiff connections between them are depicted in black.

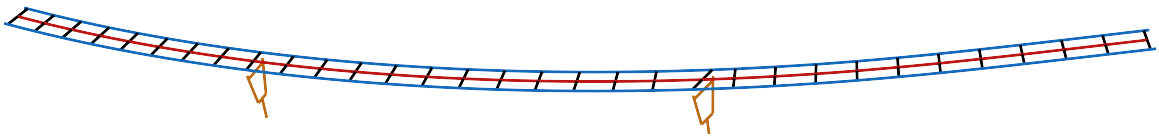


**Figure 4.7:** Beam model of the Solvalla bridge.

As the construction is of composite type, involving an interaction between steel girders and concrete slabs, the characteristics of the modeled beams must reflect their combined behaviour. This was achieved through the estimation of equivalent sections, translating the different properties of steel and concrete into an equivalent steel section. Of significant importance are the concrete areas at risk of cracking. In initial designs, the segments near the supports, within 15% of the span length of the support, were considered cracked, thus excluding the contribution of stiffness from concrete. This assumption is in accordance with SS-EN 1994-2 5.4.2.3. For cracked sections, only steel girders and reinforcements are considered. Therefore, a step-wise decrease in stiffness is assumed, where the concrete contributes either fully or not at all. Furthermore, the effective width of the concrete slab was estimated according to SS-EN 1994-2 5.4.1.2.

The modelled material strength properties was set to S460 for the steel girders and C35/45 for concrete, in accordance to specified characteristics for the bridge.

### 4.2.5 Methodology for model updating

A major complication in optimisation was choosing appropriate parameters to match the FEM behaviour to measured behaviour. When comparing the established FEM model against the measured behaviour, it was observed that the real construction showed significantly stiffer behaviour compared to the initial designs. Therefore, in efforts to stiffen the model behaviour, different assumptions and measures were taken. Primarily, steel girders were assumed to have geometric deviations corresponding to the upper limits of the allowed tolerances. The flange thickness was increased by 1.8 mm and the web height increased by 25 mm, according to SS-EN 10029. Additionally, the steel E-modulus was also increased to the upper tolerance limit, by 3%.

Apart from this, one of the more drastic changes made with noticeable impact towards achieving the correct behaviour was altering the bearing behaviour for the bridge. The loading was believed to not be sufficient to mobilise movement through the bearings, causing the bridge to behave as if having pinned the bearings on all supports. This assumption could be plausible as the test load might not have been sufficient to overcome the friction forces on the bearings. Additionally, with the behaviour corresponding to pinned bearings, the cambering of steel girders over the middle span, see Figure 4.3), could contribute to arch effects. To verify, the more detailed shell model, which incorporates the cambering of the beams was used to investigate the horizontal force in the bearings. The load case with both trucks in the middle span would translate roughly 138 kN of horizontal force at max. With a minimum load of 4000 kN in Serviceability Limit State (SLS) for the bearings, together with the assumption of a friction coefficient of 5%, showed that at least 200 kN of horizontal force was required to mobilise movements in the bearings. Therefore, the assumption that the bearings behaved as pinned was deemed accurate.

The employed beam model had horizontal beam elements, which would not capture any arch effects. To compensate, the bridge was assumed to be "raised" from its bearings, with stiff vertical links between the supports and bridge deck, hinged at the support. Thus, a reluctance towards free rotation was established through a frame action instead. This was implemented, and the effects were checked against a shell model to verify that the arch effects were not overcompensated for. This method did contribute to stiffening effects, however the true arch effect could not be fully modeled. This thereby was a known shortcoming of the beam model employed. However, with all above mentioned assumptions, the behaviour of the FEM model more closely resembled that of the real construction's stiffness.

The still evident deviance between the FEM model and measured behaviour was addressed through an optimisation routine exclusively optimising against concrete behaviour. The main parameters optimised were the E-modulus for the concrete slab and the degree of cracking along the bridge, as concrete was believed to be the likeliest reason for the deviance. Additionally, geometrical anomalies might arise for the in-situ casted concrete, making deviance in thickness a plausible variable. However, as both a slab height and E-modulus would impact the global behaviour

in likewise manner, they are likely to conflict and overcompensate one of the two. Therefore, the potential geometrical anomaly was fixed to a plausible maximum of 10% increased thickness, thus making the slab 30mm thicker then intended. The chosen methodology for the optimisation was to linearly interpolate the otherwise step wise differentiation between fully cracked and uncracked sections. Thus, the likelihood of cracked segments, in addition to smoother steps in crack propagation, could vary along the bridge. For the case of a 25% increased E-modulus for the concrete slab and fully uncracked sections, the bridge behaved stiffer compared to the measured behaviour. Therefore, this was set as the upper boundary for conducted optimisation.

Using the chosen parameters, the updating was carried out through the more promising methods identified in the previously mentioned test cases, mainly utilising the conventional method of a genetic algorithm. This methodology was deemed most appropriate due to the available data, limiting the possible completeness of the objective function. As genetic algorithm proved to be stable against discrete and incomplete data, it was chosen for the case-study. With 32 variables to optimise, corresponding to 31 values representing the cracking degree along the bridge and 1 value representing increased concrete E-modulus, a population of 640 was chosen. This was based on the results from the test cases, indicating that a sufficient population size was approximately 20 times the number of variables.

Additionally, as the Nelder-Mead simplex indicated good performance, optimisation though Nelder-Mead were also tested. Despite the reasoning that it performed well in the test cases due to its simpler nature, it was still of relevance to verify. However, the implementation of Nelder-Mead simplex proved to be problematic. Firstly, the methodology for Nelder-Mead simplex in the used toolbox seamed to implement "soft" boundaries. Thus, even if boundaries was stated, it had a tendency to disregard those. This would post difficulties for certain parameters, such as cracking, which was assigned 0 for fully cracked and 1 for uncracked. Any values beyond these would lose reasonability. Secondly, through the genetic algorithm, from the same tool box, there were simple principles to implement discrete values. Thus, cracking degree could be strictly considered only at 0.2 steps for the optimiser. This feature, Nelder-Mead simplex lacked. Either way, in attempts to verify the choice of using genetic algorithm, Nelder-Mead simplex were verified. As this methodology does require an initial guess, two different scenarios were used. The first of which assumed the crack distribution to be equal to that in preliminary design, in addition to 10% increased E-modulus for the concrete. Additionally, a worse guess of assigning all sections to half cracked, with 10% increased E-modulus was tested. This to test performance change in case of a less optimal starting guess.

The objective function chosen was similar to that explained for the conventional methods, see Section 4.1.1, where true displacements are correlated to those acquired through the model, see Equation 4.17, normalised through max-min normalisation, see Equation 4.18. It is important to note that the test case 67SN were weighted less compared to the others. This choice was made because measurements from

case 67SN were believed to deviate from expected behaviour in a manner that could indicate measurement errors, see Appendix A.24. Additionally, this load case suffers from a loss of multiple data points, as some were missed during measuring. Thus, the effect of this load case was minimised to reduce the potential risk of divergent values or anomalies adversely affecting the result.

$$\begin{aligned} \text{OBJ} \quad = \frac{1}{n} \sum_{i=1}^{n} \Big[ \quad & 1.1 \cdot \left(U_{i,45SN} - U_{i,45SN,\text{true}}\right)^2 \\ & + 1.1 \cdot \left(U_{i,56SN} - U_{i,56SN,\text{true}}\right)^2 \\ & + 0.8 \cdot \left(U_{i,67SN} - U_{i,67SN,\text{true}}\right)^2 \Big] \end{aligned} \tag{4.17}$$

$$X_{Scaled} = \frac{X - X_{min,true}}{X_{max,true} - X_{min,true}} \tag{4.18}$$

Furthermore, for each iteration of the optimisation, input data and corresponding displacements was stored for potential training of neural networks. With these, a simple implementation into NN was evaluated through same principle networks presented and developed for the test cases. For this case-study, this was preformed as a proof of concept, thus optimal settings were not found or used.

As a final remark, only the even loading cases were chosen for optimisation. This was done to keep computational times low while still yielding acceptable results. Since the load cases of two trucks per span result in higher deflections, these cases are likely to show the largest effects. With a prescribed measuring error of 1 mm, larger deflections result in a lower percentile measurement error. Thus, they were deemed the most appropriate cases for optimisation.

# 5
# Results of model updating

According to previously described methodology, the corresponding studies and their results are presented and discussed. The order of the results will follow the same as described under methodology, thereby, each category of convectional methods, neural network and AI are discussed one at the time. Lastly, based on the results acquired from the benchmarking, the case-study will be evaluated through the most promising and also proposed method. This was done with the purpose of evaluating the scaling between theory and application, but also to test the feasibility of the proposed methods.

## 5.1 Results of benchmarking

Before a method could be proposed, each update principle must be evaluated in order to determine their strengths and weaknesses. This was evaluated through a benchmarking process that tested each principle against identical base problems.

### 5.1.1 Conventional methods

The more conventional and often preferred methods of optimisation through the Nelder-Mead simplex, pattern search, genetic algorithm, and particle swarm were evaluated and tested against the three constructed test cases. For a more complete description of the studies performed, the circumstances, and the settings used, see Section 4.1.1.

#### Case A

Case A, being the simplest of the cases, representing the first Euler-Bernoulli equation, led to the expectation that the simpler methods of the Nelder-Mead simplex and pattern search would perform the strongest. This was because more advanced methods, such as genetic algorithms and particle swarm optimisation, have been shown to perform poorly in simple single-objective problems (Majid Solat Yavari et al., 2016).

For the initial time limit, Nelder-Mead simplex was run on auto settings, reaching its tolerance criteria of $1 \times 10^{-4}$ at a total runtime (RT) of 1447.4 seconds. This was thus the upper limit set for each of the remaining methods.

The results from test Case A are presented in Table 5.1. It can be noted that the pattern search method outperformed the Nelder-Mead simplex method, requiring less than half the runtime while achieving greater accuracy. Additionally, both biologically inspired models, the genetic algorithm and particle swarm, struggled with the set maximum runtime. The genetic algorithm method reached the time limit for each test, resulting in the least accurate solutions. In contrast, the particle swarm method converged within the time limit, achieving satisfactory accuracy in the case of three particles. However, even with the most promising setup, it matched the accuracy of pattern search but required more than double the runtime.

**Table 5.1:** Optimiser performance for test Case A, based on a single result set with runtime (RT) limit of 1447.4 seconds.

| | Optimiser performance | | |
|---|---|---|---|
| Optimiser type | Error, E [%] | Iterations | Runtime, RT [s] |
| Nelder-Mead Simplex | 0.0011 | 45 | 1447.4 |
| Pattern Search | $3.5667 \times 10^{-4}$ | 32 | 648.6 |
| Genetic Algorithm (20) | 0.95 | 6 | 1620.1 |
| Genetic Algorithm (30) | 0.27 | 4 | 1717.8 |
| Genetic algorithm (40) | 1.05 | 3 | 1869.7 |
| Particle Swarm (3) | 0.0017 | 30 | 1142.7 |
| Particle Swarm (5) | $4.12 \times 10^{-4}$ | 24 | 1490.4 |
| Particle Swarm (10) | 0.10 | 11 | 1459.3 |

It can be concluded that pattern search offers benefits in terms of computational demand and accuracy. The least successful method in performance was the genetic algorithm (GA), producing its best performance with a population of 30 individuals. With too few solutions, GA tended to create a narrow solution space, failing to achieve the diversity needed for convergence. In contrast, the particle swarm achieved its most accurate solution with a more modest size of only five particles. In this setting, it matched the pattern search in accuracy, but with a noticeable increase in computational time. A further increase in the number of particles resulted in erratic movements around the optimum without converging to a better solution.

The repeatability among the methods varied significantly. Both the Nelder-Mead simplex and pattern search methods demonstrated high repeatability in their results, in contrast to the biologically inspired models. For the relatively low number of individuals or particles in the genetic algorithm and particle swarm optimisation, the diversity of solutions was limited. The limited success of these methods was highly dependent on the randomised starting solutions. This approach differs from that of other models, such as the Nelder-Mead simplex and pattern search, which required initial guess values. To visualise repeatability, Table 5.2 illustrates the variability in solutions in ten identical runs. The optimal settings for the population and particles used, as determined from previous results, are presented in Table 5.1.

**Table 5.2:** Optimiser performance for test Case A, repeatability represented by variety on error and standard deviation of E-modulus.

| | Error, Young´s modulus (E) [%] | | | |
|---|---|---|---|---|
| Optimiser type | Lowest | Mean | Highest | Std Dev |
| Nelder-Mead Simplex | -0.0011 | -0.0011 | -0.0011 | 0 |
| Pattern Search | $3.57 \times 10^{-4}$ | $3.57 \times 10^{-4}$ | $3.57 \times 10^{-4}$ | 0 |
| Genetic Algorithm (30) | -1.15 | 0.013 | 0.69 | $2.32 \times 10^{8}$ |
| Particle Swarm (5) | -0.031 | -0.0047 | 0.036 | $3.75 \times 10^{6}$ |

In summary, the high variety and instability in solutions can be observed in the biological methods, with the genetic algorithm having the largest variation. This instability was attributed to the relatively low maximum runtime, which prevented both biological methods from converging within a reasonable solution space. Test Case A clearly demonstrates the apparent lack of robustness in biological methods. Furthermore, it is observed that particle swarm optimisation with five particles does not yield a more accurate solution, on average, compared to pattern search. Consequently, the pattern search demonstrates superior performance and stability for test Case A.

### Case B

Next, Case B slightly increased complexity. Following similar reasoning as in Case A, the simpler methods were not expected to struggle, while the more advanced methods were expected to catch up in performance.

Nelder-Mead simplex, run on auto settings, reached its tolerance criteria of $1 \times 10^{-4}$ at a higher runtime of 2932.6 seconds for Case B. This was thereby the upper limit set for each of the remaining methods.

The results of the optimisers can be observed in Table 5.3. It can be seen that the Nelder-Mead simplex outperformed the pattern search for increased complexity. Compared to the simpler test Case A, the pattern search lost its advantage in both accuracy and computational time compared to the Nelder-Mead simplex method. However, both of these simpler methods continued to outperform the more complex biological methods, generally achieving better accuracy within the same runtime.

**Table 5.3:** Optimiser performance for test Case B, based on a single result set with runtime (RT) limit of 2932.6 seconds.

| | Optimiser performance | | | | |
|---|---|---|---|---|---|
| Optimiser type | Error, E [%] | Error, K [%] | Mean Error [%] | Iterations | RT [s] |
| Nelder-Mead Simplex | $1.08 \times 10^{-4}$ | $5.13 \times 10^{-5}$ | $7.99 \times 10^{-5}$ | 88 | 2932.6 |
| Pattern Search | 0.0017 | 0.010 | 0.006 | 92 | 2967.8 |
| Genetic Algorithm (40) | 0.76 | 6.26 | 3.51 | 6 | 3322.1 |
| Genetic Algorithm (60) | 0.083 | 1.80 | 0.94 | 4 | 3579.5 |
| Genetic Algorithm (75) | 0.24 | 2.40 | 1.32 | 2 | 3605.6 |
| Particle Swarm (10) | 0.15 | 0.71 | 0.43 | 23 | 2958.4 |
| Particle Swarm (15) | 0.049 | 0.48 | 0.26 | 15 | 2963.2 |
| Particle Swarm (20) | 0.26 | 10.75 | 2.75 | 11 | 2957.2 |

To summarise, for the two-variable problem, the simpler Nelder-Mead simplex method still outperformed both the genetic algorithm and particle swarm optimisation. Analysis shows that particle swarm optimisation performed noticeably better in accuracy compared to the genetic algorithm for a particle size of 15. Interestingly, the genetic algorithm improved its relative performance with increasing complexity. Furthermore, it is observed that both the genetic algorithm and particle swarm optimisation seemed less sensitive to changes in solution size (population/particles), although they remained sensitive to some extent.

Similarly to test Case A, both the Nelder-Mead simplex and pattern search methods were observed to be stable in their solutions, proving to be completely repeatable. It was further noted that the simpler methods once again demonstrated better performance compared to those inspired by nature. Therefore, a study was conducted to assess the impact of initial guesses. The study of the initial values is presented in Table 5.4. From this, it can be concluded that the initial guess had an insignificant impact on performance for Case B, with guesses near the boundaries sometimes even slightly improving performance. It was noted that pattern search struggled slightly more in the case of the upper bound, while the Nelder-Mead simplex seemed to converge within a shorter time span. This study highlights that the initial guess had an insignificant effect and thus could not be considered the cause of the outstanding performance observed in the previous study, as shown in Table 5.3.

**Table 5.4:** Optimiser performance for test Case B, with different initial guesses, still limited to 2932.6 seconds in runtime (RT).

| | Optimiser performance | | | | |
|---|---|---|---|---|---|
| Optimiser, LB Guess | Error, E [%] | Error, K [%] | Mean Error [%] | Iterations | RT [s] |
| Nelder-Mead Simplex | $2.58 \times 10^{-5}$ | $8.42 \times 10^{-6}$ | $1.71 \times 10^{-5}$ | 88 | 2948.6 |
| Pattern Search | 0.0017 | 0.015 | 0.0084 | 89 | 2967.8 |
| Optimiser, Mean Guess | Error, E [%] | Error, K [%] | Mean Error [%] | Iterations | RT [s] |
| Nelder-Mead Simplex | $1.08 \times 10^{-4}$ | $5.13 \times 10^{-5}$ | $7.97 \times 10^{-5}$ | 88 | 2932.6 |
| Pattern Search | 0.0017 | 0.010 | 0.0059 | 92 | 2961.8 |
| Optimiser, UB Guess | Error, E [%] | Error, K [%] | Mean Error [%] | Iterations | RT [s] |
| Nelder-Mead Simplex | $5.92 \times 10^{-5}$ | $2.1 \times 10^{-5}$ | $4.01 \times 10^{-5}$ | 77 | 2644.2 |
| Pattern Search | 0.0101 | 0.050 | 0.030 | 89 | 2972.8 |

Test Case B demonstrated that the simpler methods still performed better for an equal runtime. However, it is evident that simpler methods, especially pattern search, have lost their advantages in terms of runtime, with the genetic algorithm showing increased accuracy and better stability. Although particle swarm optimisation lost accuracy compared to test Case A, its results more closely resembled those of the genetic algorithm, indicating a more even performance between the two biological methods.

### Case C

For the last case, test Case C, with the highest complexity of the test cases. The increase in complexity and computational demand was expected to slightly benefit biological methods, namely the genetic algorithm and particle swarm optimisation.

For the increase in complexity, the Nelder-Mead simplex method required 5968.8 seconds to reach its tolerance criteria, almost double the runtime limit compared to Case B.

The results from test Case C are presented in Table 5.5, where K1 represents the left spring support, K2 the corresponding right spring support, and KR the intermediate rotational spring, as illustrated in Figure 4.1. All methods reached the maximum runtime limit established by the Nelder-Mead simplex method. From the results, it can once again be noted that the Nelder-Mead simplex achieved high accuracy for all variables and did not seem to struggle with scaling complexity. The pattern search continued to perform well, ranking second, ahead of particle swarm optimisation. For the third consecutive test, the genetic algorithm showed the lowest performance.

**Table 5.5:** Optimiser performance for test Case C, based on a single result set with a runtime limit of 5968.8 seconds.

| | Variable Error [%] | | | | |
|---|---|---|---|---|---|
| Optimiser type | E | K1 | K2 | KR | Mean |
| Nelder-Mead Simplex | $1.30 \times 10^{-5}$ | $7.83 \times 10^{-5}$ | $6.49 \times 10^{-6}$ | $5.35 \times 10^{-5}$ | $3.78 \times 10^{-5}$ |
| Pattern Search | 0.030 | 0.022 | 0.064 | 0.022 | 0.034 |
| Genetic Algorithm (80) | 6.89 | 0.39 | 52.55 | 28.28 | 22.03 |
| Genetic Algorithm (100) | 1.23 | 4.81 | 17.98 | 0.25 | 6.07 |
| Genetic Algorithm (120) | 0.22 | 15.90 | 3.48 | 38.03 | 14.41 |
| Particle Swarm (10) | 2.43 | 7.84 | 1.50 | 11.46 | 5.81 |
| Particle Swarm (20) | 0.50 | 0.98 | 1.03 | 2.63 | 1.28 |
| Particle Swarm (30) | 5.93 | 4.46 | 12.79 | 0.73 | 5.98 |

In conclusion of the primary study for Case C, the simpler methods yielded more accurate results within the set computational time. It is also observed that particle swarm optimisation outperformed the genetic algorithm, offering the most accurate and efficient solutions among the two biologically inspired methods.

It is important to acknowledge that the primary objective function was chosen to be detailed, containing deflections, sectional forces, and sectional moments. For final comparisons, Case C with a simplified objective function, considering only deflections, was used to evaluate performance with less comprehensive data. Once again, the Nelder-Mead simplex sets the maximum runtime allowed, restricting runtime for other methods. A new runtime limit was set for this "side-track," which took 6778.9 seconds.

The most beneficial population and particle sizes from the previous study were selected for the test, as shown in Table 5.5. The results of this study indicated that the Nelder-Mead simplex method again achieved the highest accuracy, see Table 5.6. In contrast to the previous study, the genetic algorithm yielded the second-highest accuracy for this analysis. Particle swarm optimisation ranked third in accuracy, with pattern search showing the lowest performance.

**Table 5.6:** Optimiser performance for test Case C, based on a single result set with a runtime limit of 6778.9 seconds.

| | Variable Error [%] | | | | |
|---|---|---|---|---|---|
| Optimiser type | E | K1 | K2 | KR | Mean |
| Nelder-Mead Simplex | $3.60 \times 10^{-4}$ | $3.43 \times 10^{-4}$ | 0.0011 | 0.0042 | 0.0015 |
| Pattern Search | 3.74 | 0.86 | 0.81 | 40.33 | 11.43 |
| Genetic Algorithm (80) | 2.20 | 4.22 | 12.18 | 18.30 | 9.22 |
| Genetic Algorithm (100) | 11.14 | 16.10 | 8.51 | 54.69 | 22.61 |
| Genetic Algorithm (120) | 5.37 | 4.25 | 11.77 | 7.81 | 7.30 |
| Particle Swarm (10) | 3.12 | 1.17 | 0.16 | 33.59 | 9.51 |
| Particle Swarm (20) | 6.05 | 4.51 | 1.51 | 38.65 | 12.68 |
| Particle Swarm (30) | 5.90 | 3.40 | 7.60 | 35.65 | 13.14 |

Noteworthy is the change in mean error between the cases of the complete objective function and the corresponding reduced one, comparing results in Table 5.5 against Table 5.6. Although the Nelder-Mead simplex method showed the highest accuracy, the mean error increased by a factor of almost 40. The most drastic increase was observed in the pattern search, which saw an increase in mean error by a factor of 333. In contrast, both biological methods showed significantly lower increases, with the genetic algorithm indicating an increase factor of 1.16 and particle swarm optimisation showing a corresponding increase in error factor of 7.43.

In conclusion, considering all previously performed tests, the simpler Nelder-Mead simplex method generally outperformed the other methods. For the Nelder-Mead simplex, both the final accuracy and computational runtime were generally superior. However, it is evident that the genetic algorithm emerges as a more robust model in the context of less precisely formulated objective functions, making it more suitable in cases of data scarcity.

### Conclusion test cases

A final re-evaluation of each previously studied test case was carried out, focussing solely on the best configurations without runtime limitations. The final results are presented in Table 5.7. It should be noted that the performance of the Nelder-Mead simplex method was good throughout the study, demonstrating its efficiency and robustness. Subsequently, pattern search almost consistently matched accuracy and offered a comparable runtime. The genetic algorithm appeared to have difficulty converging, yielding the least accurate results over notably the longest duration. Lastly, particle swarm yielded relatively accurate results in a shorter duration compared to the genetic algorithm, approaching the performance of pattern search.

**Table 5.7:** Optimiser performance for each test case, with unlimited runtime (RT) and default convergence criteria.

| | Magnitude of Error | | | | | Training Time | |
|---|---|---|---|---|---|---|---|
| Optimiser Case A | E [%] | K1 [%] | K2 [%] | KR [%] | Mean [%] | Iterations | RT [s] |
| Nelder-Mead Simplex | 0.0011 | - | - | - | 0.001 | 45 | 1447.4 |
| Pattern Search | $3.57 \times 10^{-4}$ | - | - | - | $3.57 \times 10^{-4}$ | 32 | 648.6 |
| Genetic Algorithm(30) | 0.19 | - | - | - | 0.19 | 50 | 17424.5 |
| Particle Swarm(5) | $5.33 \times 10^{-4}$ | - | - | - | $5.33 \times 10^{-4}$ | 21 | 1320.4 |
| Optimiser Case B | E [%] | K1 [%] | K2 [%] | KR [%] | Mean [%] | Iterations | RT [s] |
| Nelder-Mead Simplex | $1.08 \times 10^{-5}$ | $5.13 \times 10^{-5}$ | - | - | $7.99 \times 10^{-5}$ | 88 | 2932.6 |
| Pattern Search | $1.15 \times 10^{-5}$ | $1.1 \times 10^{-4}$ | - | - | $6.06 \times 10^{-5}$ | 88 | 3224.9 |
| Genetic Algorithm(60) | 0.23 | 1.26 | - | - | 0.74 | 50 | 37673.8 |
| Particle swarm(15) | $1.05 \times 10^{-4}$ | 0.0057 | - | - | 0.0029 | 34 | 6312.1 |
| Optimiser Case C | E [%] | K1 [%] | K2 [%] | KR [%] | Mean [%] | Iterations | RT [s] |
| Nelder-Mead Simplex | $1.30 \times 10^{-5}$ | $7.83 \times 10^{-5}$ | $6.49 \times 10^{-6}$ | $5.35 \times 10^{-5}$ | $3.78 \times 10^{-5}$ | 197 | 5968.8 |
| Pattern Search | $3.57 \times 10^{-4}$ | $1.54 \times 10^{-4}$ | 0.0011 | $1.47 \times 10^{-4}$ | $4.31 \times 10^{-4}$ | 162 | 12513.7 |
| Genetic Algorithm(100) | 1.55 | 2.22 | 25.28 | 2.31 | 7.84 | 46 | 72577.5 |
| Particle Swarm(20) | 0.011 | 0.050 | 0.029 | 0.068 | 0.039 | 56 | 14717.4 |

Considering these results, in combination with those presented in each test case subsection, the Nelder-Mead simplex method evidently performs the best overall. It produced the least error in two out of three cases, yielded reproducible solutions, was relatively unaffected by the initial guess, and performed well with scarce input. This method was successful in almost all tests. However, it must also be acknowledged that the complexity might be insufficient to reach the limitations of the method. However, the results of Table 5.6 revealed an unexpected degree of robustness for the genetic algorithm. Although all other methods experienced significant decreases in accuracy, the genetic algorithm exhibited little to no change in mean error. This showcases the generality of the genetic algorithm in finding solutions, even with data scarcity and increased complexity.

### 5.1.2 Neural Network method

The following and also proposed alternative method of optimisation was through neural networks. For each case, developments, tests, and comparisons between different NN models are carried out. This includes both the ordinary feedforward (FF) and the feedforward reversed (FFR). For descriptions and details regarding their setup, underlying intentions, and methodology for the studies performed, see Chapter 4.1.2.

#### Case A

As previously described, Case A involves the simplest of the cases. It sets a performance basis considering the least demanding system through a single variable analysis and is the perfect candidate for development and evaluation of the NN.

##### FF - Feedforward

As a primary study, determining the appropriate architectural size of the NN was important to verify the appropriate models. Thus, an extensive study of the effects of both the number and the size of the hidden layers was conducted. With the convergence criteria described in Section 4.1.2, the time to train individual models (TM), the number of trainable parameters in the network, and the acquired Mean Absolute Percentage Error (MAPE) in deflection can be found in Table 5.8.

**Table 5.8:** Neural network size comparison. Performance of the prediction of feedforward for Case A. Training based on 80% of the data set, with remaining data were kept as validation of the solutions. Based on 5070 sets of solutions.

| | Optimiser performance for deflection | | |
|---|---|---|---|
| Model | MAPE, U [%] | Runtime, TM [s] | Parameters [nr] |
| FF-2L8N | 5.59 | 104.3 | 137 |
| FF-4L8N | 3.17 | 112.2 | 281 |
| FF-8L8N | 2.44 | 129.9 | 569 |
| FF-2L16N | 6.14 | 103.8 | 401 |
| FF-4L16N | 2.21 | 109.4 | 945 |
| FF-8L16N | 1.45 | 124.1 | $2.033 \times 10^3$ |
| FF-2L32N | 1.59 | 101.9 | $1.313 \times 10^3$ |
| FF-4L32N | 1.61 | 91.9 | $3.225 \times 10^3$ |
| FF-8L32N | 2.20 | 112.7 | $7.649 \times 10^3$ |
| FF-2L64N | 1.35 | 97.8 | $4.673 \times 10^3$ |
| FF-4L64N | 1.19 | 101.9 | $12.993 \times 10^3$ |
| FF-8L64N | 1.45 | 102.5 | $29.633 \times 10^3$ |
| FF-2L128N | 1.39 | 93.9 | $17.537 \times 10^3$ |
| FF-4L128N | 1.28 | 106.1 | $50.561 \times 10^3$ |
| FF-8L128N | 1.39 | 118.9 | $116.609 \times 10^3$ |
| FF-2L256N | 1.25 | 103.3 | $67.841 \times 10^3$ |
| FF-4L256N | 1.26 | 106.4 | $199.425 \times 10^3$ |
| FF-8L256N | 1.37 | 202.2 | $462.593 \times 10^3$ |
| FF-2L512N | 1.53 | 106.1 | $266.753 \times 10^3$ |
| FF-4L512N | 1.22 | 309.4 | $792.065 \times 10^3$ |
| FF-8L512N | 1.48 | 398.2 | $1.843 \times 10^6$ |
| FF-2L1024N | 1.79 | 610.8 | $1.058 \times 10^6$ |
| FF-4L1024N | 1.01 | 932.5 | $3.157 \times 10^6$ |
| FF-8L1024N | 1.49 | 1200.2 | $7.355 \times 10^6$ |

The results from Table 5.8 indicate a general decrease in error for larger network sizes. In addition, with an increase in the number of neurones, networks tend to produce more stable outcomes. The results show larger errors for systems with few neurones per layer, with improvements as the number of neurones increases. However, once 32 neurones were reached, the rate of improvement decreased and eventually stopped. Lastly, it should be noted that a neurone size of 64 to 256 indicated little change in training time, while sizes beyond 256 required significantly longer time to train.

Apart from architectural size, the method of optimising weights and biases may change the outcome. Therefore, a further "side-track" study of sensitivity was conducted with respect to different optimisation methods, in relation to different network sizes. For the number of layers, the upper and lower boundaries of two and eight from the previous study were used (see Table 5.8), in combination with a range span of neurones each. The results of the optimiser study can be seen in Table 5.9.

**Table 5.9:** Neural network optimiser and size comparison. Feedforward prediction performance for Case A. Training based on 80% of the data set, with remaining data were kept as validation of the solutions. Based on 5070 sets of solutions.

| | | Optimiser performance for deflection | |
|---|---|---|---|
| Model | Optimiser | MAPE, U [%] | Runtime, TM [s] |
| FF-2L32N | SGD | 11.05 | 107.1 |
| FF-2L512N | SGD | 6.89 | 163.5 |
| FF-8L32N | SGD | 8.59 | 123.6 |
| FF-8L512N | SGD | 3.19 | 417.4 |
| FF-2L32N | Adam | 1.82 | 106.5 |
| FF-2L512N | Adam | 1.77 | 93.1 |
| FF-8L32N | Adam | 1.51 | 91.4 |
| FF-8L512N | Adam | 1.03 | 587.1 |
| FF-2L32N | RMSprop | 3.18 | 62.6 |
| FF-2L512N | RMSprop | 1.75 | 167.7 |
| FF-8L32N | RMSprop | 3.18 | 67.1 |
| FF-8L512N | RMSprop | 2.65 | 342.4 |

The optimisation study highlights some major benefits and limitations of various systems. The SGD optimiser proved the least performing method regardless of network size. Following, with the second-lowest performance, the method of RMSprop is found. Although RMSprop indicated better performance for one of the performed setups, high sensitivity to network size could be observed. The best performing optimiser was Adam, which commonly yielded the lowest errors while being relatively insensitive to network size. It should be noted that Adam was used as the default optimiser for all remaining analyses presented. With an appropriate model of choice, the final deflection prediction was visualised. Figure 5.1 shows the deflected behaviour along the beam, comparing the "true" analytical solution against the predicted deflections. The model of choice was FF-4L512N due to the promising performance indicated by the study presented in Table 5.8. From the deflected shape, it can be concluded that the network adequately captures the correct behaviour.

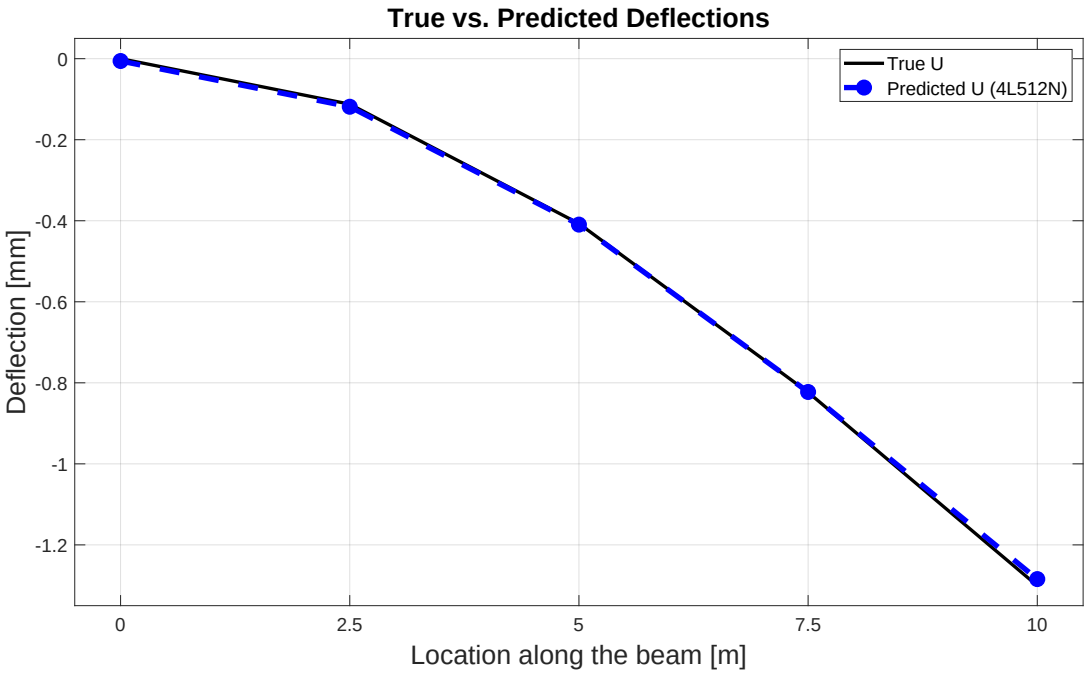



**Figure 5.1:** Visualisation of neural networks feedforward prediction of displacement (U) along the beam, for test Case A.

#### FFR - Feedforward reversed

With a better understanding of appropriate network sizes and the effect of different optimisers, the FFR model was established and evaluated, deriving unknown variables through input of known displacements. For various configurations, average errors for the predicted E-modulus are presented, both as mean and median values, along with the required training time, as shown in Table 5.10.

**Table 5.10:** Neural network size and batch comparison. Feedforward reversed prediction performance for test Case A. Training based on 80% of the total data set, with remaining data were kept as validation of the solutions. Based on 5070 sets of solutions.

| | Optimiser performance for deflection | | |
|---|---|---|---|
| Model, Batch (beams/batch) | Mean Error, E [%] | Median Error, E [%] | Runtime, TM [s] |
| FFR-4L512N, Batch: 128 (32) | 3.35 | 1.60 | 146.7 |
| FFR-4L512N, Batch: 256 (64) | 3.80 | 1.98 | 108.1 |
| FFR-4L512N, Batch: 512 (128) | 3.53 | 1.63 | 111.6 |

The results indicated rather stable solutions across three different batch sizes, indicating insensitivity against such a setting. Notable is the large difference between the mean and median errors. As the model required a displacement input for each node, it also outputs a guessed variable for each respective node. Thus, along the

beam, the guessed E-modulus showed some variance. The higher mean value results from the large anomalies encountered in locations where the model struggled. For instance, predictions near the fixed boundary. The model thus produced noticeable anomalies at certain nodes more often than others, which impacted an average mean value. However, for the majority of the nodes, a less deviant approximation was made, which was reflected in the median value.

For reference, a test set collected through FFR-4L512N with a batch size of 256 (see Table 5.10) is visualised in Figure 5.2. Note that two curves are depicted: the results of the FFR models with or without consideration of the point at x = 0 m. The result highlights that the network that considers the fixed point at x = 0 generally performed worse along the beam. The reason was that the model cannot determine a unique value for the fixed node, thus consistently assigning a mean value based on the whole training data set. For the range of 30 - 40 GPa for the E-modulus, regardless of which deflections were given as input, an initial guess of around 35 GPa was given for the first node. With this discovery, the remaining FFR results are obtained with the exclusion of the node at x = 0.

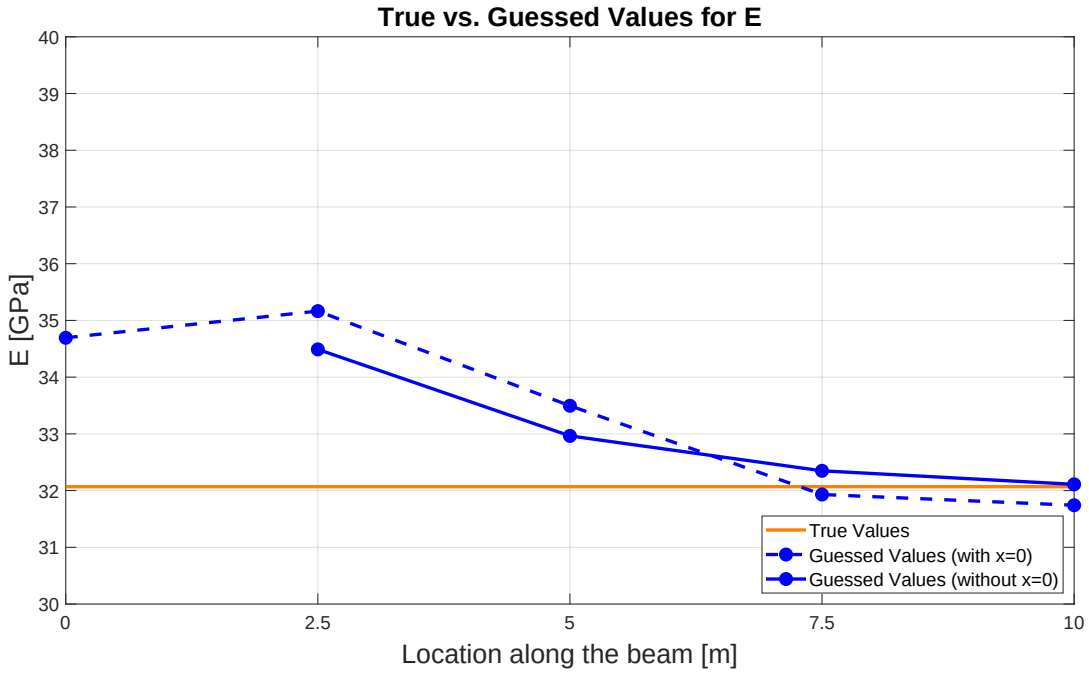



**Figure 5.2:** Visualisation of the neural networks variation of approximation along the beam, for test Case A.

Although the results in Table 5.10 indicate low training times, this only holds for the complete training data set. The time required to produce the training data was 30 times the training time, while simultaneously resulting in an accuracy that was worse compared to the least efficient model presented for Case A regarding conventional methods (see Table 5.1). Therefore, there was no need to evaluate performance for down-scaled training data, as the FFR model was evidently less efficient.

### Case B

For Case B, similar principles laid the basis for continued studies. With the extensive study of network size performed for Case A, appropriate ranges were assumed to facilitate network scaling. Thus, similar settings and methods are carried over, excluding the models of the smallest sizes that indicated less performance (see previous analysis in Table A.1). This as the more complex case was expected to require larger models in response.

#### FF - Feedforward

Primarily, the main size study was re-evaluated for Case B. The convergence criteria described in Section 3.3.1 were still relevant, and the time required to train individual models (TM) in combination with the acquired mean absolute percentage error (MAPE) in deflection can be found in Table 5.11.

**Table 5.11:** Neural network size comparison. Feedforward prediction performance for Case B. Training based on 80% of the data set, with remaining data were kept as validation of the solutions. Based on 11856 sets of solutions.

| | Optimiser performance for deflection | |
|---|---|---|
| Model | MAPE, U [%] | Runtime, TM [s] |
| FF-4L128N | 0.83 | 202.5 |
| FF-8L128N | 0.82 | 240.0 |
| FF-16L128N | 18.93 | 524.9 |
| FF-4L256N | 0.73 | 366.2 |
| FF-8L256N | 0.79 | 374.3 |
| FF-16L256N | 0.72 | 601.8 |
| FF-4L512N | 0.99 | 493.5 |
| FF-8L512N | 0.79 | 832.9 |
| FF-16L512N | 1.15 | 2208.1 |
| FF-4L1024N | 1.12 | 1435.5 |
| FF-8L1024N | 0.61 | 4557.3 |
| FF-16L1024N | 0.84 | 8973.1 |

The results presented in Table 5.11 generally correlated with those of the architecture size analysis study for Case A, as seen in Table 5.8. With a larger network size, a decrease in error can be observed. Furthermore, the high error anomaly for the 16 layer and 128 neurone (16L128N) network should be noted. The high error for an increased network size indicates that the model started overfitting the data as a result of having too few neurones for too many hidden layers. Additionally, it can be observed that a size of 256 neurones per hidden layer yields a more stable outcome with respect to the number of layers. Beyond 256 neurones, significant increases in training time and greater sensitivity to the number of hidden layers could be observed.

Additionally, the previous assumption that the activation function of ReLU was appropriate was tested for small and large architectural sizes, considering either min-max scaling or standard scalar methods. The results of such an analysis can be seen in Table 5.12.

**Table 5.12:** Neural network activation function and scaling comparison. Feedforward prediction performance for Case B. Training based on 80% of the data set, with remaining data were kept as validation of the solutions. Based on 11856 sets of solutions.

| | | Min-max scalar | | Standard scalar | |
|---|---|---|---|---|---|
| Model | Activation function | MAPE, U [%] | TM [s] | MAPE, U [%] | TM [s] |
| FF-4L32N | ReLU | 1.47 | 352.7 | 1.21 | 211.6 |
| FF-4L512N | ReLU | 1.05 | 379.1 | 0.85 | 439.2 |
| FF-16L32N | ReLU | 1.35 | 278.4 | 1.01 | 347.3 |
| FF-16L512N | ReLU | 0.82 | 2685.9 | 1.19 | 1049.0 |
| FF-4L32N | Tanh | 0.96 | 205.3 | 0.84 | 245.6 |
| FF-4L512N | Tanh | 4.89 | 608.1 | 4.79 | 481.1 |
| FF-16L32N | Tanh | 1.75 | 295.8 | 2.62 | 192.0 |
| FF-16L512N | Tanh | 105.68 | 1460.3 | 91.36 | 770.8 |
| FF-4L32N | ELU | 1.36 | 92.6 | 1.45 | 143.5 |
| FF-4L512N | ELU | 2.71 | 1022.2 | 2.14 | 641.5 |
| FF-16L32N | ELU | 0.88 | 293.6 | 1.57 | 175.8 |
| FF-16L512N | ELU | 26.25 | 655.1 | 30.28 | 648.3 |

The analysis studying the impact of activation functions in combination with scaling, as seen in Table 5.12, indicated that the activation function of ReLU resulted in the most robust results. Although it yielded a relatively low error, it is also consistent regardless of the scaling method. Activation functions such as Tanh or ELU resulted in higher errors while also being highly sensitive to network size. Although some functions could match the ReLU error, consistency and robustness were not observed for the other methods.

Through determination of appropriate architectural size and activation function, the final deflection prediction was once again visualised. For results, see Figure 5.3 which compares correct against predicted deflections. The model of choice was FF-8L512N as it indicated the most appropriate relation of accuracy and computational demand, see Table 5.11. The figure indicates a close correlation, indicating sufficient accuracy for the approximated deflection.

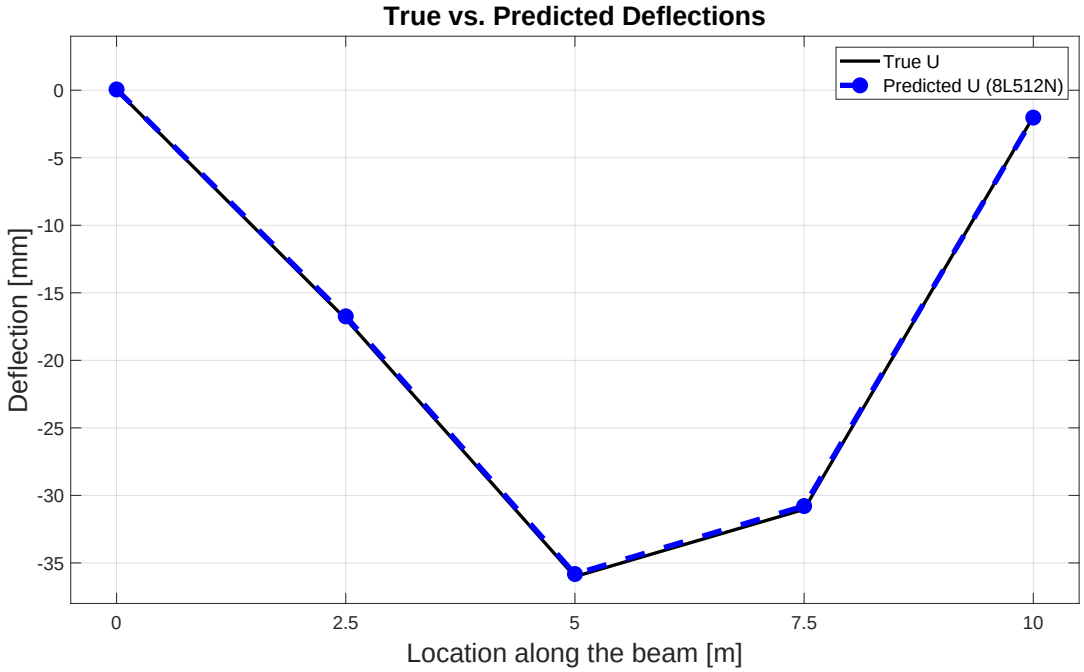



**Figure 5.3:** Visualisation of neural networks feedforward prediction of displacement (U) along the beam, for Case B.

#### FFR - Feedforward reversed

Similarly to Case A, with the findings of appropriate settings, the FFR model was established. Likewise, the variables of the E-modulus and K stiffness were derived from known displacements. As the study of FFR models for Case A revealed negative effects if considering the first node, this was therefore excluded for this application. The results, considering both mean and median values, can be found in Table 5.13.

**Table 5.13:** Neural network size and batch comparison. Feedforward reversed prediction performance for test Case B. Training based on 80% of the data set, the remaining data were kept as validation of the solutions. Based on 11856 sets of solutions.

| | Magnitude of Error | | | | Runtime |
|---|---|---|---|---|---|
| Model, Batch (beams/batch) | Mean, E [%] | Mean, K [%] | Median, E [%] | Median, K [%] | TM [s] |
| FFR-8L512N, Batch: 128(32) | 4.14 | 93.29 | 2.15 | 37.61 | 843.9 |
| FFR-8L512N, Batch: 256(64) | 4.08 | 89.56 | 1.91 | 36.27 | 605.5 |
| FFR-8L512N, Batch: 512(128) | 4.02 | 80.04 | 1.95 | 35.55 | 464.6 |

The result follows the trend observed from Case A, where the median value yields better results compared to the mean values. However, it should be noted that there was a significant increase in the difference between the mean and median with respect to the stiffness of the spring support (K). Upon examining how the FFR model approximates the stiffness of the spring along the beam, it became apparent that the best guesses are made at the locations where the spring was found (see Figure 5.4). This follows the trend of the FFR analysis for Case A, where better approximations were established where the largest impact of variables was seen. Thus, for the spring support, clear improvements in the approximation were observed at the end of the beam, at location x = 10. Furthermore, the opposite was observed for the approximation of the E-modulus, where the accuracy decreased at locations where other parameters had a high effect. Thus, a trend of internal variable conflict was observed, where improvements in one variable often resulted in worse outcomes for others.

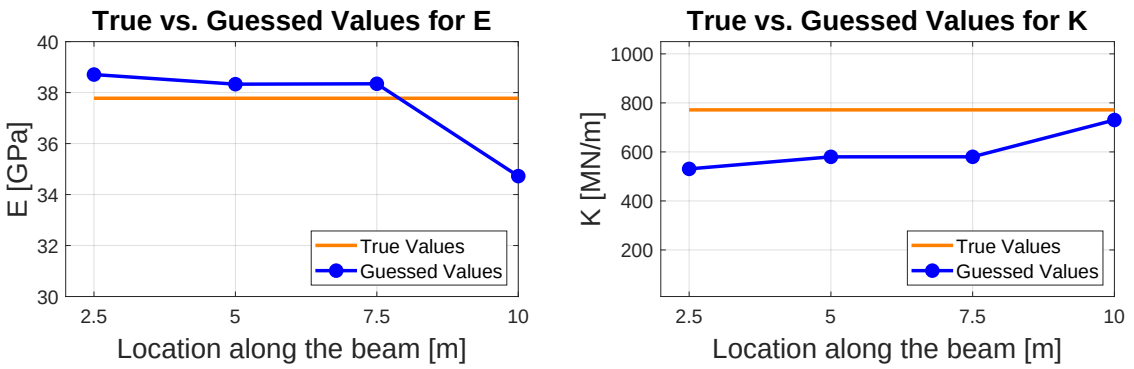



**Figure 5.4:** Visualisation of the variation in neural network approximations along the beam, for Case B.

Therefore, if the variable was recovered based on where it approximated the best, it could drastically improve the outcome. This can be seen in Table 5.14, where K was derived from the approximations made at the end of the beam.

**Table 5.14:** Neural network. Feedforward reversed prediction performance for test Case B. Considers best locations of variable retrieval. Training based on 80% of the data set, with the remaining data were kept as validation of the solutions. Based on 11856 sets of solutions.

| | Custom Point Error | | Runtime |
|---|---|---|---|
| Model, Batch (beams/batch) | E [%] | K [%] | TM [s] |
| FFR-8L512N, Batch: 128 (32) | 2.15 | 19.97 | 843.9 |
| FFR-8L512N, Batch: 256 (64) | 1.91 | 21.74 | 605.5 |
| FFR-8L512N, Batch: 512 (128) | 1.95 | 22.03 | 464.6 |

Although significant improvements were noted, the error achieved was noticeably larger compared to that retrieved by conventional methods. Although the training time was roughly only 20% of the runtime achieved through corresponding conventional methods, the result relied on the full training data set. Thus, considering the time required to acquire training data and train the model, the FFR system required at least 40 times the runtime compared to conventional methods. Therefore, it was deemed unnecessary to downscale the data as the network could not compete even with the entire data set.

### Case C

Lastly, the complexity was further increased in Case C. This complexity arose from the extension of the beam element by an additional beam segment. The extension introduced two additional variables. An support with a spring stiffness, K2, and an intermediately connected rotational spring with spring stiffness, KR, as seen in Figure 4.1. The support and rotational spring results in potential impact for both the FF and FFR studies.

#### FF - Feedforward

For the last case, Case C, a new study of the appropriate size of the network was performed. The convergence criteria described in Section 3.3.1 were still considered, and the time required to train individual models (TM) in combination with the acquired mean absolute percentage error (MAPE) in deflection can be found in Table 5.15.

**Table 5.15:** Neural network size comparison. Feedforward prediction performance for Case C. Training based on 80% of the data set, with the remaining data were kept as validation solutions. Based on 15435 sets of solutions.

| | Optimiser performance for deflection | |
|---|---|---|
| Model | MAPE, U [%] | Runtime (TM) [s] |
| FF-4L128N | 1.49 | 672.2 |
| FF-8L128N | 1.22 | 784.9 |
| FF-16L128N | 0.66 | 1079.1 |
| FF-32L128N | 0.70 | 4285.1 |
| FF-4L256N | 0.89 | 830.4 |
| FF-8L256N | 1.04 | 830.1 |
| FF-16L256N | 0.95 | 1084.3 |
| FF-32L256N | 1.72 | 4769.8 |
| FF-4L512N | 1.63 | 2071.5 |
| FF-8L512N | 0.98 | 1893.5 |
| FF-16L512N | 0.70 | 1976.0 |
| FF-32L512N | 98.09 | 3212.7 |
| FF-4L1024N | 1.07 | 3756.6 |
| FF-8L1024N | 0.90 | 11740.2 |
| FF-16L1024N | 1.20 | 28099.8 |
| FF-32L1024N | 98.59 | 13317.7 |

The results presented in Table 5.15 follow previous trends of generally lower error for larger systems. Although, this seems to hold true, a trend was also noted in 3 out of 4 cases where 16 hidden layers generate a lower error for a given set of neurones. Furthermore, the time required to train each model drastically increased for systems with 32 layers, often also with higher error.

After examining appropriate network sizes, the final deflection prediction was for the last test case, Case C. For results, see Figure 5.5 which compares correct against predicted deflections. The model of choice was FF-16L512N for Case C, in accordance with the good performance obtained in the previous study, see Table 5.15. The figure indicates a close correlation, indicating a sufficient accuracy in approximated deflection.

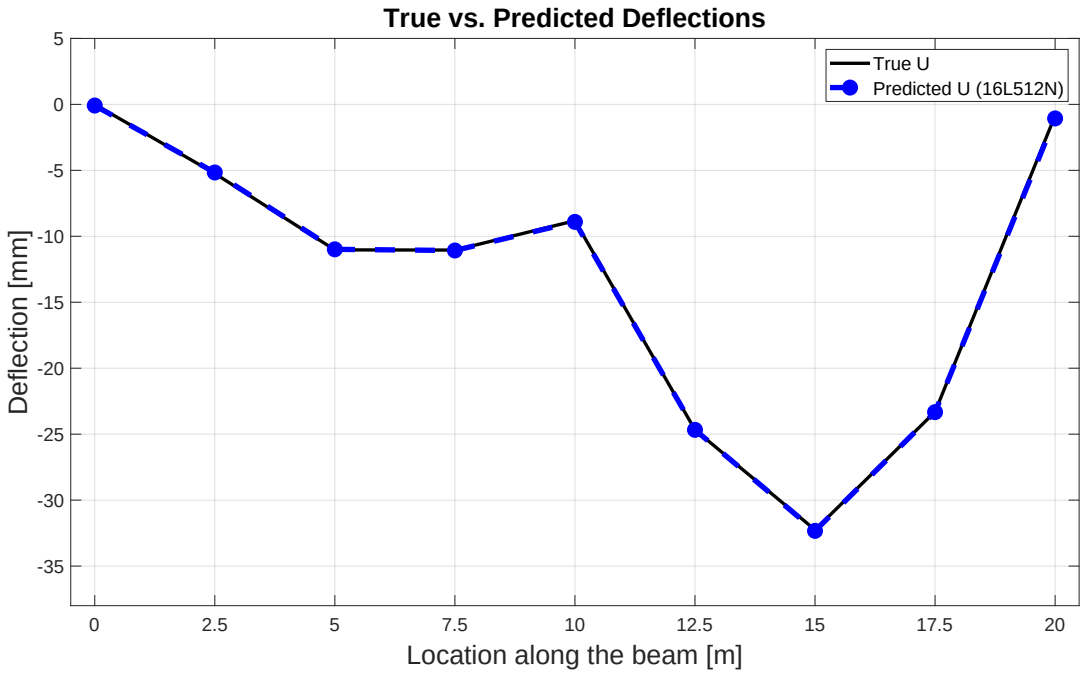



**Figure 5.5:** Visualisation of neural networks feedforward prediction for displacement (U) along the beam, for test Case C.

#### FFR - Feedforward reversed

For the reversed-order system, FFR, which approximates variables based on known displacements, a more thorough batch size analysis was conducted to evaluate the possible effect different batch sizes may have. The results of this analysis can be studied in Table 5.16. It is important to note that the table assumes the number of batches as chosen sets of known solutions. As the beam has 8 individual data point locations (excluding x = 0), a batch size of 8 would result in a single batch of complete beam solutions.

**Table 5.16:** Neural network size and batch comparison. Feedforward reversed prediction performance for test Case C. Training based on 80% of the data set, with remaining data were kept as validation solutions. Based on 15435 sets of solutions.

| | Mean error | | | | Median error | | | | Runtime |
|---|---|---|---|---|---|---|---|---|---|
| Model, Batch (beams/batch) | E [%] | K1 [%] | K2 [%] | KR [%] | E [%] | K1 [%] | K2 [%] | KR [%] | TM [s] |
| FFR-16L512N, Batch:8(1) | 8.70 | 721.27 | 679.82 | 661.32 | 8.70 | 721.27 | 679.82 | 661.31 | 5135.7 |
| FFR-16L512N, Batch:32(4) | 7.50 | 168.88 | 362.09 | 354.37 | 7.23 | 64.75 | 366.56 | 274.51 | 4028.2 |
| FFR-16L512N, Batch:64(8) | 7.45 | 156.21 | 379.43 | 363.37 | 7.26 | 60.56 | 369.49 | 303.32 | 3490.4 |
| FFR-16L512N, Batch:128(16) | 7.23 | 153.65 | 372.49 | 383.43 | 7.02 | 57.18 | 354.91 | 314.10 | 2828.1 |
| FFR-16L512N, Batch:256(32) | 7.37 | 163.00 | 366.78 | 319.93 | 7.36 | 51.57 | 365.43 | 240.93 | 2361.7 |
| FFR-16L512N, Batch:512(64) | 7.22 | 125.77 | 344.31 | 309.90 | 6.97 | 49.60 | 313.21 | 252.95 | 2016.8 |
| FFR-16L512N, Batch:1024(128) | 7.10 | 111.79 | 405.30 | 309.40 | 6.85 | 48.90 | 371.04 | 239.62 | 2047.1 |

The results highlight multiple areas of interest. For the smallest batch size, representing only one beam, the network stopped learning after some iterations and suffered from dying neurones. Additionally, the resulting error for the rotational spring (KR) seemed to be the most difficult for the network to learn. A trend of improved performance towards the batch size of 512, or 64 whole beams was noted.

The most significant finding from the FFR study performed, as seen in Table 5.16, was the significant decrease in error for K1, K2, and KR when the median was considered instead of the mean. This was probably a result of the varying effects of variables along the beam. For example, when studying the variable solution along a single beam (see Figure 5.6), it was apparent that both K1 and K2 have the greatest effects at locations close to their actual positions. Thus, values at the location of effect may yield a much better solution. The most significant was the effect of K2, which was consistently far from correct except at the location of x = 20, where the spring was located.

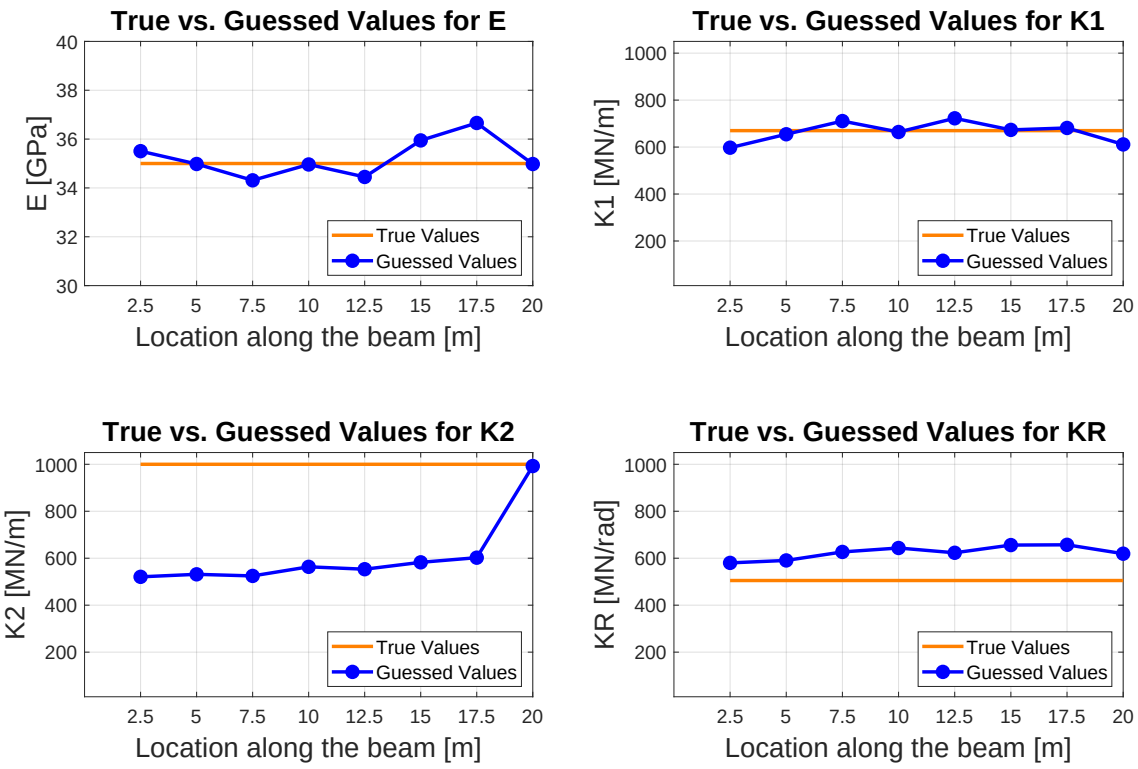



**Figure 5.6:** Visualisation of the variation in neural network approximations along the beam, for test Case C.

Therefore, further attempts were made to derive the guesses at the most appropriate locations, as seen in Table 5.17. Here, the median value of E was presented, along with K1 derived at x = 10 m, K2 at x = 20 m, and KR at x = 12.5 m was chosen. An unclear and unexplained outcome was that KR often yielded the most accurate results one node away from the actual spring location. Thus, x = 12.5 m was beneficial even though the rotational spring was found at x = 10 m.

**Table 5.17:** Neural network. Feedforward reversed prediction performance for test Case C. Considers best location of variable retrieval. Training based on 80% of the data set, with remaing data were kept as validation solutions. Based on 15435 sets of solutions.

| | Custom Points Error | | | | Runtime |
|---|---|---|---|---|---|
| Model, Batch (beams/batch) | E [%] | K1 [%] | K2 [%] | KR [%] | TM [s] |
| FFR-16L512N, Batch: 8 (1) | 8.70 | 721.27 | 679.82 | 661.32 | 2090.2 |
| FFR-16L512N, Batch: 32 (4) | 7.23 | 46.35 | 25.98 | 191.60 | 4028.2 |
| FFR-16L512N, Batch: 64 (8) | 7.26 | 22.28 | 40.42 | 227.86 | 3490.4 |
| FFR-16L512N, Batch: 128 (16) | 7.02 | 19.59 | 57.36 | 268.54 | 2828.1 |
| FFR-16L512N, Batch: 256 (32) | 7.36 | 12.57 | 26.61 | 247.62 | 2361.7 |
| FFR-16L512N, Batch: 512 (64) | 6.97 | 12.33 | 22.12 | 233.28 | 2016.8 |
| FFR-16L512N, Batch: 1024 (128) | 6.85 | 16.32 | 28.87 | 249.89 | 2047.1 |

Through variable retrieval from specific locations, the resulting errors were drastically reduced. The model with a batch size of 512 still resulted in the least error, now indicating much more accurate solutions. However, with the lowest error for KR still close to 200%, performance was far from the desired range. This is also based on the full training set time, which was significantly longer than that used for conventional methods. Thus, there was no point in testing the performance of FFR with a decreased amount of training data for Case C.

## Conclusion of NN test cases

As prior analysis regarding the feedforward in reversed order (FFR) models indicated poor accuracy with long training times, insufficient performance was noted. However, for the ordinary feedforward (FF) system, a high accuracy of deflections was studied for each case. Thus, the methods of predicting deflections based on input variables perform significantly better then anticipated. Therefore, as a final analysis, the sensitivity of the FF systems to a decrease in training data was studied based on the best setups found throughout the studies. The results can be found in Figures 5.7, 5.8, and 5.9 for cases A, B, and C, respectively. Each figure shows the relationship of accuracy to the prediction of deflections for the corresponding set percentage of the training data considered. The normalised error represents an increase in factorial error compared to the lowest error achieved. In addition, actual error percentages are presented for different configurations. For complete results of this study, see Appendix A.1. The results indicate that significant decreases in training data can be performed while conserving accuracy. It should also be noted that excessive amount of training data was used for each case. This was intentionally chosen to ensure that each model had more then enough to learn potentially patterns.

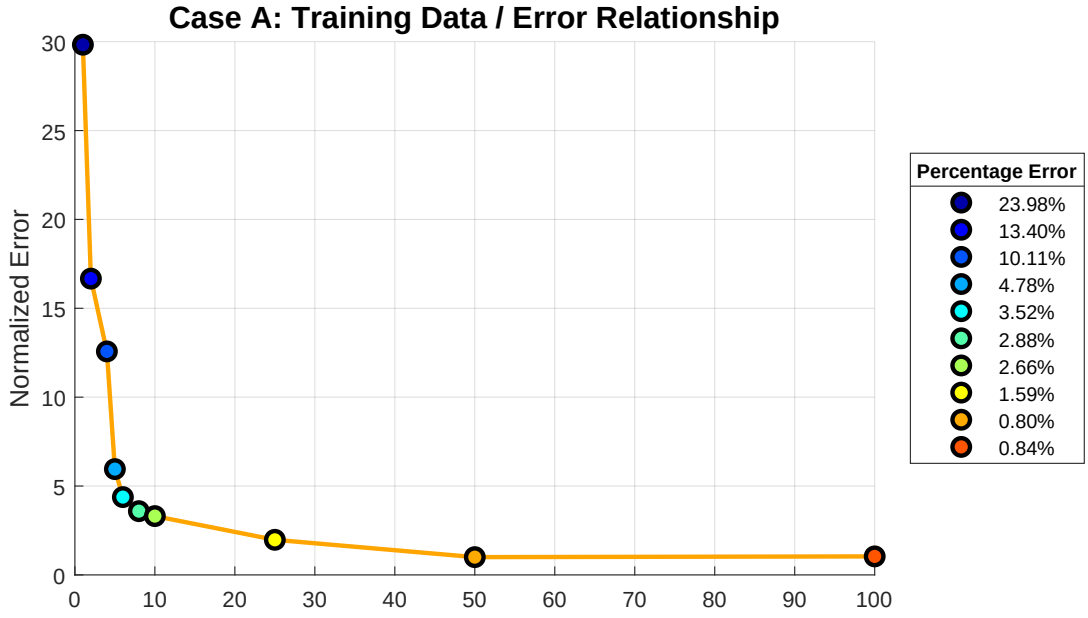



**Figure 5.7:** Case A, neural network feedforward performance for decreased training data. Based on training set of 5070 solutions. For percentage of considered data, 80% is trained against and the remainder kept as validation data.

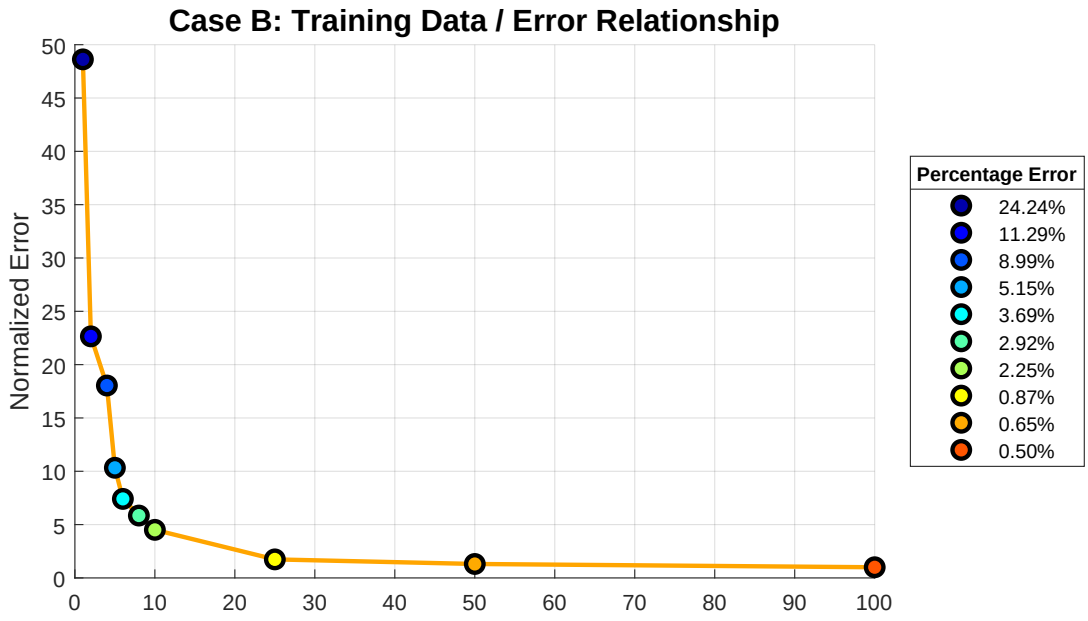



**Figure 5.8:** Case B, neural network feedforward performance for decreased training data. Based on training set of 11856 solutions. For percentage of considered data, 80% is trained against and the remainder kept as validation data.

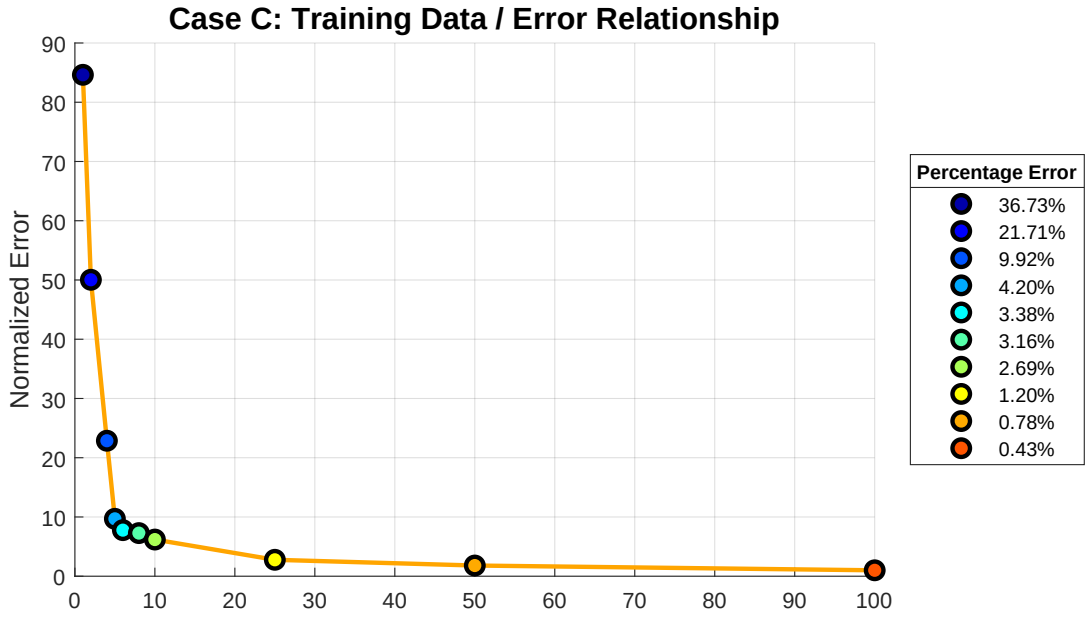



**Figure 5.9:** Case C, neural network feedforward performance for decreased training data. Based on training set of 15435 solutions. For percentage of considered data, 80% is trained against and the remainder kept as validation data.

### 5.1.3 Physics Informed Neural Network method

The implementation of physics informed neural network were done with the aim of proving its applicability and efficiently. As described in Chapter 4.1.3, only test Case A and B were considered as a "proof of concept". For the two test cases, developments and results for the studies will be presented below, of which both feedforward in combination of the inverse problem are presented.

#### Case A

The simplest of test cases was used as preliminary application. With the purpose to study PINN, both forward and inverse problem was solved and evaluated.

##### Feedforward prediction

Primarily, the feedforward performance was evaluated for the neural networks performance in predicting deflections, see Table 5.18. High accuracy in predicting deflection was noted, exceeding the approximations made by the regular neural network without the guidance of an equation describing the physical behaviour. For the non-dimensional experiment, the test with the L-BFGS optimiser showed that the model converged using only the derivatives of the governing equation. Minimising the loss function only considering the loss from the boundaries and the residual of the fourth derivative, with respect to the position on the beam (x). In contrast to regular NN, the PINN converged with a limited amount of training data, without considering the loss from the true solution. The non-dimensional experiment showed great performance, with a mean absolute percentage error of $4 \times 10^{-2}$ %. Since Case A is a statically determinate problem, the deflections could be represented through the Euler-Bernoulli beam equations itself. Therefore, to evaluate the impact of the beam equation in the training process, Gaussian noise of 10% or 20% was introduced into the input parameters of the training data, which had a small impact on the prediction of displacement by the PINN. However, even though noise results in a small but noticeable change compared to the case of no noise, the models achieve significantly more accurate results compared to ordinary NN.

**Table 5.18:** PINN performance in relation to input noise. Performance for predicting deflections for test Case A.

| | | Optimiser performance for deflection |
|---|---|---|
| Model | Gaussian Noise [%] | MAPE, U [%] |
| PINN - LBFGS - 2L32N | 0% | 0.042 |
| PINN - Adam - 2L32N | 0% | 0.361 |
| PINN - Adam - 2L32N | 10% | 0.972 |
| PINN - Adam - 2L32N | 20% | 0.664 |

To visualise the deflected behaviour, with and without introduction of Gaussian noise in the training data, the "true" deflection are plotted together with and without noise in Figures 5.10 and 5.11 respectively. The first figure represents the predicted deflection compared to the "true" solution. It both reflects the deflected shape of test Case A while also highlighting an almost perfect prediction of deflection. In contrast, the second figure shows the "true" deflected behaviour against the predictions considering Gaussian noise. Although some deviations can be observed along the predictions, the overall shape is captured with high accuracy.

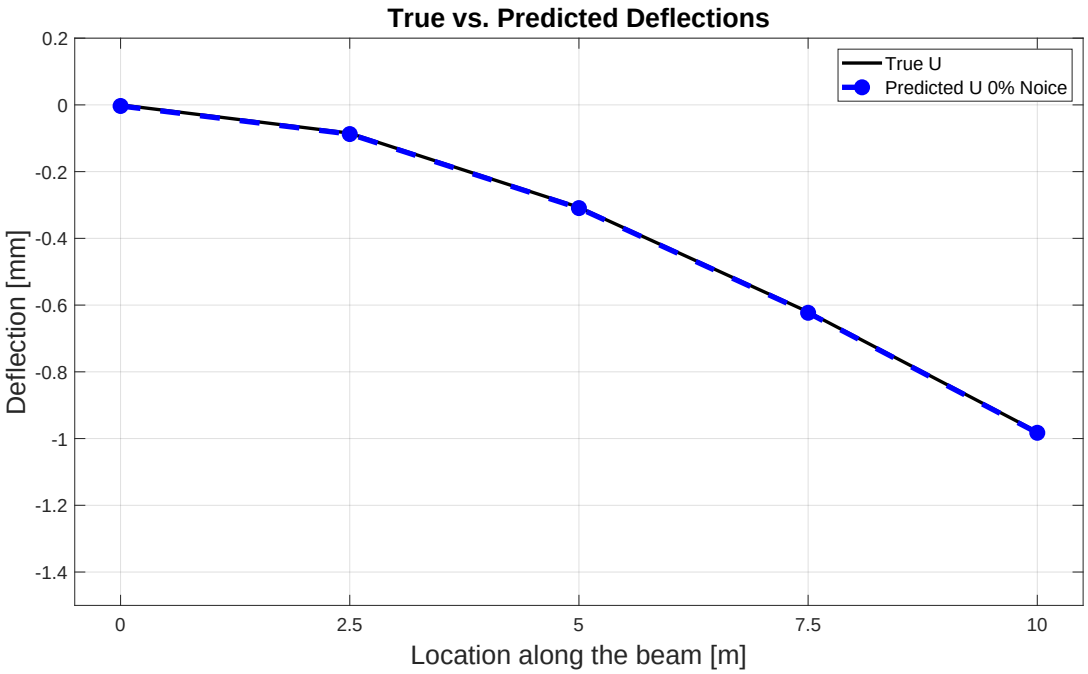



**Figure 5.10:** Visualisation of PINN's predicted displacement (U) along the beam, for Case A. Without introduced Gaussian noise.

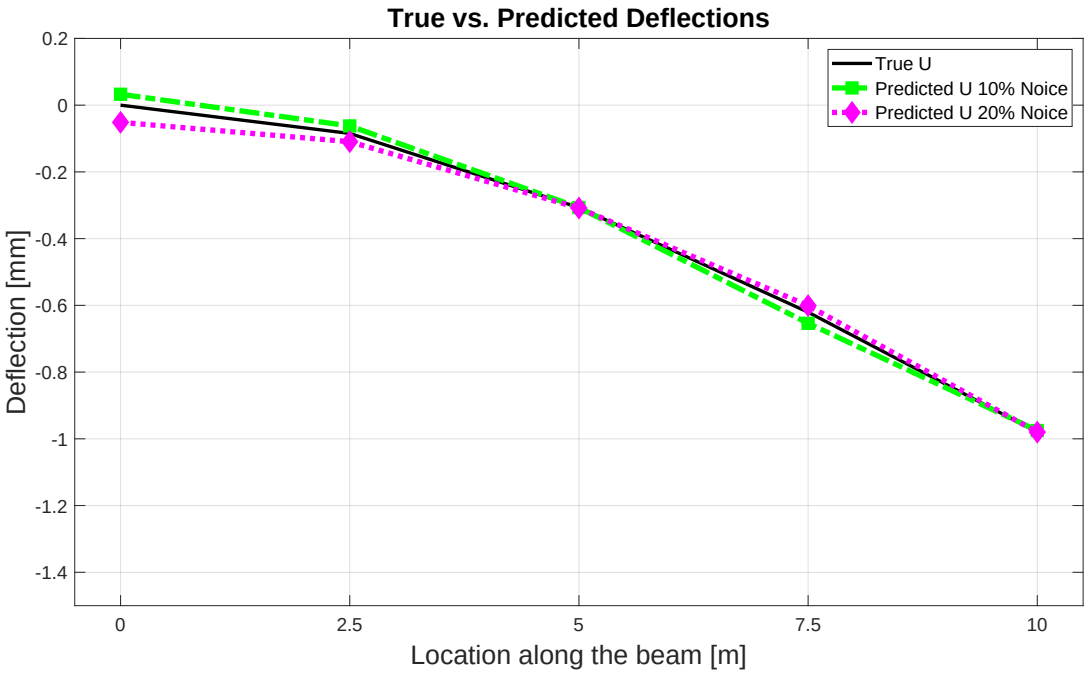



**Figure 5.11:** Visualisation of PINN's predicted displacement (U) along the beam, for Case A. With introduction of 10 - 20% Gaussian noise.

### Inverse problem

With established models predicting deflections with high precision, the same models were now used to derive the corresponding E-modulus through a trainable variable. The results of the experiment for an inverse problem for Case A can be seen in Table 5.19. Upon studying the results through PINN, high accuracy was evident for both cases with and without Gaussian noise. Although noise did affect the results, it should be noted that the absolute percentages indicate a significantly more accurate solution compared to ordinary NN without the guidance of the underlying physical equation. The least accurate model for PINN, using 20% noise, still yields a solution that was close to 15 times better accuracy than the best ordinary neural network of the FFR model.

**Table 5.19:** Performance for the inverse problem for Case A, with implementation of the PINN framework. Prediction of Young´s modulus (E) in relation to Gaussian noise in the training data.

| | | Optimiser performance for variables |
|---|---|---|
| Model | Gaussian Noise [%] | Error, E [%] |
| PINN - Adam - 2L32N | 0% | 0.023 |
| PINN - Adam - 2L32N | 10% | 0.080 |
| PINN - Adam - 2L32N | 20% | 0.101 |

### Case B

In attempts to evaluate whether Case A proved good results due to the simplicity of the statically determined problem and the potentially strong dependence of the equation, the complexity was increased. Case B was used to investigate the scalability of the PINN method, both for the increased complexity and the increased number of variables to solve. Thus, through an indeterminate problem, the verification of potential benefits of PINN was further evaluated.

#### Feedforward prediction

The estimated deflection for test Case B was performed in a similar way as for Case A. The results, see Table 5.11, indicated high accuracy for the corresponding low amount of training data. For the experiment setup with no Gaussian noise, similar accuracy for deflection was achieved in comparison to the equivalent NN for test Case B. Furthermore, the introduction of Gaussian noise led to a decline in accuracy. However, the error size remained low even with significant noise levels, indicating the robustness of the PINN.

**Table 5.20:** PINN performance in relation to input noise. Performance for predicting deflections for test Case B.

| | | Optimiser performance for deflection |
|---|---|---|
| Model | Gaussian Noise [%] | MAPE, U [%] |
| PINN - Adam - 2L32N | 0% | 0.70 |
| PINN - Adam - 2L32N | 10% | 1.46 |
| PINN - Adam - 2L32N | 20% | 2.34 |

To visualise the accuracy achieved, the "true" solution was plotted against predictions without noise, see Figure 5.12, and against corresponding predictions with noise, see Figure 5.13. From these, it can be concluded that the deflection behaviour was captured with high accuracy, both with and without noise.

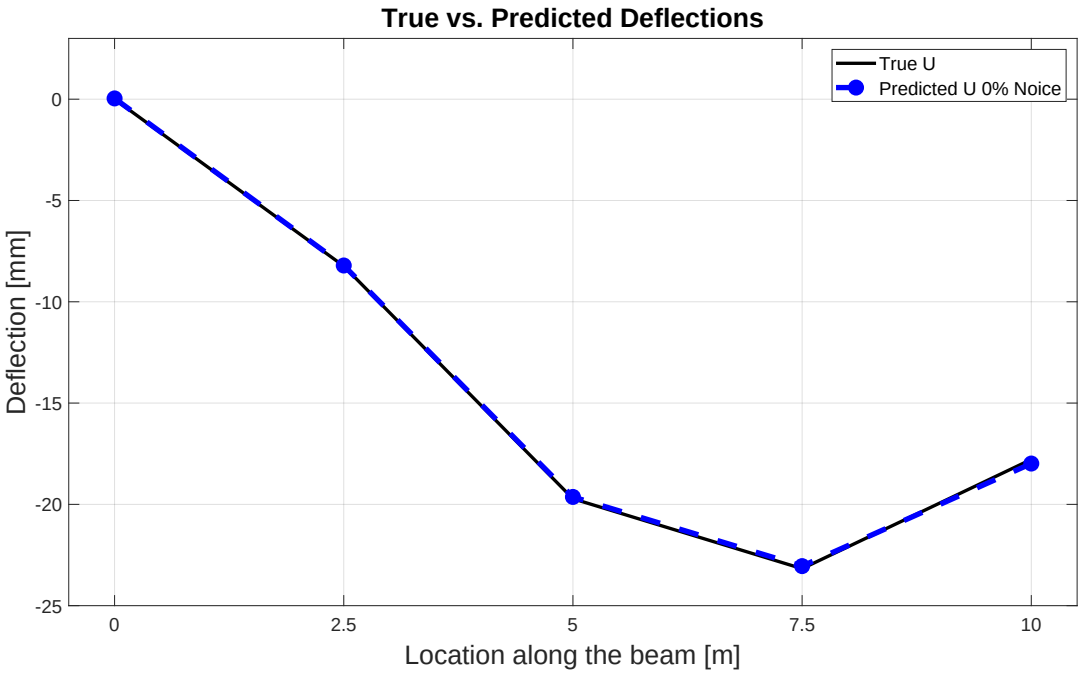



**Figure 5.12:** Visualisation of the PINN's predicted displacement (U) along the beam, for Case B. Without introduced Gaussian noise.

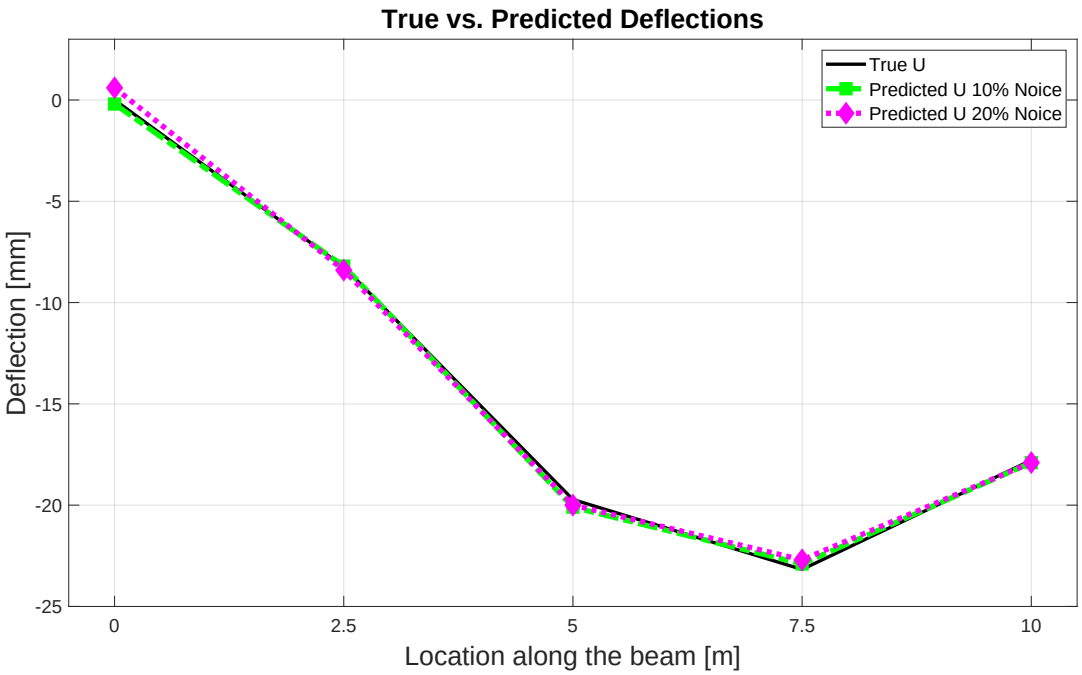



**Figure 5.13:** Visualisation of the PINN's predicted displacement (U) along the beam, for Case B. With introduction of 10 - 20% Gaussian noise.

**Inverse problem**

With proven capabilities considering both accuracy in deflection and sensitivity to noise, accuracy in variable derivations was of interest. To determine whether the PINN would suffer a similar fate as the NN, which experienced a drastic performance decline for multivariable derivations. For test Case B, the inverse problem aimed to derive the Young´s modulus (E) and the support stiffness (K). Three different experimental setups were conducted to evaluate the performance in predicting either the Young's modulus, the spring stiffness, or both. The study, see Table 5.21, indicates low error for each of the three experiments. Furthermore, the PINN does not appear to experience any significant decline in accuracy compared to the ordinary NN, since the model deriving both variables scaled well with increasing complexity. In these experiments, the accuracy of the E-modulus did not decrease between single-variable and multivariable derivation, while the spring stiffness showed a notable decline. Although the decline in accuracy was noticeable in terms of relative increase, the final percentage error was significantly smaller compared to that derived from an ordinary NN. This study highlights the superior performance of the physics-informed neural networks framework compared to the regular neural network, specifically in managing multivariable problems and more complex tasks. The physics or domain-specific guidance ensures more accurate predictions and improved generalisation, particularly in scenarios with limited training data and varying data quality, where regulating the impact of measurements on model tuning is crucial.

**Table 5.21:** Performance for the inverse problem for test Case B. Prediction of Young´s modules (E) and/or spring stiffness (K) through the PINN framework.

| | | Optimiser performance for variables | |
|---|---|---|---|
| Model | Solved variables | Error, E [%] | Error, K [%] |
| PINN - Adam - 2L32N | E | 0.13 | - |
| PINN - Adam - 2L32N | K | - | 1.31 |
| PINN - Adam - 2L32N | E & K | 0.12 | 2.40 |

### 5.1.4 OpenAI method

In attempts to achieve a more general and complex neural network, the pre-made AI of ChatGPT was used for optimisation. Based on the same test cases as previous studies, different prompting techniques were tested with the desire to achieve an efficient optimiser AI. For the underlying methodology, details on the prompt techniques used and motivations, see Chapter 4.1.4. For complete prompts, progression and underlying methods employed, see Appendix 5.22 for full details.

#### Case A

The results of prompt engineering and the model engine are presented in Table 5.22. The first technique, which involved providing complete context to ChatGPT, often failed in the very first response. Although the AI indicated an understanding of the context, the type of problem and stated appropriate equations, it failed to perform numerical calculations, resulting in sporadic iterations without convergence. The technique of using more concise prompts, skipping context and avoiding numerical calculations avoided failure, but revealed an absence of analytical reasoning between iterations. Sporadic results were once again studied between iterations. To promote analytical thinking, chain-of-thought (COT) reasoning was introduced with stepwise subtasks. Through COT, the model demonstrated a more logical progression, considering history and used logical reasoning. However, through the ChatGPT 3.5 engine, the AI encountered failures in multiple sets of tests. ChatGPT deviated from the stated instructions, violating both the stated rules and intervals and ceased chain-of-thought reasoning. Even if convergence were met, it did not adhere to the specified output formatting and thus would fail in automation. Lastly, the same prompt through the ChatGPT 4 engine demonstrated improved capabilities in following instructions and identified the optimal value after seven iterations.

**Table 5.22:** Summary of prompt and model. Solving for Young´s modulus (E) in test Case A, prompt development.

| | Performance | | | | |
|---|---|---|---|---|---|
| | Error, E [%] | Iterations | Failure | Failed Iter. | Failure Type |
| Full context (3.5 & 4) | No convergance | 30 | Yes | 1 | Misscalculations |
| Concise 3.5 | No convergance | 30 | No | - | - |
| Concise 4 | No convergance | 30 | No | - | - |
| Concise COT 3.5 | 1.12 | 9 | Yes | 7 | Rule violations |
| Concise COT 4 | 0 | 7 | No | - | - |

In accordance to literature study in Chapter 3.3.3, ChatGPT demonstrated poor performance in mathematics and numerical analysis. Through miscalculations, the concept of favoured numbers could be studied as the model often misplaced errors by the number seven or five, rarely misplacing for the number one. Furthermore, in methods that lack chain-of-thought reasoning, the AI frequently generates numerical sequences. A tendency to fixate on linear sequences, such as consecutive numbers, was noted. In further studies, GPT-4 was used to assess the magnitude

of miscalculations. Using the consistent prompt used for the case of *full context* in Table 5.23, a noticeable variation in the output was observed. The standard deviation of the calculations via ChatGPT can be observed in Figure 5.14. The significant variation in the calculated E-modulus using the same equations and values, ranging from 30.30 to 39.29, indicated a low precision. Although the standard deviation curve centres on reasonable solutions, less than half of the output provided solutions of reasonable size. Note that the standard deviation curve excludes instances when GPT-4 did not produce an output, stating that it was "unable to perform the numerical calculations" and extreme outliers such as E = 12.56. This behaviour often led to instantaneous failure in automated processes, as the magnitude of miscalculation produced sporadic and illogical values that failed to converge.

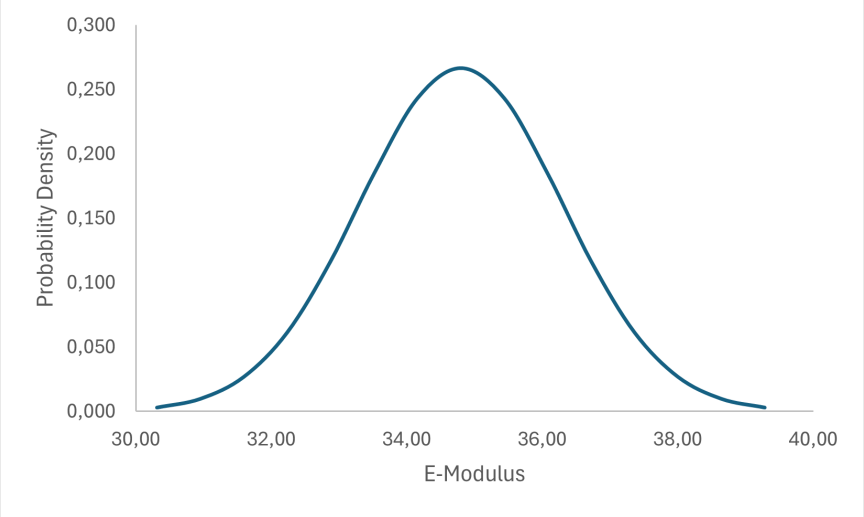



**Figure 5.14:** Standard deviation regarding numerical Calculation for GPT-4 over identical prompts. Correct solution were E = 35.60.

As a result, the strategy to minimise the burden of numerical calculations was tested and found to be more promising. Upon evaluating sequences of objective function values, ChatGPT appears to focus on assessing patterns and interpolations rather than performing direct calculations, leading to improved performance. Furthermore, ChatGPT 4 demonstrated the most promising results, being the only version that achieved success. Based on above mentioned methods, more detailed result of the most promising combination is found in Table 5.23.

**Table 5.23:** Summary of best prompt and model for test Case A.

| | Performance | | | | |
|---|---|---|---|---|---|
| | Error, E-modulus [%] | Iterations | Runtime | Tokens | Cost [$] |
| Concise COT 4 | 0 | 7 | 297.5 | 19978 | 0.60 |

As final remarks, as per the realise of the new engine of ChatGPT 4o, a sample test were reevaluated concerning capabilities in numerical calculations. The results from these can be found in Appendix A.4, see Figure A.10. The study indicated no noticeable improvements concerning numerical evaluations, thus the results hold.

### Case B

With the proven applicability of ChatGPT in Case A, the complexity was increased to assess its scalability. Similarly to the previous analysis, Test Case B involved two unknown parameters, namely E-modulus and spring stiffness K. Initially, the proven prompting method from Case A was applied directly, followed by any necessary adjustments if needed.

The results for Case B are presented in Table 5.24, where the prompt method derived from Case A is denoted as B1. A notable outcome was that the method yielded a seemingly convergent result. However, it should be noted that this method yielded a relatively poor accuracy and performance. ChatGPT indicated a convergence early on and thus ceased further exploration of variable values. In addition, note that the test is marked as "failed". This, as ChatGPT failed to respond beyond 38 iterations. Although this may be considered an anomaly, its occurrence should not be overlooked, as it could cease an automated application. Nevertheless, the method once again demonstrated promising outcomes, took history into account, made appropriate assumptions and correctly followed general instructions.

The modified prompt, denoted as B2 introduced a minimum step-size for the variable change and a specified convergence criteria. The results for B2 are shown in Table 5.24, which indicated a decrease in performance. In addition to decreased accuracy, failures were noted in the first iteration. ChatGPT struggled to adhere to the stated rules, failing to obey chain-of-thought reasoning. ChatGPT further stated that convergence was achieved after 39 iterations. This violated the convergence criteria, which ChatGPT explicitly claimed to be faulty and unreachable. Hence, it recommended the user to "correct" the objective function and stated that convergence criteria could not be reached, which was incorrect.

**Table 5.24:** Summary of prompt and model. Case B, prompt development.

| | Performance | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Error, E [%] | Error, K [%] | Iterations | Runtime | Failure | Failed Iter. | Failure Type | Tokens | Cost [$] |
| Concise COT 4-B1 | 2.25 | 12.46 | 38 | 2197.5 | Yes | 38 | Respond failure | 229325 | 6.88 |
| Concise COT 4-B2 | 4.38 | 23.74 | 39 | 849.2 | Yes | 1 | Rule violations | 136409 | 4.09 |

To determine whether ChatGPT converged at a potential local minimum, the solution space was visualised using a surface plot, see Figure 5.15. From this analysis, it was observed that no local minimum was apparent. These illustrations underscore the capabilities and limitations of ChatGPT. While each iteration seemingly progressed towards the minimum, the fixation on variables can be studied. For the method of B1, the E-modulus was prematurely considered converged, consequently

a fixation upon incorrect values was noted. For the case of B2, ChatGPT had a linear change of variables, not deviating for this linear relation. Beyond iteration 4, see Figure 5.16, ChatGPT found its current direction unfavourable and changed. However, a reverse direction was chosen following the relations already tested, as convergence was stated to be close after iteration 4. Apart from B1 and B2, additional tests were carried out in an effort to avoid premature convergence, but without success. See Appendix A.9 for reference. Although it was challenging to prevent the behaviour of premature convergence through prompt engineering, the illustrations also reveal the potential of these methods. Throughout each iteration, the AI demonstrated logical reasoning, positive progression and advanced understanding.

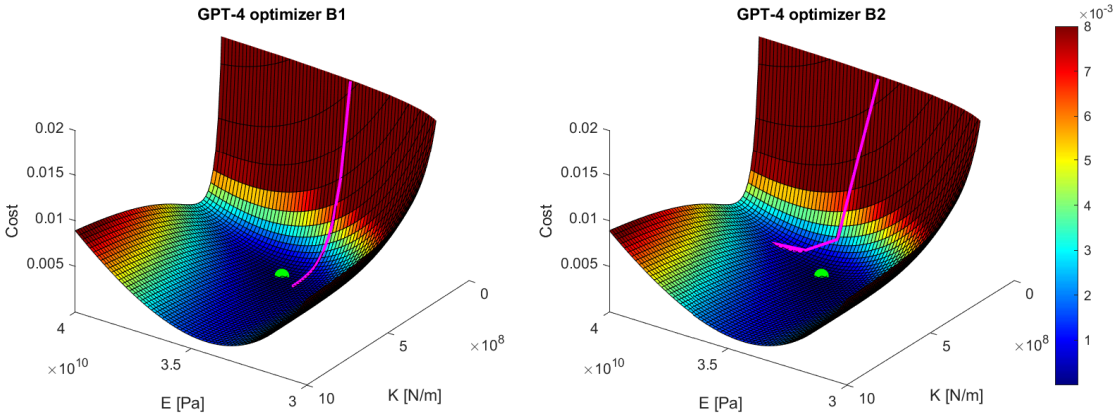



**Figure 5.15:** Solution space with GPT-4's attempt at solving. 3D view. Green dot, marks the actual minimum.

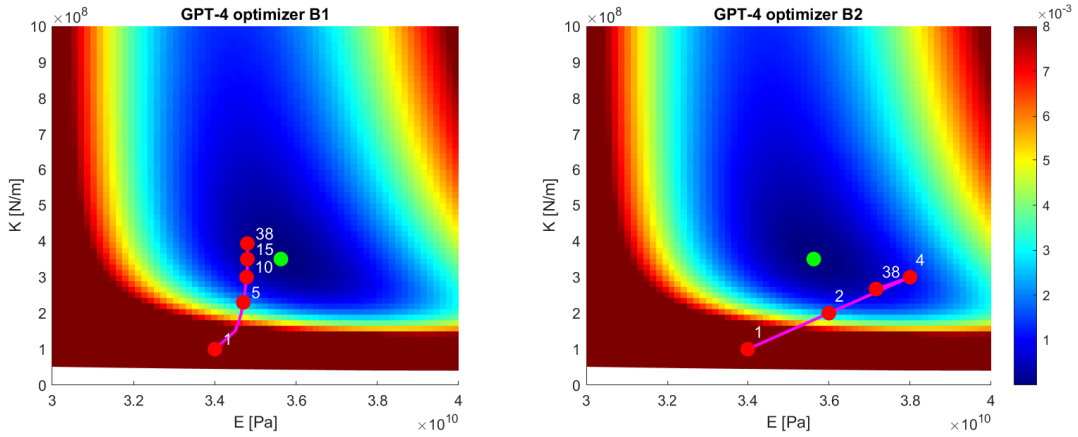



**Figure 5.16:** Solution space with GPT-4's attempt at solving. Contour plot with iteration numbering and progression. Green dot, marks the actual minimum.

Despite the demonstrated competence, within the time frame of this thesis, no developed prompt could overcome the shortcomings of ChatGPT. Throughout trials and errors, ChatGPT would malfunction or deviate from the stated rules, leading to

inaccurate and unpredictable behaviour. It must be acknowledged that failure does not render ChatGPT useless in structural applications. The results presented above indicate appropriate behaviour, minimising the objective value compared to start values and showed correct reasoning. However, in terms of developing an automated and rational model updating principle, the lack in robustness and repeatability was detrimental. Furthermore, the accuracy achieved was below a reasonable range, resulting in an optimisation not applicable in the case of structural engineering.

Likewise for Case A, the newer release of ChatGPT 4o was tested for the prompting techniques of B1 and B2, see Appendix A.4 for results. The reevaluation of prompt method of B1 indicated some slight changes, see Figure A.11. Primarily, the final results seemed to indicate slight increased accuracy, however, iterations encountered a new failure. The new engine seemed to be introduced with a harsher limit on tokens per minute, resulting in a complete stop after 23 iterations. Such behaviour further strengthen the intention to limit token use which was employed in B2. For the reevaluation of B2, see Figure A.12, no improvements were noted. With a decline in accuracy, rule violations and a further tendency of premature convergence, ChatGPT 4o proved to suffer from similar shortcomings.

### Case C

As no robust or successful prompt could be developed for Case B, further development was stopped for Case C. Before increasing complexity, it was essential to establish a robust principle. Given the previous results of Cases A and B, it was clear that achieving scalability with ChatGPT presents significant challenges. Therefore, it was determined that the results of Cases A and B were sufficient to assess and draw conclusions about the performance of OpenAI's ChatGPT.

## 5.2 Results case-study

With the prerequisites for the case-study described in Section 4.2.5, the FEM beam model of the Solvalla bridge was updated against measured data. Optimisation was performed through the genetic algorithm which ran during 480476 seconds, which is equivalent to 133.5 hours or approximately five and a half days. This was for a population size of 640, which led to 33 full iterations. During this time, 22240 full model evaluations were performed, thus creating the same amount of stored solutions for neural network training. As each run produced output for three load cases, consisting of 26 data points (points of deflection retrieval) each, the total training data established during the conventional optimisation became just above 2 million individual data sets.

The result indicated a noteworthy increase in stiffness compared to initial designs. In addition to all stiffening changes and model tweaks described in 4.2.5, the optimisation converged for a crack distribution together with an increase in Young´s modules (E). After optimisation through the genetic algorithm, the optimised and updated FEM model indicated a 23% increase in E-modulus, for the corresponding derived configuration of the cracked sections.

If comparing the cracking propagation and degree along the bridge, two noteworthy outcomes can be seen in contrast to original assumptions. Primarily, the length of the cracked regions does not align perfectly with the previously assumed regions that cover 15% of the span lengths, concentrated at the supports. Instead, support 5, the left support, indicated less cracked regions, while support 6, the right support, showed a similar but slightly shifted degree of cracking compared to initial assumptions. However, in accordance with the initial cracking assumptions, a stepwise cracking was seemingly noted. The assumed possibility of progressive crack initiation and thus semi-considered concrete sections does not seem to be present. Figure 5.17 shown below indicates regions of cracked and uncracked regions, for initial design against the updated model, with colour gradients that describe the degree of cracking.

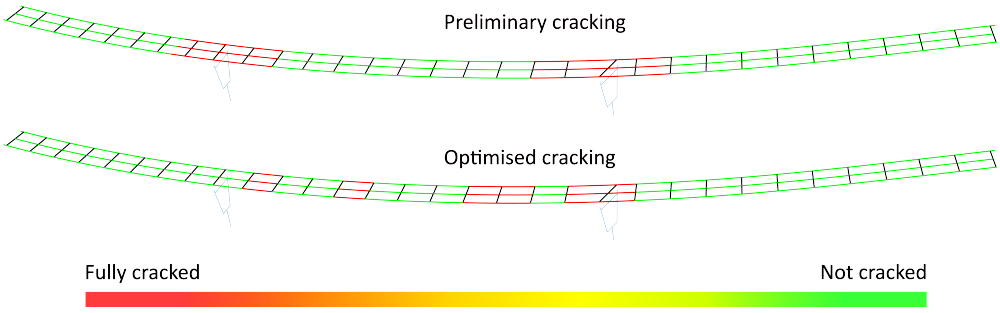



**Figure 5.17:** Case-study of Solvalla. Comparison between initial cracked regions against optimised regions using the genetic algorithm for optimisation.

With the above describe cracking distribution, along with an optimised E-modulus increase of 23%, the overall bridge deflection was found to match measured data in a much higher degree than before. Figures 5.18 to 5.20 below show how the southern and northern track deflection is represented, for initial design stages, optimised design compared to measured behaviour. From these graphs, the difference and noticeable improvements of the optimisation can be noted. Additionally, while the optimisation aims towards a perfect match, it does so through noisy data. With ± 1mm error for both reference and test readings, an error towards 2mm could be considered acceptable. With this, it can be noted that optimised results are below these, thus achieving a sufficiently optimised behaviour.

The first of considered load case was the even loading in between support 4 and 5, thus 45SN, see Figure 5.18. From the results, the impact of the stiffening effects and optimisation can be seen. For both the southern and northern track, major improvements can be seen for the loaded span, thus to the left in the figures. With almost a one to one fit, this span was almost perfectly represented. However, a more obvious struggle was noted regarding capturing correct hogging effects at the adjacent spans. Even though the effects were captured better in comparisons against the initial model, the match was not perfect. This was however not seen as detrimental as the higher and more critical deflection downwards were represented accurately enough.

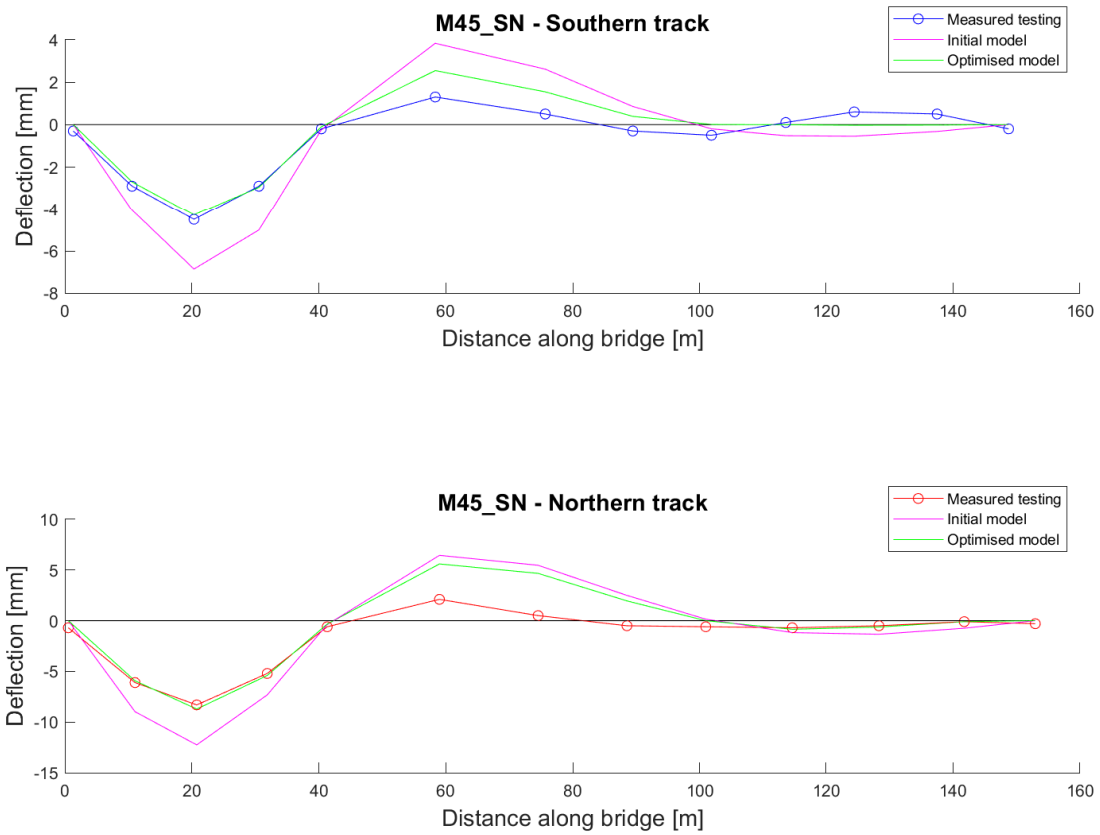



**Figure 5.18:** Results of model updating of case-study. Measurements from testing, against initial prediction and the optimised model, for load case 45SN.

Moving forward, the next loading condition considered was the even loading between support 5 and 6, thus 56SN, see Figure 5.19. This span also has existing railroads passing below, and a large downward deflection would be even more critical. The model updating indicates an improved model behaviour after the model update was performed. With this, a much more representative behaviour was captured at the maximum deflection for the northern track. Although, even if the model was improved, larger deviance can be noted compared to the previous loading case. However, with a maximum measured displacement of 20.6mm and a derived equivalent of 22.6mm, this was just at the verge of the expected error rate of 2mm. Thus, even if deviance was noted, it could be attributed towards measurement errors.

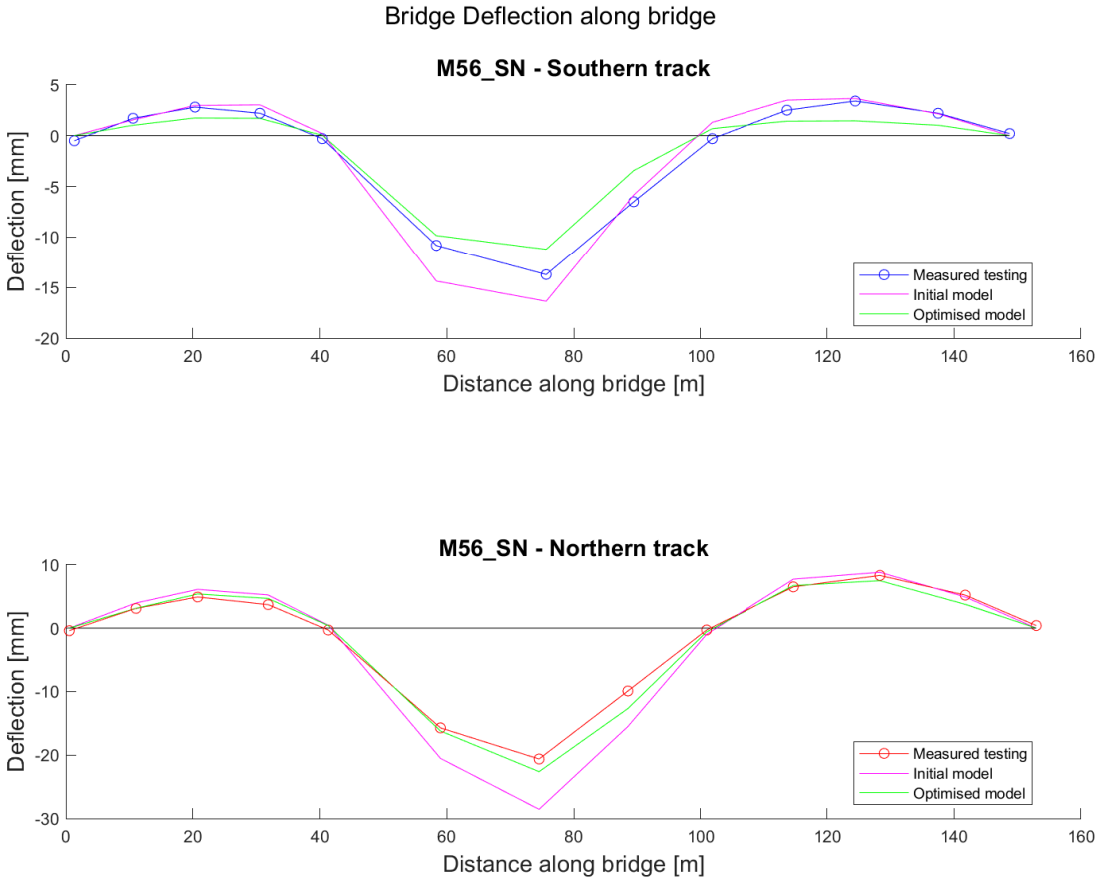



**Figure 5.19:** Results of model updating of case-study. Measurements from testing, against initial prediction and optimised model, for load case 56SN.

Lastly, the loading of the even case between support 6 and 7, 67SN was considered, see Figure 5.20. While this case does seem to produce the least accuracy, for both northern and southern track, this was to some extend expected. Being the case with the less weight in the objective function, it was expected to show least match. Nevertheless, after model updating, the mismatch was never beyond 2mm, thus within error tolerance of measuring. Additionally, it must be noted that the measuring point along the northern track at mid span, specifically at the distance along the bridge of 60 meters, was not considered as it was deemed unrealistic and likely to be a measuring error.

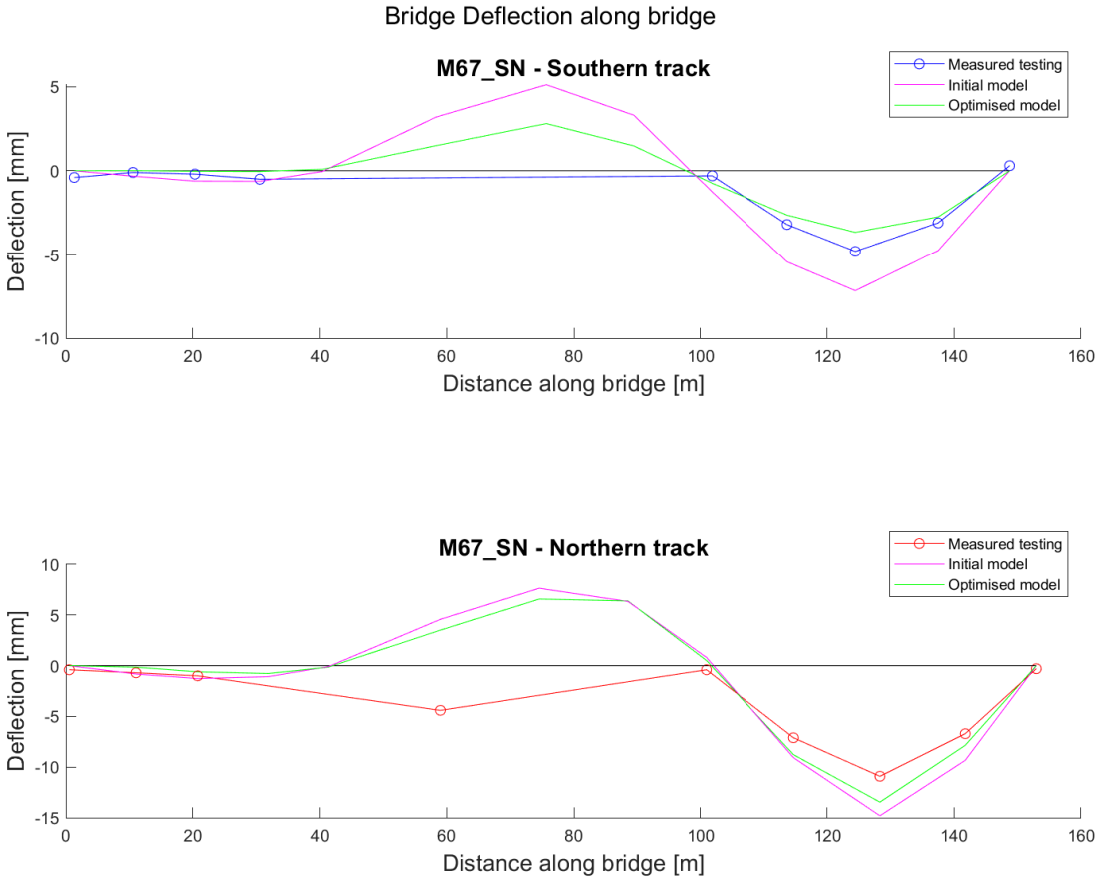



**Figure 5.20:** Results of model updating of case-study. Measurements from testing, against initial prediction and the optimised model, for load case 67SN.

Through the establishment of the training data acquired, a proof of concept was established regarding NN implementation. The chosen neural network consisted out of 8 layers with 512 neurons each. By evaluation against validation data, corresponding to 20% of the total data set, an average accuracy of 86.83% was established for predicting deflections. With just above 2 million data sets, consisting of 3 different load cases, having 26 data sets each, the data density was approximately 90 datasets per variable. Through this, the corresponding ratios were studied and compare against those previously examined for test cases. Figure 5.21 illustrates how the acquired data density and corresponding prediction accuracy relates to that of the test cases. The grey line represents the equivalent data density and accuracy relation, for each test case, combined into one averaged trend line. The result indicate that there seem to be a rather close correlation between datasets per variable and accuracy. With the averaged relations for all considered load cases, the relation for the case-study proved to perfectly align with that acquired from test cases. Further, the NN proved to find patterns for the loading 67SN more accurately compared to the others, with 45SN proving the most difficult to learn.

The differentiation in achieved accuracy between the cases might seem counter intuitive considering all data was gathered during the same FEM optimisation. However, if examining the objective function employed, see Equation 4.17, it should be noted that the case of 67SN was the less weighted loading, as it indicated large likelihoods of measuring errors. If this reasoning holds true, even if weighted less, the optimisation likely struggled to fit against the corresponding case. Consequently, it is possible that optimisation put more effort in attempts to satisfy the model against the more deviant cases, thus producing a proportionally larger quantity of data sets containing variations affecting that case. Thus, where the conventional optimiser struggles, more training data is created for corresponding characteristic.

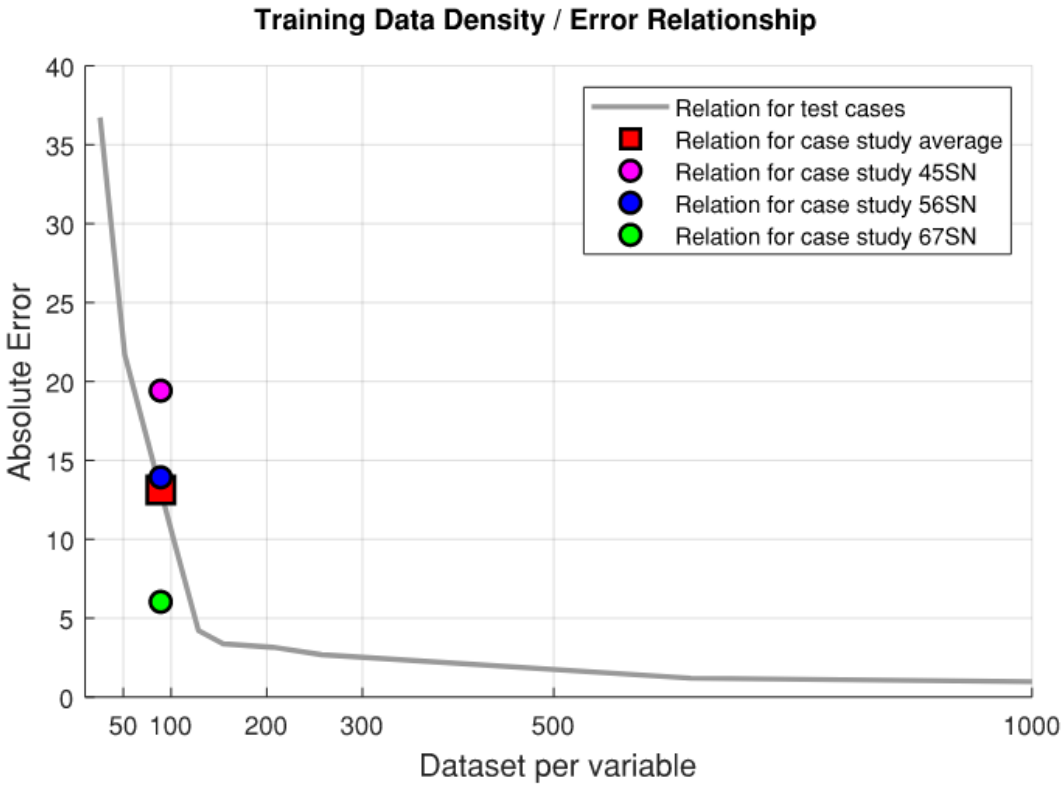



**Figure 5.21:** Training data density and corresponding accuracy for the neural network, feedforward prediction for displacement.

As a very last study, the methodology of Nelder-Mead simplex were implemented to further verify the above used choice of genetic algorithm. For the attempt of providing a decent start guess, thus equal to the initial design assumptions regarding cracking, Nelder-Mead simplex was setup. From Figure 5.22 seen below, the rather close resemblance in initial and optimised state can be seen. It was noted that it acquired an objective function value which was comparable to that acquired through genetic algorithm, thus resulted in a adequately accurate solution. However, with the provided guess, the Nelder-Mead simplex showed very low optimisation progress as the optimiser more or less kept the initial guess as the output. Thus, it was considered a too good guess, at least in attempts to evaluate optimisation principles. For visualisation of the optimisation attempt, see Appendix A.5 and Figure A.14.

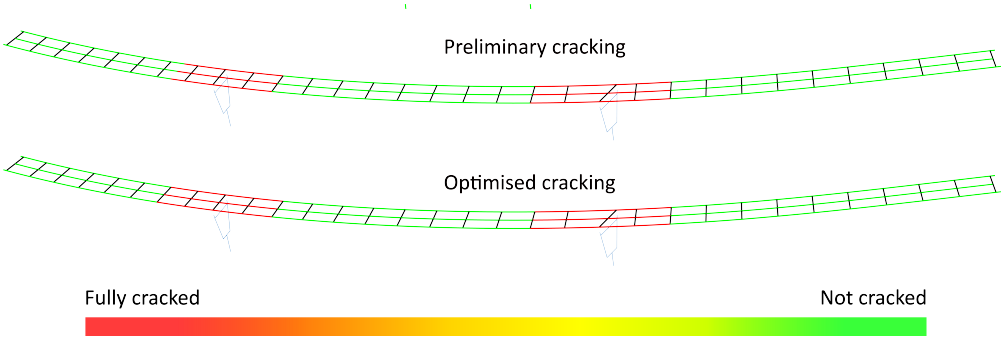



**Figure 5.22:** Case-study of Solvalla. Comparison between initial cracked regions against optimised regions using the Nelder-Mead simplex optimisation.

Therefore, the less optimal initial guess that all sections being half cracked was tested. This in an attempt to force the optimiser to optimise and essentially take progressive steps. However, with such a guess, the Nelder-Mead simplex struggled, eventually failing after several days. This run led to the solver breaking stated boundaries and evaluating solutions which are unreasonable. The failed progression can be seen in Appendix A.5, in Figure A.15. With this in mind, it could be noted that the genetic algorithm does seem most appropriate, yielding an adequate and stable output compared to the Nelder-Mead simplex, even with regard to a more beneficial initial guess. The genetic algorithm further proved to be more intuitive, providing appropriate functionalities for large-scale implementation, which the Nelder-Mead simplex lacked.

# 6
# Conclusion

Throughout the thesis, three main categories of optimisation were studied: traditional optimisers, custom-built neural networks, and the pre-made AI of ChatGPT. Through various studies, evaluating performance changes due to scaling of the problem, fine-tuning of models, relative performance, and future applicability were assessed.

The initial studies concerning the traditional optimisers not only set the reference point for performance but also highlighted differences among the methods of Nelder-Mead simplex, pattern search, genetic algorithm, and particle swarm. The results indicated similar abilities in problem scaling, with the simpler methods of Nelder-Mead simplex and pattern search consistently outperforming the others for each test case. Out of the four, the genetic algorithm showed the least performance with respect to accuracy and computational time. While the constructed test cases might have been too simple to truly demonstrate scalability, the outcomes among the four methods indicated clear differences. However, the study evaluating a less coherent and complete objective function showed significant decrease in performance for the simpler methods, while the genetic algorithm showed almost no decrease in performance, indicating superior insensitivity to discrete data.

Subsequent studies evaluated the performance of the developed neural networks, which performed similar tasks of updating a model. These studies highlighted performance differences between various configurations and scalability with increasing complexity. The FFR methods, which predicted variables based on deflection input data, were directly comparable to the other principles, producing similar procedures. However, the studies indicated significantly lower performance compared to the traditional methods, with much longer computational times. Additionally, the feedforward reverse (FFR) models, aimed to derive the material parameters from known deflection, showed poor scalability and struggling with increased complexity. In contrast, the feedforward (FF) methods, which estimated deflections based on input variables, demonstrated remarkable accuracy with computational demands comparable to those of the conventional methods.

In attempts to identify improved methods of acquiring variables through a neural network, the concept of PINN was studied. Even though this was a more concise "proof of concept" study compared to others performed, the results displayed some of its capabilities. In contrast to an ordinary neural network (NN), with use of the PINN framework, the network could predict both deflections and variables through

the same model. Thus, a more versatile network was developed in terms of applicable tasks. It further showed huge improvements compared to ordinary NN, with drastic improvements in displayed accuracy as well as reduced dependency on large sets of training data. Therefore, through test Case A and B, the potential and significant gains of PINN were obvious. However, with these results, three remarks can be made. Firstly, the adaptation of PINN requires a governing equation that represents the behaviour the network aim to learn, and for the specific problem, correct boundaries need to be implemented to help the learning process of the network. In real applications, difficulties might arise due to the complexity of deriving a correct equation for the specific problem. Additionally, determining the correct boundary conditions is crucial and overly simplified assumptions may hinder the network's ability to learn effectively. The PINN model might prove difficult to implement appropriately, thus losing its rationality. Secondly, the tuning of the hyperparameter can be difficult for regular NN. There are tools developed to aid in the tuning of hyperparameters, like the Optuna package for Python used in the study. But even so, by adding more parameters to the loss function, the contributions of different losses can make turning of the additional hyperparameters challenging. Finally, even if the PINN model was tested with various amounts of Gaussian noise to simulate non-perfect input parameters or a mismatch in the governing equation, the remarkable performance displayed might be a direct result of benchmarking against too simplistic test cases. The performance of the physics informed neural networks in real-world case studies, remains uncertain and requires further investigation.

The final study evaluated and developed principles of ChatGPT adaptation within model updating. This analysis assessed performance in relation to different prompt techniques. The results highlighted the limitations of ChatGPT, particularly its lacking capabilities in performing numerical calculations and its tendency to break rules. While it succeeded in the primary test case, the second case, which involved the first increase in complexity, resulted in a complete halt in progress as the AI failed. This indicated that scalability was alarmingly poor for ChatGPT. Even though the analysis was prematurely halted, it underscored the significance of prompt engineering, especially through chain-of-thought prompting. It is important to note that these findings apply to ChatGPT engines prior to the update released on $13^{\text{th}}$ May 2024. Although, limited sampled reevaluations indicate that the updated and new engine of ChatGPT 4o suffers a similar fate, however only limited reevaluations were possible within the time frame of this thesis. As the update may have shifted performance, results may need reevaluation for further and final statements. While the direct comparison among the three different categories indicates better consistency, accuracy, and low computational demands for the traditional methods, the other two showed potential benefits and applications elsewhere. Therefore, in the strict consideration of updating variables based on known measured deflections, no method could compete or compare with the traditional methods. However, both neural networks and ChatGPT showed noticeable potential.

In the case of neural networks, while not suitable to replace the traditional methods of model updating, parallel use could still be valuable. The FF model, which predicted deflections based on known input variables, showed high accuracy for relatively small sets of training data and required computational time comparable to that of traditional model updating methods. Thus, the necessary training data could be produced in parallel with traditional model updating, without introducing extra demands, as the data could be collected simultaneously. Once trained, the model could produce reevaluations and deflection estimations within a fraction of the time with remarkable accuracy. Therefore, while unable to replace the principles of traditional methods, a trained model could generate continuous and similar approximations without the need for a FEM model. This implies significant benefits in computational demands if continued evaluation of a studied structure is of interest. This model could potentially also be updated and fine tuned even further with continuous measurements over time, increasing the accuracy even further with more training data being added to the training data.

Therefore, for contemporary applications, the appropriate method of updating is through traditional methods, with the ability to ease future re-evaluations through the parallel establishment of a neural network. Through such a principle, an additional tool could be established without significant extra effort. This methodology would not impact the process of model updating today but could facilitate continued study, parametric study, changes of variables within a known range, and quick overviews of structural responses. In addition, the established neural network requires minimal computational demands, thus allowing instantaneous approximations with minimal computational power. This could mean that engineers could perform quick reevaluations without the need for a powerful workstation, enabling estimations, for instance, on-site. Further, such a tool could facilitate faster communications, decision-making and problem solving, as fast estimations could be performed without extensive reevaluations. For instance, through an preliminary NN prediction, fast and reasonable estimations on behaviour could ease engineers in providing more profound engineering judgments.

Based on these recommendations, the implementation of the conventional method of the genetic algorithm alongside parallel training of a neural network was carried out for the case-study. While the optimisation did reach convergence through the updating of an increased E-modulus for concrete in combination with crack redistribution, these results are theoretical as no physical quantities for deviations are known with certainty. Therefore, derived causes from the case-study should be treated as theoretical quantities. The derived 23% increase in E-modulus for the concrete slab seemed rather drastic, so more plausible causes were reasoned. Concrete of strength class C35/45 was intended, however, tests acquired through the provided concrete journal indicated an actual strength closer to C60/75. This would mean an increase of E-modulus of approximately 15%, responsible for the majority of the derived increase. An alternative and plausible reason for the deviation is the arch effects created as a result of prevented bearing movement. As the rough control of horizontal forces showed that friction forces are likely not surpassed, the

arch effects were likely present in reality. Since the beam model had insufficient complexity to capture the true arch effects, a likely and important strengthening effect is excluded from the optimisation. Therefore, it must be noted that the real construction probably does not have parameters corresponding to those derived in the optimisation. Regardless of whether the parameters are physically correct, the case-study served its purpose of being optimised to capture a more accurate global behaviour. However, it must be noted that the deviations presented in the results of the case-study did produce noticeable errors. Considering the possible error in field measurements being up to ±2mm, the optimisation did satisfy this range. With this notable error, it must be emphasised that optimisation cannot find a single solution that satisfies all the considered load cases if notable errors exist. This could likely be enhanced by employing more accurate measuring principles, however, the techniques of levelling was deemed accurate enough for the intention of the test loading. Finally, in tests of parallel training data gathering, a fairly reasonable training density was established compared to that from test cases. Thus, with comparable data densities, considering the test cases proved solvable even with significant data reductions, the concept seems plausible for adaptation. Further, this was additionally noted for the results of a proof-of-concept network, which achieved an accuracy above 85% in deflection estimations. Additionally, a relationship between training data density and achievable accuracy was noted, resulting in a potential methodology to estimate the required number of data sets for corresponding accuracy. Thus, the potential for estimating the feasibility of creating a sufficiently accurate model can be evaluated beforehand, prior to initiating an updating principle.

As final remarks, even though the FFR system and ChatGPT could not match the traditional methods in this study, it must be acknowledged that both principles are in early development and thus could show drastic improvements in future applications. Therefore, the conclusion that they are not appropriate alternatives to the traditional methods does not necessarily hold for long, and both of them possess significant potential.

Therefore, for future work and continued evaluations, neither neural networks nor AI should be overlooked. Within neural networks, different approaches and methodologies could be beneficial. Furthermore, the concepts of the PINN framework seem promising in aiding networks for more complex and realistic implementations. Further exploration of techniques for adapting to real-world scenarios and assessing their efficacy and performance in such intricate environments would be of great interest. Additionally, the concepts of PINN in some form, could be evaluated in terms of more complex and realistic implementation, both to study methods of adaptation towards reality and to verify its success and performance in more complex scenarios. Additionally, considering AI implementation, changed and improved models are persistently released. For example, the update of $13_{\text{th}}$ May 2024 posted significant changes to the tested ChatGPT engines, with the introduction of a new model. Thus, the principles tested and the results presented in this thesis are likely only valid for the tested AI versions, making ongoing studies, developments, and attempts relevant.

# Bibliography


Agarwal, R. (2023, September). Complete Guide to the Adam Optimization Algorithm. https://builtin.com/machine-learning/adam-optimization

Aggarwal, C. C. (2018). *Neural Networks and Deep Learning*. https://doi.org/10.1007/978-3-319-94463-0

Aggarwal, N. (2023, April). Difference between Batch Gradient Descent and Stochastic Gradient Descent. https://www.geeksforgeeks.org/difference-between-batch-gradient-descent-and-stochastic-gradient-descent/

Ali, A. F., Hassanien, A. E., & Snášel, V. (2013, August). The Nelder-Mead Simplex Method with Variables Partitioning for Solving Large Scale Optimization Problems. In *Advances in intelligent systems and computing* (pp. 271–284). https://doi.org/10.1007/978-3-319-01781-5{_}25

Anyoha, R. (2017, August). The History of Artificial Intelligence. https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/

Azaria, A. (2022). ChatGPT Usage and Limitations. *HAL Open Science*, *hal-03913837*.

Baba, A. (2020, January). The beam finite element explained. https://www.sesamx.io/blog/beam_finite_element/

Baier-Saip, J. A., Baier, P. A., de Faria, A. R., Oliveira, J. C., & Baier, H. (2020). Shear locking in one-dimensional finite element methods. *European Journal of Mechanics, A/Solids*, *79*. https://doi.org/10.1016/j.euromechsol.2019.103871

Behmanesh, I., Moaveni, B., Lombaert, G., & Papadimitriou, C. (2015). Hierarchical Bayesian model updating for structural identification. *Mechanical Systems and Signal Processing*, *64-65*. https://doi.org/10.1016/j.ymssp.2015.03.026

Bock, S., & Weis, M. (2019). A Proof of Local Convergence for the Adam Optimizer. *2019 International Joint Conference on Neural Networks (IJCNN)*, 1–8. https://doi.org/10.1109/IJCNN.2019.8852239

Bower, A. F. (2008). Theory and Implementation of the Finite Element Method. https://solidmechanics.org/text/Chapter8_6/Chapter8_6.htm

Brmf. (2020, November). Tvärbanan. https://en.wikipedia.org/wiki/Tv%C3%A4rbanan#/media/File:Tvarbana3.png

Broo, H., Lundgren, K., & Plos, M. (2008). *A guide to non-linear finite element modelling of shear and torsion in concrete bridges* (tech. rep.). Chalmers University of Technology. Gothenburg.

Brownlee, J. (2023, October). A Gentle Introduction to k-fold Cross-Validation. https://machinelearningmastery.com/k-fold-cross-validation/

Brownlee, J. (2021, October). Gradient Descent Optimization With Nadam From Scratch. https://machinelearningmastery.com/gradient-descent-optimization-with-nadam-from-scratch/

Buhl, N. (2023, July). Activation Functions in Neural Networks: With 15 examples. https://encord.com/blog/activation-functions-neural-networks/

CAE Assistant. (2022, December). Introduction to Finite Element Method | Finite Element Analysis. https://caeassistant.com/blog/finite-element-method/

Chang, K. H. (2012). Stochastic Nelder-Mead simplex method - A new globally convergent direct search method for simulation optimization. *European Journal of Operational Research*, *220*(3). https://doi.org/10.1016/j.ejor.2012.02.028

Charlotte, M., Núnez, I. F., Gourinat, Y., & Matignon, D. (2023). Port-Hamiltonian Formulations of Some Elastodynamics Theories of Isotropic and Linearly Elastic Shells: Naghdi–Reissner's Moderately Thick Shells. *Applied Sciences (Switzerland)*, *13*(4). https://doi.org/10.3390/app13042608

Chen, H., Batchelor-McAuley, C., Kätelhön, E., Elliott, J., & Compton, R. G. (2022). A Critical Evaluation of Using Physics-Informed Neural Networks for Simulating Voltammetry: Strengths, Weaknesses and Best Practices. *Journal of Electroanalytical Chemistry*, *925*. https://doi.org/10.1016/j.jelechem.2022.116918

Chen, L., Zaharia, M., & Zou, J. (2023, October). *How is ChatGPT's behavior changing over time?* (Tech. rep.). Stanford University & UC Berkeley. https://arxiv.org/abs/2307.09009v3

Cheng Yu, J. (n.d.). Numerical Optimization. https://www.jade-cheng.com/au/coalhmm/optimization/

Chinosi, C., Della Croce, L., & Scapolla, T. (1998). Hierarchic finite elements for thin naghdi shell model. *International Journal of Solids and Structures*, *35*(16). https://doi.org/10.1016/S0020-7683(97)83328-8

Chowdhary, K. R. (2020). *Fundamentals of artificial intelligence*. https://doi.org/10.1007/978-81-322-3972-7

Coombs, W. M., Charlton, T. J., Cortis, M., & Augarde, C. E. (2018). Overcoming volumetric locking in material point methods. *Computer Methods in Applied Mechanics and Engineering*, *333*. https://doi.org/10.1016/j.cma.2018.01.010

Copeland, B. (2024, February). artificial intelligence. https://www.britannica.com/technology/artificial-intelligence/Methods-and-goals-in-AI

Das, S. (2023, March). Optimization Algorithms and Interactive Visualization (Part 2). https://medium.com/deepkapha-notes/optimization-algorithms-and-interactive-visualization-part-2-4d6d9791e1d3

Devriendt, M. (2012). Trigger levels for displacement monitoring. *The Canadian Geotechnical Society*, 23–25. https://cgs.ca/pdf/GeoTechNews/2012/GIN%203001.pdf

Ehrlinger, L., & Wöß, W. (2022). A Survey of Data Quality Measurement and Monitoring Tools. https://doi.org/10.3389/fdata.2022.850611

Eiselt, H. A., & Sandblom, C.-L. (2007). The Simplex Method. In *Linear programming and its applications* (pp. 129–165). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-73671-4{_}5

Ekin, S. (2023). Prompt Engineering for ChatGPT: A Quick Guide to Techniques, Tips, and Best Practices. *IEEE TechRxiv*, *58*.

Enckell, M. (2006, September). *Structural Health Monitoring using Modern Sensor Technology* (tech. rep.). KTH. Stockholm.

Erdem, K. (2020, May). Introduction to Extreme Learning Machines. https://towardsdatascience.com/introduction-to-extreme-learning-machines-c020020ff82b

Ereiz, S., Duvnjak, I., & Fernando Jiménez-Alonso, J. (2022). Review of finite element model updating methods for structural applications. https://doi.org/10.1016/j.istruc.2022.05.041

Felippa, C. A. (2001). *INTRODUCTION to FINITE ELEMENT METHODS*. University of Colorado. https://quickfem.com/wp-content/uploads/IFEM.Ch00.pdf

Ferreira, A. J., & Fantuzzi, N. (2009). MATLAB codes for finite element analysis: Solids and structures. *Solid Mechanics and its Applications*, *157*.

Franco, F. (2024, January). Activation Functions in Neural Networks. https://ai.plainenglish.io/activation-functions-in-neural-networks-9f0170334bf5

Fulford, I., & Ng, A. (n.d.-a). Building Systems with the ChatGPT API. https://learn.deeplearning.ai/courses/chatgpt-building-system/lesson/2/language-models%2C-the-chat-format-and-tokens

Fulford, I., & Ng, A. (n.d.-b). ChatGPT Prompt Engineering for Developers. https://learn.deeplearning.ai/courses/chatgpt-prompt-eng/lesson/1/introduction?_gl=1*1vwrcg5*_ga*ODc1NTk1NDAuMTcxMDIyNjkyOA..*_ga_PZF1GBS1R1*MTcxMTAwNzcyMC41LjEuMTcxMTAwNzc2OC4xMi4wLjA.

Giray, L. (2023). Prompt Engineering with ChatGPT: A Guide for Academic Writers. *Annals of Biomedical Engineering*, *51*(12), 2629–2633. https://doi.org/10.1007/s10439-023-03272-4

Hanly, S. (n.d.). Accelerometers: Taking the Guesswork out of Accelerometer Selection. https://blog.endaq.com/accelerometer-selection

Harish, A. (2023, November). What is Convergence in Finite Element Analysis? https://www.simscale.com/blog/convergence-finite-element-analysis/

Harish, A. B., & Matikainen, M. K. (2023). Alleviation techniques for volumetric locking in elements based on the absolute nodal coordinate formulation. *Finite Elements in Analysis and Design*, *224*. https://doi.org/10.1016/j.finel.2023.103990

Harsoor, S. (2023, June). Transformer Model And variants of Transformer (ChatGPT). https://pub.aimind.so/transformer-model-and-variants-of-transformer-chatgpt-3d423676e29c

Hassan, E., Shams, M. Y., Hikal, N. A., & Elmougy, S. (2023). The effect of choosing optimizer algorithms to improve computer vision tasks: a comparative study. *Multimedia Tools and Applications*, *82*(11). https://doi.org/10.1007/s11042-022-13820-0

Hermawanto, D. (2013, August). *Genetic Algorithm for Solving Simple Mathematical Equality Problem* (tech. rep.). Indonesian Institute of Sciences (LIPI). Indonesia. https://arxiv.org/pdf/1308.4675

Hutanu, A. (2023, February). How ChatGPT works and AI, ML & NLP Fundamentals. https://www.pentalog.com/blog/tech-trends/chatgpt-fundamentals/

IBM. (n.d.). What is gradient descent? https://www.ibm.com/topics/gradient-descent

IBM Data and AI Team. (2023, October). Understanding the different types of artificial intelligence. https://www.ibm.com/blog/understanding-the-different-types-of-artificial-intelligence/

Jiang, L. (2020, June). A Visual Explanation of Gradient Descent Methods (Momentum, AdaGrad, RMSProp, Adam). https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

Kapoor, T., Wang, H., Nunez, A., & Dollevoet, R. (2024). Physics-Informed Neural Networks for Solving Forward and Inverse Problems in Complex Beam Systems. *IEEE Transactions on Neural Networks and Learning Systems*, *35*(5). https://doi.org/10.1109/TNNLS.2023.3310585

Khoa, V. V., & Takayama, S. (2018). Wireless sensor network in landslide monitoring system with remote data management. *Measurement: Journal of the International Measurement Confederation*, *118*. https://doi.org/10.1016/j.measurement.2018.01.002

Khouri Chalouhi, E. (2022, September). *Optimal design solutions of road bridges considering embedded environmental impact and cost* [Doctoral dissertation, KTH Royal Institute of Technology].

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*.

Kittur, M. G., & Huston, R. L. (1990). Finite element mesh refinement criteria for stress analysis. *Computers and Structures*, *34*(2). https://doi.org/10.1016/0045-7949(90)90368-C

Kostadinov, S. (2019, October). Understanding Backpropagation Algorithm. https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd

Kramer, O., Ciaurri, D. E., & Koziel, S. (2011). Derivative-free optimization. *Studies in Computational Intelligence*, *356*. https://doi.org/10.1007/978-3-642-20859-1{_}4

Lewis, R. M., Torczon, V., & Trosset, M. W. (2000). Direct search methods: Then and now. *Journal of Computational and Applied Mathematics*, *124*(1-2). https://doi.org/10.1016/S0377-0427(00)00423-4

Machine Learning Mastery. (2023). Prompt Engineering for Effective Interaction with ChatGPT. *Machine Learning Mastery*. https://machinelearningmastery.com/prompt-engineering-for-effective-interaction-with-chatgpt/

Majid Solat Yavari, Costin Pacoste, & Raid Karoumi. (2016). Structural Optimization of Concrete Slab Frame Bridges Considering Investment Cost. *Journal of Civil Engineering and Architecture*, *10*(9). https://doi.org/10.17265/1934-7359/2016.09.002

Markowska-Kaczmar, U., & Kosturek, M. (2021). Extreme learning machine versus classical feedforward network: Comparison from the usability perspective. https://doi.org/10.1007/s00521-021-06402-y

Marwala, T. (2010). *Finite-element-model updating using computional intelligence techniques: Applications to structural dynamics*. https://doi.org/10.1007/978-1-84996-323-7

Mashwani, W. K., Haider, R., & Brahim Belhaouari, S. (2021). A Multiswarm Intelligence Algorithm for Expensive Bound Constrained Optimization Problems. *Complexity*, *2021*. https://doi.org/10.1155/2021/5521951

MathWorks. (n.d.). What Is Overfitting? https://se.mathworks.com/discovery/overfitting.html

Meethal, R. E., Kodakkal, A., Khalil, M., Ghantasala, A., Obst, B., Bletzinger, K. U., & Wüchner, R. (2023). Finite element method-enhanced neural network for forward and inverse problems. *Advanced Modeling and Simulation in Engineering Sciences*, *10*(1). https://doi.org/10.1186/s40323-023-00243-1

Moseley, B. (2021, October). So, what is a physics-informed neural network? https://benmoseley.blog/my-research/so-what-is-a-physics-informed-neural-network/

Moseley, B., Markham, A., & Nissen-Meyer, T. (2023). Finite basis physics-informed neural networks (FBPINNs): a scalable domain decomposition approach for solving differential equations. *Advances in Computational Mathematics*, *49*(4). https://doi.org/10.1007/s10444-023-10065-9

Murel, J., & Kavlakoglu, E. (2023, November). What is regularization? https://www.ibm.com/topics/regularization

Neto, M. A., Amaro, A., Luis, R., Cirne, J., & Leal, R. (2015). *Engineering computation of structures: The finite element method*. https://doi.org/10.1007/978-3-319-17710-6

Nocedal, J., & Wright, S. (1996). *Numerical Optimization Second Edition* (Vol. 17).

Oden, J. T., & Fost, R. B. (1973). Convergence, accuracy and stability of finite element approximations of a class of non-linear hyperbolic equations. *International Journal for Numerical Methods in Engineering*, *6*(3). https://doi.org/10.1002/nme.1620060307

OpenAI. (n.d.-a). GPT-4 and GPT-4 Turbo. https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo

OpenAI. (n.d.-b). Transforming work and creativity with AI. https://openai.com/product

OpenAI. (n.d.-c). Usage tiers. https://platform.openai.com/docs/guides/rate-limits/usage-tiers?context=tier-free

Patnaik, S. N., Berke, L., & Gallagher, R. H. (1991). Compatibility conditions of structural mechanics for finite element analysis. *AIAA Journal*, *29*(5). https://doi.org/10.2514/3.10662

Peck, R. B. (1969). Advantages and limitations of the observational method in applied soil mechanics. *Geotechnique*, *19*(2). https://doi.org/10.1680/geot.1969.19.2.171

Ranghar, A. (2023, December). Introduction to Recurrent Neural Network. https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

Reschetiuk, Y., Rylander, M., Haas, H., Andersson, R., Mohlind, B., & Ekström, I. (2012, May). *MÄTMETODER FÖR RÖRELSEÖVERVAKNING AV FYLLNINGSDAMMAR [MEASUREMENT METHODS FOR MOVEMENT OBSERVATION FOR EMBANKMENT DAMS]* (tech. rep.). Energiforsk. https://roctest.com/wp-content/uploads/2017/03/C95.pdf

Roy, R. (2023, May). ML | Stochastic Gradient Descent (SGD). https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/

Singh, A. (2010). *COST FUNCTIONS FOR SUPERVISED LEARNING BASED ON A ROBUST SIMILARITY METRIC* [Doctoral dissertation, University of Florida]. https://ufdcimages.uflib.ufl.edu/UF/E0/04/17/82/00001/singh_a.pdf

Singh, A. (2020). Heuristic algorithms. https://optimization.cbe.cornell.edu/index.php?title=Heuristic_algorithms

SL. (n.d.). Tvärbanan. https://sl.se/reseplanering/var-trafik/tvarbanan

Stockholms stad. (2024, February). Tvärbanan till Helenelund. https://vaxer.stockholm/projekt/kista/tvarbanan-till-helenelund/

Svedholm, C., Kollén, M., & Uhlán, M. (n.d.). SL:S NYA SPÅRVÄGSBRO ÖVER MÄLARBANAN. https://www.stalbyggnad.se/stalbroar/sls-nya-sparvagsbro-over-malarbanan/

Tam, A. (2021, October). A Gentle Introduction to Particle Swarm Optimization. https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/

TE Connectivity. (2017, July). CHOOSING THE RIGHT TYPE OF ACCELEROMETER. https://www.mouser.com/pdfdocs/choosing-the-right-accelerometer-white-paper.pdf

Van Veen, F. (2016, September). The Neural Network Zoo. https://www.asimovinstitute.org/neural-network-zoo/

Venkatesh, K., Prakash, S., & Srinivasa Murthy, P. (2018). Completeness-Compatibility and Reviews for Finite Element Analysis Results. *International Journal of Engineering Research & Technology*, *1*(10). https://www.ijert.org/research/completeness-compatibility-and-reviews-for-finite-element-analysis-results-IJERTV2IS1123.pdf

Wan, H. P., & Ren, W. X. (2016). Stochastic model updating utilizing Bayesian approach and Gaussian process model. *Mechanical Systems and Signal Processing*, *70-71*. https://doi.org/10.1016/j.ymssp.2015.08.011

Wikipedia. (2024a, January). Activation function. https://en.wikipedia.org/wiki/Activation_function

Wikipedia. (2023, August). Bending. https://en.wikipedia.org/wiki/Bending

Wikipedia. (2024b, April). MEMS. https://en.wikipedia.org/wiki/MEMS

Wikipedia. (2024c, March). Tvärbanan. https://en.wikipedia.org/w/index.php?title=Tv%C3%A4rbanan&action=history&offset=&limit=500

Wright, S. J. (2022, October). simplex method. https://www.britannica.com/topic/simplex-method

Yehia, S., Abudayyeh, O., Abdel-Qader, I., Zalt, A., & Meganathan, V. (2008). Evaluation of sensor performance for concrete applications. *Transportation Research Record*, (2050). https://doi.org/10.3141/2050-10

Yu Cheng, J., & Mailund, T. (2015, March). An iteration of the Nelder-Mead method over two-dimensional space. https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method#/media/File:An-iteration-of-the-Nelder-Mead-method-over-two-dimensional-space-showing-point-p-min.png

# A

# Appendix

## A.1 General plan for case-study

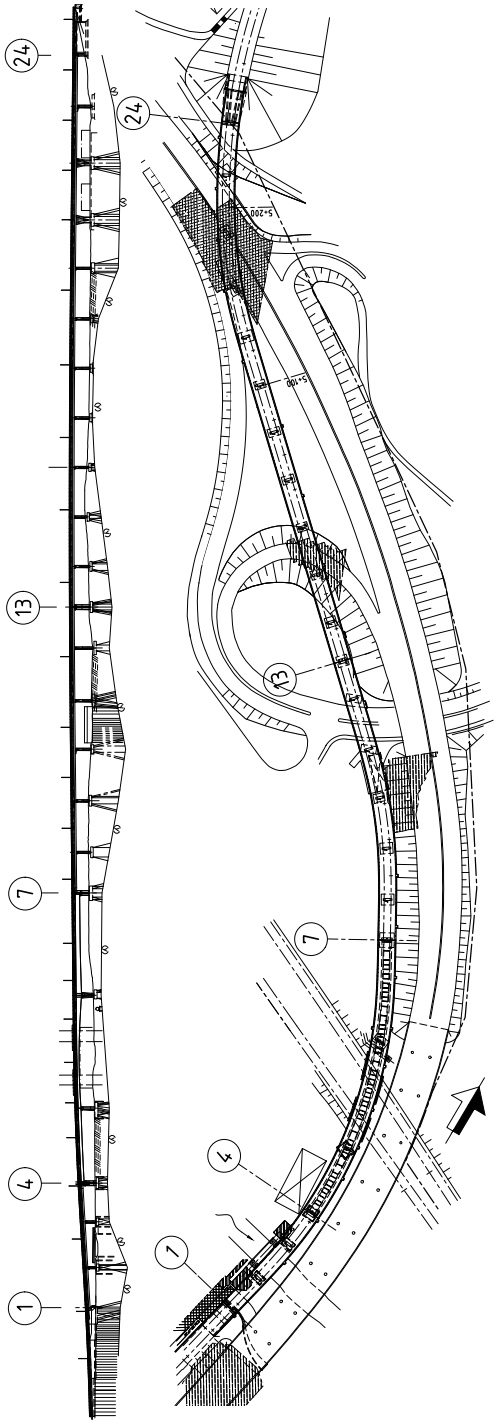



**Figure A.1:** Plans and drawings of the brigade in Solvalla. Segments of interest for this thesis consists of those between support 4 and 7.

## A.2 Optimisation via Neural Network

### A.2.1 FFR. Sensitivity against decrease in training data

**Table A.1:** Neural Network, Forward Feed prediction performance for test cases with decreased training data. Total run time (RT) is represented by times required through produce traning data (TD) and to train the model (TM).

| | Optimizer performance for deflection (U) | | |
|---|---|---|---|
| Model, amount of data (data [%]) | MAPE U [%] | Run time (RT) | (TD / TM) [s] |
| Test case A, batch = 256 | | | |
| FF-4L512N-ReLU 4055, (100%) | 0.8371 | 61492.9 | (61354.4/ 138.5) |
| FF-4L512N-ReLU 2028, (50%) | 0.8038 | 30783.0 | (30677.2/ 105.8) |
| FF-4L512N-ReLU 1014, (25%) | 1.5874 | 15421.9 | (15338.6/ 83.3) |
| FF-4L512N-ReLU 406, (10%) | 2.6595 | 6161.7 | (6135.4/ 26.3) |
| FF-4L512N-ReLU 312, (8%) | 2.8831 | 4928.5 | (4908.4/ 20.1) |
| FF-4L512N-ReLU 239, (6%) | 3.5189 | 3705.8 | (3681.3/ 24.5) |
| FF-4L512N-ReLU 203, (5%) | 4.7848 | 3091.0 | (3067.7/ 23.3) |
| FF-4L512N-ReLU 163, (4%) | 10.1070 | 2480.1 | (2454.2/ 25.9) |
| FF-4L512N-ReLU 82, (2%) | 13.3984 | 1248.7 | (1227.1/ 21.6) |
| FF-4L512N-ReLU 41, (1%) | 23.9770 | 637.5 | (613.5/ 24.0) |
| Test case B, batch = 256 | | | |
| FF-8L512N-ReLU 9485, (100%) | 0.4985 | 125851.1 | (125018.2/ 832.9) |
| FF-8L512N-ReLU 4743, (50%) | 0.6522 | 62674.9 | (62509.1/165.8 ) |
| FF-8L512N-ReLU 2372, (25%) | 0.8664 | 31338.1 | (31254.6/ 83.5 |
| FF-8L512N-ReLU 948, (10%) | 2.2471 | 12577.3 | (12501.8/ 75.5) |
| FF-8L512N-ReLU 730, (8%) | 2.9153 | 10041.2 | (10001.5/ 39.7) |
| FF-8L512N-ReLU 558, (6%) | 3.6936 | 7530.4 | (7501.1/ 29.3) |
| FF-8L512N-ReLU 475, (5%) | 5.1483 | 6277.4 | (6250.9/ 26.5) |
| FF-8L512N-ReLU 380, (4%) | 8.9936 | 5015.1 | (5000.7/ 14.4) |
| FF-8L512N-ReLU 189, (2%) | 11.2906 | 2522.4 | (2500.4/ 22.0) |
| FF-8L512N-ReLU 95, (1%) | 24.2388 | 1261.9 | (1250.2/ 11.7) |
| Test case C, batch = 512 | | | |
| FF-16L512N-ReLU 12345, (100%) | 0.4340 | 200280.2 | (198304.2/ 1976.0) |
| FF-16L512N-ReLU 6138, (50%) | 0.7833 | 99977.2 | (99152.1/ 825.1) |
| FF-16L512N-ReLU 3087, (25%) | 1.2031 | 50001.5 | (49576.1/ 425.4) |
| FF-16L512N-ReLU 1235, (10%) | 2.6857 | 20148.1 | (19830.4/ 317.7) |
| FF-16L512N-ReLU 950, (8%) | 3.1588 | 16029.7 | (15864.3/ 165.4) |
| FF-16L512N-ReLU 727, (6%) | 3.3752 | 12104.8 | (11898.3/ 206.5) |
| FF-16L512N-ReLU 618, (5%) | 4.2048 | 10085.2 | (9915.2/ 170.0) |
| FF-16L512N-ReLU 494, (4%) | 9.9244 | 8004.7 | (7932.2/ 72.5) |
| FF-16L512N-ReLU 247, (2%) | 15.3414 | 4052.6 | (3966.1/ 86.5) |
| FF-16L512N-ReLU 124, (1%) | 36.7289 | 2040.2 | (1983.1/ 57.1) |

## A.3 Optimisation via ChatGPT 3.5 & 4

Case: A

Method: General and consise prompt, minimal numerical alaysis. ChatGPT 3.5 turbo and 4 (first half 3.5 and secound 4)

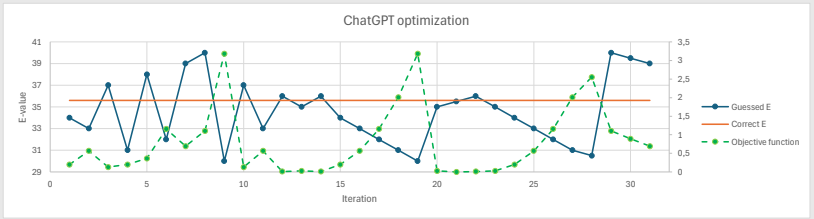
Prompt instructions: "Rules: You are suppose to optimise a problem. As input, i will supply you with the objective function value that is to be minimized followed by corresponding variable value. You can impact it by chaning the variable parameter E , which must lay between 30000000000 and 40000000000. Give the new E value and explanation. Dont provide code, you should answer an estimation on E"

User input prompt: Obj=XX for E=XX

Remarks: Model repeat itself, do not seem to analyse or consider previous steps. Seamingly lacking analytical capabilities as unlogical variations are given.

Results:

| Iteration | Objective function | Guessed E | Correct E | E error [%] |
|---|---|---|---|---|
| 1 | 0,2023 | 34 | 35,6 | 4,49% |
| 2 | 0,5670 | 33 | 35,6 | 7,30% |
| 3 | 0,1307 | 37 | 35,6 | 3,93% |
| 4 | 0,2011 | 31 | 35,6 | 12,92% |
| 5 | 0,3642 | 38 | 35,6 | 6,74% |
| 6 | 1,1560 | 32 | 35,6 | 10,11% |
| 7 | 0,6939 | 39 | 35,6 | 9,55% |
| 8 | 1,1051 | 40 | 35,6 | 12,36% |
| 9 | 3,1827 | 30 | 35,6 | 15,73% |
| 10 | 0,1307 | 37 | 35,6 | 3,93% |
| 11 | 0,5670 | 33 | 35,6 | 7,30% |
| 12 | 0,0113 | 36 | 35,6 | 1,12% |
| 13 | 0,0268 | 35 | 35,6 | 1,69% |
| 14 | 0,0113 | 36 | 35,6 | 1,12% |
| 15 | 0,2023 | 34 | 35,6 | 4,49% |
| 16 | 0,5670 | 33 | 35,6 | 7,30% |
| 17 | 1,1560 | 32 | 35,6 | 10,11% |
| 18 | 2,0110 | 31 | 35,6 | 12,92% |
| 19 | 3,1827 | 30 | 35,6 | 15,73% |
| 20 | 0,0268 | 35 | 35,6 | 1,69% |
| 21 | 0,0007 | 35,5 | 35,6 | 0,28% |
| 22 | 0,0113 | 36 | 35,6 | 1,12% |
| 23 | 0,0268 | 35 | 35,6 | 1,69% |
| 24 | 0,2023 | 34 | 35,6 | 4,49% |
| 25 | 0,5670 | 33 | 35,6 | 7,30% |
| 26 | 1,1560 | 32 | 35,6 | 10,11% |
| 27 | 2,0110 | 31 | 35,6 | 12,92% |
| 28 | 2,5537 | 30,5 | 35,6 | 14,33% |
| 29 | 1,1051 | 40 | 35,6 | 12,36% |
| 30 | 0,8901 | 39,5 | 35,6 | 10,96% |
| 31 | 0,6939 | 39 | 35,6 | 9,55% |



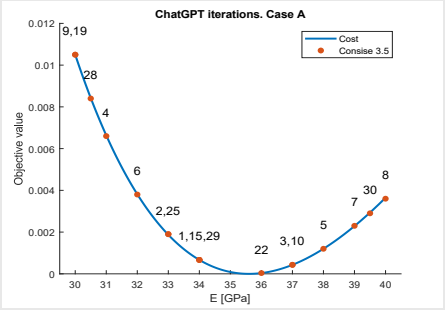




**Figure A.2:** Results and iterations of ChatGPT. Case A, concise prompt. Depicts instructions, prompts, remarks, iterations, development of values over iterations and objective function value over iterations.

Case: A

Method: General prompt, minimal numerical alaysis, introduction of chain of thought reasoning. ChatGPT 3.5 turbo

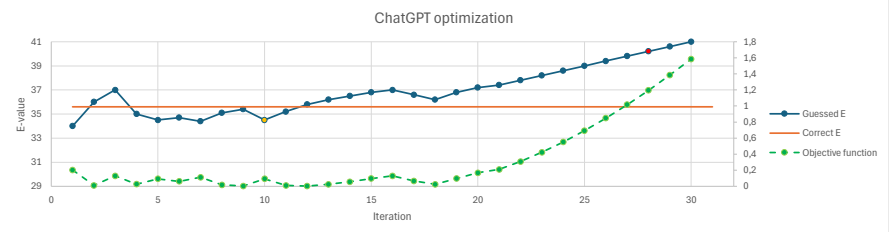
Prompt instructions: Rules: "You are suppose to optimise a problem. As input, i will supply you with the objective function value (obj) that is to be minimized followed by corresponding variable value. You can impact it by chaning the variable parameter (E) , which must lay between 30000000000 and 40000000000. Try to estimate an appropriate uppdated E based on previouse number of trials and values. Dont provide code, instead after resoning i want the new E in chat. Reason in steps.
1) Summarize all previouse obj values and thair corresponding E
2) What are the 5 best values soo far?
3) What is the mean of 5 best E?
4) Based on above, state the new E estimate. Avoid duplicates of already tried E"

User input prompt: Obj=XX for E=XX

Remarks: When forces analytical steps, forcing backtracking, previous values seemed to be considered. More logical iterations, however seemed to yield somewhat unreasonable solutions, stepping in faulty directions. Further fails to follow bounderies and instructions, with duplicate solution at 10 and exceeding boundery after 28 iterations.

Results:

| Iteration | Objective function | Guessed E | Correct E | E error [%] |
|---|---|---|---|---|
| 1 | 0,2023 | 34 | 35,6 | 4,49% |
| 2 | 0,0113 | 36 | 35,6 | 1,12% |
| 3 | 0,1307 | 37 | 35,6 | 3,93% |
| 4 | 0,0268 | 35 | 35,6 | 1,69% |
| 5 | 0,0929 | 34,5 | 35,6 | 3,09% |
| 6 | 0,0615 | 34,7 | 35,6 | 2,53% |
| 7 | 0,1113 | 34,4 | 35,6 | 3,37% |
| 8 | 0,0186 | 35,1 | 35,6 | 1,40% |
| 9 | 0,0029 | 35,4 | 35,6 | 0,56% |
| 10 | 0,0929 | 34,5 | 35,6 | 3,09% |
| 11 | 0,0118 | 35,2 | 35,6 | 1,12% |
| 12 | 0,0028 | 35,8 | 35,6 | 0,56% |
| 13 | 0,0251 | 36,2 | 35,6 | 1,69% |
| 14 | 0,0555 | 36,5 | 35,6 | 2,53% |
| 15 | 0,0970 | 36,8 | 35,6 | 3,37% |
| 16 | 0,1307 | 37 | 35,6 | 3,93% |
| 17 | 0,0681 | 36,6 | 35,6 | 2,81% |
| 18 | 0,0251 | 36,2 | 35,6 | 1,69% |
| 19 | 0,0970 | 36,8 | 35,6 | 3,37% |
| 20 | 0,1688 | 37,2 | 35,6 | 4,49% |
| 21 | 0,2115 | 37,4 | 35,6 | 5,06% |
| 22 | 0,3093 | 37,8 | 35,6 | 6,18% |
| 23 | 0,4230 | 38,2 | 35,6 | 7,30% |
| 24 | 0,5514 | 38,6 | 35,6 | 8,43% |
| 25 | 0,6939 | 39 | 35,6 | 9,55% |
| 26 | 0,8493 | 39,4 | 35,6 | 10,67% |
| 27 | 1,0168 | 39,8 | 35,6 | 11,80% |
| 28 | 1,1956 | 40,2 | 35,6 | 12,92% |
| 29 | 1,3850 | 40,6 | 35,6 | 14,04% |
| 30 | 1,5841 | 41 | 35,6 | 15,17% |



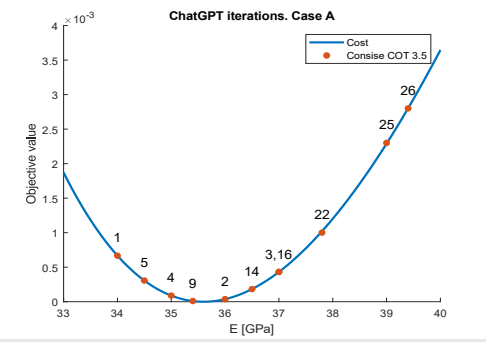




**Figure A.3:** Results and iterations of ChatGPT. Case A, concise prompt through chain of thought. Depicts instructions, prompts, remarks, iterations, development of values over iterations and objective function value over iterations.

Case: A

Method: General prompt, minimal numerical alaysis, introduction of chain of thought reasoning. ChatGPT 3.5 turbo

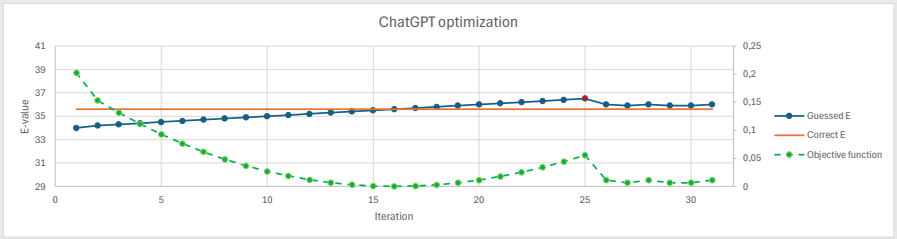
Prompt instructions: Rules: You are suppose to optimise a problem. As input, i will supply you with the objective function value (obj) that is to be minimized followed by corresponding variable value. You can impact it by chaning the variable parameter (E) , which must lay between 30 and 40. Try to estimate an appropriate uppdated E, by atleast 2 decimals , based on previouse number of trials and values. Dont provide code, instead after resoning i want the new E in chat. Reason in following steps
"1 Summarize all previouse obj values and thair corresponding E
2 What are the 5 best values so far?
3 What is the mean of 5 best E?
4 Does the last 3 iterations follow a bad direction, should E change in current direction?
5 Based on above best mean values and previouse trails, state the new E estimate. Avoid duplicates of already tried E and stricly consider the bounderies."

User input prompt: Obj=XX for E=XX

Remarks: When forces analytical steps, forcing backtracking, historical values seemed to be considered. However, linear steps of the E-modulus were again given, stepping away from optimal. Lack of analytical consideration is noted even if vocabulary reasoning often is correct. Fails simple math. Breaks rules completly after 25 iterations and gets stuck at 36Gpa even if better has been previously tried.

Results:

| Iteration | Objective function | Guessed E | Correct E | E error [%] |
|---|---|---|---|---|
| 1 | 0,2023 | 34 | 35,6 | 4,49% |
| 2 | 0,1531 | 34,2 | 35,6 | 3,93% |
| 3 | 0,1312 | 34,3 | 35,6 | 3,65% |
| 4 | 0,1112 | 34,4 | 35,6 | 3,37% |
| 5 | 0,0929 | 34,5 | 35,6 | 3,09% |
| 6 | 0,0764 | 34,6 | 35,6 | 2,81% |
| 7 | 0,0615 | 34,7 | 35,6 | 2,53% |
| 8 | 0,0483 | 34,8 | 35,6 | 2,25% |
| 9 | 0,0368 | 34,9 | 35,6 | 1,97% |
| 10 | 0,0268 | 35 | 35,6 | 1,69% |
| 11 | 0,0186 | 35,1 | 35,6 | 1,40% |
| 12 | 0,0118 | 35,2 | 35,6 | 1,12% |
| 13 | 0,0066 | 35,3 | 35,6 | 0,84% |
| 14 | 0,0029 | 35,4 | 35,6 | 0,56% |
| 15 | 0,0007 | 35,5 | 35,6 | 0,28% |
| 16 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 17 | 0,0007 | 35,7 | 35,6 | 0,28% |
| 18 | 0,0028 | 35,8 | 35,6 | 0,56% |
| 19 | 0,0064 | 35,9 | 35,6 | 0,84% |
| 20 | 0,0113 | 36 | 35,6 | 1,12% |
| 21 | 0,0175 | 36,1 | 35,6 | 1,40% |
| 22 | 0,0251 | 36,2 | 35,6 | 1,69% |
| 23 | 0,0339 | 36,3 | 35,6 | 1,97% |
| 24 | 0,0441 | 36,4 | 35,6 | 2,25% |
| 25 | 0,0555 | 36,5 | 35,6 | 2,53% |
| 26 | 0,0113 | 36 | 35,6 | 1,12% |
| 27 | 0,0064 | 35,9 | 35,6 | 0,84% |
| 28 | 0,0113 | 36 | 35,6 | 1,12% |
| 29 | 0,0064 | 35,9 | 35,6 | 0,84% |
| 30 | 0,0064 | 35,9 | 35,6 | 0,84% |



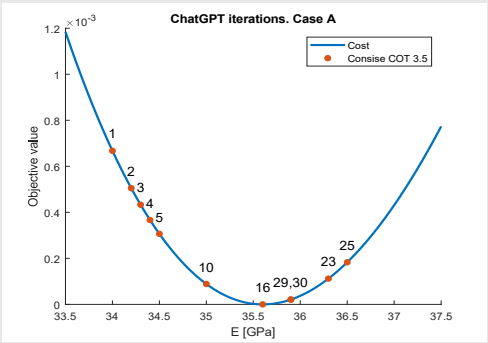




**Figure A.4:** Results and iterations of ChatGPT. Case A, concise prompt through chain of thought. Depicts instructions, prompts, remarks, iterations, development of values over iterations and objective function value over iterations.

Case: A

Method: General prompt, minimal numerical alaysis, introduction of chain of thought reasoning. ChatGPT 3.5 turbo

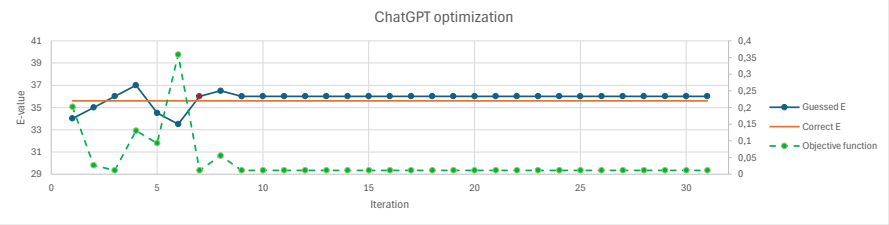
Prompt instructions: Rules: You are suppose to optimise a problem. As input, i will supply you with the objective function value (obj) that is to be minimized followed by corresponding variable value. You can impact it by chaning the variable parameter (E) , which must lay between 30 and 40. Try to estimate an appropriate uppdated E based on previouse number of trials and values. Optimize E to atleast 2 decimal places. Dont provide code, instead after resoning i want the new E in chat. Reason in following steps
"1 Summarize all previouse obj values and thair corresponding E
2 What are the 5 best values so far?
3 What is the mean of 5 best E. Stricly evaluate for only 5 values?
4 Does the last 3 iterations follow a bad direction, should E change in current direction? If it is a promising direction, larger stepsize of E might be appropriate.
5 Based on above best mean values and previouse trails, state the new E estimate. Avoid duplicates of already tried E and stricly consider the

User input prompt: Obj=XX for E=XX

Remarks: When forces analytical steps, forcing backtracking, historical values seemed to be considered. More logical behavior is seen early on, however there do not not seam to be correct and complete analytically reason as new values are stated without fully considering history. Fails early after just 7 iterations. Both instruction following and logic

Results:

| Iteration | Objective function | Guessed E | Correct E | E error [%] |
|---|---|---|---|---|
| 1 | 0,2023 | 34 | 35,6 | 4,49% |
| 2 | 0,0268 | 35 | 35,6 | 1,69% |
| 3 | 0,0113 | 36 | 35,6 | 1,12% |
| 4 | 0,1307 | 37 | 35,6 | 3,93% |
| 5 | 0,0929 | 34,5 | 35,6 | 3,09% |
| 6 | 0,3590 | 33,5 | 35,6 | 5,90% |
| 7 | 0,0113 | 36 | 35,6 | 1,12% |
| 8 | 0,0555 | 36,5 | 35,6 | 2,53% |
| 9 | 0,0113 | 36 | 35,6 | 1,12% |
| 10 | 0,0113 | 36 | 35,6 | 1,12% |
| 11 | 0,0113 | 36 | 35,6 | 1,12% |
| 12 | 0,0113 | 36 | 35,6 | 1,12% |
| 13 | 0,0113 | 36 | 35,6 | 1,12% |
| 14 | 0,0113 | 36 | 35,6 | 1,12% |
| 15 | 0,0113 | 36 | 35,6 | 1,12% |
| 16 | 0,0113 | 36 | 35,6 | 1,12% |
| 17 | 0,0113 | 36 | 35,6 | 1,12% |
| 18 | 0,0113 | 36 | 35,6 | 1,12% |
| 19 | 0,0113 | 36 | 35,6 | 1,12% |
| 20 | 0,0113 | 36 | 35,6 | 1,12% |
| 21 | 0,0113 | 36 | 35,6 | 1,12% |
| 22 | 0,0113 | 36 | 35,6 | 1,12% |
| 23 | 0,0113 | 36 | 35,6 | 1,12% |
| 24 | 0,0113 | 36 | 35,6 | 1,12% |
| 25 | 0,0113 | 36 | 35,6 | 1,12% |
| 26 | 0,0113 | 36 | 35,6 | 1,12% |
| 27 | 0,0113 | 36 | 35,6 | 1,12% |
| 28 | 0,0113 | 36 | 35,6 | 1,12% |
| 29 | 0,0113 | 36 | 35,6 | 1,12% |
| 30 | 0,0113 | 36 | 35,6 | 1,12% |



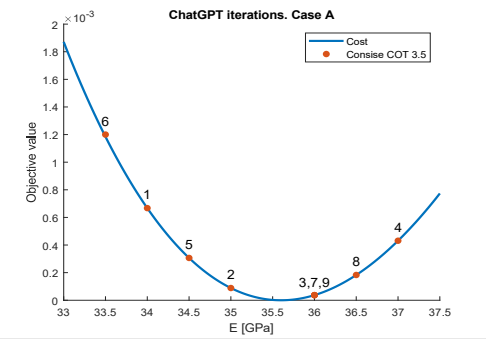




**Figure A.5:** Results and iterations of ChatGPT. Case A, concise prompt through chain of thought. Depicts instructions, prompts, remarks, iterations, development of values over iterations and objective function value over iterations.

Case: A

Method: General prompt, minimal numerical alaysis, introduction of chain of thought reasoning. ChatGPT 4

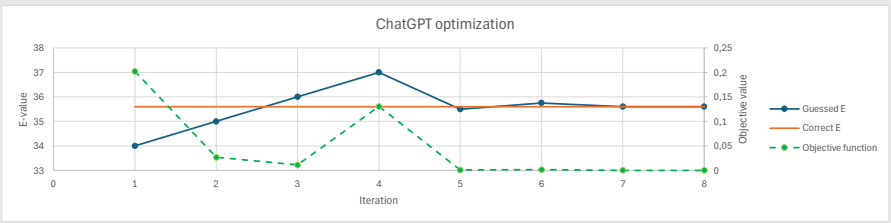
Prompt instructions: Rules: You are suppose to optimise a problem. As input, i will supply you with the objective function value (obj) that is to be minimized followed by corresponding variable value. You can impact it by chaning the variable parameter (E) , which must lay between 30 and 40. Try to estimate an appropriate uppdated E based on previouse number of trials and values. Optimize E to atleast 2 decimal places. Dont provide code, instead after resoning i want the new E in chat. Reason in following steps
"1 Summarize all previouse obj values and thair corresponding E
2 What are the 5 best values so far?
3 What is the mean of 5 best E. Stricly evaluate for only 5 values?
4 Does the last 3 iterations follow a bad direction, should E change in current direction? If it is a promising direction, larger stepsize of E might be appropriate.
5 Based on above best mean values and previouse trails, state the new E estimate. Avoid duplicates of already tried E and stricly consider the bounderies."

User input prompt: Obj=XX for E=XX

Remarks: Same promt as previous however with ChatGPT 4. This time it considers history and logically utilizes previous data. Converges and gives final and correct value after only 7 iterations.

Results:

| Iteration | Objective function | Guessed E | Correct E | E error [%] |
|---|---|---|---|---|
| 1 | 0,2023 | 34 | 35,6 | 4,49% |
| 2 | 0,0268 | 35 | 35,6 | 1,69% |
| 3 | 0,0113 | 36 | 35,6 | 1,12% |
| 4 | 0,1307 | 37 | 35,6 | 3,93% |
| 5 | 0,0007 | 35,5 | 35,6 | 0,28% |
| 6 | 0,0016 | 35,75 | 35,6 | 0,42% |
| 7 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 8 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 9 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 10 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 11 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 12 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 13 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 14 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 15 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 16 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 17 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 18 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 19 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 20 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 21 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 22 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 23 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 24 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 25 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 26 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 27 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 28 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 29 | 0,0000 | 35,6 | 35,6 | 0,00% |
| 30 | 0,0000 | 35,6 | 35,6 | 0,00% |



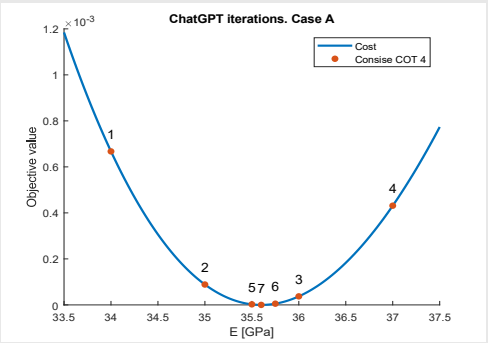




**Figure A.6:** Results and iterations of ChatGPT. Case A, concise prompt through chain of thought. Depicts instructions, prompts, remarks, iterations, development of values over iterations and objective function value over iterations.

Case: B

Method: General prompt, minimal numerical alaysis, introduction of chain of thought reasoning. ChatGPT 4

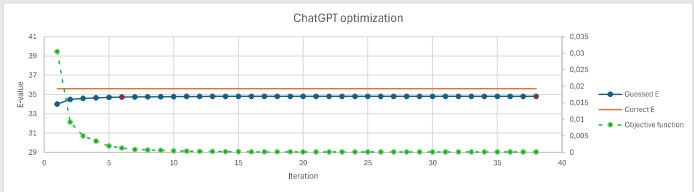
Prompt instructions: Rules: You are suppose to optimise a problem. As input, i will supply you with the objective function value (obj) that is to be minimized followed by corresponding variable values. You can impact it by chaning two variable parameter (E) and (K).E must lay between 30 and 40 while K bust lay between 1 and 100. Try to estimate an appropriate uppdated E and K based on previouse number of trials and values. Optimize E and K to atleast 2 decimal places. Dont provide code, instead after resoning i want the new E and K in chat. Reason in following steps
"1 Summarize all previouse obj values and thair corresponding E and K
2 What are the 5 best values so far?
3 What is the mean of 5 best E and K values. Stricly evaluate for only 5 values?
4 Does the last 3 iterations follow a bad direction, should E and K change in current direction? If it is a promising direction, larger variable stepsize might be appropriate.
5 Based on above best mean and previouse trails, state the new E and K estimate. Avoid duplicates of already tried E and K while stricly consider the bounderies."

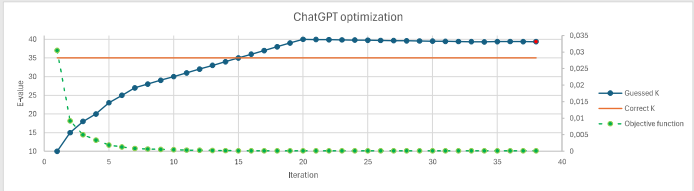
User input prompt: Obj=XX for E=XX and K=XX

Remarks: Same prompt and still ChatGPT 4, but extended to case B. Model now significantly struggles, indicating a false convergece early on, thus starts tweaking parameters by too small intervals, even if told not to. Also states converge after 34 iterations towards a fualty solution. ChatGPT fully fails to respond after 38 iterations

Results:

| Iteration | Objective function | Guessed K | Correct K | Guessed E | Correct E | E error [%] | E error [%] |
|---|---|---|---|---|---|---|---|
| 1 | 0,03050 | 10 | 35 | 34 | 35,6 | 4,49% | 71,43% |
| 2 | 0,00920 | 15 | 35 | 34,5 | 35,6 | 3,09% | 57,14% |
| 3 | 0,00500 | 18 | 35 | 34,6 | 35,6 | 2,81% | 48,57% |
| 4 | 0,00340 | 20 | 35 | 34,65 | 35,6 | 2,67% | 42,86% |
| 5 | 0,00190 | 23 | 35 | 34,7 | 35,6 | 2,53% | 34,29% |
| 6 | 0,00130 | 25 | 35 | 34,73 | 35,6 | 2,44% | 28,57% |
| 7 | 0,00086 | 27 | 35 | 34,75 | 35,6 | 2,39% | 22,86% |
| 8 | 0,00070 | 28 | 35 | 34,76 | 35,6 | 2,36% | 20,00% |
| 9 | 0,00058 | 29 | 35 | 34,77 | 35,6 | 2,33% | 17,14% |
| 10 | 0,00047 | 30 | 35 | 34,78 | 35,6 | 2,30% | 14,29% |
| 11 | 0,00038 | 31 | 35 | 34,79 | 35,6 | 2,28% | 11,43% |
| 12 | 0,00031 | 32 | 35 | 34,795 | 35,6 | 2,26% | 8,57% |
| 13 | 0,00026 | 33 | 35 | 34,798 | 35,6 | 2,25% | 5,71% |
| 14 | 0,00022 | 34 | 35 | 34,799 | 35,6 | 2,25% | 2,86% |
| 15 | 0,00019 | 35 | 35 | 34,7995 | 35,6 | 2,25% | 0,00% |
| 16 | 0,00017 | 36 | 35 | 34,7997 | 35,6 | 2,25% | 2,86% |
| 17 | 0,00015307 | 37 | 35 | 34,7998 | 35,6 | 2,25% | 5,71% |
| 18 | 0,00014408 | 38 | 35 | 34,7999 | 35,6 | 2,25% | 8,57% |
| 19 | 0,00015028 | 39 | 35 | 34,79995 | 35,6 | 2,25% | 11,43% |
| 20 | 0,00014097 | 40 | 35 | 34,79997 | 35,6 | 2,25% | 14,29% |
| 21 | 0,00014084 | 39,95 | 35 | 34,79997 | 35,6 | 2,25% | 14,14% |
| 22 | 0,00014072 | 39,9 | 35 | 34,79997 | 35,6 | 2,25% | 14,00% |
| 23 | 0,00014060 | 39,85 | 35 | 34,79997 | 35,6 | 2,25% | 13,86% |
| 24 | 0,00014060 | 39,8 | 35 | 34,79997 | 35,6 | 2,25% | 13,71% |
| 25 | 0,00014041 | 39,75 | 35 | 34,79997 | 35,6 | 2,25% | 13,57% |
| 26 | 0,00014033 | 39,7 | 35 | 34,79997 | 35,6 | 2,25% | 13,43% |
| 27 | 0,00014026 | 39,65 | 35 | 34,79997 | 35,6 | 2,25% | 13,29% |
| 28 | 0,00014020 | 39,6 | 35 | 34,79997 | 35,6 | 2,25% | 13,14% |
| 29 | 0,00014015 | 39,55 | 35 | 34,79997 | 35,6 | 2,25% | 13,00% |
| 30 | 0,00014011 | 39,5 | 35 | 34,79997 | 35,6 | 2,25% | 12,86% |
| 31 | 0,00014008 | 39,45 | 35 | 34,79997 | 35,6 | 2,25% | 12,71% |
| 32 | 0,00014006 | 39,4 | 35 | 34,79997 | 35,6 | 2,25% | 12,57% |
| 33 | 0,00014004 | 39,35 | 35 | 34,79997 | 35,6 | 2,25% | 12,43% |
| 34 | 0,00014005 | 39,3 | 35 | 34,79997 | 35,6 | 2,25% | 12,29% |
| 35 | 0,00014005 | 39,37 | 35 | 34,79997 | 35,6 | 2,25% | 12,49% |
| 36 | 0,00014006 | 39,4 | 35 | 34,79997 | 35,6 | 2,25% | 12,57% |
| 37 | 0,00014006 | 39,38 | 35 | 34,79997 | 35,6 | 2,25% | 12,51% |
| 38 | 0,00014005 | 39,36 | 35 | 34,79997 | 35,6 | 2,25% | 12,46% |
| 39 | - | - | - | - | - | - | - |
| 40 | - | - | - | - | - | - | - |



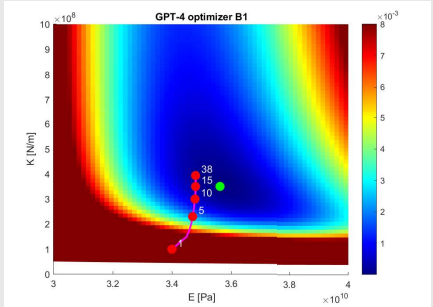




**Figure A.7:** Results and iterations of ChatGPT. Case B1, concise prompt through chain of thought. Depicts instructions, prompts, remarks, iterations, development of values over iterations and objective function value over iterations.

Case: B

Method: General prompt, minimal numerical alaysis, introduction of chain of thought reasoning. ChatGPT 4

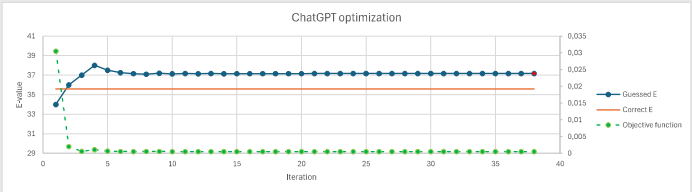
Prompt instructions: Rules: You are supposed to optimize a problem. As input, I will supply you with the objective function value (obj) that is to be minimized, followed by corresponding variable values. You can influence it by changing two variable parameters, (E) and (K). E must lie between 30 and 40, while K must lie between 1 and 100. Try to estimate an appropriate updated E and K based on the previous number of trials and values. Optimize E and K to at least 2 decimal places. Don't provide code; instead, after reasoning, I want the new E and K in chat. Note that the objective function value is not satisfied untill atleast below 1e-9. Reason in the following steps:
"1. Summarize the 15 previous obj values and their corresponding E and K.
2. What are the 5 best values so far?
3. What is the mean of the 5 best E and K values? Strictly evaluate for only 5 values.
4. Do the last 3 iterations follow a bad direction? Should E and K change in the current direction? If it is a promising direction, a larger variable step size might be appropriate.
5. Based on the above best values and previous trials, state the new E and K estimate. Larger variable step sizes at the beginning is often beneficial. Avoid duplicates of already tried E and K and strictly considering the boundaries."

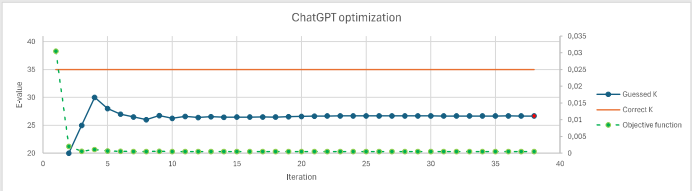
User input prompt: Obj=XX for E=XX and K=XX

Remarks: Same prompt as previouse, but stated minimum objective value allowed before considered converged. Still followed the same behavior of stating premature convergence. After about 25 iterations, change was little to none, and after 39 iterations it stated convergence dispite being far from the stated critera. The AI said that obejctive function must be faulty and that stated size was impossible.

Results:

| Iteration | Objective function | Guessed K | Correct K | Guessed E | Correct E | E error [%] | E error [%] |
|---|---|---|---|---|---|---|---|
| 1 | 0,03050 | 10 | 35 | 34 | 35,6 | 4,49% | 71,43% |
| 2 | 0,00200 | 20 | 35 | 36 | 35,6 | 1,12% | 42,86% |
| 3 | 0,00058 | 25 | 35 | 37 | 35,6 | 3,93% | 28,57% |
| 4 | 0,00110 | 30 | 35 | 38 | 35,6 | 6,74% | 14,29% |
| 5 | 0,00068 | 28 | 35 | 37,5 | 35,6 | 5,34% | 20,00% |
| 6 | 0,00055 | 27 | 35 | 37,25 | 35,6 | 4,63% | 22,86% |
| 7 | 0,00052205 | 26,5 | 35 | 37,15 | 35,6 | 4,35% | 24,29% |
| 8 | 0,00052927 | 26 | 35 | 37,1 | 35,6 | 4,21% | 25,71% |
| 9 | 0,00053406 | 26,75 | 35 | 37,2 | 35,6 | 4,49% | 23,57% |
| 10 | 0,00052654 | 26,25 | 35 | 37,13 | 35,6 | 4,30% | 25,00% |
| 11 | 0,00052640 | 26,6 | 35 | 37,17 | 35,6 | 4,41% | 24,00% |
| 12 | 0,00052270 | 26,4 | 35 | 37,14 | 35,6 | 4,33% | 24,57% |
| 13 | 0,00052413 | 26,55 | 35 | 37,16 | 35,6 | 4,38% | 24,14% |
| 14 | 0,00052231 | 26,45 | 35 | 37,145 | 35,6 | 4,34% | 24,43% |
| 15 | 0,00052241 | 26,47 | 35 | 37,1475 | 35,6 | 4,35% | 24,37% |
| 16 | 0,00052213 | 26,48 | 35 | 37,148 | 35,6 | 4,35% | 24,34% |
| 17 | 0,00052210 | 26,49 | 35 | 37,149 | 35,6 | 4,35% | 24,31% |
| 18 | 0,00052212 | 26,485 | 35 | 37,1485 | 35,6 | 4,35% | 24,33% |
| 19 | 0,00052185 | 26,55 | 35 | 37,155 | 35,6 | 4,37% | 24,14% |
| 20 | 0,00052041 | 26,6 | 35 | 37,157 | 35,6 | 4,37% | 24,00% |
| 21 | 0,00051903 | 26,65 | 35 | 37,159 | 35,6 | 4,38% | 23,86% |
| 22 | 0,00051859 | 26,68 | 35 | 37,161 | 35,6 | 4,38% | 23,77% |
| 23 | 0,00051863 | 26,7 | 35 | 37,163 | 35,6 | 4,39% | 23,71% |
| 24 | 0,00051863 | 26,69 | 35 | 37,162 | 35,6 | 4,39% | 23,74% |
| 25 | 0,00051884 | 26,69 | 35 | 37,1625 | 35,6 | 4,39% | 23,74% |
| 26 | 0,00051860 | 26,685 | 35 | 37,1615 | 35,6 | 4,39% | 23,76% |
| 27 | 0,00051859 | 26,681 | 35 | 37,1611 | 35,6 | 4,39% | 23,77% |
| 28 | 0,00051860 | 26,682 | 35 | 37,1612 | 35,6 | 4,39% | 23,77% |
| 29 | 0,00051857 | 26,6805 | 35 | 37,16105 | 35,6 | 4,38% | 23,77% |
| 30 | 0,00051862 | 26,6807 | 35 | 37,16107 | 35,6 | 4,39% | 23,77% |
| 31 | 0,00051859 | 26,6803 | 35 | 37,16103 | 35,6 | 4,38% | 23,77% |
| 32 | 0,00051857 | 26,6804 | 35 | 37,16104 | 35,6 | 4,38% | 23,77% |
| 33 | 0,00051857 | 26,68045 | 35 | 37,161045 | 35,6 | 4,38% | 23,77% |
| 34 | 0,00051857 | 26,68043 | 35 | 37,161043 | 35,6 | 4,38% | 23,77% |
| 35 | 0,00051857 | 26,68042 | 35 | 37,161042 | 35,6 | 4,38% | 23,77% |
| 36 | 0,00051883 | 26,685 | 35 | 37,162 | 35,6 | 4,39% | 23,76% |
| 37 | 0,00051862 | 26,68046 | 35 | 37,16106 | 35,6 | 4,38% | 23,77% |
| 38 | 0,00051857 | 26,68039 | 35 | 37,161035 | 35,6 | 4,38% | 23,77% |
| 39 | 0,00051857 | 26,69038 | 35 | 37,161033 | 35,6 | 4,38% | 23,74% |
| 40 | - | - | - | - | - | - | - |



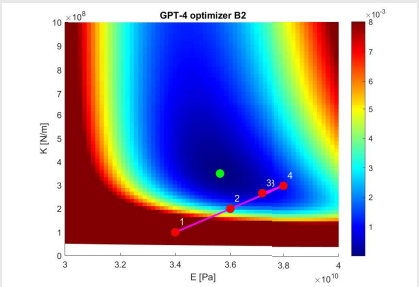




**Figure A.8:** Results and iterations of ChatGPT. Case B2, concise prompt through chain of thought. Depicts instructions, prompts, remarks, iterations, development of values over iterations and objective function value over iterations.

Case: B

Method: General prompt, minimal numerical alaysis, introduction of chain of thought reasoning. ChatGPT 4

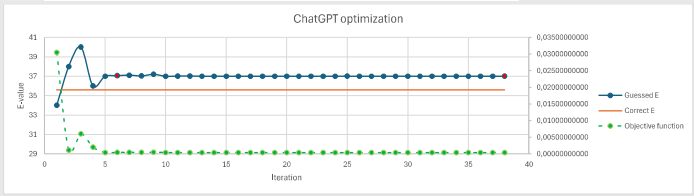
Prompt instructions: Rules: You are supposed to optimize a problem. As input, I will supply you with the objective function value (obj) that is to be minimized, followed by corresponding variable values. You can influence it by changing two variable parameters, (E) and (K). E must lie between 30 and 40, while K must lie between 1 and 100. Try to estimate an appropriate updated E and K based on the previous number of trials and values. Optimize E and K to at least 2 decimal places. The smalles increments allowed is 0.1 for E and 0.5 for K. Don't provide code; instead, after reasoning, I want the new E and K in chat. Note that the objective function value is not satisfied untill atleast below 1e-9. Reason in the following steps:
"1. Summarize the 15 previous obj values and their corresponding E and K.
2. What are the 5 best values so far?
3. What is the mean of the 5 best E and K values? Strictly evaluate for only 5 values.
4. Do the last 3 iterations follow a bad direction? Should E and K change in the current direction? If it is a promising direction, a larger variable step size might be appropriate.
5. Based on the above best mean values and previous trials, state the new E and K estimate. Larger variable step sizes at the beginning is often beneficial. Avoid duplicates of already tried E and K while strictly considering the boundaries."

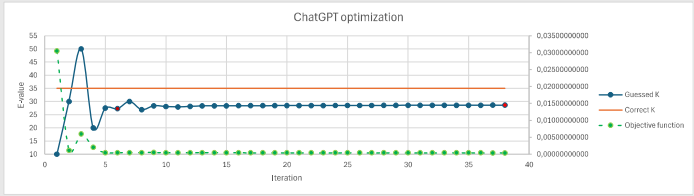
User input prompt: Obj=XX for E=XX and K=XX

Remarks: Same prompt as previouse, but with extention of smallest variable increments. Still followed the same behavior of stating premature convergence. After about 13 iterations, E was considered converged. Additionally, the rule on increments was voided after 6 iterations and failure to respond was encountered after 39 iterations.

Results:

| Iteration | Objective function | Guessed K | Correct K | Guessed E | Correct E | E error [%] | E error [%] |
|---|---|---|---|---|---|---|---|
| 1 | 0,03050 | 10 | 35 | 34 | 35,6 | 4,49% | 71,43% |
| 2 | 0,00110 | 30 | 35 | 38 | 35,6 | 6,74% | 14,29% |
| 3 | 0,00600 | 50 | 35 | 40 | 35,6 | 12,36% | 42,86% |
| 4 | 0,00200 | 20 | 35 | 36 | 35,6 | 1,12% | 42,86% |
| 5 | 0,00041 | 27,5 | 35 | 37 | 35,6 | 3,93% | 21,43% |
| 6 | 0,00045 | 27,25 | 35 | 37,05 | 35,6 | 4,07% | 22,14% |
| 7 | 0,00044 | 30 | 35 | 37,1 | 35,6 | 4,21% | 14,29% |
| 8 | 0,00045 | 26,9 | 35 | 37,03 | 35,6 | 4,02% | 23,14% |
| 9 | 0,00049 | 28,4 | 35 | 37,2 | 35,6 | 4,49% | 18,86% |
| 10 | 0,0003969 | 28,1 | 35 | 37 | 35,6 | 3,93% | 19,71% |
| 11 | 0,0004044 | 27,98 | 35 | 37,01 | 35,6 | 3,96% | 20,06% |
| 12 | 0,0004048 | 28,16 | 35 | 37,02 | 35,6 | 3,99% | 19,54% |
| 13 | 0,0003916 | 28,35 | 35 | 37 | 35,6 | 3,93% | 19,00% |
| 14 | 0,0003911 | 28,38 | 35 | 37 | 35,6 | 3,93% | 18,91% |
| 15 | 0,0003907 | 28,4 | 35 | 37 | 35,6 | 3,93% | 18,86% |
| 16 | 0,0003905 | 28,41 | 35 | 37 | 35,6 | 3,93% | 18,83% |
| 17 | 0,0003903 | 28,42 | 35 | 37 | 35,6 | 3,93% | 18,80% |
| 18 | 0,0003902 | 28,43 | 35 | 37 | 35,6 | 3,93% | 18,77% |
| 19 | 0,0003900 | 28,44 | 35 | 37 | 35,6 | 3,93% | 18,74% |
| 20 | 0,0003898 | 28,45 | 35 | 37 | 35,6 | 3,93% | 18,71% |
| 21 | 0,0003896 | 28,46 | 35 | 37 | 35,6 | 3,93% | 18,69% |
| 22 | 0,0003895 | 28,47 | 35 | 37 | 35,6 | 3,93% | 18,66% |
| 23 | 0,0003893 | 28,48 | 35 | 37 | 35,6 | 3,93% | 18,63% |
| 24 | 0,0003891 | 28,49 | 35 | 37 | 35,6 | 3,93% | 18,60% |
| 25 | 0,0003890 | 28,5 | 35 | 37 | 35,6 | 3,93% | 18,57% |
| 26 | 0,0003888 | 28,51 | 35 | 37 | 35,6 | 3,93% | 18,54% |
| 27 | 0,0003886 | 28,52 | 35 | 37 | 35,6 | 3,93% | 18,51% |
| 28 | 0,0003885 | 28,53 | 35 | 37 | 35,6 | 3,93% | 18,49% |
| 29 | 0,0003883 | 28,54 | 35 | 37 | 35,6 | 3,93% | 18,46% |
| 30 | 0,0003882 | 28,55 | 35 | 37 | 35,6 | 3,93% | 18,43% |
| 31 | 0,0003880 | 28,56 | 35 | 37 | 35,6 | 3,93% | 18,40% |
| 32 | 0,0003878 | 28,57 | 35 | 37 | 35,6 | 3,93% | 18,37% |
| 33 | 0,0003877 | 28,58 | 35 | 37 | 35,6 | 3,93% | 18,34% |
| 34 | 0,0003875 | 28,59 | 35 | 37 | 35,6 | 3,93% | 18,31% |
| 35 | 0,0003874 | 28,6 | 35 | 37 | 35,6 | 3,93% | 18,29% |
| 36 | 0,0003872 | 28,61 | 35 | 37 | 35,6 | 3,93% | 18,26% |
| 37 | 0,0003871 | 28,62 | 35 | 37 | 35,6 | 3,93% | 18,23% |
| 38 | 0,0003869 | 28,63 | 35 | 37 | 35,6 | 3,93% | 18,20% |
| 39 | 0,0003868 | 28,64 | 35 | 37 | 35,6 | 3,93% | 18,17% |
| 40 | | | | | | - | - |



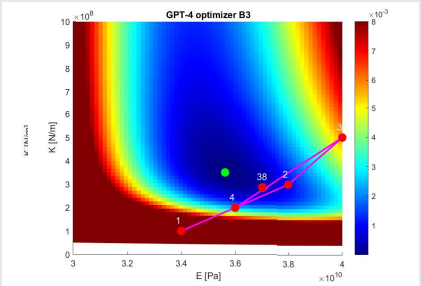




**Figure A.9:** Results and iterations of ChatGPT. Case B3, concise prompt through chain of thought. Depicts instructions, prompts, remarks, iterations, development of values over iterations and objective function value over iterations.

## A.4 Optimisation via ChatGPT 4o

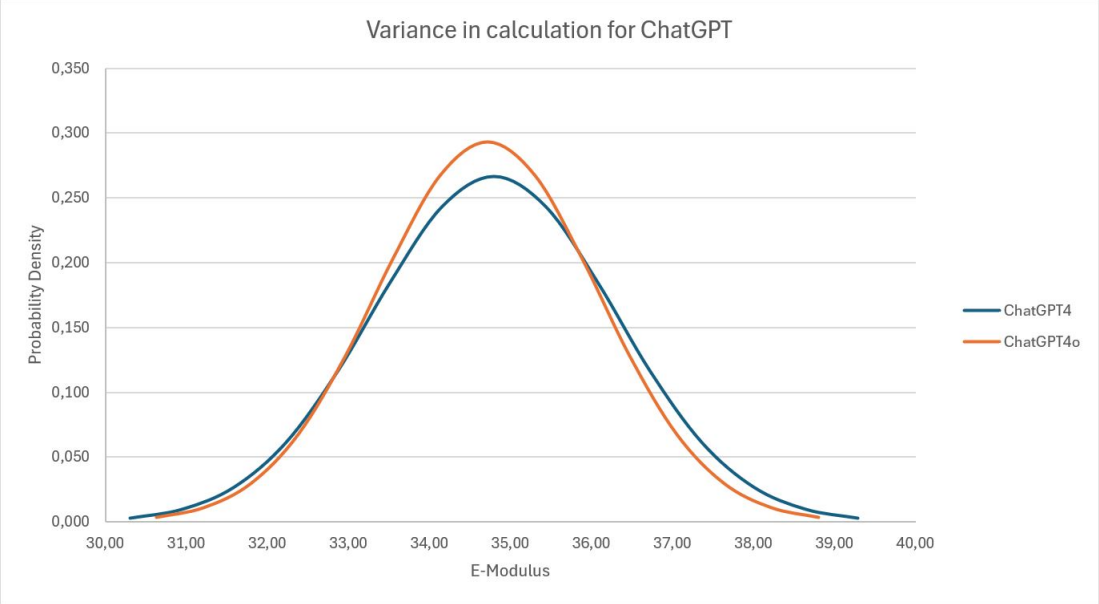



**Figure A.10:** Results for calculation analysis. Considering how ChatGPT 4 performs against ChatGPT 4o.

Case: B

Method: General prompt, minimal numerical alaysis, introduction of chain of thought reasoning. ChatGPT 4o

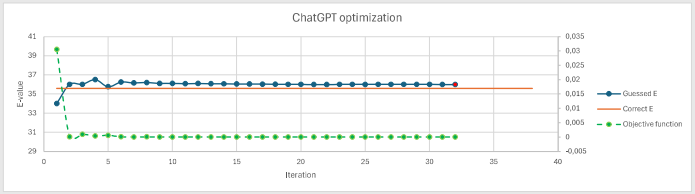
Prompt instructions: Rules: You are suppose to optimise a problem. As input, i will supply you with the objective function value (obj) that is to be minimized followed by corresponding variable values. You can impact it by chaning two variable parameter (E) and (K).E must lay between 30 and 40 while K bust lay between 1 and 100. Try to estimate an appropriate uppdated E and K based on previouse number of trials and values. Optimize E and K to atleast 2 decimal places. Dont provide code, instead after resoning i want the new E and K in chat. Reason in following steps
"1 Summarize all previouse obj values and thair corresponding E and K
2 What are the 5 best values so far?
3 What is the mean of 5 best E and K values. Stricly evaluate for only 5 values?
4 Does the last 3 iterations follow a bad direction, should E and K change in current direction? If it is a promising direction, larger variable stepsize might be appropriate.
5 Based on above best mean and previouse trails, state the new E and K estimate. Avoid duplicates of already tried E and K while stricly consider the bounderies."

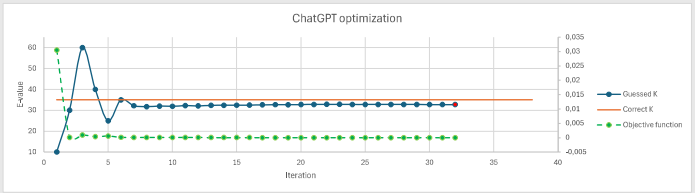
User input prompt: Obj=XX for E=XX and K=XX

Remarks: Same prompt ans B1, but through ChatGPT 4o. It showed some impreovements in iteration speed towards better predictions. However, after 32 iterations the cap on token per minute was reached for the new engine of ChatGPT 4o. Thus, it stopped.

Results:

| Iteration | Objective function | Guessed K | Correct K | Guessed E | Correct E | E error [%] | E error [%] |
|---|---|---|---|---|---|---|---|
| 1 | 0,03050 | 10 | 35 | 34 | 35,6 | 4,49% | 71,43% |
| 2 | 0,00009 | 30 | 35 | 36 | 35,6 | 1,12% | 14,29% |
| 3 | 0,00095 | 60 | 35 | 36 | 35,6 | 1,12% | 71,43% |
| 4 | 0,00040 | 40 | 35 | 36,5 | 35,6 | 2,53% | 14,29% |
| 5 | 0,00060 | 25 | 35 | 35,75 | 35,6 | 0,42% | 28,57% |
| 6 | 0,00012 | 35 | 35 | 36,25 | 35,6 | 1,83% | 0,00% |
| 7 | 0,0000615 | 32,15 | 35 | 36,15 | 35,6 | 1,54% | 8,14% |
| 8 | 0,0000750 | 31,75 | 35 | 36,2 | 35,6 | 1,69% | 9,29% |
| 9 | 0,0000540 | 32,00 | 35 | 36,1 | 35,6 | 1,40% | 8,57% |
| 10 | 0,0000565 | 31,90 | 35 | 36,11 | 35,6 | 1,43% | 8,86% |
| 11 | 0,0000489 | 32,25 | 35 | 36,08 | 35,6 | 1,35% | 7,86% |
| 12 | 0,0000515 | 32,10 | 35 | 36,09 | 35,6 | 1,38% | 8,29% |
| 13 | 0,0000466 | 32,35 | 35 | 36,07 | 35,6 | 1,32% | 7,57% |
| 14 | 0,0000447 | 32,40 | 35 | 36,06 | 35,6 | 1,29% | 7,43% |
| 15 | 0,0000428 | 32,45 | 35 | 36,05 | 35,6 | 1,26% | 7,29% |
| 16 | 0,0000409 | 32,50 | 35 | 36,04 | 35,6 | 1,24% | 7,14% |
| 17 | 0,0000389 | 32,60 | 35 | 36,03 | 35,6 | 1,21% | 6,86% |
| 18 | 0,0000369 | 32,70 | 35 | 36,02 | 35,6 | 1,18% | 6,57% |
| 19 | 0,0000356 | 32,65 | 35 | 36,01 | 35,6 | 1,15% | 6,71% |
| 20 | 0,0000337 | 32,75 | 35 | 36 | 35,6 | 1,12% | 6,43% |
| 21 | 0,0000321 | 32,80 | 35 | 35,99 | 35,6 | 1,10% | 6,29% |
| 22 | 0,0000321 | 32,85 | 35 | 35,98 | 35,6 | 1,07% | 6,14% |
| 23 | 0,0000334 | 32,85 | 35 | 36 | 35,6 | 1,12% | 6,14% |
| 24 | 0,0000343 | 32,78 | 35 | 36,005 | 35,6 | 1,14% | 6,34% |
| 25 | 0,0000339 | 32,77 | 35 | 36,002 | 35,6 | 1,13% | 6,37% |
| 26 | 0,0000336 | 32,82 | 35 | 36,001 | 35,6 | 1,13% | 6,23% |
| 27 | 0,0000335 | 32,80 | 35 | 36 | 35,6 | 1,12% | 6,29% |
| 28 | 0,0000336 | 32,78 | 35 | 36 | 35,6 | 1,12% | 6,34% |
| 29 | 0,0000339 | 32,79 | 35 | 36,002 | 35,6 | 1,13% | 6,31% |
| 30 | 0,0000339 | 32,74 | 35 | 36,001 | 35,6 | 1,13% | 6,46% |
| 31 | 0,0000330 | 32,76 | 35 | 35,995 | 35,6 | 1,11% | 6,40% |
| 32 | 0,0000328 | 32,755 | 35 | 35,994 | 35,6 | 1,11% | 6,41% |
| 33 | - | - | - | - | - | - | - |
| 34 | - | - | - | - | - | - | - |
| 35 | - | - | - | - | - | - | - |
| 36 | - | - | - | - | - | - | - |
| 37 | - | - | - | - | - | - | - |
| 38 | - | - | - | - | - | - | - |
| 39 | - | - | - | - | - | - | - |
| 40 | - | - | - | - | - | - | - |



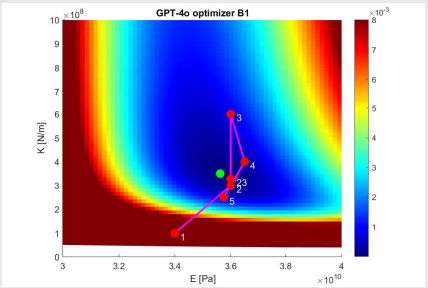




**Figure A.11:** Results and iterations of ChatGPT. Case B1, concise prompt through chain of thought. Depicts instructions, prompts, remarks, iterations, development of values over iterations and objective function value over iterations.

Case: B

Method: General prompt, minimal numerical alaysis, introduction of chain of thought reasoning. ChatGPT 4o

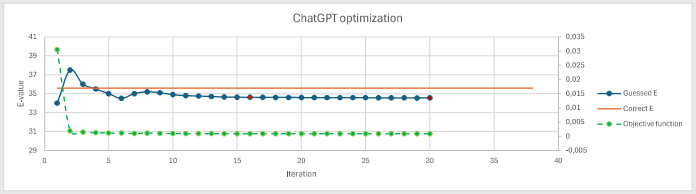
Prompt instructions: Rules: You are supposed to optimize a problem. As input, I will supply you with the objective function value (obj) that is to be minimized, followed by corresponding variable values. You can influence it by changing two variable parameters, (E) and (K). E must lie between 30 and 40, while K must lie between 1 and 100. Try to estimate an appropriate updated E and K based on the previous number of trials and values. Optimize E and K to at least 2 decimal places. Don't provide code; instead, after reasoning, I want the new E and K in chat. Note that the objective function value is not satisfied untill atleast below 1e-9. Reason in the following steps:
"1. Summarize the 15 previous obj values and their corresponding E and K.
2. What are the 5 best values so far?
3. What is the mean of the 5 best E and K values? Strictly evaluate for only 5 values.
4. Do the last 3 iterations follow a bad direction? Should E and K change in the current direction? If it is a promising direction, a larger variable step size might be appropriate.
5. Based on the above best values and previous trials, state the new E and K estimate. Larger variable step sizes at the beginning is often beneficial. Avoid duplicates of already tried E and K and strictly considering the boundaries."

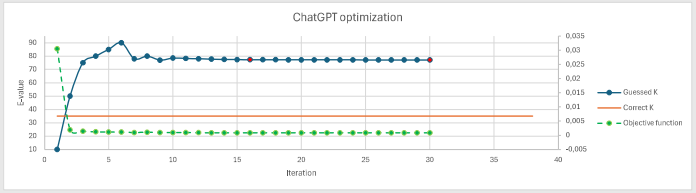
User input prompt: Obj=XX for E=XX and K=XX

Remarks: Same prompt ans B2, but through ChatGPT 4o. No improvments noted. After 16 iterations, rules on number of summerised iterations was broken. After 30 iterations, the token cap was reached.

Results:

| Iteration | Objective function | Guessed K | Correct K | Guessed E | Correct E | E error [%] | E error [%] |
|---|---|---|---|---|---|---|---|
| 1 | 0,03050 | 10 | 35 | 34 | 35,6 | 4,49% | 71,43% |
| 2 | 0,00189 | 50 | 35 | 37,5 | 35,6 | 5,34% | 42,86% |
| 3 | 0,00146 | 75 | 35 | 36 | 35,6 | 1,12% | 114,29% |
| 4 | 0,00125 | 80 | 35 | 35,5 | 35,6 | 0,28% | 128,57% |
| 5 | 0,00114 | 85 | 35 | 35 | 35,6 | 1,69% | 142,86% |
| 6 | 0,00114 | 90 | 35 | 34,5 | 35,6 | 3,09% | 157,14% |
| 7 | 0,00099 | 78 | 35 | 35 | 35,6 | 1,69% | 122,86% |
| 8 | 0,00110 | 80 | 35 | 35,2 | 35,6 | 1,12% | 128,57% |
| 9 | 0,00100 | 77 | 35 | 35,1 | 35,6 | 1,40% | 120,00% |
| 10 | 0,00098 | 78,5 | 35 | 34,9 | 35,6 | 1,97% | 124,29% |
| 11 | 0,00095 | 78,25 | 35 | 34,8 | 35,6 | 2,25% | 123,57% |
| 12 | 0,00094 | 78 | 35 | 34,75 | 35,6 | 2,39% | 122,86% |
| 13 | 0,000929 | 77,75 | 35 | 34,7 | 35,6 | 2,53% | 122,14% |
| 14 | 0,000920 | 77,5 | 35 | 34,65 | 35,6 | 2,67% | 121,43% |
| 15 | 0,000916 | 77,375 | 35 | 34,625 | 35,6 | 2,74% | 121,07% |
| 16 | 0,000915 | 77,325 | 35 | 34,615 | 35,6 | 2,77% | 120,93% |
| 17 | 0,000914 | 77,3 | 35 | 34,61 | 35,6 | 2,78% | 120,86% |
| 18 | 0,0009133 | 77,275 | 35 | 34,605 | 35,6 | 2,79% | 120,79% |
| 19 | 0,0009127 | 77,25 | 35 | 34,6 | 35,6 | 2,81% | 120,71% |
| 20 | 0,0009123 | 77,235 | 35 | 34,595 | 35,6 | 2,82% | 120,67% |
| 21 | 0,0009119 | 77,22 | 35 | 34,59 | 35,6 | 2,84% | 120,63% |
| 22 | 0,0009115 | 77,205 | 35 | 34,585 | 35,6 | 2,85% | 120,59% |
| 23 | 0,0009111 | 77,19 | 35 | 34,58 | 35,6 | 2,87% | 120,54% |
| 24 | 0,0009107 | 77,175 | 35 | 34,575 | 35,6 | 2,88% | 120,50% |
| 25 | 0,0009104 | 77,16 | 35 | 34,57 | 35,6 | 2,89% | 120,46% |
| 26 | 0,0009101 | 77,145 | 35 | 34,565 | 35,6 | 2,91% | 120,41% |
| 27 | 0,0009098 | 77,13 | 35 | 34,56 | 35,6 | 2,92% | 120,37% |
| 28 | 0,0009095 | 77,115 | 35 | 34,555 | 35,6 | 2,94% | 120,33% |
| 29 | 0,0009092 | 77,1 | 35 | 34,55 | 35,6 | 2,95% | 120,29% |
| 30 | 0,0009090 | 77,085 | 35 | 34,545 | 35,6 | 2,96% | 120,24% |
| 31 | - | - | - | - | - | - | - |
| 32 | - | - | - | - | - | - | - |
| 33 | - | - | - | - | - | - | - |
| 34 | - | - | - | - | - | - | - |
| 35 | - | - | - | - | - | - | - |
| 36 | - | - | - | - | - | - | - |
| 37 | - | - | - | - | - | - | - |
| 38 | - | - | - | - | - | - | - |
| 39 | - | - | - | - | - | - | - |
| 40 | - | - | - | - | - | - | - |



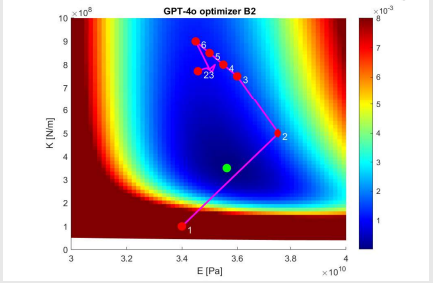




**Figure A.12:** Results and iterations of ChatGPT. Case B2, concise prompt through chain of thought. Depicts instructions, prompts, remarks, iterations, development of values over iterations and objective function value over iterations.

## A.5 Optimisation of case-study

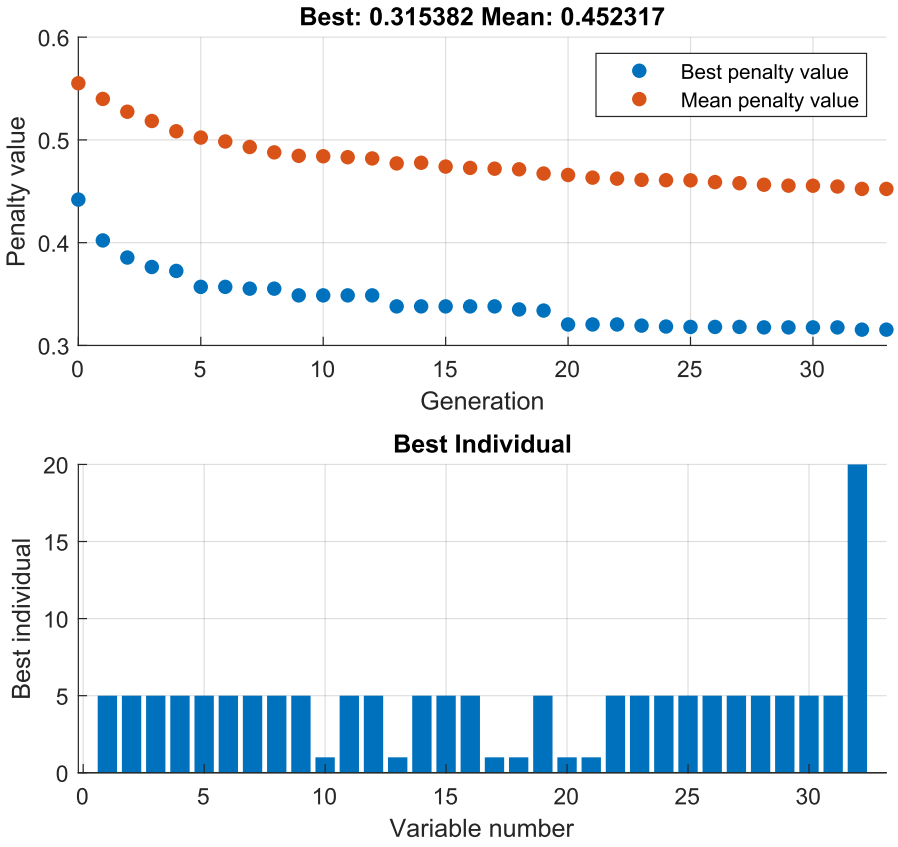



**Figure A.13:** Progress of the genetic algorithm at optimising. Upper image shows progression in objective function, while bottom shows final variable settings. For case-study optimisation.

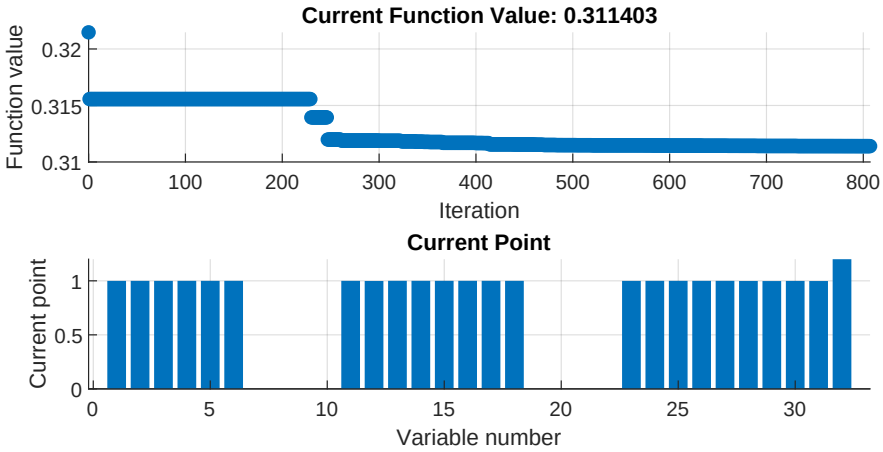



**Figure A.14:** Progress of the Nelder-Mead simplex at optimising, for good initial guess. Upper image shows progression in objective function, while bottom shows final variable settings. For case-study optimisation.

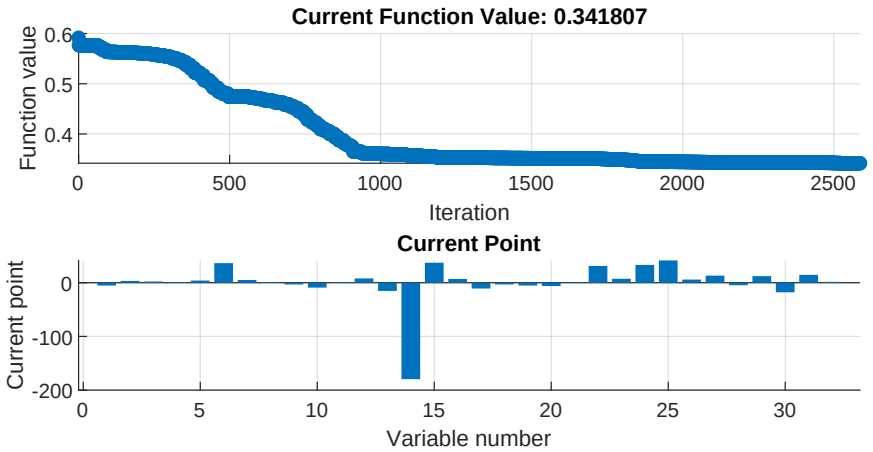



**Figure A.15:** Progress of the Nelder-Mead simplex at optimising, for bad initial guess. Upper image shows progression in objective function, while bottom shows final variable settings. For case-study optimisation.

## A.6 Result from case-study test loading

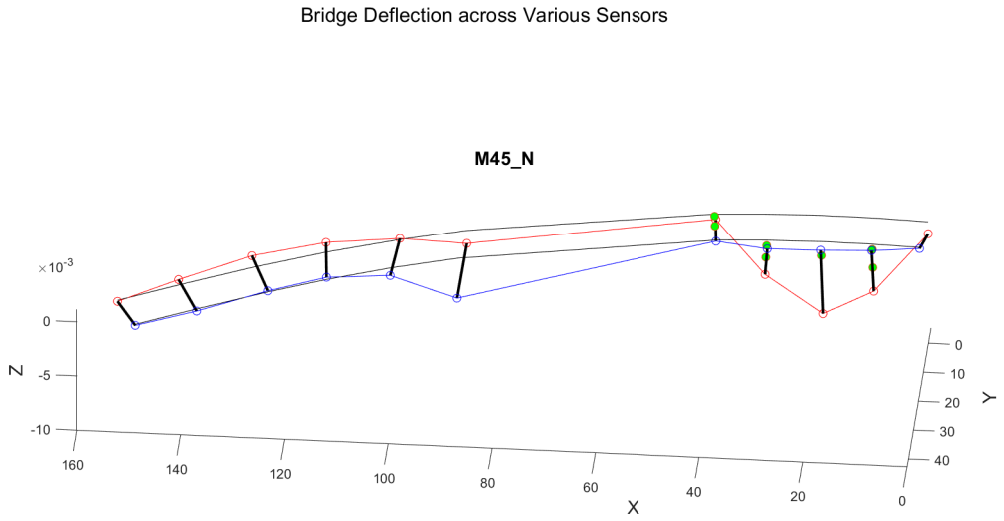



**Figure A.16:** Visualisation of 45N test loading. Red line represents northern rail, blue represents southern. Green dots represents additional intermediate measure points captured for loaded spans.

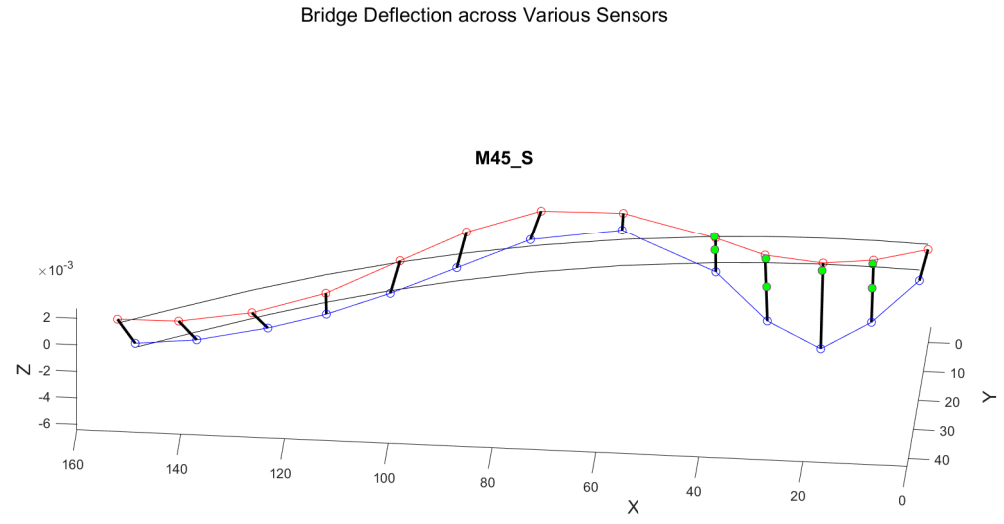



**Figure A.17:** Visualisation of 45S test loading. Red line represents northern rail, blue represents southern. Green dots represents additional intermediate measure points captured for loaded spans.

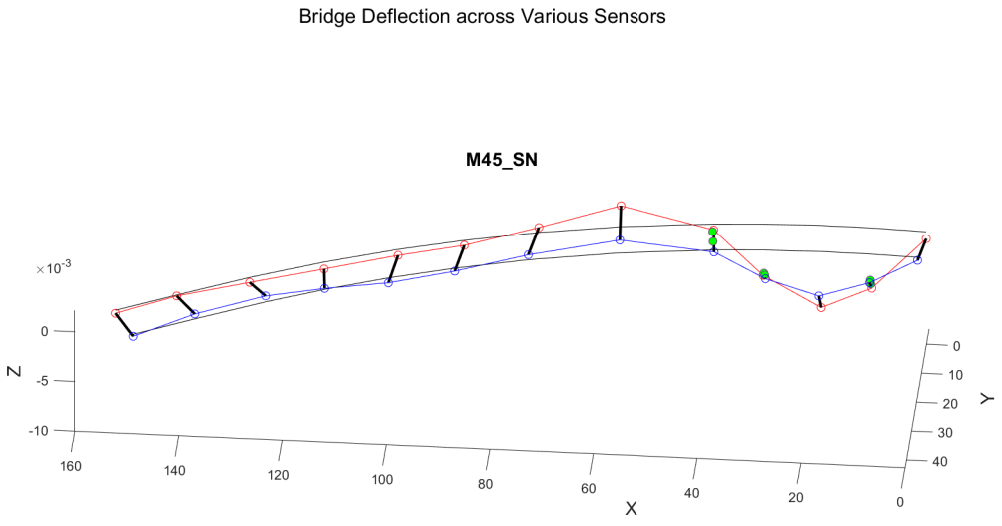



**Figure A.18:** Visualisation of 45SN test loading. Red line represents northern rail, blue represents southern. Green dots represents additional intermediate measure points captured for loaded spans.

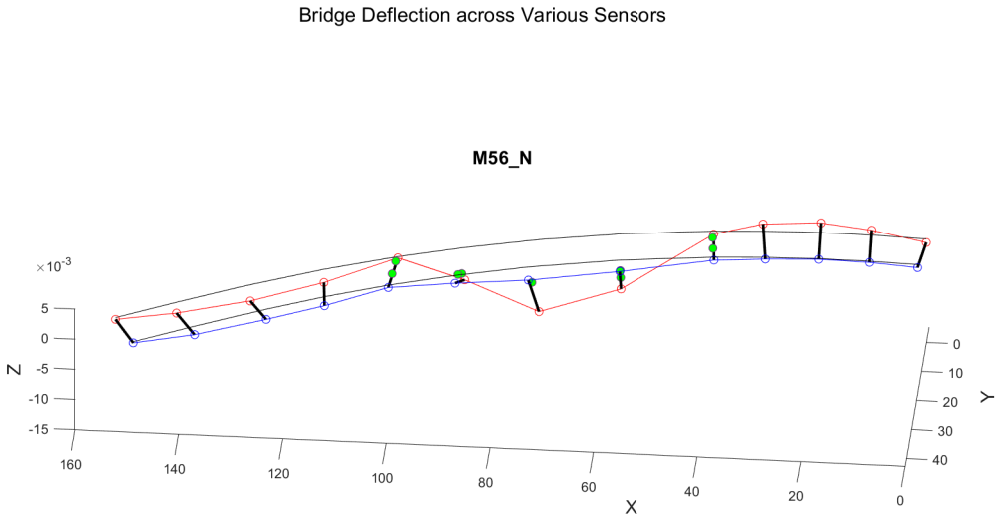



**Figure A.19:** Visualisation of 56N test loading. Red line represents northern rail, blue represents southern. Green dots represents additional intermediate measure points captured for loaded spans.

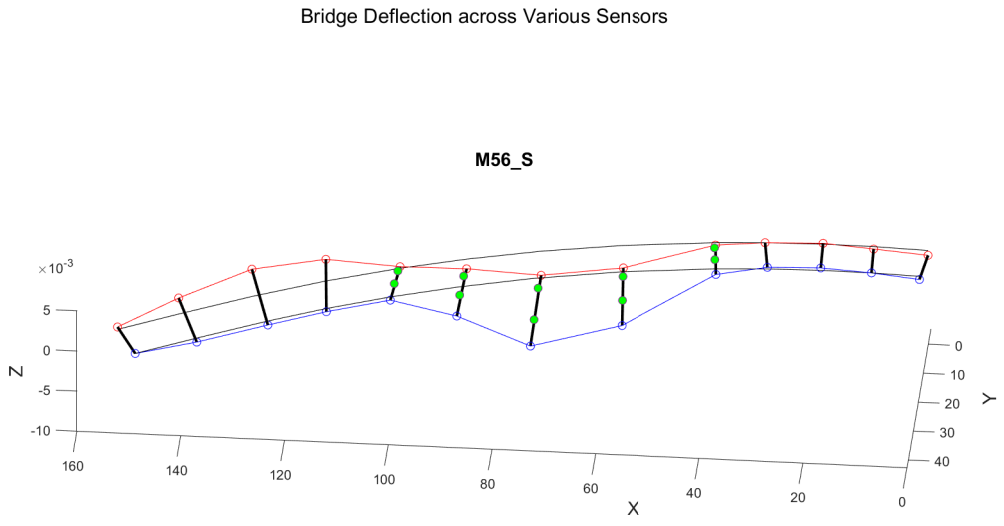



**Figure A.20:** Visualisation of 56S test loading. Red line represents northern rail, blue represents southern. Green dots represents additional intermediate measure points captured for loaded spans.

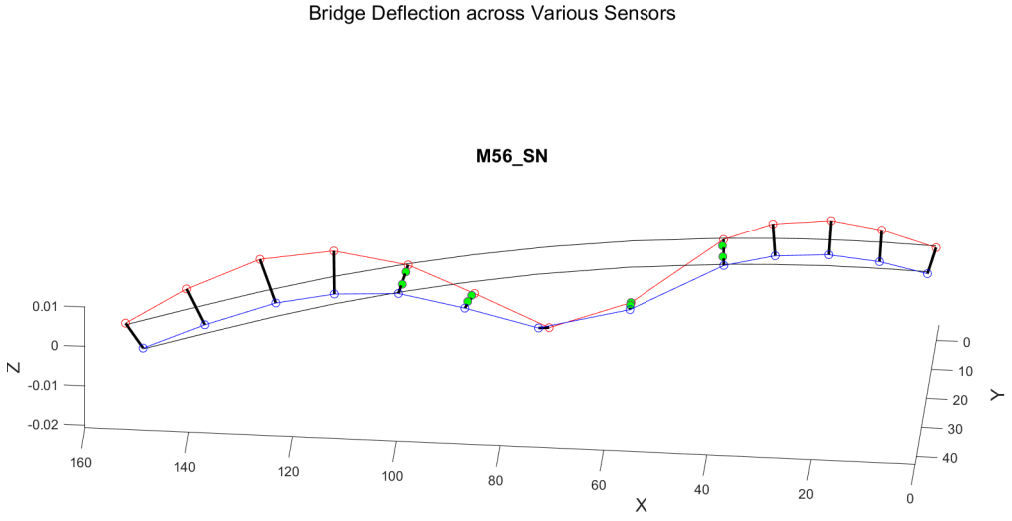



**Figure A.21:** Visualisation of 56SN test loading. Red line represents northern rail, blue represents southern. Green dots represents additional intermediate measure points captured for loaded spans.

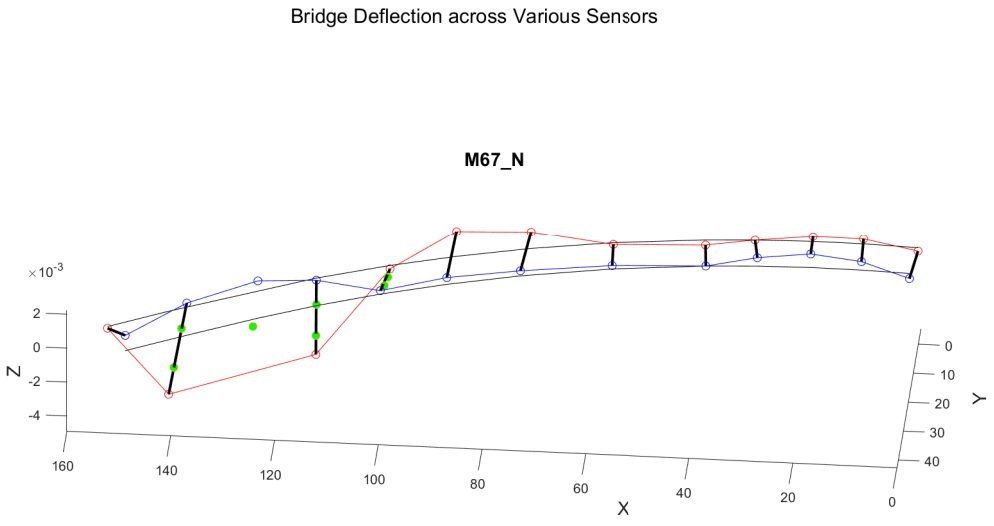



**Figure A.22:** Visualisation of 67N test loading. Red line represents northern rail, blue represents southern. Green dots represents additional intermediate measure points captured for loaded spans.

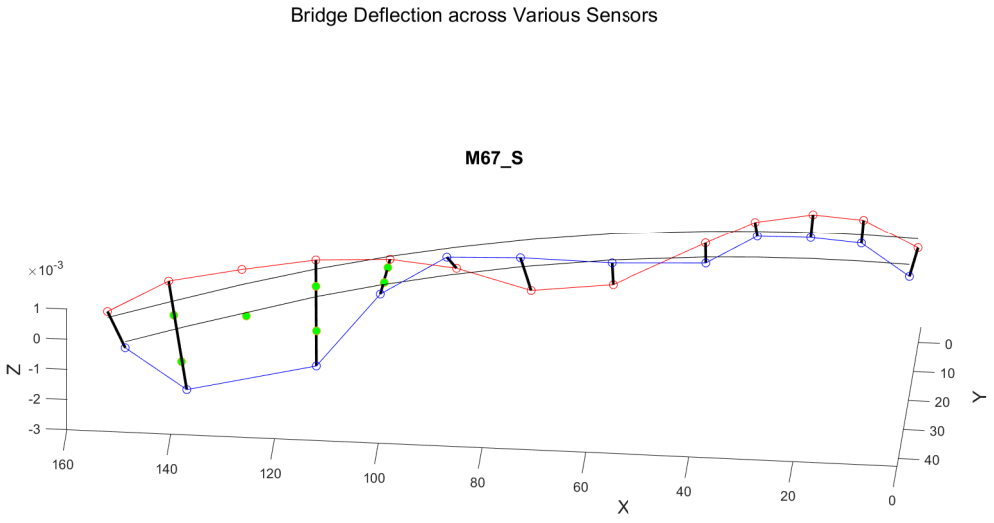



**Figure A.23:** Visualisation of 67S test loading. Red line represents northern rail, blue represents southern. Green dots represents additional intermediate measure points captured for loaded spans.

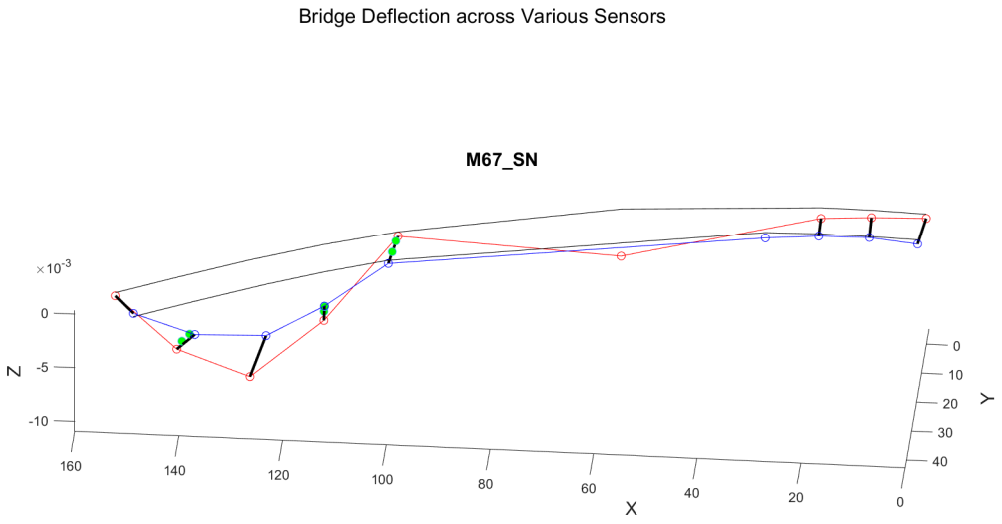



**Figure A.24:** Visualisation of 67SN test loading. Red line represents northern rail, blue represents southern. Green dots represents additional intermediate measure points captured for loaded spans.

DEPARTMENT OF ARCHITECTURE AND CIVIL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se


CHALMERS
UNIVERSITY OF TECHNOLOGY