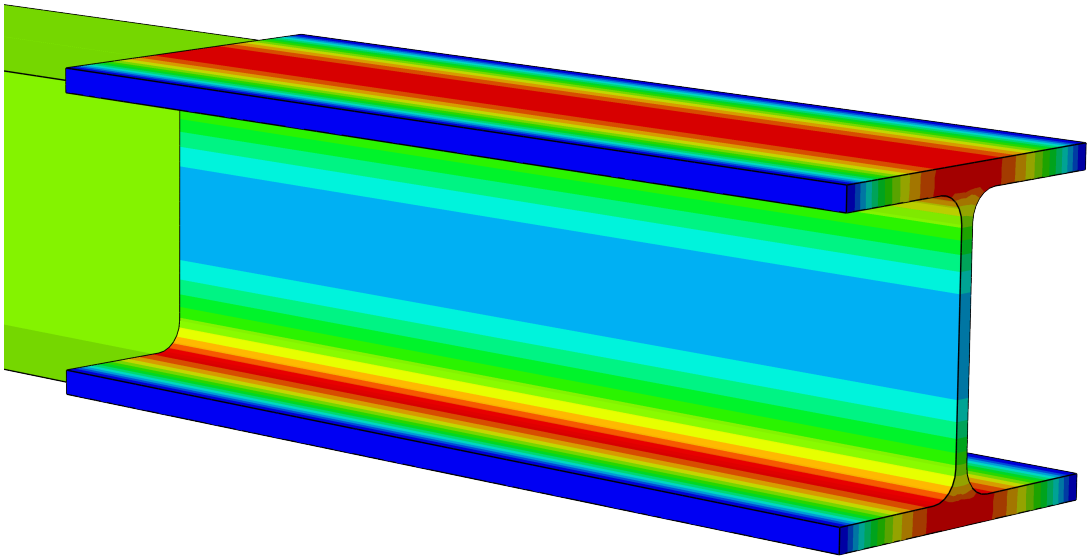
CHALMERS
UNIVERSITY OF TECHNOLOGY



Innovative method for deriving residual stress distribution in hot-rolled steel beams

An inverse analysis based on finite element modelling and strain data measured with optical fibres

Master's thesis in Structural Engineering

Filip Karlström
Borik Ronnby

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

Master's thesis 2024

Innovative method for deriving residual stress distribution in hot-rolled steel beams

An inverse analysis based on finite element modelling and strain data measured with optical fibres

Filip Karlström
Borik Ronnby

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Architecture and Civil Engineering
Division of Structural Engineering
Chalmers University of Technology
Gothenburg, Sweden 2024

Innovative method for deriving residual stress distribution in hot-rolled steel beams
An inverse analysis based on finite element modelling and strain data measured with optical fibres
Filip Karlström
Borik Ronnby

Examiner:
Mohammad al-Ermrani, Division of Structural Engineering

Supervisors:
Farshid Zamiri, AFRY Infastructure
Carlos Gil Berrocal, Division of Structural Engineering

Master's Thesis 2024
Department of Architecture and Civil Engineering
Division of Structual Enginering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

Innovative method for deriving residual stress distribution in hot-rolled steel beams
Filip Karlström & Borik Ronnby
Department of Architecture and Civil Engineering
Chalmers University of Technology

Abstract

In this masters project, an innovative method of determining residual stresses in steel beams has been developed. The method is based on an inverse analysis method which utilises finite element modelling and strain data collected with optical fibres to estimate the variables describing the residual stress distribution. The problem solved is of a non-linear type since the material behaviour during loading is non-linear, resulting in that a direct-search method was used in the inverse analysis. The variables describing the residual stress distribution are therefore derived using a Nelder-Mead optimisation algorithm where the difference between the finite element model's strains and strains obtained from testing is minimised.

The method was tested by performing four point bending tests on six hot-rolled profiles split between three HEA200 and three HEB200 beams. It can be concluded that the method is highly dependent on the quality and amount of test data that can be used. Especially the ability to capture data after the point of yielding is crucial in order to accurately mirror the non-linear behaviour of the material. However, if the data is good enough the method works as expected and will find a suitable residual stress distribution of the specimen's tested.

The accuracy of the method in its current state is however questionable as simplifications made in the FE-model might have significantly affected the result. Several improvements of the method is thereby proposed to make it more accurate and reliable.

Keywords: Residual stress distribution, Inverse analysis, Optical fibres, Hot-rolled steel beam, Nelder-Mead

Innovativ metod för att bestämma egenspänningsfördelningen i varmvalsade balkar
Filip Karlström & Borik Ronnby
Institutionen för Arkitektur och Samhällsbyggnadsteknik
Chalmers Tekniska Högskola

Sammanfattning

I detta mastersprojekt har en innovativ metod för att bestämma egenspänningar i stålbalkar framtagits. Metoden är baserad på en inversanalys som använder sig av finit elementmodellering och töjningsdata från optiska fibrer för att estimera variablerna som beskriver egenspänningarna. Problemet som löses är icke-linjärt eftersom materialets beteende under lastning är icke-linjärt, vilket resulterade i att en direkt sökmetod användes i inversanalysen. Variablerna för egenspänningsfördelningen är framtagna genom en Nelder-Mead optimeringsmetod där skillnaden mellan töjningsdatan från den finita element modellen och de fysiska testerna minimeras.

Metoden var testad genom att utföra fyrpunktsböjningstester på sex varmvalsade profiler, varav tre var av typ HEA200 och tre var av typ HEB200. Det kunde konstateras att metoden är ytterst beroende av kvalitén och mängden testdata som kan användas. Framför allt möjligheten att mäta data efter att stålet flyter är viktigt för att noggrant kunna modellera det icke-linjära beteendet hos materialet. Om datan är bra nog så fungerar dock metoden som väntat och den finner en passande egenspänningsfördelning för den testade provkroppen.

Träffsäkerheten hos metoden i sitt nuvarande skede är dock ifrågasättbar eftersom förenklingar av balkmodellen kan ha avsevärt påverkat resultatet. Flertalet förbättringar av metoden är därmed föreslagna för att göra den mer träffsäker och trovärdig.

Nyckelord: Egenspänningsfördelning, Inversanalys, Optiska fibrer, Varmvalsad stålbalk, Nelder-Mead

Preface

This Master's thesis was a collaboration between Chalmers University of Technology, Department of Architecture and Civil Engineering and AFRY AB's, department of Inspection and investigation. The work on this thesis was carried out at AFRY AB and at Chalmers University of Technology's Structural lab from January 2024 to June 2024. It is part of a pilot study examining the ability to use optical fibres to measure strain in steel members.

The authors want to express their deepest gratitude to the projects supervisors, Ph.D Farshid Zamiri at AFRY AB and Senior Lecturer Carlos Gil Berrocal at the Department of Architecture and Civil engineering, as well as to examiner Prof. Mohammad al-Emrani at the Department of Architecture and Civil Engineering. Thanks for your constant support and guidance throughout the project.

Farshid, for your constant help and deep understanding of the topic at hand. Thanks, it has been a pleasure.

Carlos, for selflessly spending your time to advance this project. Thanks for you time and expertise using optical fibres.

Mohammad, for your constant enthusiasm and dedication to the project as well as our development and understanding. Thanks, it has been a privilege to have you as our examiner.

A special thanks to Sebastian Almfeldt and Anders Karlsson at Chalmers Structural lab, without your engagement and expertise this project would not have been possible.

Filip Karlström & Borik Ronnby, Gothenburg, 2024/06/27

Table of Contents

List of Figures

List of Tables

Nomenclature

Greek

σ_c	Compressive stress
σ_f	Flange stress
σ_t	Tensile stress
σ_w	Web stress

Latin

B/b	Width of cross section
b_f	Flange width
d	Depth
d_{Abaqus}	Deflection used in Abaqus
$d_{PointLoad}$	Deflection at point load
$d_{Support}$	Deflection at support
f_y	Material yield limit
h	Height of cross section
t_f	Flange thickness
t_w	Web thickness

1
Introduction

This section describes the background of the project and what its aim and limitations are.

1.1 Background

Residual stresses have a large influence on the fatigue life of manufactured steel products (James et al., 2007). Since the mean stress is inversely related to the fatigue capacity, an increase in the mean stress in the form of residual stress will lower the fatigue life of the product. In addition, residual stresses affect when parts of the cross-section yield, which in compression make the products more sensitive to buckling (Young, 1975). Because of these effects, it is of interest to be able to accurately determine the residual stresses of steel members.

There are several methods to determine the residual stresses, and they can be categorised into three types: destructive, semi-destructive, and non-destructive (Rossini et al., 2012). Destructive and semi-destructive methods are usually easy to perform and can be used for any type of material, but they leave the specimen damaged after the procedure. In addition, errors are introduced in the process of drilling or cutting the specimen for the measurements, and the data from the tests could be difficult to interpret (Rossini et al., 2012). Methods that are non-destructive, on the other hand, leave the specimen unchanged but usually require lab conditions and tweaking for the specific specimen to give proper results (Rossini et al., 2012).

In search of an alternative way of measuring residual stresses, a method based on inverse analysis of strains obtained using optical fibre sensors has been proposed. The fibres allow for high amounts of strain measurements along the length of the cable compared to traditional strain gauges (Sabri et al., 2013), which could improve the accuracy of the residual stress profile. In this thesis, the method described above was tested and then evaluated.

1.2 Method overview

The method compares strain data from physical testing with that of a numerical model. By assuming a residual stress distribution and evaluating each assumption the method derives the initial residual stress distribution of the beam. This is a type of inverse analysis were the result is known, in this case the strain data, the

initial state of the problem is then to be solved, in this case the residual stress distribution. To solve the problem, an optimisation algorithm is used which makes new assumptions based on the result of the previous assumption, until it reaches a set convergence criteria.
Measurements are made using optical fibres glued to a specimen subjected to loading, with at least one section going beyond yielding. The fibres allow for high amounts of strain measurements along the length of the cable compared to traditional strain gauges (Sabri et al., 2013), which could improve the accuracy of the residual stress profile.

1.3 Aim

The aim of this master project was to develop and evaluate a method of deriving the residual stress distribution of steel beams based on strain data measured with optical fibres.

1.4 Limitations

For the physical tests conducted, no cyclic loading was performed. This decreased the number of variables that could affect the result, leading to a reduced work load in calibrating the model based on the test results.

Only hot-rolled H-profiles were tested, and their sizes were limited by the fact that the maximum available force in four-point bending was 200 kN and that plasticity had to be reached in the cross-section. Because of the limited available force, the steel strength was also limited, not including high strength steels. Only a steel grade of S355J2+M was studied.

The residual stress distribution that is analysed in this thesis is only the residual stress distribution of the macroscopic scale in the axial direction. The microstructure stress distribution and the transverse stress are disregarded.

1.5 Societal, ethical, and ecological aspects

This thesis does not include discussions regarding societal or ethical aspects as it focuses on the behaviour of steel, not its implementation in the real world regarding sourcing, production, etc. There is a connection to the ecological aspect in the sense that it might be possible to use the results of the investigation to optimise cross-sections of beams, which would result in less material being used. This would in turn reduce carbon emissions from steel constructions built using this optimisation.

2

Literature review

There are three types of residual stresses that are categorised by either the scale on which they are in self-equilibrium or by the scale on which they are measured (Withers & Bhadeshia, 2001b). The three types are: macrostresses (type I), which are on the scale of structures, intergranual stresses (type II), which are on the scale of a small number of grains, and atomic scale stresses (type III), which are on the scale smaller than a singular grain (Withers & Bhadeshia, 2001a).

For this thesis, only axial residual stress of type I is of interest since the stress distribution of interest is the one over the whole cross-section. In addition, only hot-rolled profiles are investigated in this thesis. Therefore, the following text mainly focuses on information relevant to hot-rolled profiles and type I axial residual stresses.

2.1 Residual stress in hot-rolled profiles

The residual stresses in hot-rolled beams are introduced during cooling of the beam after the rolling process. The stresses appear due to different points of the cross-section cooling at different rates, the parts that cool first will end up in compression, and the ones that cool later will end up in tension (Young, 1975).

The parts that cool first depend on the geometry of the section and the cooling conditions, such as how close together the beams lay on the cooling bed and in which orientation the individual beams are placed (Alpsten, 1968). If the beam is allowed to radiate heat freely, the orientation of the beam has far less impact, and the tips of the flanges as well as the web will end up in compression while the area around the web-flange junctions will end up in tension (Alpsten, 1968). However, if the beams on the cooling bed are placed close together, the orientation of the beam matters greatly as it determines which surfaces face each other and their ability to radiate heat (Alpsten, 1968).

According to Alpsten's study (1968), the whole web can end up being in tension if the beams are placed close enough with the webs in the vertical direction. If the webs are instead placed horizontally, the study shows that the stress distribution differs much less from the case with free heat transfer compared to the case with vertical webs (Alpsten, 1968). The stress distribution for the case with horizontal webs seems to follow the same pattern as the case with free heat radiation, but with an increased magnitude of compression and tension in the web and flange respec-

tively.

After the cooling process, the beams usually have to be straightened since they curve when cooling (Alpsten, 1975). Straightening is done either in a rotorizing machine or in a gag press. The process of straightening the beam causes plastic deformations which alter the residual stress pattern in the beam to a sort of asymmetric zig-zag pattern (Alpsten, 1975). Furthermore, in Alpstens study of rotorized beams, the average residual stress in the flanges of all the rotorized beams was significantly reduced while the residual stresses in the web were just slightly changed. Thereby, straightening could be seen as a sort of stress relief (Young, 1975)

However, there is no guarantee that the beams are straightened. Based on measurements, many delivered products have a simple residual stress pattern from cooling, also called thermal residual stress pattern. (Alpsten, 1968). For this reason, assuming that the beam's residual stress pattern is the thermal residual stress pattern in the design results in a design that is on the safe side (Young, 1975) (Alpsten, 1975).

The resulting residual stress, from cooling and possible straightening of the beam, will negatively affect its fatigue life, susceptibility to stress corrosion, and buckling strength (Young, 1975). Areas subjected to residual tension will have a higher mean stress during cyclic loading, which results in a lower fatigue life for the whole beam (James et al., 2007), while areas subjected to residual compression will yield earlier when loaded in compression, increasing the risk of in-elastic buckling (Young, 1975). In addition, the areas in residual tension are at risk of stress corrosion cracking even without any external load (Ghosh et al., 2011).

2.2 Methods of measuring residual stresses

As mentioned in Chapter 1, there are several methods used for measuring residual stresses ranging from destructive to non-destructive methods, with some in between (Rossini et al., 2012). In general, destructive and semi-destructive methods are easier to apply and are valid for a wider range of materials. The drawback is that the method will permanently damage the object studied as well as that the data gathered are usually hard to interpret as errors can easily occur during the processes (Rossini et al., 2012). Non-destructive methods revolve around measuring physical properties. For example, electromagnetism or the spacing between the crystalline structure of the steel, usually referred to as diffraction methods (Rossini et al., 2012). Some of the most common destructive and non-destructive methods are further described below.

Hole-drilling is one of the most used methods today and is classified as a semi-destructive method (Withers & Bhadeshia, 2001b). Drilling a hole in a region with residual stresses allows the material to release or relax its in-plane stresses in the area around the insertion. During the relaxation of the stresses, the material around the hole will deform, allowing the strain to be measured (Withers & Bhadeshia, 2001b). Important to note is the reduction of accuracy as the depth of the hole approaches

and later surpasses its diameter. Also, high residual stresses in an area may cause local yielding, disrupting the relaxation and resulting in incorrect measurements. The effects of this will begin to appear as the residual stresses exceed half of the yield stress (Withers & Bhadeshia, 2001b).

Sectioning is another method that has frequently been used for measuring residual stresses in metallic materials (Rossini et al., 2012). The method involves cutting the specimen into strips while measuring the strain changes, from which the residual stresses along the cutting lines can be calculated (Rossini et al., 2012). It is important that no plasticity or heat is introduced during cutting to achieve accurate results (Rossini et al., 2012). Overall, the method is easy to perform, economical, and gives accurate results (Tebedge et al., 1972), but it is destructive and cannot be performed on the site of the structure or object.

Two diffraction methods in regular use are X-ray and Neutron diffraction. X-ray diffraction has a small penetration depth in steel, allowing for the assumption of plane-strain in the studied area (Withers & Bhadeshia, 2001b). The geometries of elements are an important factor in the availability of x-ray diffraction, as they need to allow the rays to hit the desired measurement area and then diffract directly onto the detector (Rossini et al., 2012). Neutron diffraction is, at large, very similar to X-ray diffraction, but it has some distinctions. The main advantage is the higher penetration depth that allows for measurements up to 60 mm (Fitzpatrick & Lodini, 2003). Drawbacks today are largely the cost and lack of portability (Liu et al., 2023), making it unsuitable for any type of field work.

The ultrasonic method is also a non-destructive technique that has gained traction due to its ease of use, cheap procurement of equipment, and lightweight, making it suitable for on-site inspection (Withers & Bhadeshia, 2001b). Compared to x-ray diffraction, the penetration depth of ultrasonic waves is higher but cannot give as high a resolution of the data collected. Another limitation of the technique is mainly that, for thicker plates only the average stress can be measured (Rossini et al., 2012), requiring supplementary methods to get a better picture of the residual stress distribution.

2.3 Models of residual stress distribution

Several different models of residual stress distribution have been proposed throughout the 20th century. One of the earliest models widely applied was developed by Galaambos and Ketter in the late 1950s (Galambos & Ketter, 1959). Based on results from tests conducted at Fritz Engineering Laboratory, Leigh University in Bethlehem, Pennsylvania (Huber & Beedle, 1953), was applicable only to hot-rolled beams.

The model developed based on the measurements assumes a bi-linear distribution of the residual stresses in the flanges, with the outer-flanges in compression and the

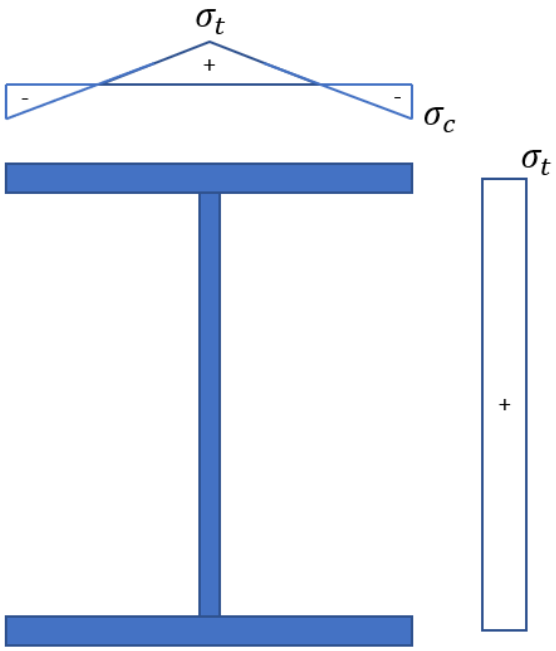


Figure 2.1: American model of residual stress distribution.

middle part in tension (Abambres & Quach, 2016). In the web, a uniform distribution is assumed with tension matching the magnitude of the middle part of the flange, the model is illustrated in Figure 2.1. Equations 2.1 are the equations used to describe the stress distribution according to the model (Abambres & Quach, 2016).

$$\sigma_c = 0.3 f_y \tag{2.1a}$$

$$\sigma_t = \sigma_c \cdot \frac{b_f t_f}{b_f t_f + t_w(h - 2t_f)} \tag{2.1b}$$

In the mid-1970s, B. W. Young at the University of Sussex performed his own tests and examined experimental data done in previous decades regarding distribution of residual stresses (Young, 1975). Young found that in most cases, the distribution of residual stresses could be described with a parabolic curve for both the web and flanges, where the peak stresses are functions of only the geometry (Abambres & Quach, 2016), see Figure 2.2 and Equations 2.2a-c.

$$\sigma_{c1} = 165 \cdot \left(1 - \frac{h \cdot t_w}{2.4 \cdot B t_f}\right) \tag{2.2a}$$

$$\sigma_{c2} = 100 \cdot \left(1.5 + \frac{h \cdot t_w}{2.4 \cdot B t_f}\right) \tag{2.2b}$$

$$\sigma_t = 100 \cdot \left(0.7 + \frac{h \cdot t_w}{2 \cdot B t_f}\right) \tag{2.2c}$$

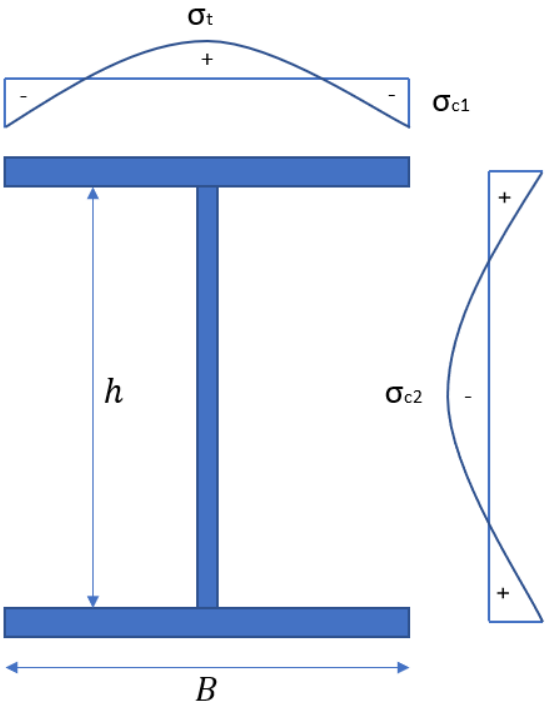


Figure 2.2: Young's model of residual stress distribution.

Certain column sections tested in the USA showed a lean towards a bi-linear distribution over the flanges, a distribution that could be recreated, although the distribution in the web differed. For the specimen tested in the USA, a uniform tensile stress was observed in the web, a phenomenon that was not observed in other studies or tests recreated in the UK (Young, 1975). These tendencies to a more bi-linear distribution are the results used in the model described in Figure 2.1 . Young reasons against some of those assumptions in that, due to the process that generates the residual stress, a bi-linear distribution does not seem reasonable as it introduces a discontinuity in the residual stress (Young, 1975). Introducing a rounded middle point or a heavily weighted parabolic curve would instead be more fitting and follow the results from other experiments according to Young.

Another proposed model is the one from the European Convention of Constructional Steelwork (ECCS), which suggests that the distribution is bi-linear in both web and flanges (Abambres & Quach, 2016). ECCS's model of the residual stress distribution depends on the yield stress of the material and the ratio between the height and the width of the cross section. The distribution is described by Equation 2.3 and Figure 2.3 (Abambres & Quach, 2016).

$$\begin{cases} \frac{h}{b} \leq 1.2 \Rightarrow \alpha = 0.5 \\ \frac{h}{b} > 1.2 \Rightarrow \alpha = 0.3 \end{cases} \tag{2.3}$$

More recently, a model with quadratic distributions was proposed by Skiadopoulos et al. (2023). The model was developed by applying a constrained least square method to measured residual stresses, with an assumption of a quadratic residual

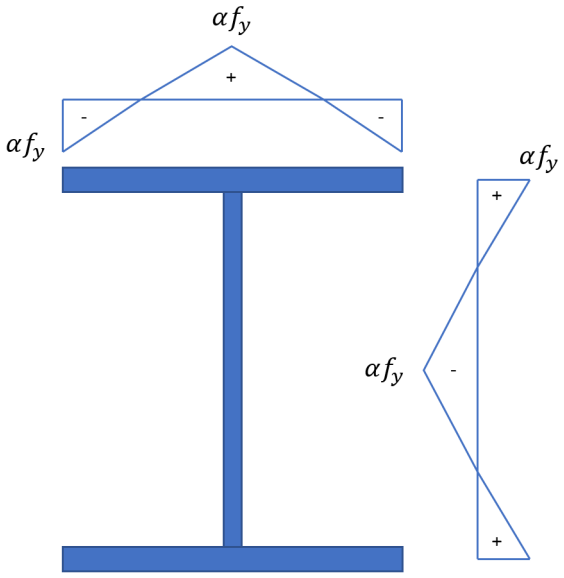


Figure 2.3: ECCS's model of residual stress distribution.

stress distribution according to Equations 2.4a and 2.4b (Skiadopoulos et al., 2023) and Figure 2.4. The constraints in the least square problem were the criteria of force equilibrium in the cross section and stress continuity in the web-flange junction according to Equations 2.4c and 2.4d. An important thing to note is that the fillet radii is not included in the equilibrium condition, which simplifies the equation.

$$\sigma_{0,f}(x) = a + b\left(x - \frac{b_f}{2}\right)^2 \tag{2.4a}$$

$$\sigma_{0,w}(y) = c + d\left(y - \frac{h}{2}\right)^2 \tag{2.4b}$$

$$(2t_f b_f)a + \left(\frac{2t_f b_f^3}{12}\right)b + [t_w(h - 2t_f)]c + \left[\frac{t_w(h - 2t_f)^3}{12}\right]d = 0 \tag{2.4c}$$

$$a = c + d\frac{(h - t_f)^2}{4} \tag{2.4d}$$

The proposed values for coefficients a and c were 37 MPa and 81 MPa, respectively. With the values of a and c, coefficients b and d are determinable by Equations 2.4c and 2.4d (Skiadopoulos et al., 2023).

2.4 Optimisation procedure

To solve the inverse analysis problem, the use of an optimisation algorithm is needed. Several types of algorithms exist to solve this: direct search, gradient, or higher-order derivative methods. For this thesis, a direct search method is to be implemented, a type of heuristic method. Contrasted with derivative methods that uses gradients and derivatives of a function to find a solution, direct search use only the function

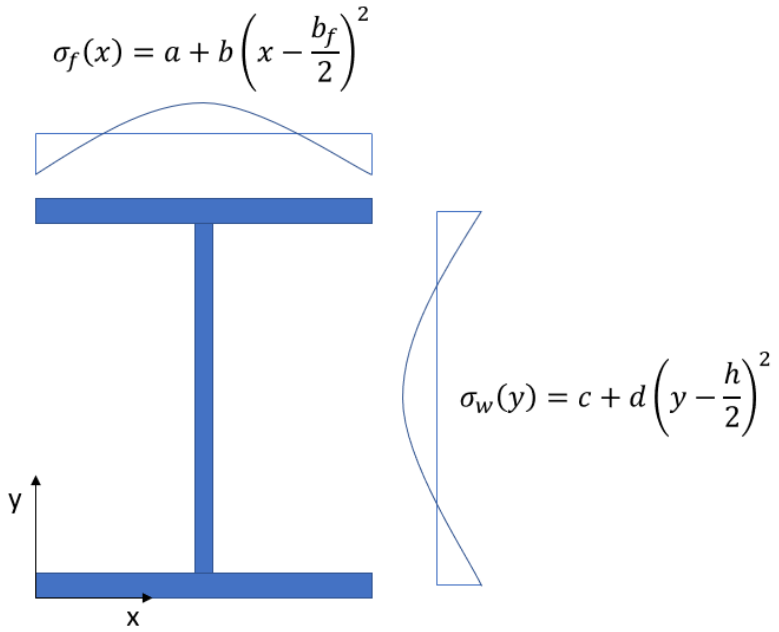


Figure 2.4: Skiadopoulos model of residual stress distribution

value itself to minimise the function. The particular variation to be implemented is called the Nelder-Mead method (Nelder & Mead, 1965) and it is usually used in non-linear problem, such as the one presented in this thesis.

First published in the mid-1960s, Nelder-Mead, as well as a lot of other direct search algorithms, use a simplex to derive an approximate solution (Kolda et al., 2003). A simplex can be described as a geometric object with n+1 vertices in an n-dimensional space and is used in the Nelder-Mead procedure as a tool that is manipulated to find a solution. The initial simplex, either assigned to or created by the algorithm, is of great importance since if the initial simplex is small, the algorithm cannot easily find its way out of a potential local minima. It is therefore recommended to pick a large initial simplex that almost covers the whole area where the solution is expected to be found (Wessing, 2019). It is also recommended to normalise the search space (Wessing, 2019).

To describe the algorithm in an intuitive way, the use of a visual example is commonly used, as in Lagarias et al. (1998). Considering a two-variable problem, the simplex then becomes a triangle. Below, the algorithm is described in six steps, with corresponding figures explaining the process.

The algorithm procedure consist of six steps:

- **Order**
  Sort the vertices so that the value of the function follows $f(x_1) < f(x_2) < ... < f(x_{n+1})$

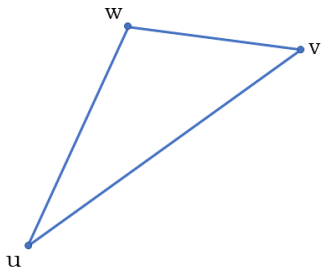


Figure 2.5: Illustration of simplex in two-dimensions.

- **Reflect**
  Calculate the reflection point x_r and evaluate $f(x_r)$. If $f(x_1) < f(x_r) < f(x_n)$ accept x_r as the new point and terminate the iteration.

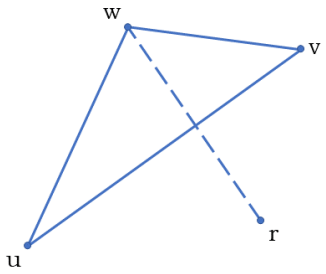


Figure 2.6: Illustration of the reflection step in the Nelder-Mead procedure for a two-dimensional space.

- **Expand**
  If $f(x_r) < f(x_1)$ a guess expanding along the same line as x_r is made and a expansion point is calculated x_e. If $f(x_e) < f(x_r)$ chose x_e and end the iteration else choose x_r.

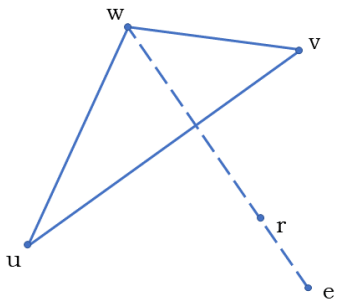


Figure 2.7: Illustration of the expansion step in the Nelder-Mead procedure for a two-dimensional space.

- **Contract**
  If $f(x_r) \geq f(x_n)$ calculate the contraction points x_{c1} or x_{c2} located on the line formed by x_n and x_r.

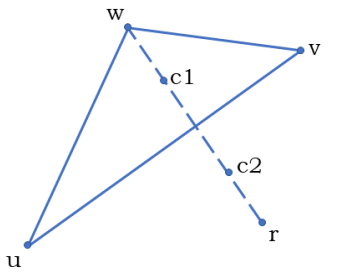


Figure 2.8: Illustration of the contraction step in the Nelder-Mead procedure for a two-dimensional space.

- **Shrink**
  If it follows that neither point x_{c1} or x_{c2} generates a value lower than $f(x_n)$ the simplex is shrunk towards the vertices that has the lowest value.

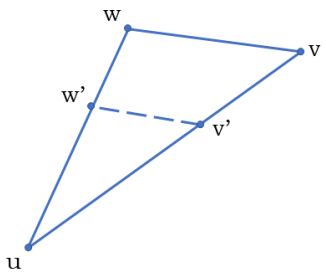


Figure 2.9: Illustration of the shrinkage step in the Nelder-Mead procedure for a two-dimensional space.

- **Convergence**
  The iteration continues until a set convergence criteria is met.

2.5 Strain measurement using optical fibres

Optical fibres can be used to measure several physical parameters, where the most common are strain or temperature changes. The scattering of electromagnetic waves is measured inside the fibres which is divided into three parts where each part contains a different type of spectral data, Rayleigh, Raman and Brillouin scattering.(Jansson, 2024). In essence, the majority of methods compare the current state of the light in the fibre with a base state and the shift in wavelength/frequency is then multiplied by an empirically determined factor to obtain the strain (Kreger et al., 2016).

Compared to traditional strain gauges, distributed optical fibre sensors have a higher sensitivity (Sabri et al., 2013) and they are able to measure at several densely spaced points along their length, allowing more measurements to be done more easily (Kreger et al., 2016) compared to traditional gauges. These traits of the optical fibres result in that the strain field can be measured with high detail and accuracy.

3

Methodology

The first part of the thesis work was to establish a good understanding of the concepts regarding residual stress: how it is created, measured, taken into consideration and modelled. The result of this literature study is presented in Chapter 2. The work after the literature study was composed of physical testing of beams to extract strain data, the development of a numerical model of the test, and the implementation of a optimisation algorithm to find suitable variables describing the residual stress pattern.

3.1 Laboratory tests

Based on the literature findings presented in Chapter 2 and limitations of the available lab equipment, it was concluded that performing a four-point bending test was the most suitable alternative. For a steel grade of S355J2+M and a lever arm of 1.5 m, the largest viable beam profile was calculated to be a HEB220. To ensure a margin of error regarding equipment capacities and other uncertainties, the profiles HEA200 and HEB200 were chosen to be used in the tests. Two beams with a length of 12.1 m were cut into three pieces of roughly 4 m in length by the supplier. Based on this length, the span length was chosen to be 3.6 m to make sure the beams would not roll off the supports during the tests. The distance between the loading points was 600 mm, out of which the middle 200 mm were chosen for strain measurements. An illustration of the setup can be seen in Figure 3.1.

For load application, a hydraulic jack was used. To distribute the load from the hydraulic jack to the two load points, a strengthened rectangular hollow steel beam was used with two rollers attached to the bottom. The rectangular section was 200 mm high and 120 mm wide with a steel thickness of 10 mm. At the bottom of it, a 140 mm wide steel plate with a thickness of 10 mm was spot welded to the rectangular hollow section, reinforcing the beam as well as providing an area for the rollers to be attached to, see Figure 3.2 for a picture of the distribution beam.

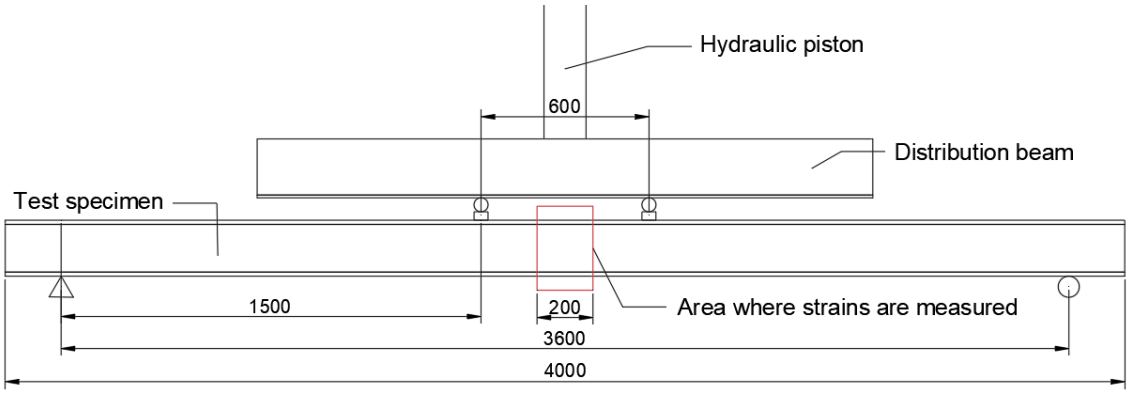


Figure 3.1: Sketch of test setup.

Figure 3.2: Picture of the distribution beam.

Figure 3.3: Beams in different stages of surface corrosion. Top two beams are of type HEB200 and bottom two are of type HEA200.

Figure 3.4: Points for measurements on flanges.

Testing and the necessary preparations for it were performed at Chalmers Structural Lab. When delivered to the lab, the six beams were labelled according to their cross-section type, followed by a number between 1 and 3, e.g., HEA3. The beams were delivered in different stages of surface corrosion, as seen in Figure 3.3, which resulted in different amounts of preparation needed for each beam. Overall, each beam was prepared according to the following procedure:

Using a wire brush the centre of the beam was cleaned of any loose debris and then vacuumed. The area where the fibres were to be attached was marked out, and a margin of 50 mm in each direction along the beam was added. With a combination of manual labour, a drill with an attachment for sandpaper, as well as an angle grinder with the same configuration as the drill, the marked area was cleaned from both rust and scaling.

The thickness of the flange in the cleaned region was measured using a thickness gauge. It was measured from left to right along six points of the cross-section over three section, 100 mm before the middle section, middle section and then 100 mm after. For a reference see Figure 3.4.

Inside the area of raw steel a lot of dust, debris from the sanding and other oily residue that were not previously removed were wiped clean using paper towels and acetone. In cleaning the area with acetone, each swipe of the towel was done in only one direction before a new and clean part of the towel was used. This was done to ensure that the grime was removed from the area and not wiped onto the sanded surface again.

Templates showing the pattern of the fibres were cut out and taped on the beam as

seen in Figure 3.5. This made it easy to tape down the fibres accurately along the cross-section with ten strands along the outside of the flanges, four on each part of the inside of the flanges and seven every side of the web. See Figure 3.6 for a more detailed visual of the fibre patterns.

Figure 3.5: Fibre pattern taped to beam to guide fibre placement.

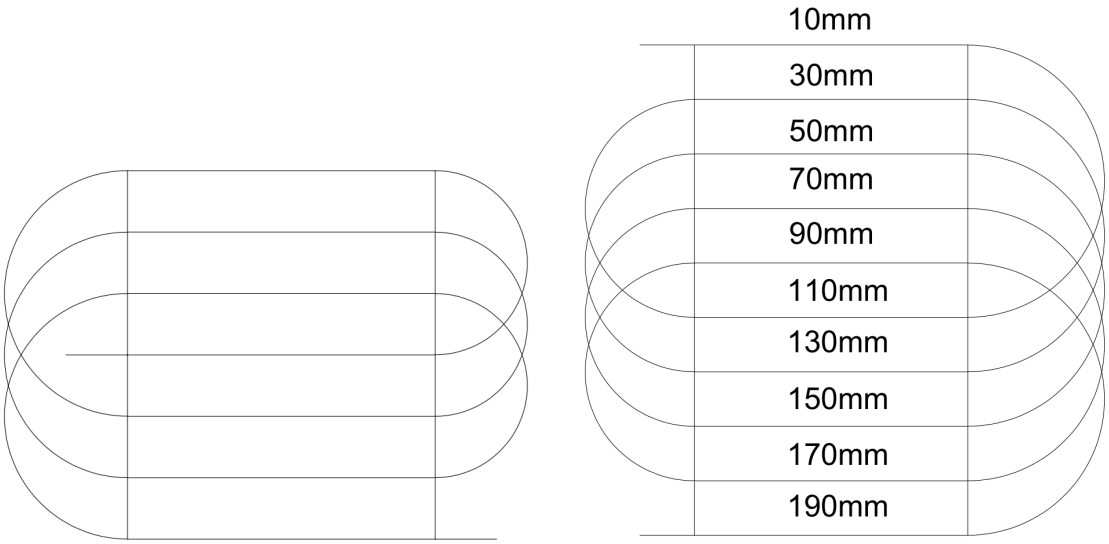
To be able to extract information from the fibres, a connection and an end terminal were needed to be welded on to the fibres. Beginning by preparing the connection end, the protective layer was scaled off and then the inner coating was removed as well by using a wire stripper. The remaining glass fibre were then cleaned with an anti-static wipe and isopropanol. Placed inside a holding block, it was cut to length and then placed into the welding machine on its corresponding side.

For the other fibre, a melting sleeve was placed over the fibre before being prepared. Its only coating, corresponding to the same one as their inner one on the connection, was scrapped off, cleaned, cut to length and placed in the welder as the connection.

With both fibres placed in the welder, they were joined together under an arc created by two electrodes. Tested to see if the joining was successful before it was removed from the welding area, the sleeve was aligned over the joint and then heated in a small oven to encapsulate the weld and keep it from breaking during future handling. Preparing the end terminal, in this case a solid glass fibre with similar coating as the main fibre is done in the same way as the other and then welded onto the other end of the fibres to create a closed loop.

Before gluing the fibres to the beam, they were connected to a sensor interrogator of type Luna Odisi 6100 to be configured and checked. In the software they were labelled and assigned the right refraction index of 1.4610. By configuring each loop of fibre or channel before they were glued down, any damaged or disturbed fibre could be repaired beforehand.

Right before permanently attaching the fibres to the beam using 3M DP-190, a two-part thermoplastic glue, the surface was wiped clean a final time using anti-static wipes and a mixture of isopropanol and acetone. The glue was applied as evenly as



(a) Fibre pattern for webs.

(b) Fibre pattern for outer flanges where the measurement specifies each fibre's distance from the left flange edge.

(c) Fibre pattern for inner flanges.

Figure 3.6: Fibre patterns

possible using a manual glue-gun along each strand, stopping before reaching the tape holding down the fibres in place. Finally a cotton swab was used to push down the glue and fibres to the surface of the steel to ensure a good bond. The glued up beam was then left to cure for a minimum of four days before it was tested.

After the glue had dried, the ends of the sections of interest were marked using a touch-to-locate method in which the program can locate and mark gauges along the length of the fibres. It works by applying a local pressure with to the end of the area that is to be measured on a unloaded fibre which causes a local spike in the strain data that the program will notice and mark as an reference point. In cases where the software had issues with marking the location due to disturbances in the fibre, the gauge was selected manually through visual inspection of the strain graph.

In order to prevent lateral-torsional buckling, a set of brackets were manufactured and placed centred over the loading points. A chain was attached from each side of the bracket onto the testing rig itself and adjusted in tension to restrain lateral movement, see Figure 3.7.

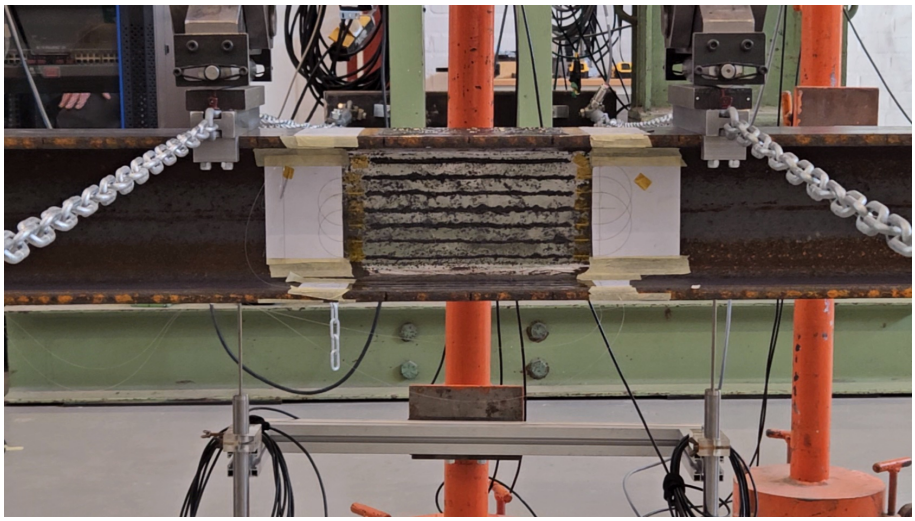


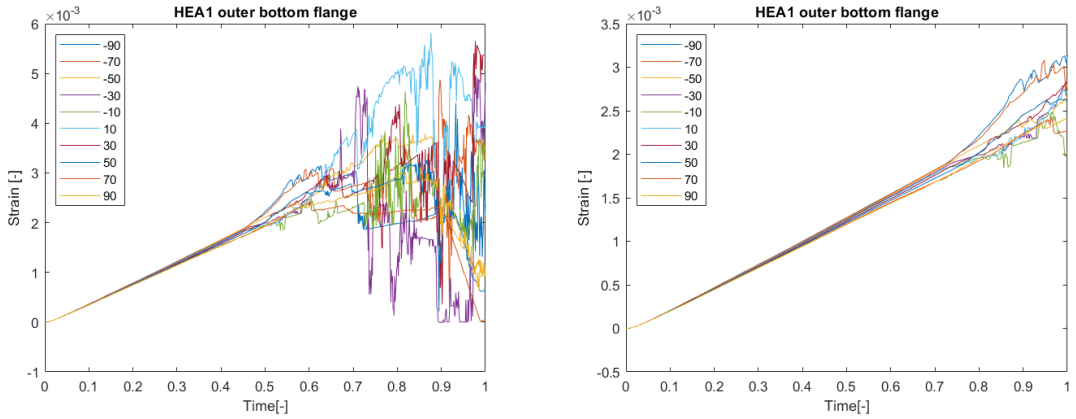
Figure 3.7: Lateral support configuration

Placed upon two cylindrical rollers, the beam was measured and adjusted to follow the specifications of the four-point bending test. Deflections were measured using linear variable differential transformers (LVDT) that were placed on the inside of the bottom flange at one side of the supports and at the outside of the bottom flange centred under the loading points.

Once the beam was placed and adjusted in the rig, the fibres were routed away from the beam and connected to the sensor interrogator. In total, four channels were connected, each one checked to be in good condition before being tared to remove any strains generated from the handling of the beam as well as hardening of the glue. When everything was set up and working correctly, the recording of the strains was started and the test began. The strain data was measured with a frequency of 12.5 Hz and a gauge length of 5.2mm and after each test the data was exported as text files for further post-processing.

The data retrieved was loaded into Matlab and reduced using a Matlab script provided by the institution which averaged every 12 time step together, resulting in roughly 1 value per second for every gauge. From the reduced data, the strain data between the previously marked gauges of each strand were extracted using a Matlab script shown in Appendix A. For each strand, the median strain for each time step was calculated from the extracted data and saved as a row vector with the first element in the vector being the strands x- or y-coordinate, depending on if the surface it was on was horizontal or vertical. The row vectors for each strand were then saved in arrays with strands of similar location on the cross section, e.g., left side of the web or topside of the top flange. Furthermore, the end of the data in time was cut off to exclude noisy data as well as the strain measurements from unloading, after which the time was normalised again. The cut-off-time was the same for every location on the same beam. See Figure 3.8 for an example and Appendix B for all original and cut versions of the strain data.

The deflection at the cut-time step to be used in Abaqus was calculated and noted by taking the average deflection below the two point loads and subtracting the average deflection of the vertical supports from it according to Equation 3.1.



(a) Full data **(b)** Cut data

Figure 3.8: Example of cut data from HEA1 bottom flange.

$$d_{\text{Abaqus}} = \frac{(d_{\text{PointLoad1}} + d_{\text{PointLoad2}})}{2} - \frac{(d_{\text{Support1}} + d_{\text{Support2}})}{2} \tag{3.1}$$

3.2 Tensile strength test of the material

In order to implement the strain data obtained from testing, the corresponding material properties for each beam were needed. As each set of profiles, HEA or HEB, all were from the same original beam only one of each beam was used to gather the testing stock. Sections of the flange and web located inside the supports were cut out using a plasma cutter. From the testing stock each test specimen or dog-bone was cut out using a water jet cutter. During the cutting process the water expands out of the nozzle causing a wider cut as it cuts through the material, resulting in a rectangular cross-section that will turn into a parallelogram. After the specimens were cut out, they were measured using digital calipers to calculate the true cross-sectional area, as well as to see if the other dimensions were according to standard.

The tensile test was performed according to SS-EN ISO 6892-1:2019 with an extensometer with a gauge of 50 mm. HEA flange specimen 1 was tested first and it was determined that the rust on the grip areas of the specimen had to be removed before testing, to ensure a proper grip in the machine. Therefore, the results from flange specimen 1 were ignored.

For the other tests the strain and force data were collected with strain given in millimetres. To get the strain, the displacement data was divided by the extensometer's gauge length, and to get the stress, the force was divided by the measured cross-sectional area. The mean of the strain-stress relation for the specimens of the same

type were then calculated and based on the mean relation, Young's modulus and the plastic behaviour were determined. See Appendix C for the calculation script.

3.3 Python script for Abaqus-model and post processing

The creation of the Abaqus model and the post processing was done through a Python script, which can be seen in Appendix D. This was done so that later changes of the parameters of the beam, more specifically the parameters describing the residual stress field could be changed without recreating the whole model manually. In addition, using a script allowed Abaqus to run without a graphical user interface (GUI), which reduced the time of the analysis.

The structure of the script can be divided in to two sections: creation and execution of functions for analysis, and comparison of data. The former section follows the overall structure of an analysis in Abaqus, beginning by creating the parts, material and instances, moving on to assembly and later loads. Finally the instances in the assembly were seeded for the mesh and element type was decided before being meshed and analysed. In the latter section of the script the data from the analysis is compared to the data from the physical tests and an "error" is calculated as the sum of the absolute value of the difference in strain for the measured points. In Figure 3.9 a simplified flowchart of the script(s) are visualised .

For the digital model some simplifications were made. Half of the beam was modelled and a symmetric boundary condition was applied to what would normally be the mid-section of the beam. All displacements and rotations out of the symmetry plane were constrained and the rest were set free. The parts of the beam outside the vertical supports were not modelled and the loads were applied over a line rather than an area as in the physical tests. Additionally, the vertical supports were only modelled as a vertical constraint rather than modelling the rollers with contact points. The lateral support were also simplified by modelling it as a fixed horizontal constraint in a line at the ends of the top flange at the location of the point loads, when in reality the beam were allowed to move slightly and the support had a rectangular contact area with the beam. Furthermore, the cross section was modelled as an ideal beam with dimensions based on the average measured dimensions where both flanges were modelled with the same thickness.
See Figure 3.10 for a visual representation of the model.

The beam was modelled with a combination of shell elements and solid elements. The parts where plastic behaviour was expected to occur were modelled with solid elements, while the parts outside the point loads and towards the supports were modelled with shell elements. This was done to increase the level of detail through out the thickness of model in the area of interest while having lower resolution in other areas by using shell elements.

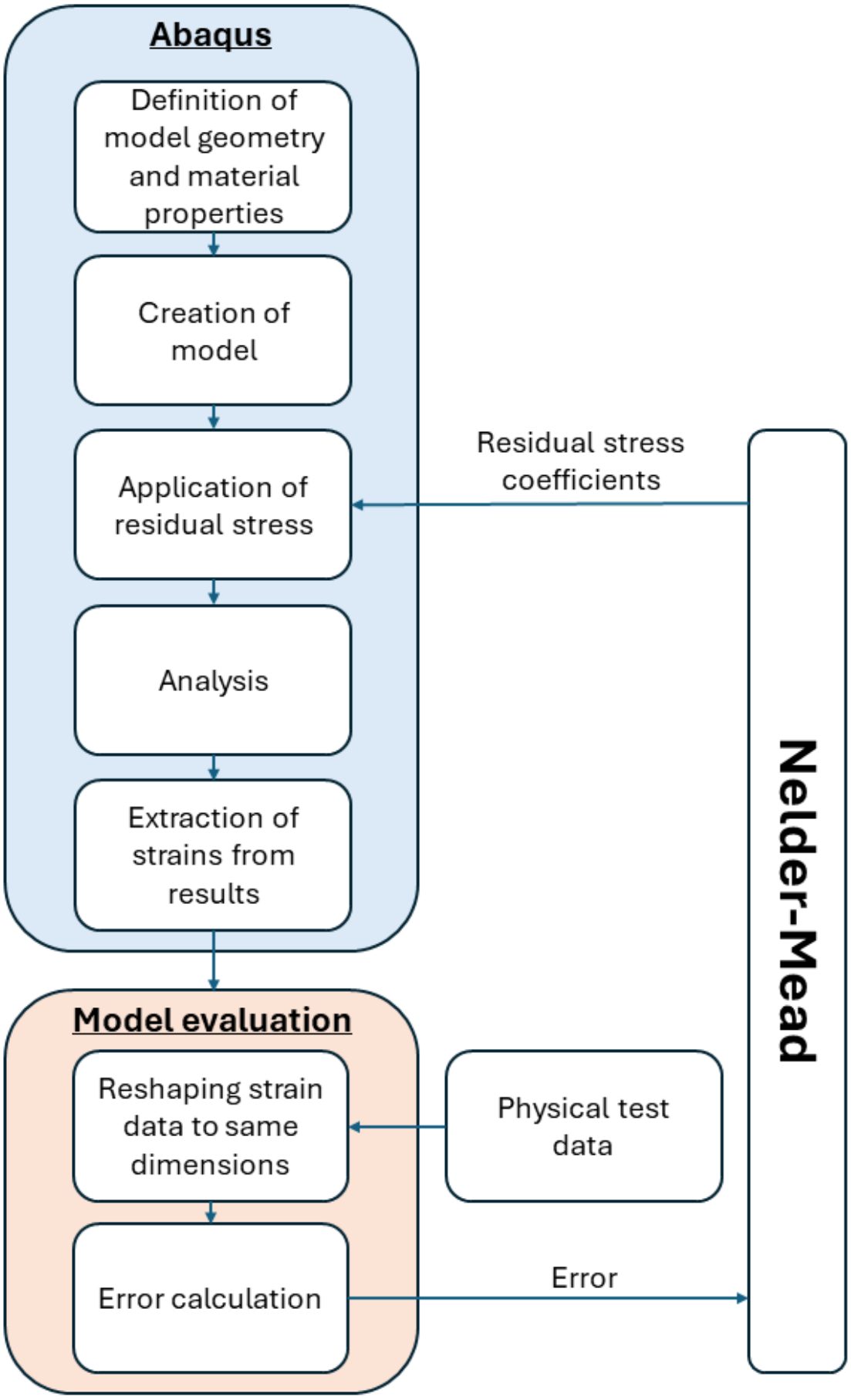


Figure 3.9: Simplified flow chart of Abaqus and error calculation script

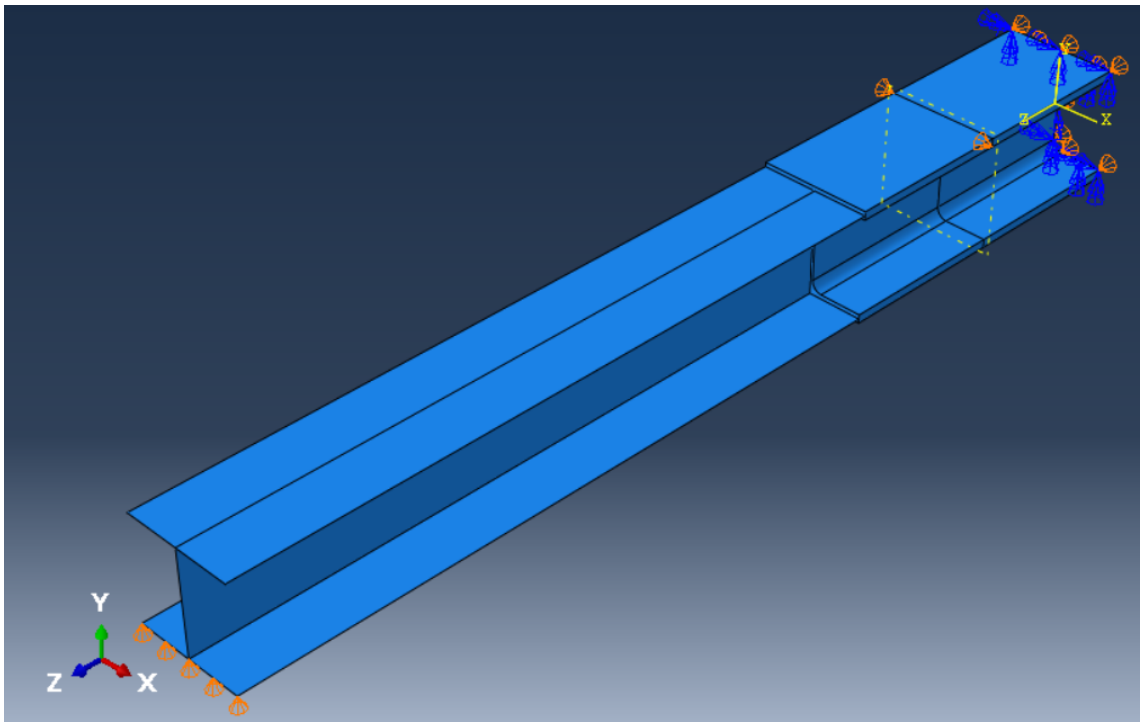


Figure 3.10: Shell and solid assembly with visible supports and symmetry conditions.

Elements used in the solid section were hexahedral with 8 nodes (C3D8) with a seed size of 5 mm whereas the shell elements were triangular (S3) with a varying seed size. At the edge between the solid and the shell sections the seed size was 5 mm while at the edge by the support the seed size was set to 50 mm. This resulted in a gradually increasing element size from the solid edge to the support. No rigorous sensitivity analysis were performed of the model, instead simulations for different seed sizes in the beginning of the development together with the need for a non-linear relationship between strain at the outside and inside of the flanges quickly guided the model to the seed sizes used.

The residual stress was applied through predefined stress fields and only to the solid elements. Instead of going through each element and applying a value, sets of elements containing the same x- and y-coordinates were created to reduce the computational load during data entry.

To determine the residual stress for each set, modified versions of Skiadopoulos equations, mentioned in Section 2.3 were used without the set values for residual stress coefficients a and c. The equations were modified such that the coordinates origin was in the middle of the cross section which resulted in Equations (3.2) where a, b, c and d are constants. Instead of the values for a and c given by Skiadopoulos, residual stress coefficients c and d were assumed to be known, wherefrom a and b could be determined using Equations (3.2c) and (3.2d).

$$\sigma_{0,f}(x) = a + bx^2 \tag{3.2a}$$

$$\sigma_{0,w}(y) = c + dy^2 \tag{3.2b}$$

$$(2t_f b_f)a + \left(\frac{2t_f b_f^3}{12}\right) b + [t_w(h - 2t_f)]c + \left[\frac{t_w(h - 2t_f)^3}{12}\right] d = 0 \tag{3.2c}$$

$$a = c + d\frac{(h - t_f)^2}{4} \tag{3.2d}$$

With all constants being known, each set of elements were given their residual stress based on their coordinates and Equations (3.2a) and (3.2b). In Figure 3.11 and 3.12 the principal idea for the sets for residual stress are shown as well as the residual stress state of the beam right after application.

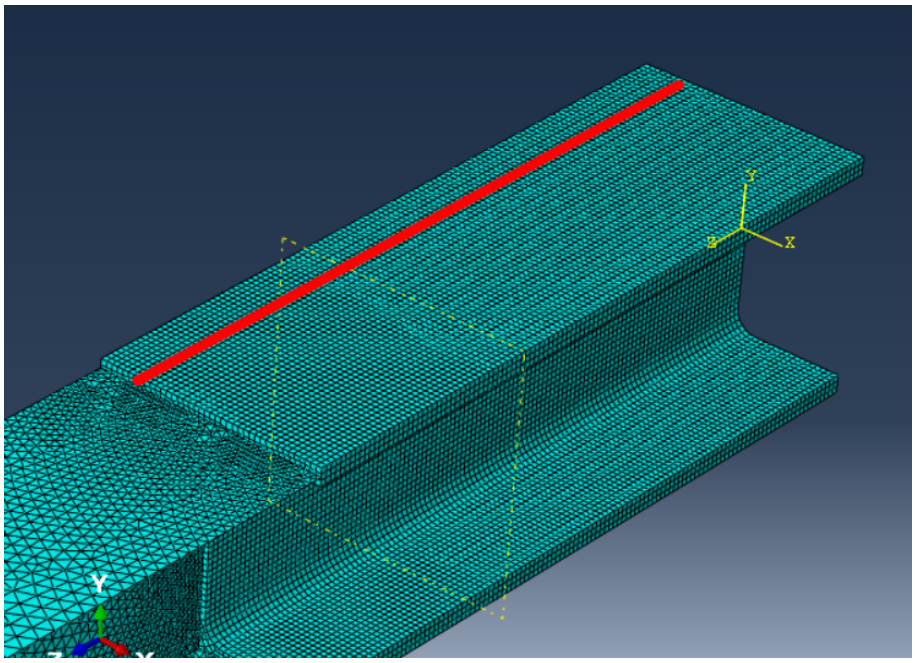


Figure 3.11: Example view of an element set for residual stress application.

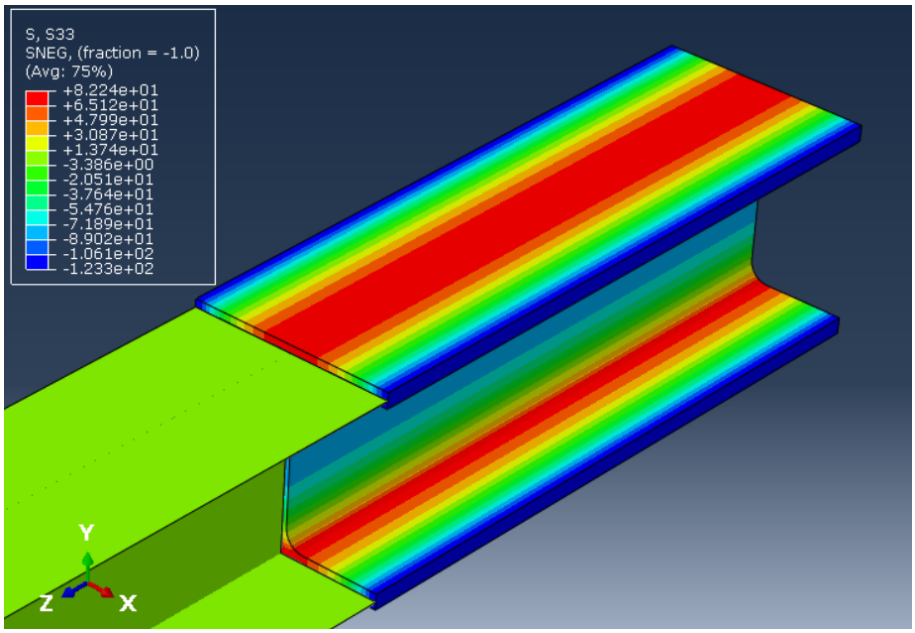


Figure 3.12: Residual stress distribution.

The analysis was split into three steps. In the first step the applied residual stress were allowed to reach self equilibrium before any loading was applied. In step two the gravitational load was applied using the gravity type load which is based on the density assigned to the material. The densities of the beams were calculated to $7.845 \cdot 10^{-9}$ $\frac{tonnes}{mm^3}$ and $7.850 \cdot 10^{-9}$ $\frac{tonnes}{mm^3}$ for the HEA and HEB beams respectively, based on the specifications given by the supplier, showwn in Appendix E. The density was calculated to $\frac{tonnes}{mm^3}$ as to match with the units used in Abaqus. During the last step the two line loads were applied using deformation control. The displacements applied to the loading lines were applied linearly until the deflection at the lines were equal to the deflection from the physical test specified by Equation 3.1.

After each simulation, the strain data of interest had to be retrieved. The data was retrieved using 8 paths along the web and flanges at the symmetry edge, i.e. in the middle of the full beam, see Figure 3.13. From the paths the longitudinal strains (E33) were extracted for each time frame in the loading step and the strain from the steps before were then subtracted from the extracted strains to simulate the tare made in the physical tests. The strain data was then compiled in arrays, one for each path. The arrays' first columns held the X- or Y-coordinates along each path and the following columns held the respective strain for each time frame. Whether it held X- or Y-coordinates were dependant on if the surface was horizontal of vertical.

To determine how well the modelled beam with its assumed residual stress distribution depicted the real behaviour, the strain data was compared to the physical test data. To compare the two data sets, the data first had to be formatted such that the two matrices in each location had the same number of coordinates and time frames. The set with the highest resolution was reduced to match the one with lower resolution by firstly checking for direct matches of coordinates/time frames and otherwise interpolating the data between two nearby values to match the coor-

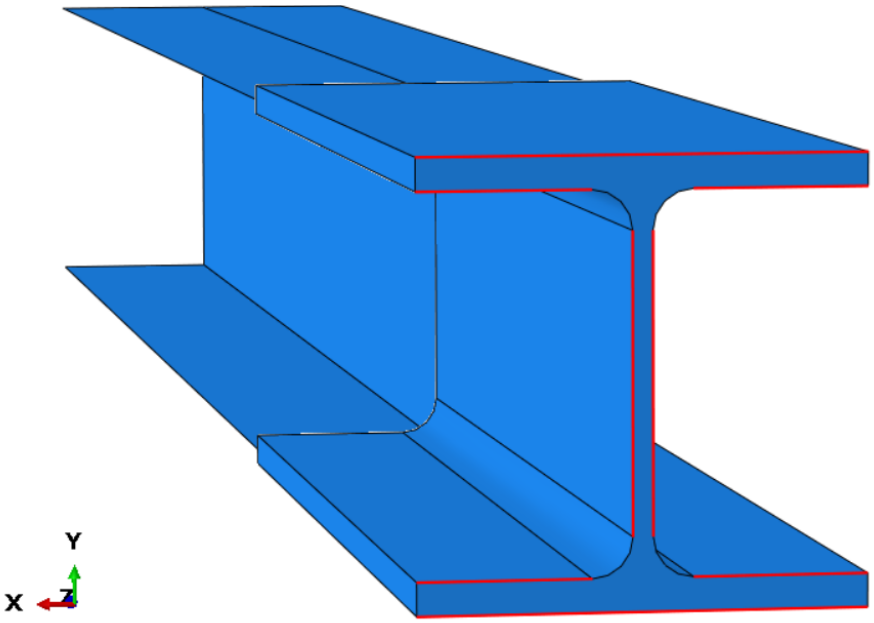


Figure 3.13: Location of paths from where longitudinal strain is extracted in Abaqus marked in red.

dinate/time frame. This resulted in that the final dimensions were based on Abaqus time frames and the physical tests coordinates.

When the data sets had the same dimensions, an error was calculated for every location by first subtracting the two matrices for the location and taking the absolute value of each element. The sum of all the elements in the resulting matrix was then calculated and the resulting number was the error of the location. The eight locations' error were summed up and saved in a text file. The error calculation could be simply described by Equation 3.3. For an example of the reshaping and error calculation process see Figure 3.14, where dimensioning coordinates are circled in blue, dimensioning time frames are circled in red, and strain data used for error calculation is underlined in green.

$$Error(c,d) = \sum_i \|\varepsilon_{measured.i} - \varepsilon_{FEM.i}(c,d)\| \tag{3.3}$$

3.4 Optimising routine

In order to obtain the residual stress distribution for the beams that were tested, an optimisation routine was implemented using a Nelder-Mead algorithm. This routine was created in a separate script from the one implementing and running the Abaqus model as well as post processing the results. The script is shown in Appendix F. Using SciPy Library's minimise function, the script extracts the error from comparing FE strain data and real strain data from the Abaqus script as well as the

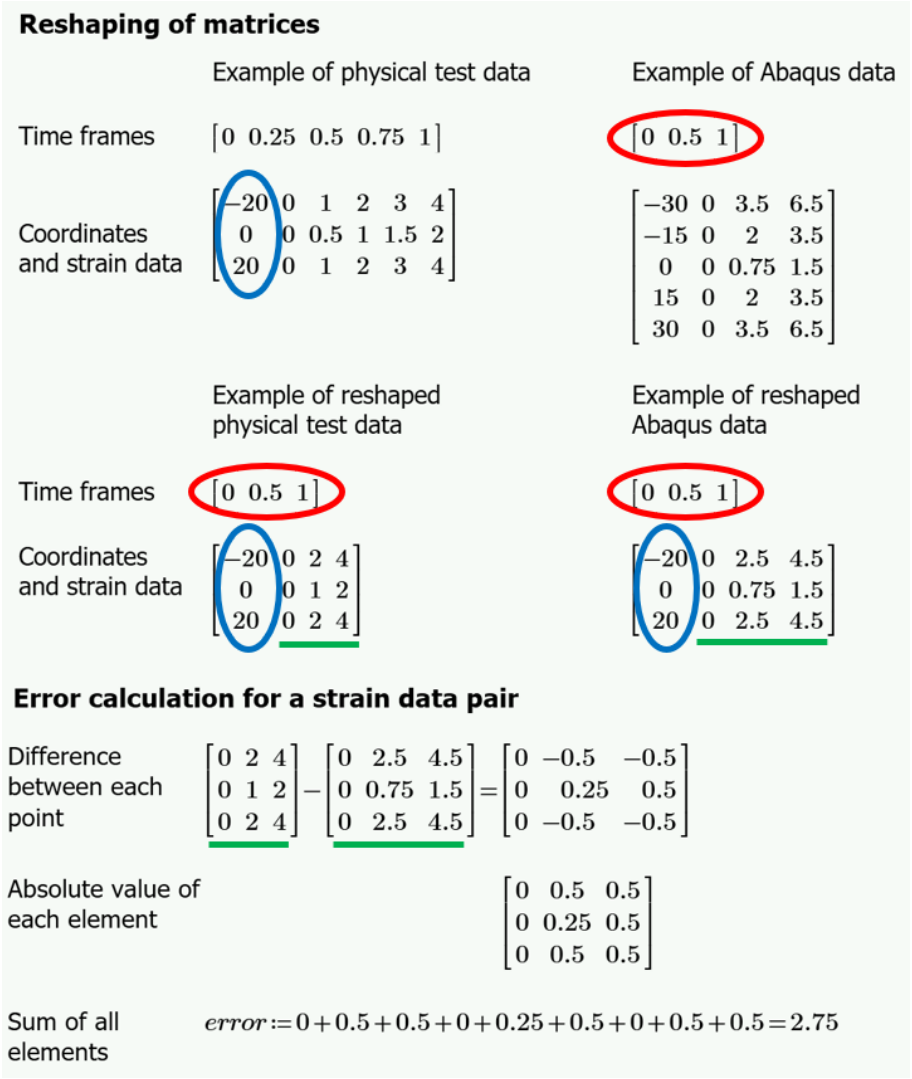


Figure 3.14: Example of data reshaping and error calculation for one location. The dimensioning coordinates are circled in blue and dimensioning time frames are circles in red. After reshaping the data sets are of the same dimensions. The strains used for calculating the error are underlined in green.

residual stress coefficients used for each simulation. The algorithm tries to minimise the error by changing the residual stress coefficients as described in Section 2.4.

The tolerances for the Nelder-Mead algorithm were set to 0.01 for input values and 0.0001 for the error which meant that the change in input and error from the previous assumption had to be lower than its respective tolerance to consider that the optimisation algorithm had converged. When both criteria were met, the final residual stress coefficients were assumed to describe the true residual stress distribution. A flowchart of the procedure is shown in Figure 3.15.

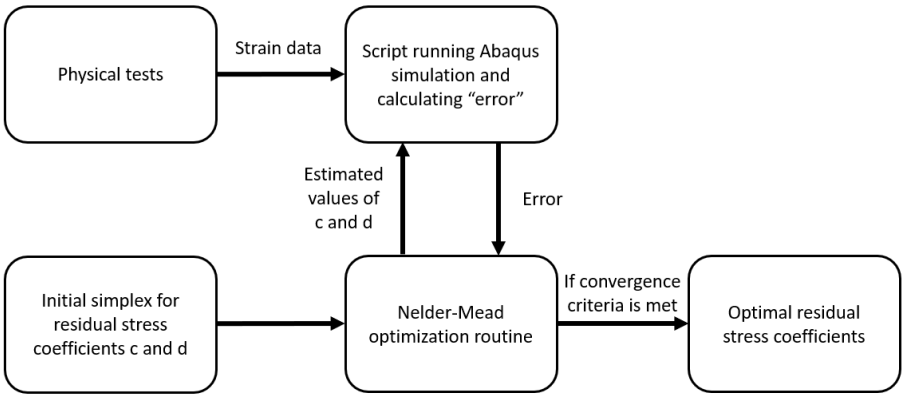


Figure 3.15: Flow chart of the inverse analysis

To reduce the possibility that the algorithm closes in on a local minima or not expanding its search from an initial guess of the residual stress coefficients, a staring simplex was implemented in the script. By forcing the initial search area to cover a larger area, it will force the algorithm to change the coefficients in a more volatile fashion for each simplex node, compared to a single initial guess. In Figure 3.16 a visualisation of the coverage obtained from using starting simplexes is illustrated. In total, three starting simplexes were implemented for each beam to further reduce the potential of the script finding local minima.

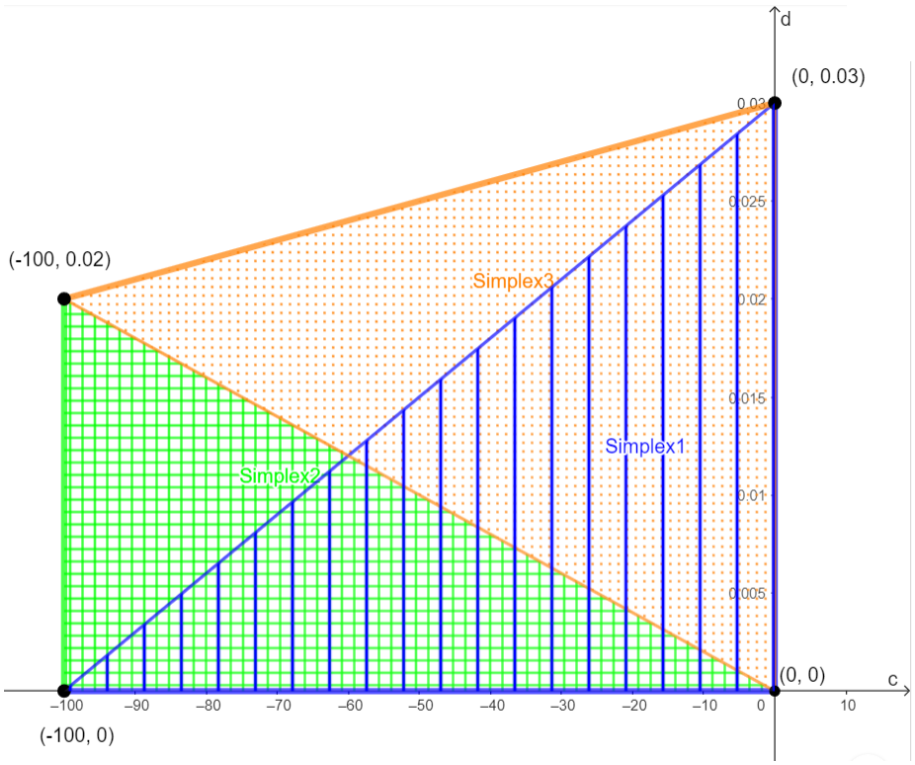


Figure 3.16: Coverage of starting simplexes with residual coefficient c on the x-axis and residual coefficient d on the y-axis.

3.5 Validation of method

To validate the developed method of estimating the residual stress coefficients, simulated test data were generated for an ideal beam and then the routine was run once for each simplex mentioned in Section 3.4. If the routine would be able to reach the residual stress parameters used for creating the simulated test data multiple simplex, the routine could be considered valid. The simulated test data was created with the measurements of an ideal HEB200 beam loaded until 30 mm deflection at the point loads and the residual stress parameters c and d were set to -88.75 and 0.02 respectively. For web and flanges Young's modulus was set to 200 GPa and the plastic behaviour was described by Table 3.1.

Stress [Mpa]	355	420	570	610	640	675
Plastic strain [-]	0	0.2	0.4	0.6	0.8	1

Table 3.1: Plastic behaviour used when creating the simulated data.

For the first simplex the routine successfully found the original values ending the optimisation at $c = -88.7538$ and $d = 0.020001$. Using the second simplex the original values were also found, ending at $c = -88.7469$ and $d = 0.019999$. The run with the third simplex was also successful and the optimisation ended at $c = -88.7542$ and $d = 0.020001$. In Figure 3.17 the changes of the residual stress coefficients and error throughout the iterations for the three simplexes are shown and in Table 3.2 a summary of the runs are shown.

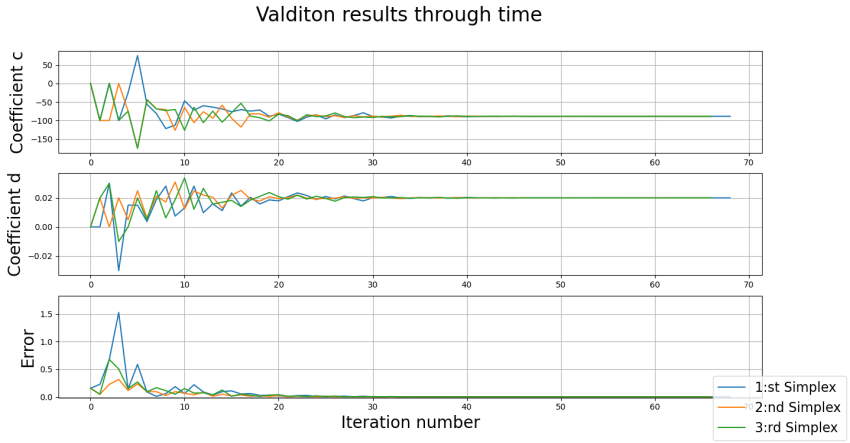


Figure 3.17: Validation of model simulations, coefficients through time

According to the validation attempts, the method works. It finds the original values of c and d even with starting simplexes that does not initially contain the optimal value.

Beam	Simplex	Number of iterations	c	d	Total error
Validation run	1	69	-88.7538	0.020001	0.001148
	2	67	-88.7469	0.019999	0.001147
	3	67	-88.7542	0.020001	0.00115
	Mean	**67.667**	**-88.7516**	**0.02**	**0.001148**

Table 3.2: Summary of the validation runs with the three simplexes

4

Results

In this chapter the results from all parts of the project will be presented.

4.1 Beam dimensions

In Table 4.1 the average thicknesses of the beams are shown. For all measurements see Appendix G.

Beam	Average flange thickness [mm]	Average web thickness [mm]
HEA1	9.522	6.775
HEA2	9.529	6.708
HEA3	9.500	6.750
HEB1	14.943	9.025
HEB2	14.953	14.958
HEB3	14.965	9.050

Table 4.1: Average thickness of the flanges and web for all tested beams.

4.2 Material model

Based on the data obtained from the tensile tests described in Section 3.2, four stress-strain curves were obtained that are used to describe the material properties of the test specimen. The specimens cross-sectional dimensions are shown in Table 4.2 and the stress-strain curves are shown in Figures 4.1-4.4, where the specimens' curves are shown as dashed lines and their mean value, which is used for analysis in Abaqus, as a black line. For the HEB flange, the second specimen was removed from the mean calculation since it differed greatly from the other two specimen due to issues during the testing procedure. The determined Young's modulus for the four mean curves are shown in Table 4.3.

Tensile test specimen
Cross-sectional area, units in milimeters [mm]

	b1	b2	h1	h2	b_mean	h_mean	Area [mm^2]
AF1	17.94	18.08	9.89	10.06	18.01	9.975	179.650
AF2	18.16	18.07	9.86	9.93	18.115	9.895	179.248
AF3	17.66	17.87	9.56	9.36	17.765	9.46	168.057
AF4	17.77	17.93	9.35	9.24	17.85	9.295	165.916
AW1	20.41	20.43	6.78	6.82	20.42	6.8	138.856
AW2	20.39	20.47	6.83	6.68	20.43	6.755	138.005
AW3	20.37	20.47	6.89	6.73	20.42	6.81	139.060
BF1	11.78	11.9	15.21	15.2	11.84	15.205	180.027
BF2	11.79	11.86	15.22	15.21	11.825	15.215	179.917
BF3	11.7	11.95	15.2	15.23	11.825	15.215	179.917
BW1	18.87	18.91	9.21	9.22	18.89	9.215	174.071
BW2	18.9	19.03	9.14	9.14	18.965	9.14	173.340
BW3	18.85	19.03	9.08	9.11	18.94	9.095	172.259

Table 4.2: Measurements of tensile test specimens. A denotes a HEA-profile, B a HEB-profile and F or W represent flange or web respectively.

Cross-section part	Young's modulus, E [GPa]
HEA web	201.132
HEA flange	203.580
HEB web	184.166
HEB flange	220.328

Table 4.3: Young's modulus

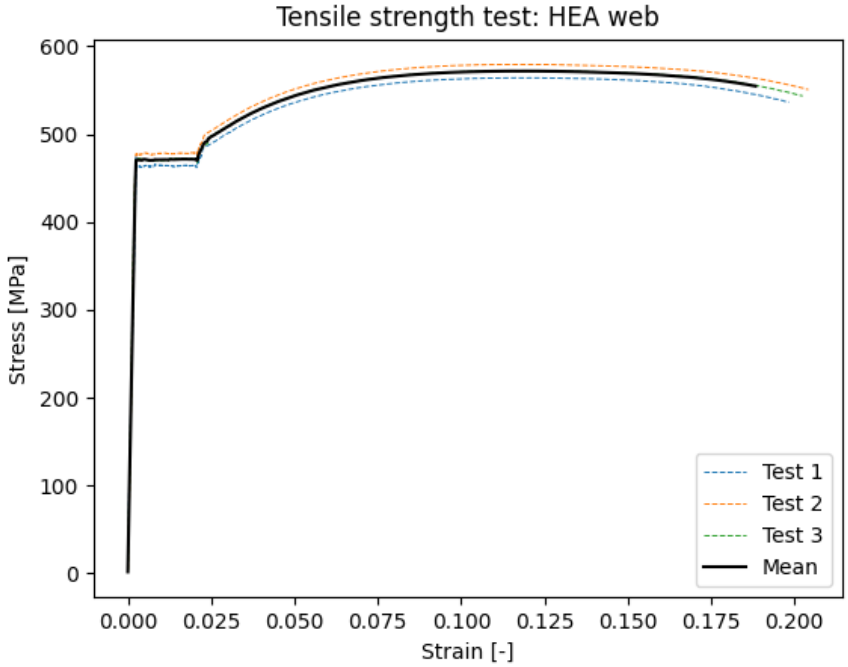


Figure 4.1: HEA web strain-stress curve

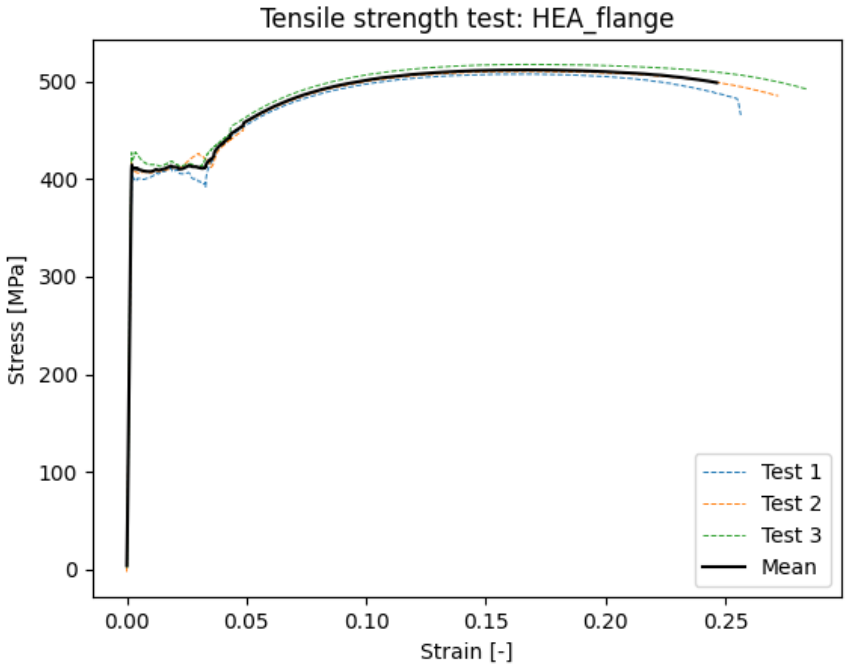


Figure 4.2: HEA flange strain-stress curve

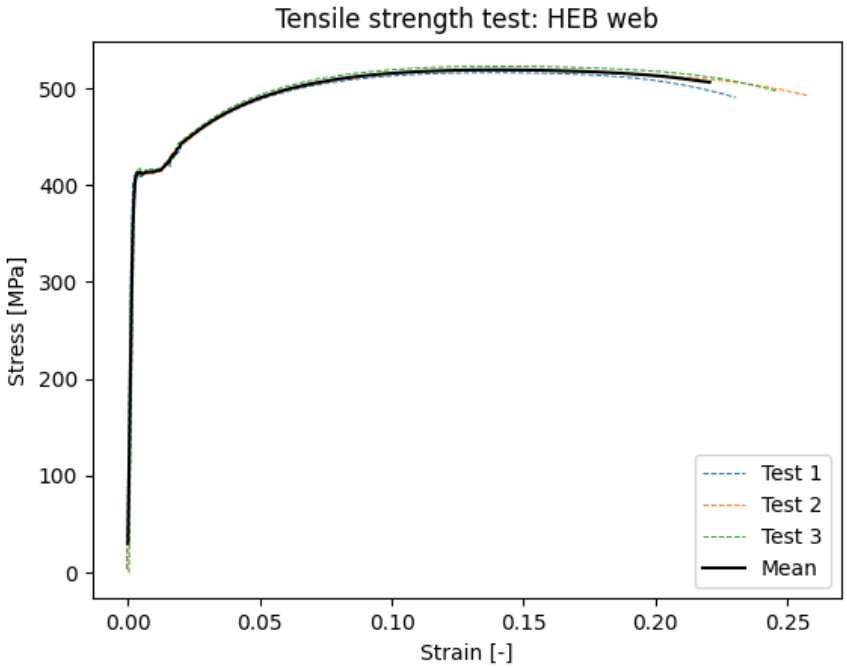


Figure 4.3: HEB web strain-stress curve

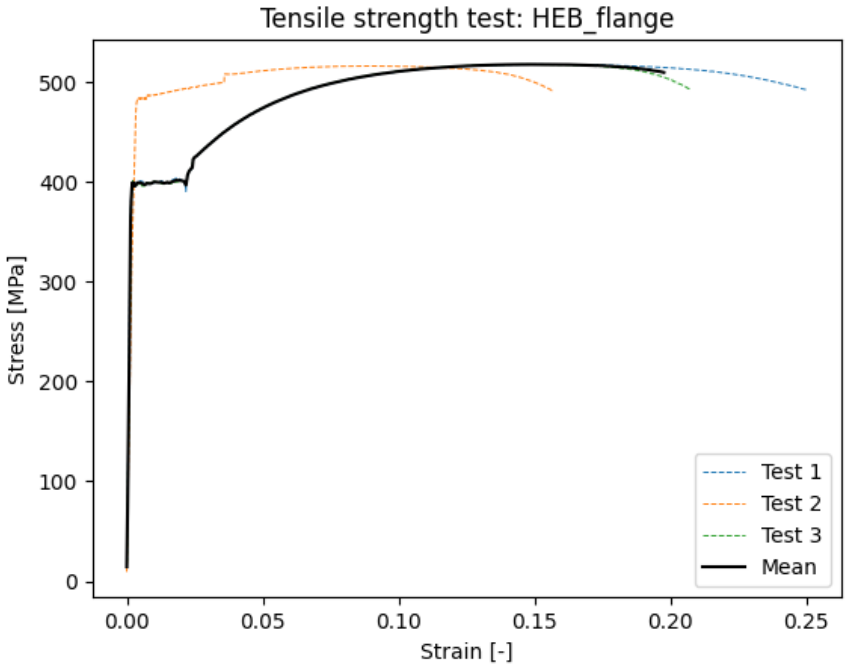


Figure 4.4: HEB flange strain-stress curve

4.3 Test data

The force-displacement curves from the four point bending tests of the six beams are shown in Figure 4.5.

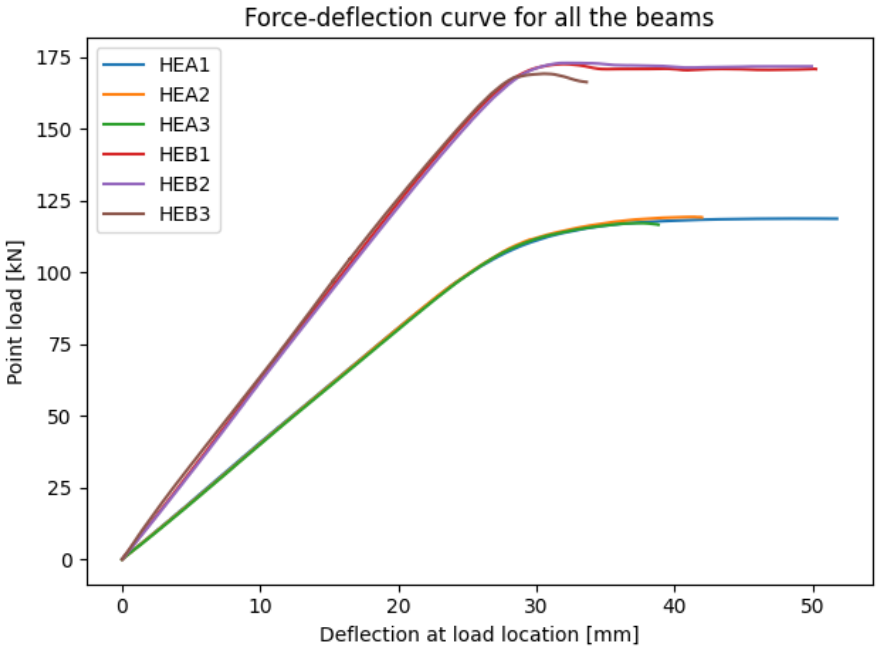
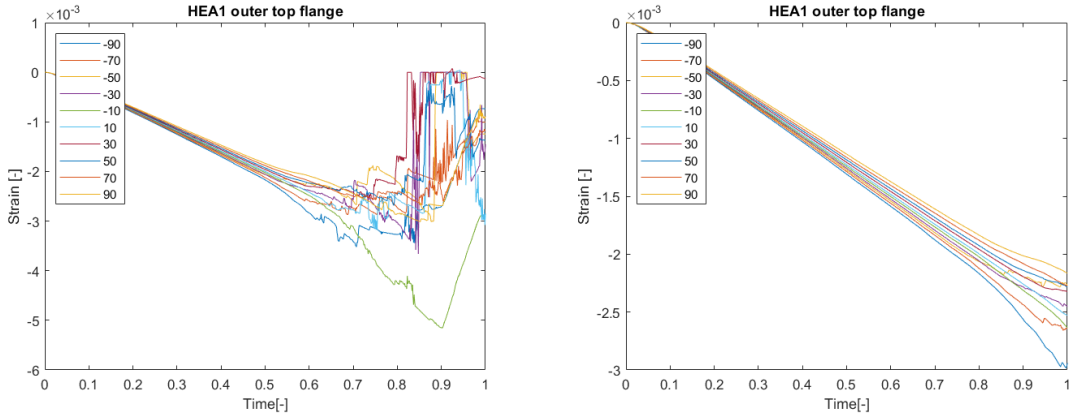


Figure 4.5: Force-deflection curves for all six beams. Deflection is the calculated deflection according to equation 3.1.

As earlier mentioned, the results from the physical tests had to be cut to remove unreasonable data. An example of full and cut strain data from the tests is shown in Figure 4.6, the rest of the strain curves are show in Appendix B. The time of the cut and what the calculated deflection at the cut-time was, for each beam, is shown in Table 4.4. This deflection is the the deflection assigned as a displacment controlled load to the Abaqus model.



(a) Full data **(b)** Cut data

Figure 4.6: Example of cut data from HEA1 top flange.

Beam name	Time of data cut [s]	Deflection at time of cut [mm]
HEA1	500	33.907
HEA2	440	29.360
HEA3	460	30.287
HEB1	480	29.193
HEB2	520	32.454
HEB3	500	28.530

Table 4.4: Time of data cut for the different beams and the deflection at the cut time.

4.4 Evaluation of coefficients through time

Results from the optimisation through time is shown in Figures 4.7-4.12, where the x-axis is the evaluation number of the Nelder-Mead algorithm. This is used in evaluating the effectiveness of the algorithm and the impact of using different starting simplexes.

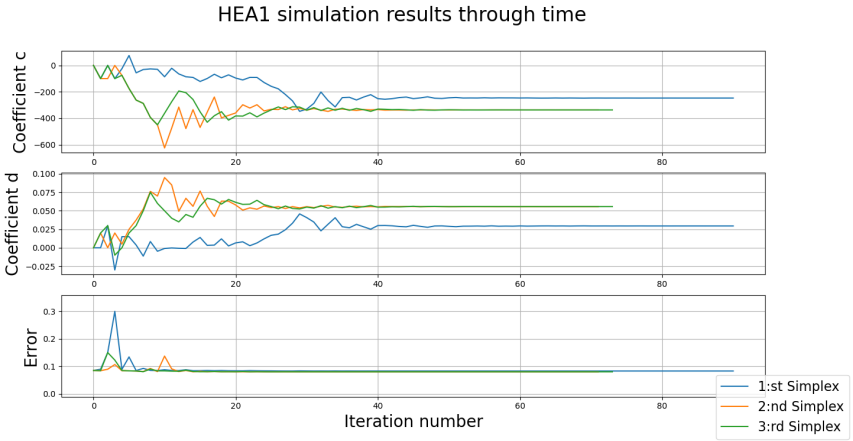


Figure 4.7: Simulation of HEA1

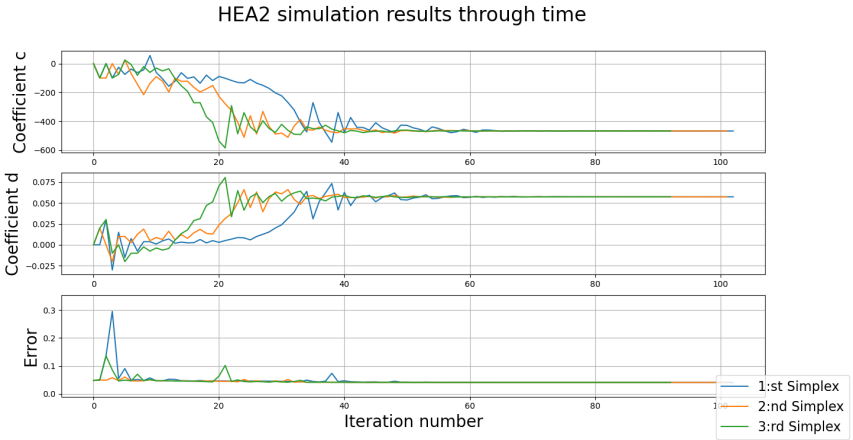


Figure 4.8: Simulation of HEA2

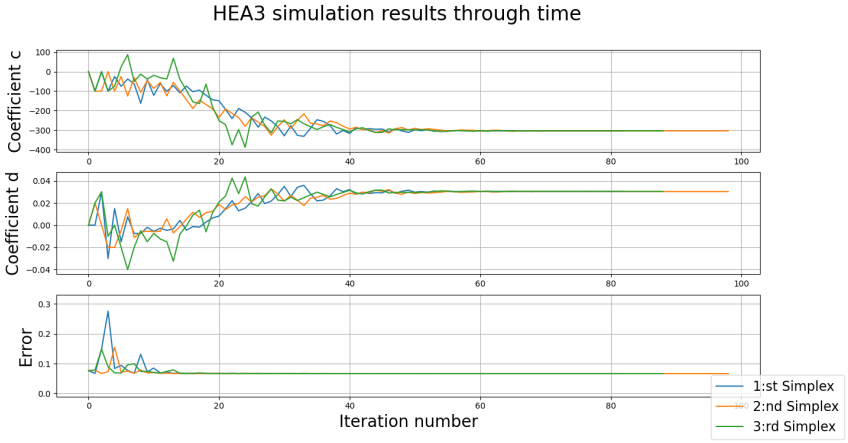


Figure 4.9: Simulation of HEA3

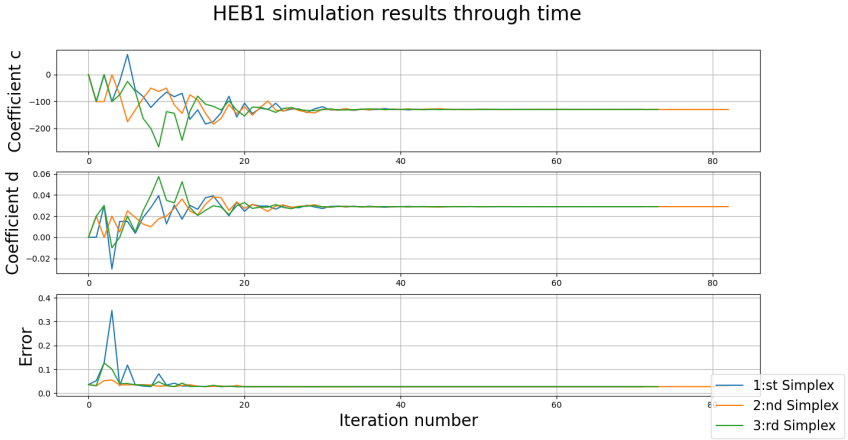


Figure 4.10: Simulation of HEB1

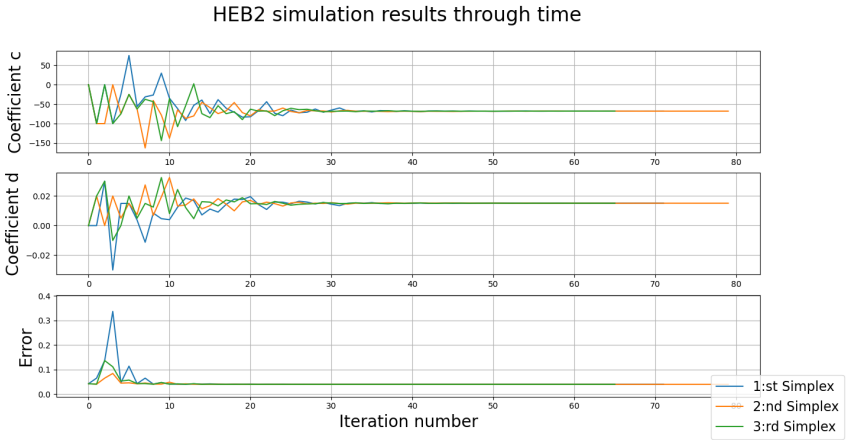


Figure 4.11: Simulation of HEB2

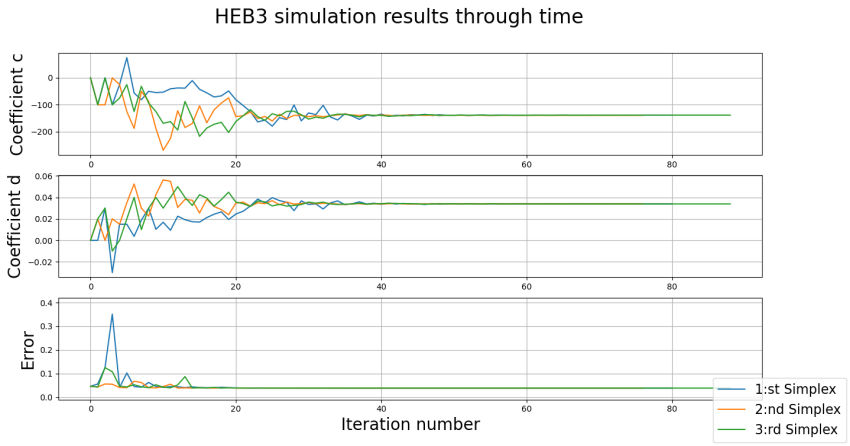


Figure 4.12: Simulation of HEB3

4.5 Optimisations results

In the Tables 4.5 and 4.6 the results from the simulations of the final routine is shown. Several attempts for each beam were done using different initial assumptions as shown in Figure 3.16 in a effort to avoid a local minimum. Standard deviation is given as a value and not a percentage of deviation. Error is given as the sum of all errors in time and space.

Beam	Simplex	a	b	c	d	Total error
HEA1	1	5.393	0.01197	-247.241	0.02952	0.08301
	2	139.564	-0.02622	-337.165	0.05570	0.08001
	3	139.503	-0.02620	-337.150	0.05687	0.08002
	Mean	**94.820**	**-0.01348**	**-307.185**	**0.04736**	**0.08101**
HEA2	1	-0.998	0.02990	-467.428	0.05728	0.04073
	2	-0.842	0.02982	-467.042	0.05726	0.04073
	3	-0.840	0.02982	-467.074	0.05726	0.04073
	Mean	**-0.893**	**0.02985**	**-467.181**	**0.05727**	**0.04073**
HEA3	1	-43.643	0.03094	-303.734	0.03036	0.06652
	2	-44.553	0.03116	-303.713	0.03028	0.06652
	3	-43.988	0.03098	-303.731	0.03035	0.06652
	Mean	**-44.061**	**0.03103**	**-303.724**	**0.03032**	**0.06652**
HEA total mean		**16.622**	**0.01580**	**-359.363**	**0.04498**	**0.06275**
HEA standard deviation		71.078	0.02536	93.389	0.01363	0.02040

Table 4.5: Results of simulation for HEA-profiles

Beam	Simplex	a	b	c	d	Total error
HEB1	1	118.806	-0.03105	-129.496	0.02901	0.02749
	2	118.818	-0.03105	-129.427	0.02900	0.02749
	3	118.771	-0.03105	-129.312	0.02898	0.02749
	Mean	**118.798**	**-0.03105**	**-129.412**	**0.02900**	**0.02749**
HEB2	1	61.691	-0.01601	-67.899	0.01514	0.03996
	2	61.660	-0.01607	-67.918	0.01514	0.03996
	3	61.663	-0.01608	-67.890	0.01514	0.03996
	Mean	**61.671**	**-0.01605**	**-67.902**	**0.01514**	**0.03996**
HEB3	1	151.933	-0.04120	-138.555	0.03394	0.03830
	2	152.137	-0.04123	-139.159	0.03403	0.03830
	3	151.847	-0.04118	-138.379	0.03391	0.03830
	Mean	**151.972**	**-0.04121**	**-138.697**	**0.03396**	**0.03830**
HEB total mean		**110.814**	**-0.02944**	**-112.004**	**0.02603**	**0.03525**
HEB standard deviation		45.677	0.01265	38.474	0.00975	0.00677

Table 4.6: Results of simulation for HEB-profiles

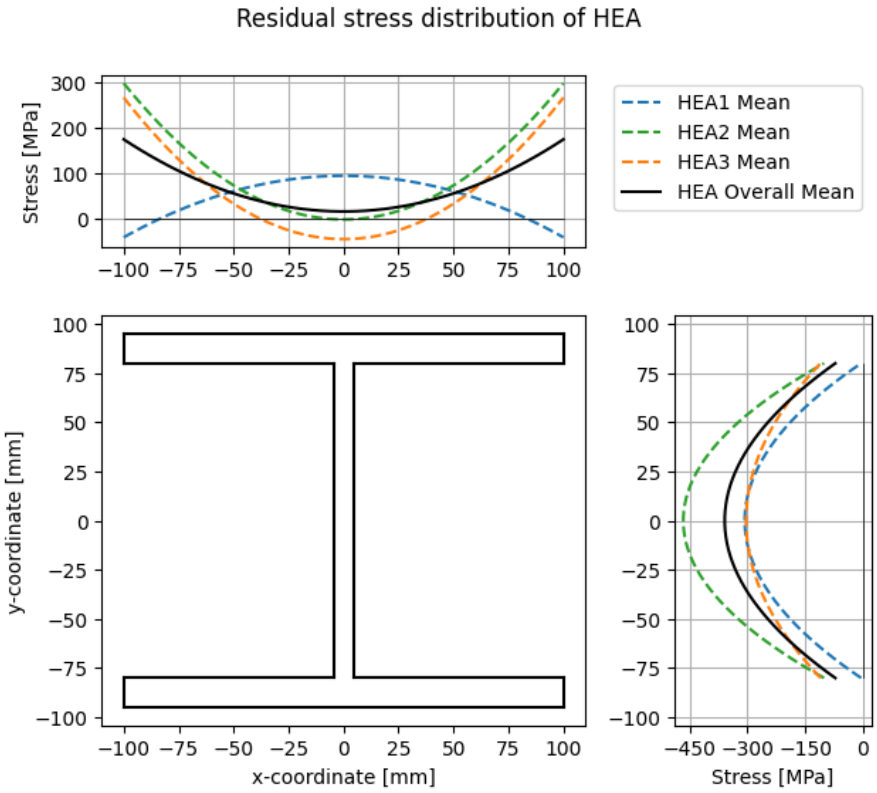


Figure 4.13: Residual stress distribution of HEA based on mean coefficients in Table 4.5

.

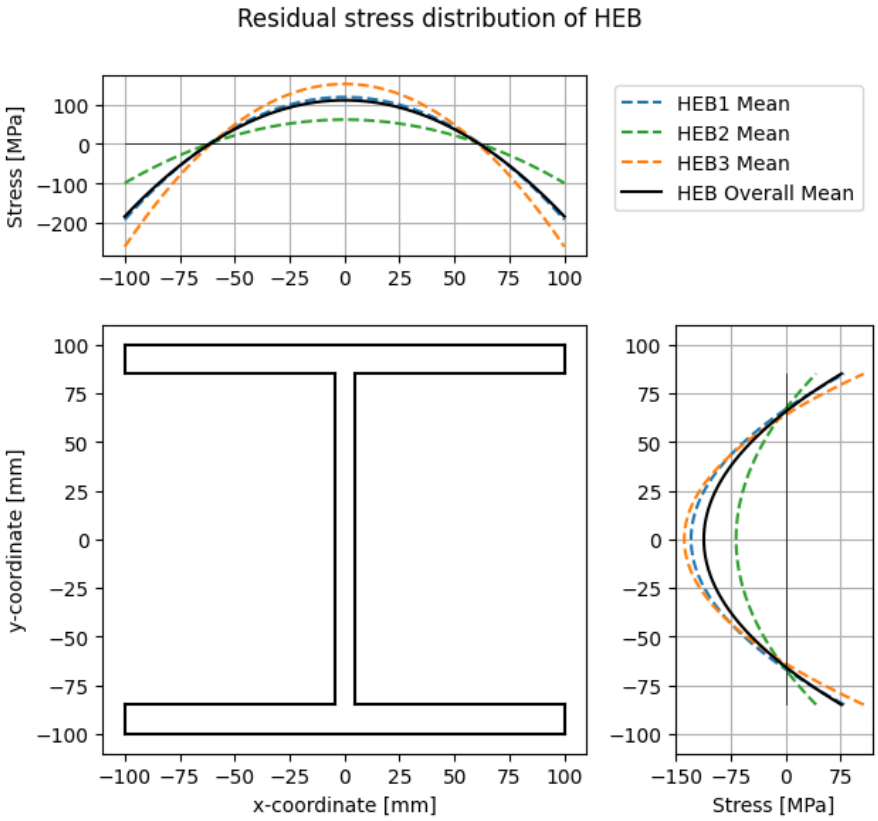


Figure 4.14: Residual stress distribution of HEB based on mean coefficients in Table 4.6

.

4.6 Comparison between finite element and real strain

Strain data obtained from Abaqus simulation using the final residual stress coefficients presented in Table 4.5 and 4.6 is compared with its corresponding data from testing in Figures 4.16-4.23. The curves in these figures were the ones used for the final error calculation of HEB1. In Appendix H, the curves for all the beams can be seen. As seen in the figures, the x-axis is the normalised time. To see how the deflection over time was for the physical test compared to Abaqus, see Figure 4.15.

All data acquired from testing starts at a strain of zero, as fibres are tared before testing is started, see Chapter 3 for further explanation. As time moves forward, it can be observed that the curves increasing in strain will at a certain point decrease or stay at the current strain level for a while until the strain rate will increase once again. This happens as the beam goes from elastic to plastic behaviour and stress redistribution occurs throughout the cross-section.

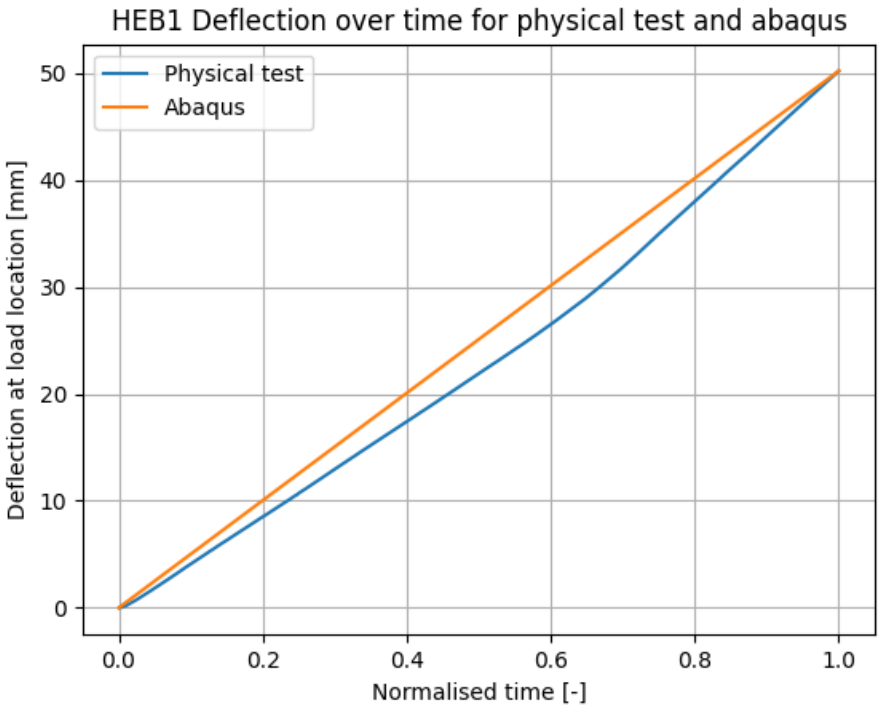


Figure 4.15: HEB1 deflection over time for Abaqus and physical test.

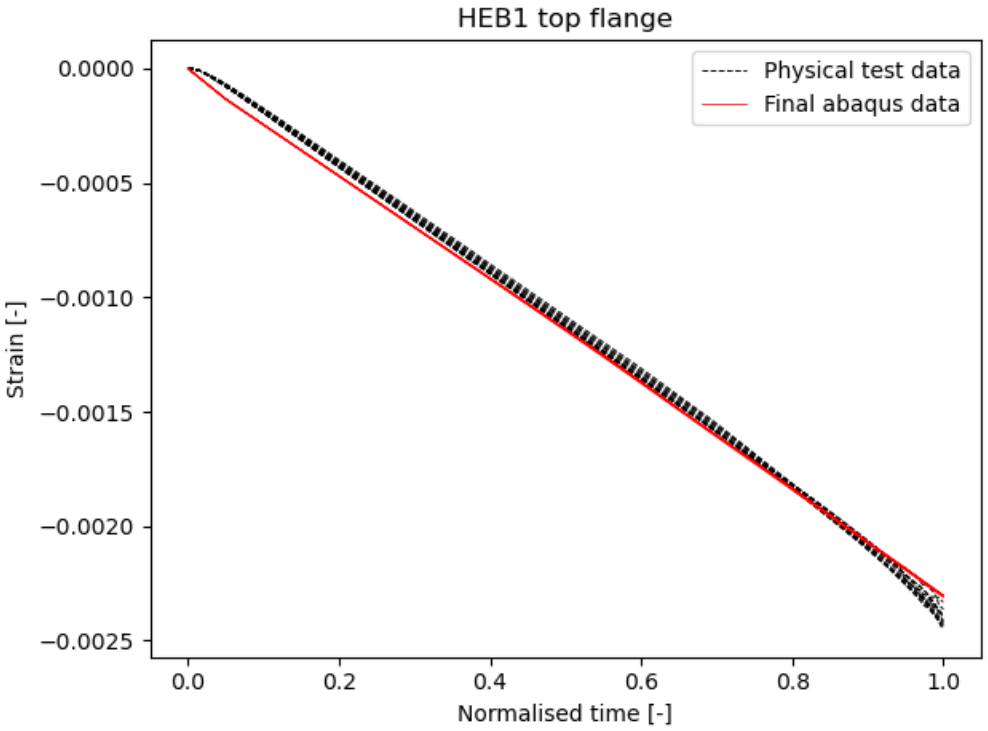


Figure 4.16: Final strain comparison for HEB1 top flange

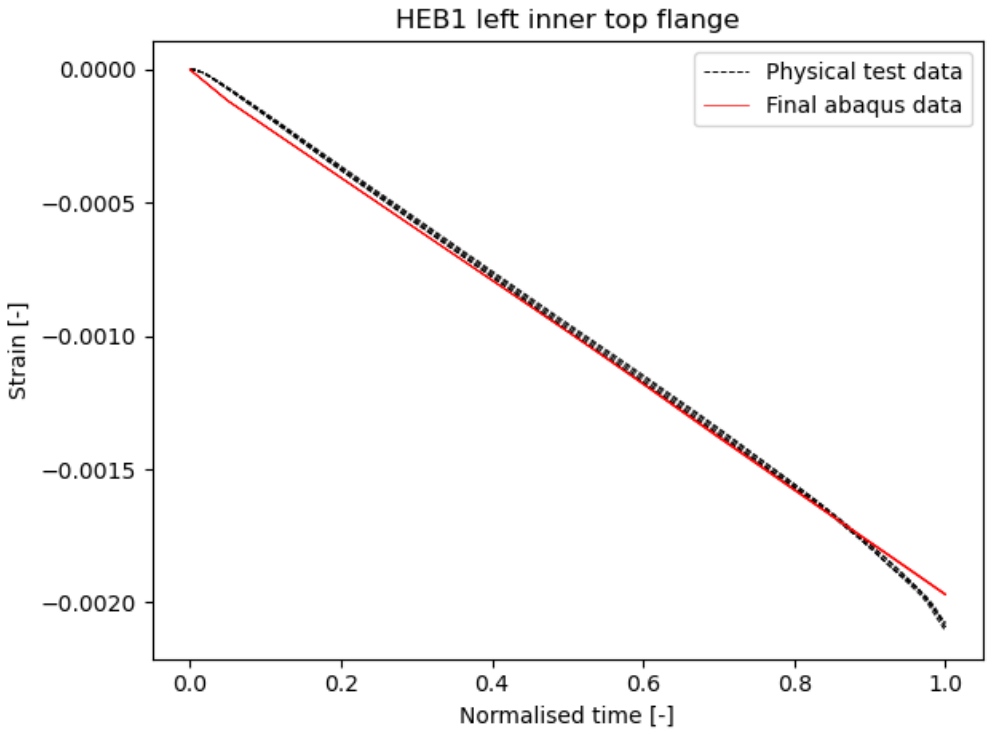


Figure 4.17: Final strain comparison for HEB1 left inner top flange

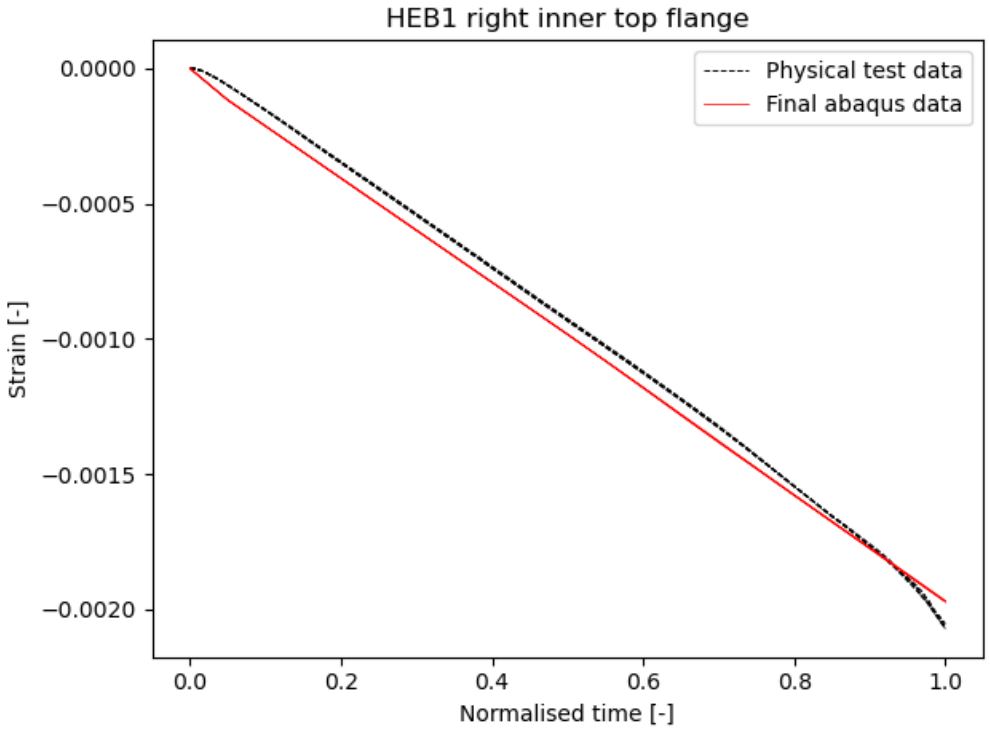


Figure 4.18: Final strain comparison for HEB1 right inner top flange

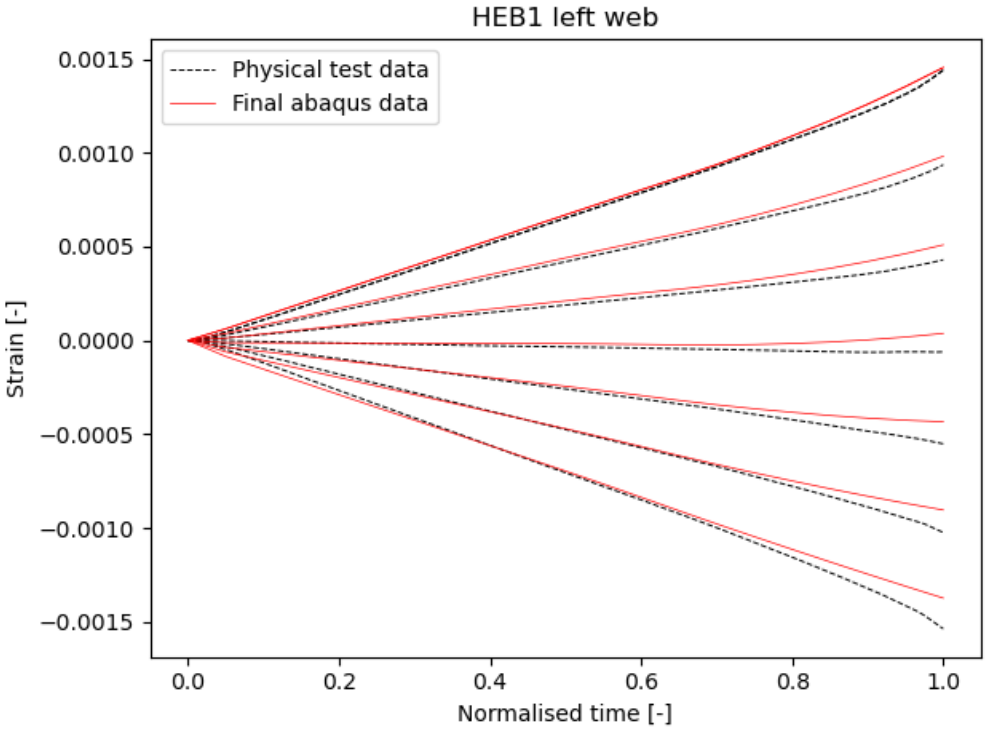


Figure 4.19: Final strain comparison for HEB1 left web

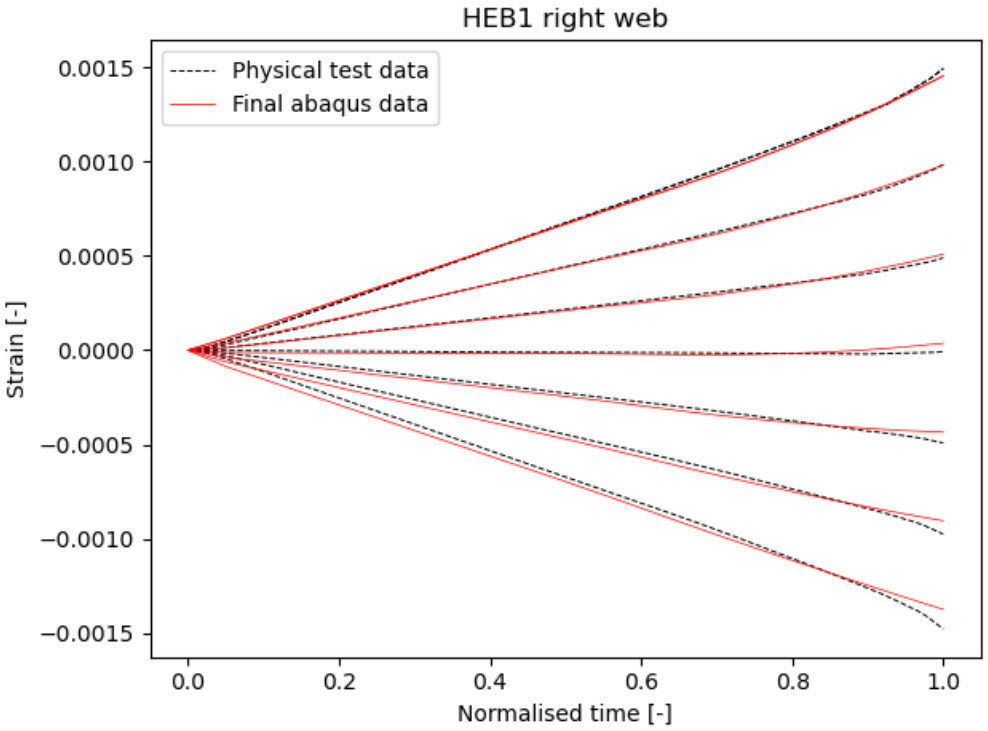


Figure 4.20: Final strain comparison for HEB1 right web

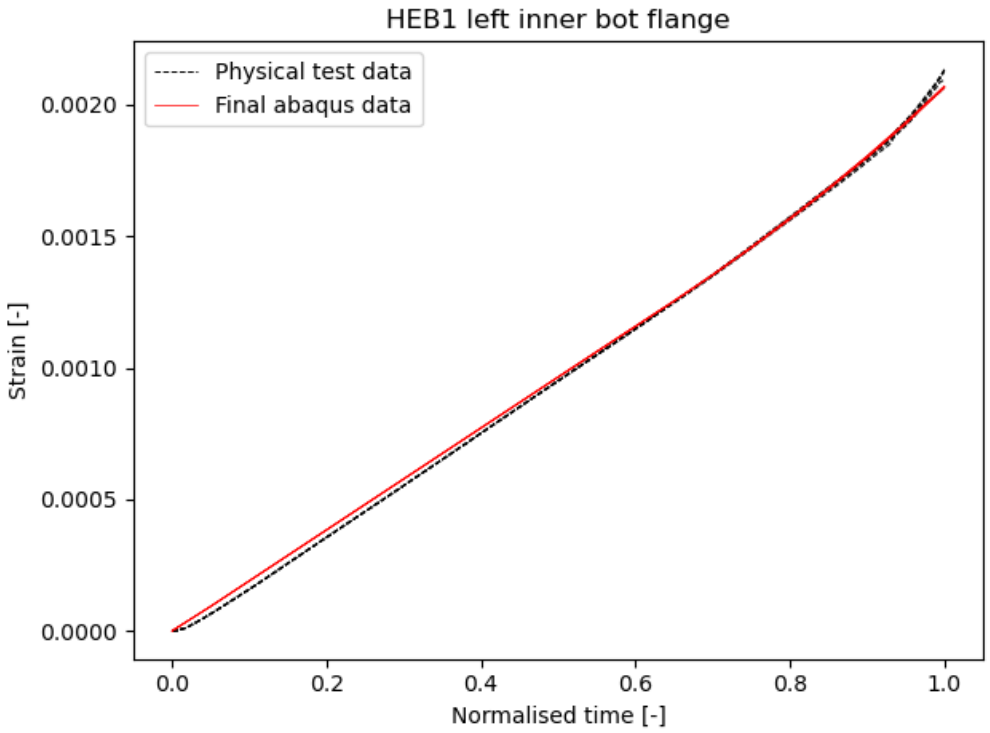


Figure 4.21: Final strain comparison for HEB1 left inner bot flange

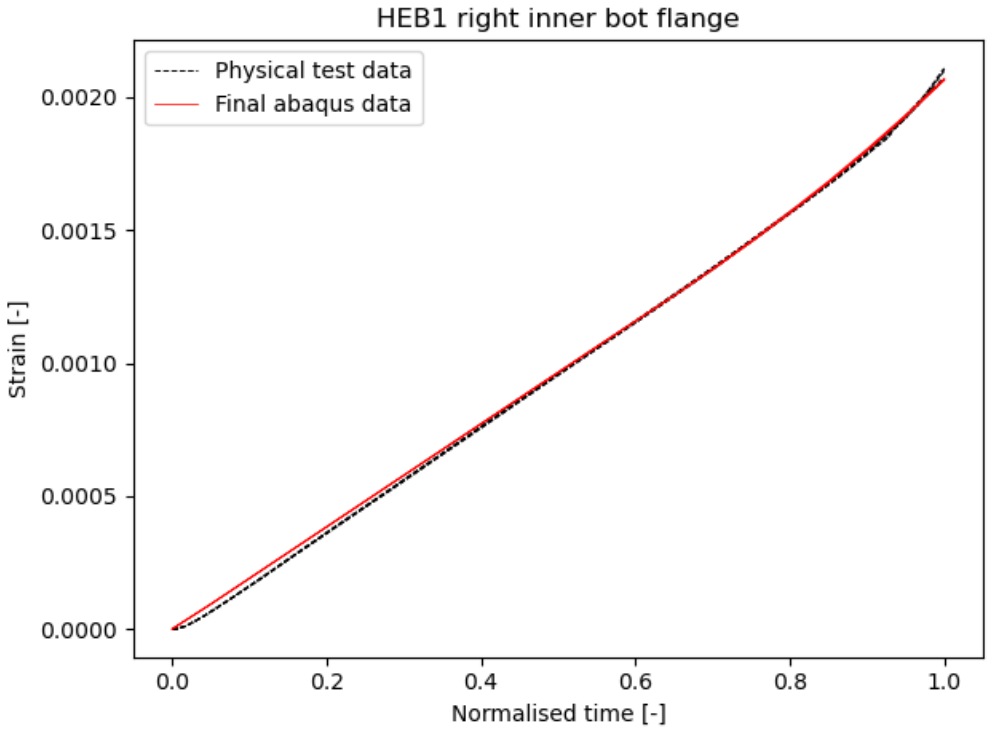


Figure 4.22: Final strain comparison for HEB1 right inner bot flange

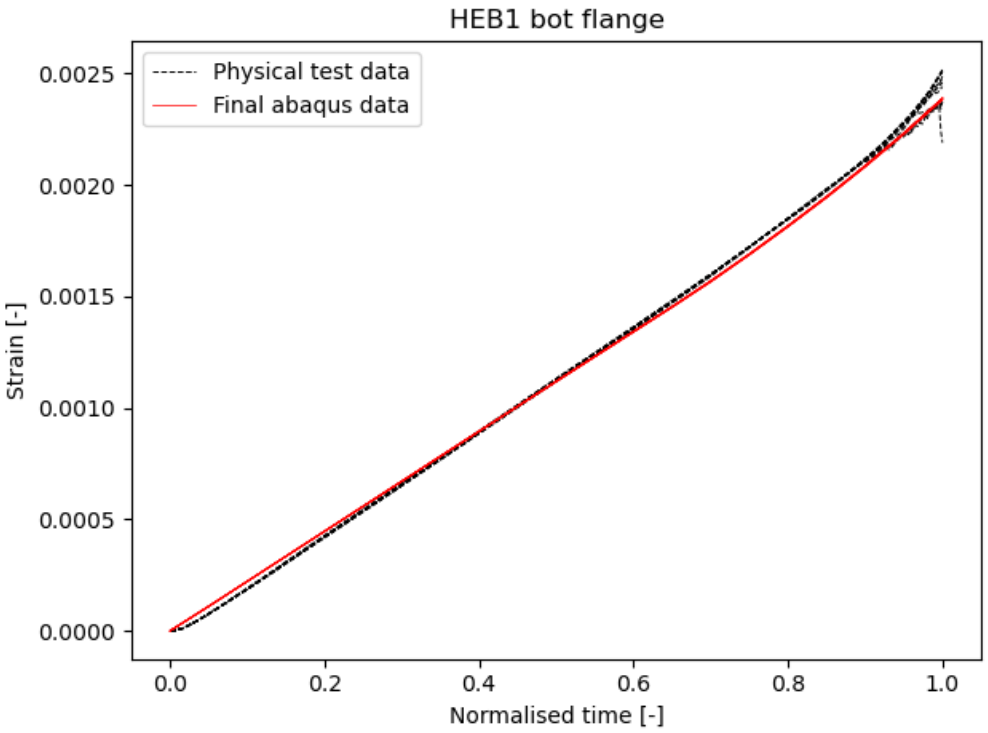


Figure 4.23: Final strain comparison for HEB1 bot flange

5
Discussion

In this chapter the choices and observations made in preparing, testing and modelling the beam are discussed as well as how they might have affected the results. In addition, the resulting residual stress distributions are discussed and analysed.

5.1 Test preparations and execution

In preparing the beams for testing, several actions and decisions may have impacted the final results obtained during testing. From the level of rust removal, thickness variations, to the cleanliness of the fibres and the tools used. One of the main concerns during preparations was how to not impact the level and distribution of residual stresses in the beams before the tests were performed. Therefore, some criteria were set up beforehand: the use of angle grinders with a grinding wheel was not to be used and final sanding was to be done by hand.

In the end, time constraints forced the abandonment of those criteria in order to move on to fibre preparations due to the long hardening times of the two-part glue used to attach the fibres to the beams. However based on the strain data obtained from the testing it is hard to say if the rushed cleaning process on four of the beams caused in any way a shift in results compared to the two first beams that were cleaned (HEA3 and HEB3). A far greater impact could be argued was the overall dimensional differences in the beams regarding different flange thicknesses and flange shape of each beam.

The beams delivered to Chalmers structural lab were all delivered at various stages of corrosion. The main impact was that the cleaning of the HEA beams that were delivered took far longer than anticipated as well as the extra dust generated during the process contaminated a large area of the lab. Furthermore, local occurrences of pit corrosion on the outside of the flanges forced the need to remove more material than needed to create a flat and clean surface to attach the optical fibres. In Figure 5.1 pit corrosion on both the flanges as well as some areas of the web that are impacted are shown.

Due to the complexity of modelling the effects from the preparations of the beams, the potential loss or redistribution of residual stresses caused by it has not been measured. During rust removal the use of power tools was needed to complete the process in time to perform the test. This caused in certain situations heating of local

Figure 5.1: Pit corrosion located on the flanges of one of the HEB beams.

areas as the machines gripped the surfaces, dug down and in other ways generated heat that could have altered the residual stress. Another problem created by the way the beams were prepared was the changing thickness of the flanges and in smaller part the thickness of the web in the centre of the beam due to material removal. Described further in the next section discussing the creation of the model, the material removal caused the need to measure the new dimensions to be used in the model.

As previously mentioned, three of the beams tested (HEA2, HEA3, and HEB3) did buckle to some extent during testing, forcing the tests to be terminated early due to risk of damage to equipment or personnel at or near the test rig. This inherent instability was known beforehand but estimated through calculations and FE-simulations to occur later than they appeared during the tests. It would be easy to assume that if the chains that held the bracket for lateral stability were not tensioned as they were after the first test, all subsequently beams would have followed the same trajectory. For the second test the beam did not buckle, it was in fact during the third test and fourth test that the beam started to show signs of lateral-torsional-buckling again and the tests were forced to terminate early.

Several reasons can be given as to why buckling may have happened. Lower accuracy during the installation of the first tested beams in the test rig is one and the asymmetry of the beams is another potential reason. For all the tests where the beam did not buckle it was observed that the bracket were taking up some amount of lateral force as it was a notable tension in the chains for some of the beams. One exception was the last HEB tested (HEB1) that did not seem to move or affect the tension of the chains connected to the bracket at all, instead it deformed straight

along its vertical axis until the maximum deflection was reached. It is interesting to note that during all the other tests, regardless if the beams buckled or the forces were taken up by the bracket, all beams tended to move or buckle in the same direction of its axis. It is speculated that this happened because of the impact of one or several of the following factors: the small tapper of the HEA beams on the flanges being different in size between each side, the angle of attack during the cleaning process, difference in weight of the chains connected the bracket, small imperfections of the load cell forcing the beams to the right. However the most plausible explanations is just that all the beams were slightly off the centre during the tests. To remove the issue of LT-buckling more robust lateral supports could have been made which could restrain the bottom flange as well.

During the tests it was noticed that after an area had yielded, the strain data in that area were not behaving according to expectations for steel during loading. This could potentially have some thing to do with the glue used, that it lost adhesion to the beam when the steel plastically deformed or that it was too stiff, resulting in damaged fibres. There was however no visual sign of the glue losing its adhesion to the beam during or after the tests. Regardless, the result of it was that the full data could not be used and instead had to be cut. Further experiments regarding different glues or other methods of attaching the fibres would have been necessary in order to evaluate if the attachment method, described in Section 3.1, was adequate for the task.

The deflections at the supports measured during the experiment might not be too accurate since the LVDT sensors were placed at the rollers. For the support which had the fastened steel cylinder the accuracy of the deflection was only dependant on how centred the placement of the sensor was, but the other support's steel cylinder were allowed to move which introduced another error source. The movement of the cylinder would increase the deflection measured since the point of the LVDT sensor would slide down the side of the cylinder, giving inaccurate results. An illustration of the issue is shown in Figure 5.2. However, the error does not have a large impact on the final result as the deflection at the supports were 60-100 times smaller than the deflection at the load positions. In addition, the error's influence is reduced further by the fact that only the average support deflection is used for calculating the deflection used in Abaqus.

5.2 Determination of material properties from tensile tests

Uncertainty in determining the cross sectional area of the dog bones may have affected the calculated stress and in turn the Young's modulus. The measurements were made with digital calipers which had a precision of ± 0.01 mm but due to the uneven surfaces of the cross-section the measurements had greater error margins. To improve the accuracy, more measurements of the specimens could have been taken or another more precise way of measuring them, like 3D-scanning, could have been

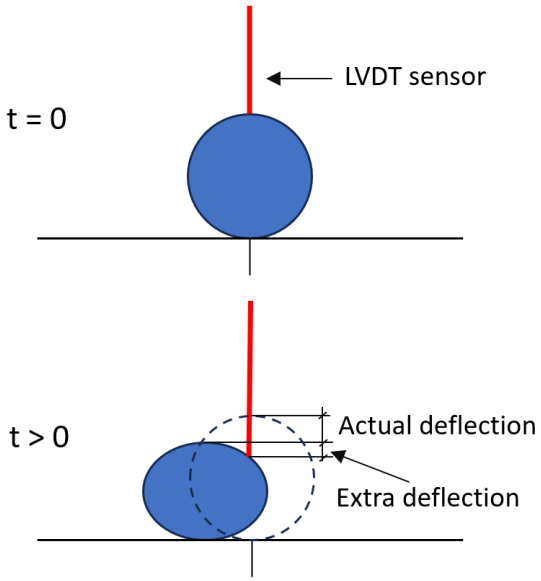


Figure 5.2: Principal sketch of error in measuring the deflection at the roller support.

carried out.

When determining the Young's modulus the result also depended greatly on what two points were chosen as reference points for the calculations. The Young's modulus for two different choices could differ by as much as tens of gigapascal. For the HEB profile the difference between the flange and webs elastic moduli were measured to be 40 GPa, a difference that seems too large to be attributed to variability of the material. The cause of this could be that the specimens were not properly gripped by the tensile test machine, as the flange parts of the HEB profile were so thick that they had to be machined down around half a millimetre to fit in the wise. Other reasons could be down to the accuracy of the cross-sectional measurements or attachment of the exstensometer.

5.3 Numerical model

In this section the process from the first model to the one that was used in the final calculations and the choices that were done along the way are discussed and examined. The first iteration of the model consisted of one solid part, a single sketch of the cross-section extruded to cover the length between the supports then assigned a simple material model, see Figure 5.3. Partitioned to apply a displacement controlled boundary condition, and a single time step was created. Coding wise the model was simple, only consisting of a few lines of code allowing it to be easily changed and understood. However it had its drawbacks, the model was far too slow to be practical, created out of only solid elements with a seed size of 5mm it consisted of 196000 elements.

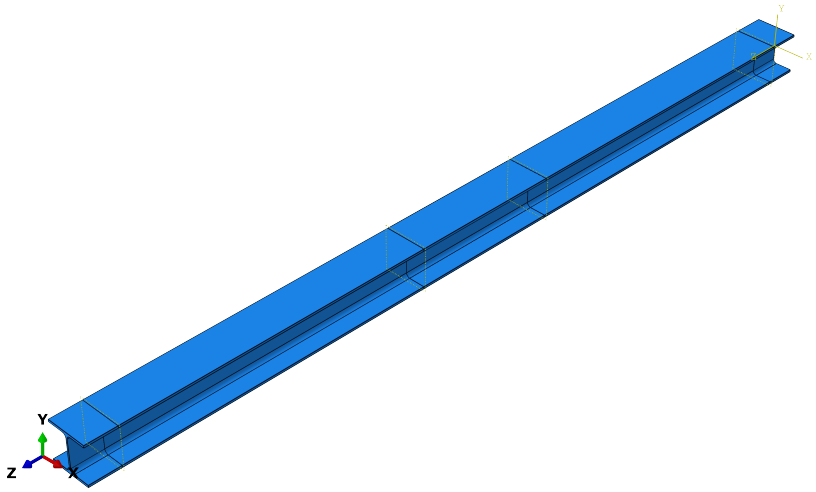


Figure 5.3: Original model

To alleviate the load of the calculation created by the massive amount of elements in the first model some changes were needed. The first step was to make use of symmetry and halving the beam in two. From there it was decided to take the element reduction even further by implementing the use of shell elements. Applying residual stress were done per element and to reduce computing times the use of sets was possible to apply residual stress to several elements at the same time. As the stress for a certain position in the cross-section will be the same throughout the length of the beam, using solid hexahedral mesh elements allows the script that is based upon element numbering to select the elements with the same cross-section coordinates in one go and then assign a pre-calculated residual stress value to that set. If the solid section had been made with a tetrahedral mesh the mesh generation changes to a a more unstructured pattern and the script would break.

Regarding the mesh size for the central section, i.e the solid meshed part, the amount of residual stress information that can be applied and extracted from the model is based on the number of elements. The choice was made so that at least two elements were needed through the thickness of the web as well as two elements throughout the flanges. In doing this it was possible the extract and compare data from the model with the data collected from the physical tests without having a linear relationship of the strain and stress between the two sides of the flange and web respectively.

For the shell part of the beam, focus was on matching the seed size to that of the solid and the other end was then manipulated to generate the least amount of elements. In this case it would be easy to say that the size of the first element should have a size of half the flange width. However, this generates more elements compared to the starting seed with a quarter of the flange width. For the HEB model a seed size of 100mm generated 9817 elements for the shell part whereas using a seed size of 50mm generated 7057 shell elements, in Figure 5.4 and 5.5 the pattern of mesh generation for both scenarios is visualised.

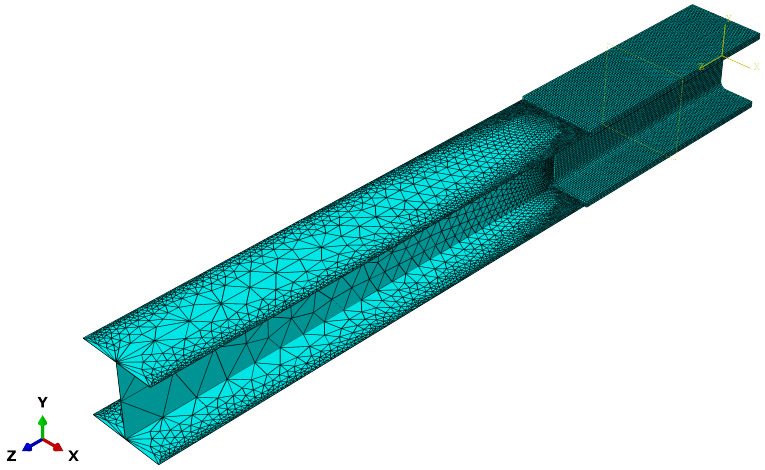


Figure 5.4: Mesh generation HEB with edge seed 100mm, 9817 shell elements

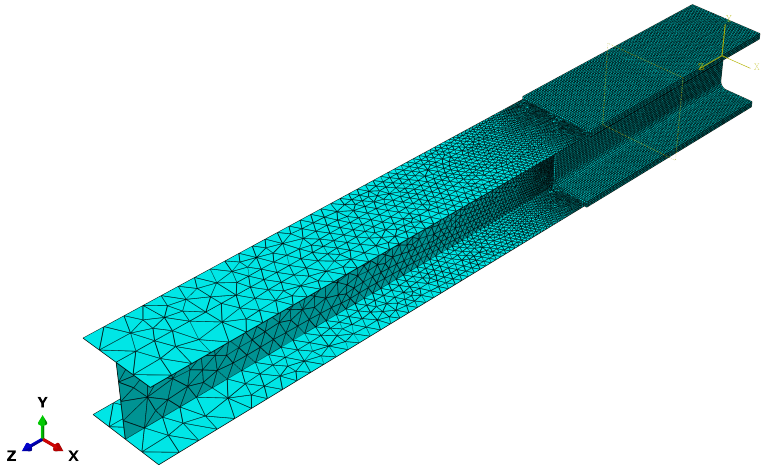


Figure 5.5: Mesh generation HEB with edge seed 50mm, 7057 shell elements

This happens as the edge with the larger starting seed creates such a high constraint for large elements, combining that with rules for the angles between each side of the element results in that too large elements is forced to be created and leaving a lot of empty space in between. Without breaking the rules of the angles the mesh will simply not create one or two elements to fill the void, instead it will slowly approach the edge of the part with smaller and smaller elements.

The seed size was optimised for the HEB model and was not changed for the HEA model. This resulted in that the HEA model had less elements than the HEB model which could have impacted the HEA results negatively. However, it still has two elements through the thickness of the web and flanges which should still yield good results.

Running the established model is a balancing act between the size of the mesh elements and the maximum size of each time step during simulation. If the time-steps

are too large, in the best case data will be lost or buckling will appear. In the worst case the simulation will terminate prematurely due to not finding convergence in the iterations. Using a seed of 5 millimetre for the solid section, the maximum time-step that was able to run without crashing was a step of 0.05 seconds. However, a lot of individual steps were needed to be recalculated resulting in the overall time of the run increasing. A simplified explanation of this process is as follows: if the simulation could not calculate the step it would first try to recalculate it for a set number of times to try and find convergence before in would decrease the size of the time step and repeat the previously step. This process continues until the minimum step size is reached, where it either finds a solution or terminates the simulation.

In the end it was decided to run the optimisation routine with the step size of 0.05 to have a shorter run time for each simulation. If one would want a more accurate solution the step size could be decreased, especially in the expected plastic region. The same could be said for the mesh size, that for a more accurate result the mesh size could be reduced at cost of the run time.

As for the dimensions of the beam in the model, the average measured thickness of the web and the combined average of the flanges was used to model the respective parts. The observed change of thickness, specifically for the flanges, was ignored in the model as the equations used to describe the residual stress distribution only are valid for double symmetric cross sections with uniform thickness. In assuming an ideal cross-section the accuracy of the results will differ from a beam with true dimensions where both deformation for the same load as well as residual stress distribution are impacted.

Better results could have been produced if a more complex model of the residual stress, that would be able to describe the residual stress distribution in non-ideal beams, was used. In addition, even for an ideal beam the assumption of a parabolic relation might not be correct. As mentioned in Section 2.3 there are several models for the residual stress.

In the Abaqus model the deflection applied at the supports was set to be linear up to the maximum deflection calculated. In reality the hydraulic jack had a linear deflection while the load positions actual deflection were dependant on the supports deflections as well. This results in a close to linear but nonetheless non-linear deflection over time, shown in the Figure 4.15. For a better result the time-deflection measured during the physical test could have been used in Abaqus, to ensure that the deflection in the model changes in the same way as in the tests.

5.4 Post processing

In processing the data from the simulations run in Abaqus and the data collected during testing as presented in Chapter 3 certain simplification or truncation of the data collected were necessary in order to easily be able to review and work with it. Looking at a single strand, for every 5.2 mm along its length a strain measurement

was collected at a sampling rate of 12.5 Hz, which for a 10 minute test means that a single 200 mm long strand will collect over 288 thousand data points. Extending this calculation to all the strands for all the beams and the amount of data to be handled would be far too much.

The first step in reducing the amount of data was done by selecting the largest gauge length available in the Odisi software of 5.2 mm. After that the data was reduced in time by taking the average of every 12th time frame. The choice of taking every 12th frame was made since the sample 12.5 Hz and by averaging every 12th frame it would the result in roughly one time step per second.

Due to the uncertainty if data could be collected for every point along the strand, be it caused from bad connection to the surface during gluing, release during yielding or in any other way stopped capturing strain, it was decided to take the median strain value instead of the mean strain value from each strand. Discrepancies in the data had therefor less impact on the final results since the mean value is more sensitive to outliers a than the median value.

As clearly visualised in Figure 3.14 there was a need to match the strain data obtained from testing with the one extracted from Abaqus. Each data set consists of two matrices, were every row corresponds to a spatial position on along the cross-section and every column corresponds to time-step. The time here is normalised for each beam so the first and final column in both the data from Abaqus and the tests are the same, the only difference is the amount of steps from start to end. In the script the data is processed using interpolation to create two matrices that are of the same size. With interpolation it is assumed that between two existing positions there is a linear relationship, increasing time steps would then lead to that the strain behaves in a linear fashion which it does as long as it is in the elastic zone. However, when strain moves into the plastic zone and the material starts to yield, the relationship changes which results in that the interpolated data will create an approximation error between the real measured points.

In the same way interpolating spatial positions over the cross-section will assume a behaviour for sections of the beam that does not mirror reality. The plastic redistribution that occurs during deformation of a member is not strictly linear through the cross-section. Taking this into consideration, it was of interest to interpolate as short distances as possible in both time and space when modifying the size of the data. Hence, the dimensions were modified to match the spatial and time data to the matrix size with the least amount of data for each category. This resulted in, as previously mentioned, that the spatial data collected from Abaqus is reduced into points matching the position of the strands on the beams and the data through time collected from the fibres is reduced to match the time for each step that the Abaqus analysis created.

5.5 Optimisation procedure

There is no guarantee that the global minimum has been found since the Nelder-Mead method finds local minima as well. To be more confident in the results, more initial simplex would have to be run through the routine.

A major drawback of the chosen optimisation algorithm for this thesis is, as discussed earlier in this chapter, the fact that the basic Nedler-Mead simplex lacks a 'kickback' feature. The 'kickback' is a part of an algorithm where it once in a while it takes a step back and makes a wild guess outside of the now established simplex to see if it has worked itself into a local minima or in any other way forced itself into a locked position. The process of the kick is not more different than the *reflect* step discussed in Chapter 2 but the change in values or put in other words the distance it travels, is far greater than that original step.

Comparing all of the data points in time and space to determine how well the estimation describes reality is somewhat redundant since any estimation of the residual stress, results in the same behaviour in the elastic zone. The method should work equally good by only checking the strains for one time frame in the plastic region. If the last time frame would be compared, there would be no need for temporal interpolation which would slightly speed up the error calculation.

No boundaries for the residual stress coefficients were given to the Nelder-Mead algorithm which resulted in that the algorithm could produce coefficients of unreasonable magnitude or sign. Boundaries could be provided for c and d to the algorithm but as a and b are calculated from equilibrium equations it is not possible to limit them directly. For c it is easy to determine good boundaries since it is equal to the stress in the middle of the web. A lower limit for d is also easily set as 0 to constrict the curve to having a positive gradient but setting a upper boundary is harder since what value is reasonable for d is dependent what c is. A possible solution to the difficulty of selecting the boundaries is to choose a and c as input variables instead of c and d.

5.6 Results

The results from the procedure are reasonable for the HEB beams but not for the HEA beams. When comparing the residual stress distribution for the HEB beams with residual stress measurements made by Young (1975) for a similar cross section, it can be seen that the magnitudes of the residual stresses are similar. Furthermore, comparing the mentioned models against the obtained mean distribution of the HEB, as seen in Figure 5.6, also shows that the results are of reasonable shape and magnitude. Therefore, the residual stress distribution obtained from the method for the HEB beams was deemed reasonable.

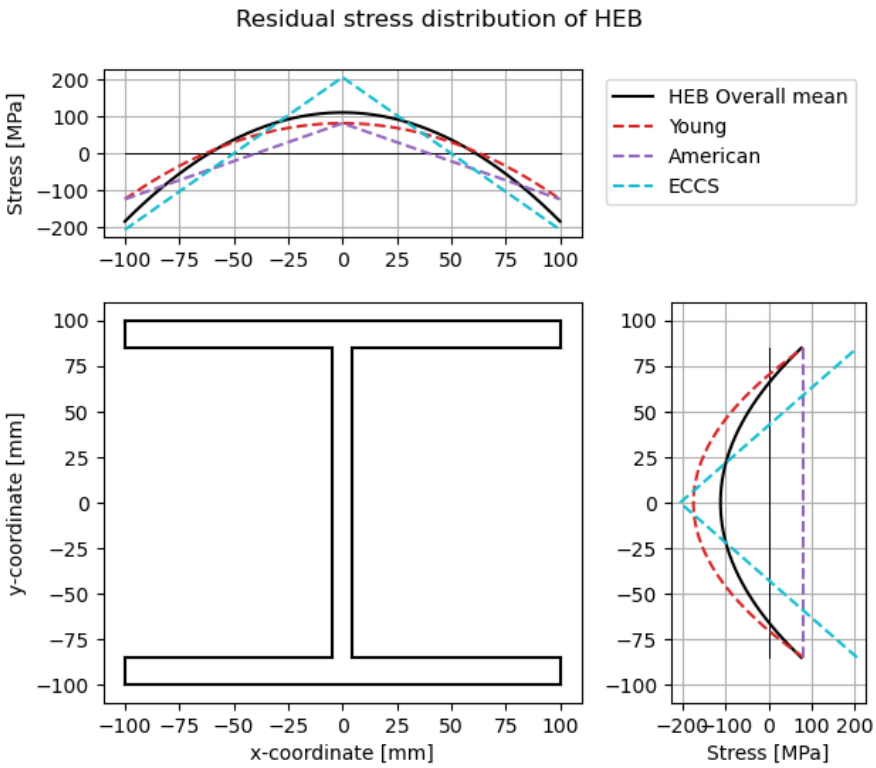


Figure 5.6: Resulting model for the HEB compared to existing models.

As seen in Figure 4.13 the residual stress distribution in the flange for HEA2 and HEA3 is of the opposite expected shape. Furthermore, the magnitudes of the residual stresses in both web and flange of the HEA beams are unreasonably large, even over the yield stress in certain parts.

There are multiple reasons to why these errors could have arisen. The simplification of modelling the beam as an ideal beam with the average dimensions is probably the largest error source since it affects the behaviour of the beam. The Nelder-Mead method itself is also a large error source since it is highly dependent on a reasonable initial guess. However, choosing a large initial simplex, as done in this project, should reduce the likelihood of the algorithm getting stuck on a local minima.

Since most of the data in the plastic region had to be cut off due to unusable data, there are less of the valuable data for the comparison. This makes it tougher for the algorithm to find the true residual stress distribution as the data is mostly linear. For a more reliable result, more data in the plastic region has to be used. In addition, the small amount of plastic data makes the method more sensitive to the determination of the material properties. If the yield stress used in the model is higher than in reality, the strain can become linear for the whole studied duration. This might be the case for some of the beams as seen in Figures 4.16-4.23.

Running Abaqus with the final residual stress coefficients for the full load duration

of the physical tests, instead of cutting the test short, shows that the behaviour of the model loses its accuracy in the long run. When comparing the physical test data from the web, which had reasonable data throughout the test, with the final Abaqus model, it can be seen that the model estimates the behaviour well in the beginning but it gets worse after the time of the cut used for optimisation. In Figure 5.7, left web data for HEB1 are compared and the model matches the physical behaviour quite well up until the cut off time of 480 s, but afterwards the model underestimates the strain. Important to note is that the unloading is included in the test data while the Abaqus data were only for the time where the deflection was still increasing. In Appendix I, figures of the comparisons made for the rest of the locations of the HEB1 beam are shown.

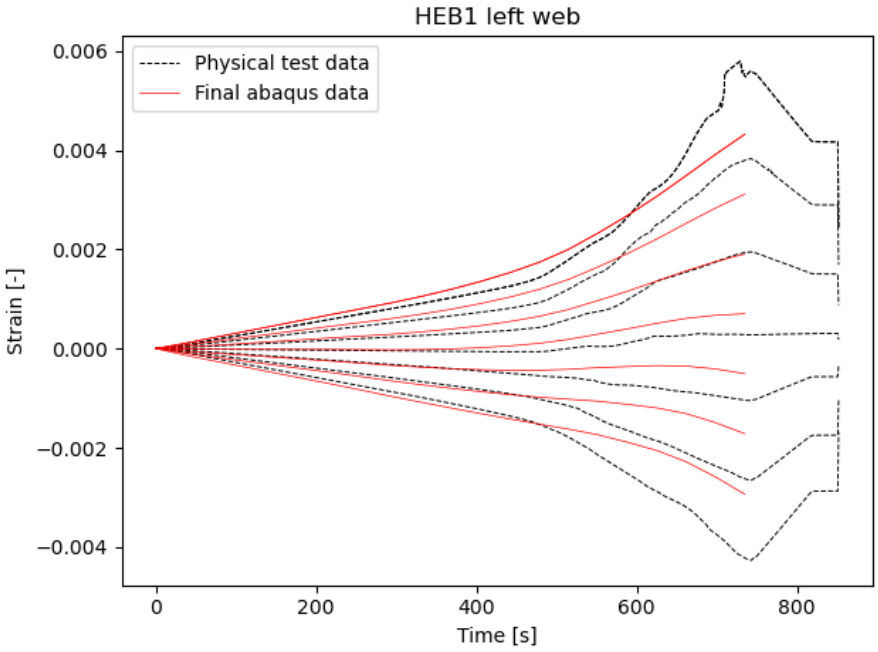


Figure 5.7: HEB1 left web for full test duration with Abaqus data based on the final residual stress coefficients. Abaqus was run up until the point where the physical test was unloaded.

When comparing the strains for the web at certain times it could also be seen that the physical beam does not fully follow Bernoulli's theory of plane section. In Figure 5.8 the strains from the HEB1 beam's left web is plotted for eight time frames where the two last frames do not follow a straight line along the height. In the finite element modelling, Bernoulli's theory is applied which could be an additional reason to why the Abaqus strain curves differ from the measured curves after extensive loading.

Interesting to note from the results in Table 4.5 and Table 4.6 is how the error for each beam is very close to each simulation even if the initial simplex or final coefficients differ. Taking a closer look at the coefficients through time does not yield any further understanding to this. However it would not be unreasonable that based on

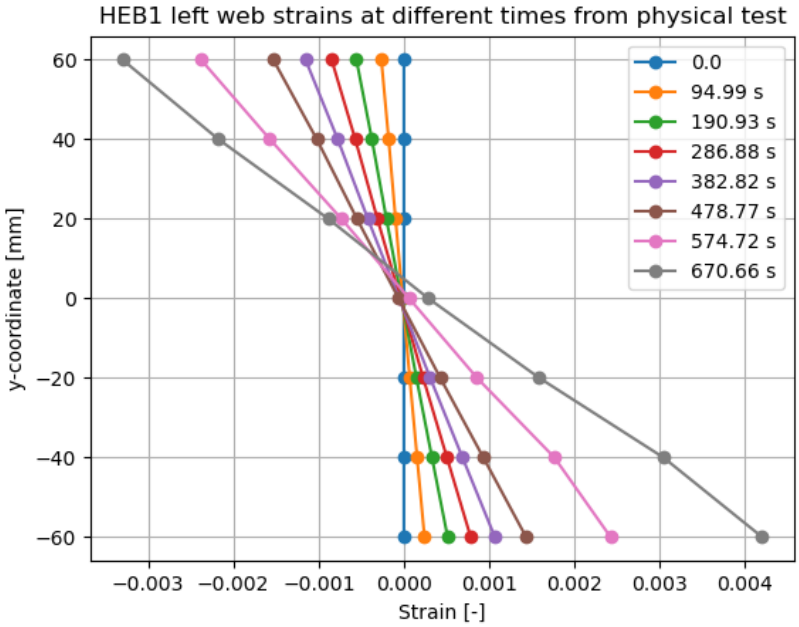


Figure 5.8: HEB1 strains over the height for certain time frames.

the fragmented data collected during testing that a more precise solution is impossible. To test this hypothesis a possible method would be to run a new simulation starting with a simplex scaled just around the final estimation created and see if it can find a new solution to the problem. It was however not tested during this thesis.

Studying the figures presented in Section 4.4 it seems that for most of the simulations the algorithm finds a near value of the final error rather early. The error shifts slightly even as the coefficients vary with relative large magnitude, as to why the error still almost stays the same is hard to tell. It could be the specifics of the algorithm used and the "size" of each step in the evaluation is so small that even if things change the overall area is the same. Most likely however is that the the equations describing the residual stress distribution forces the algorithm to deviate to far from the given value of the coefficients.

6
Conclusion

To conclude this project final thoughts and reflections on the work conducted and the results produced will be given, as well as recommendations on further work in the field of study.

6.1 Thoughts and reflections

The method developed during this project is shown to work for ideal data as demonstrated with the numerically generated data in Section 3.5. As discussed in Chapter 5, the importance of the quality of strain data fed to the script is vital to obtain accurate results. The tests performed during this project did unfortunately not yield data of the quality that was sought after. Performing physical tests is an important tool in testing and validating new hypotheses based on literature. However, the accuracy and quality of these test is highly dependent on the quality of the test specimens used, including geometric imperfections and corrosion as well as the knowledge and craftsmanship of the ones conducting the work.

For the HEB-profiles tested, a reasonable estimation of the residual stress distribution was produced by the method developed. This is thought to be mainly due to the amount of data captured after yielding as discussed in Section 5.6. The same cannot be said for the HEA-profiles, where the residual stress coefficients were far from the ones suggested by previous researchers, e.g. Skiadopoulos (Skiadopoulos et al., 2023). The difference in elastic moduli between the material properties is also of great importance to the behaviour of our model, as the stress generated for a certain displacement is quite different for the lowest modulus of around $E = 180GPa$ in the web of the HEB or the highest at around $E = 220GPa$ in the flange of the HEB. If better material data were to be gathered from the beams, either from a larger number of specimens or from a more accurate testing procedure, the material model would far better represent the real behaviour of the beam during loading.

Preparations for testing were the process that included by far the most variables to try and control during this thesis. Development of a more repeatable process would be necessary if further study of residual stress distribution were to be conducted, both for use in the method developed as well as in other endeavours regarding optical fibres and steel specimens. The adhesive used for attaching the fibres seemed to work but the quality of the surfaces they were attached to varied widely, possibly

impacting the results.

The method developed compared to existing methods has the advantage that it is based on many measuring points. Having a large amount of measuring points allows the removal of outliers in post processing of the data which results in that the data used for the determination of the residual stress coefficients is clean. Other destructive methods instead have to rely on that the length or area, on which the strain gauge is placed, is free from discrepancies.

6.2 Further work

The results from this thesis indicates that continuing development of residual stress distribution models is still needed to accurately capture the behaviour of real world specimen. For the method developed, it is clear that increasing the accuracy of the FE-model is needed to better capture the behaviour of the beam. This include the varying thickness of the flanges and the web, both throughout the cross-section as well as along the length of the beam.

Implementing different optimisation algorithms is also one area that would be needed to more confidently say if the method developed is applicable or not at large. Using an algorithm that has incorporated a kickback feature in an effort to reduce the possibility of getting stuck on a local minima of the objective function would be a natural first step. Testing this method on other hot-rolled members or even welded beams, with a different assumption of the residual stress distribution for the latter, would also be important in an effort to asses the viability and robustness of the method developed.

References

Abambres, M., & Quach, W.-M. (2016). Residual stresses in steel members: A review of available analytical expressions [Publisher: Emerald Group Publishing Limited]. *International Journal of Structural Integrity*, *7*(1), 70–94. https://doi.org/10.1108/IJSI-12-2014-0070

Alpsten, G. A. (1968). *Thermal residual stresses in hot-rolled steel members, December 1968* (tech. rep. No. 337.3). Fritz engineering laboratory. Leigh university. http://preserve.lehigh.edu/engr-civil-environmental-fritz-lab-reports/329

Alpsten, G. A. (1975). Residual stresses, yield stress, and column strength of hot-rolled and roller-straightened steel shapes [Publisher: IABSE]. https://doi.org/10.5169/SEALS-19799

Fitzpatrick, e. M. E., & Lodini, A. (2003). Analysis of Residual Stress by Diffraction Using Neutron and Synchrotron Radiation. *Measurement Science and Technology*, *14*(9), 1739. https://doi.org/10.1088/0957-0233/14/9/703

Galambos, T. V., & Ketter, R. L. (1959). Columns Under Combined Bending and Thrust [Publisher: American Society of Civil Engineers]. *Journal of the Engineering Mechanics Division*, *85*(2), 1–30. https://doi.org/10.1061/JMCEA3.0000084

Ghosh, S., Rana, V. P. S., Kain, V., Mittal, V., & Baveja, S. K. (2011). Role of residual stresses induced by industrial fabrication on stress corrosion cracking susceptibility of austenitic stainless steel. *Materials & Design*, *32*(7), 3823–3831. https://doi.org/10.1016/j.matdes.2011.03.012

Huber, A. W., & Beedle, L. S. (1953). Residual stress and the compressive strength of steel . Welding Journal, 33 (12), p. 589-s, (December 1954), Reprint No. 96 (54-3). *Welding Journal*, *33 Reprint No. 96*.

James, M. N., Hughes, D. J., Chen, Z., Lombard, H., Hattingh, D. G., Asquith, D., Yates, J. R., & Webster, P. J. (2007). Residual stresses and fatigue performance. *Engineering Failure Analysis*, *14*(2), 384–395. https://doi.org/10.1016/j.engfailanal.2006.02.011

Jansson, A. (2024). *Identifying locally induced loads in hard rock tunnel linings by distributed optical fibre sensors* [Doctoral dissertation, Chalmers University of Technology]. Retrieved May 28, 2024, from https://research.chalmers.se/en/publication/540677

Kolda, T. G., Lewis, R. M., & Torczon, V. (2003). Optimization by Direct Search: New Perspectives on Some Classical and Modern Methods. *SIAM Review*, *45*(3), 385–482. https://doi.org/10.1137/S003614450242889

Kreger, S. T., Rahim, N. A. A., Garg, N., Klute, S. M., Metrey, D. R., Beaty, N., Jeans, J. W., & Gamber, R. (2016). Optical frequency domain reflectometry: Principles and applications in fiber optic sensing. *Fiber Optic Sensors and Applications XIII*, *9852*, 218–227. https://doi.org/10.1117/12.2229057

Lagarias, J. C., Reeds, J. A., Wright, M. H., & Wright, P. E. (1998). Convergence Properties of the Nelder–Mead Simplex Method in Low Dimensions. *SIAM Journal on Optimization*, *9*(1), 112–147. https://doi.org/10.1137/S1052623496303470

Liu, H., Han, D., Deng, L., & Cao, R. (2023). Analytical model for measurement of stresses using two-hole drilling. *International Journal of Pressure Vessels and Piping*, *206*, 105019. https://doi.org/10.1016/j.ijpvp.2023.105019

Nelder, J. A., & Mead, R. (1965). A Simplex Method for Function Minimization. *The Computer Journal*, *7*(4), 308–313. https://doi.org/10.1093/comjnl/7.4.308

Rossini, N. S., Dassisti, M., Benyounis, K. Y., & Olabi, A. G. (2012). Methods of measuring residual stresses in components. *Materials & Design*, *35*, 572–588. https://doi.org/10.1016/j.matdes.2011.08.022

Sabri, N., Aljunid, S. A., Salim, M. S., Ahmad, R. B., & Kamaruddin, R. (2013). Toward Optical Sensors: Review and Applications. *Journal of Physics: Conference Series*, *423*(1), 012064. https://doi.org/10.1088/1742-6596/423/1/012064

Skiadopoulos, A., de Castro e Sousa, A., & Lignos, D. G. (2023). Experiments and proposed model for residual stresses in hot-rolled wide flange shapes. *Journal of Constructional Steel Research*, *210*, 108069. https://doi.org/10.1016/j.jcsr.2023.108069

Tebedge, N., Alpsten, G., & Tall, L. (1972). *Residual-stress measurement by the sectioning method, presented at SESA Spring Meeting, May 1972 (73-5)* (tech. rep. No. 342). Fritz engineering laboratory. Leigh university. http://preserve.lehigh.edu/engr-civil-environmental-fritz-lab-reports/342

Wessing, S. (2019). Proper initialization is crucial for the Nelder–Mead simplex search. *Optimization Letters*, *13*(4), 847–856. https://doi.org/10.1007/s11590-018-1284-4

Withers, P., & Bhadeshia, H. (2001a). Residual stress. Part 1 – Measurement techniques [Publisher: Taylor & Francis]. *Materials Science and Technology*, *17*(4), 355–365. https://doi.org/10.1179/026708301101509980

Withers, P., & Bhadeshia, H. (2001b). Residual stress. Part 2 – Nature and origins [Publisher: Taylor & Francis]. *Materials Science and Technology*, *17*(4), 366–375. https://doi.org/10.1179/026708301101510087

Young, B. W. (1975). Residual stresses in hot rolled members [Publisher: IABSE]. https://doi.org/10.5169/SEALS-19798

A

Matlab script for extracting and formatting data from physical tests

Formatting of data

```
clc
% T is the raw text file
display('Loaded data is '+ string(T{21,2}) + string(T{21,3})+string(T{21,4})+' '+st
Beam_name = T{21,2}(7:end);
% Select one data location
top_flange = true
bot_flange = false
left_web = false
right_web = false

% Check if only one location is selected
input_check = top_flange+bot_flange+left_web+right_web;
if input_check > 1 || input_check == 0
    display('Wrong ammount of selected locations!')
    return
end

% Experiment time in seconds
clock_time = 801;
% Experiment time in number of data points with 12.5 Hz where n = 12
t_experiment = round(clock_time*12.5/n+1);
% Time increments as a percentage of total time to match abaqus
plottime = linspace(0,1,t_experiment);
% Extracts the data within the specified time frame. data is the raw data average e
data_values = data(1:t_experiment,1:size(str,2));


% Positional vector based on 5.2 mm distance between every gage in fiber
xpos = zeros(size(data_values,2),1);
for k = 1:size(data_values,2)
```

```
    xpos(k) = 0.08+0.0052*(k-1);
end

% For outer flanges
if bot_flange == true || top_flange == true

    output = zeros(10, t_experiment+1);

    % Position of fibers on the beam with 0 as center coordinate
    output(:,1) = [-90 -70 -50 -30 -10 10 30 50 70 90]';

    % Takes the median of the data from each strand in space to get an
    % average. Data taken between gage positions
    output(1,2:end) = median(data_values(:,415:451),2,"omitmissing")';
    output(2,2:end) = median(data_values(:,580:622),2,"omitmissing")';
    output(3,2:end) = median(data_values(:,749:793),2,"omitmissing")';
    output(4,2:end) = median(data_values(:,916:960),2,"omitmissing")';
    output(5,2:end) = median(data_values(:,1087:1128),2,"omitmissing")';
    output(6,2:end) = median(data_values(:,498:539),2,"omitmissing")';
    output(7,2:end) = median(data_values(:,667:709),2,"omitmissing")';
    output(8,2:end) = median(data_values(:,836:878),2,"omitmissing")';
    output(9,2:end) = median(data_values(:,1006:1049),2,"omitmissing")';
    output(10,2:end) = median(data_values(:,1176:1219),2,"omitmissing")';

    % Changes the data from microstrains to strains
    output(:,2:end) = output(:,2:end)/10^6;
    % Plots
    close all
    figure()
    plot(plottime,output(:,2:end))
    xlabel('Time[-]')
    ylabel('Strain [-]')
    legend(string(output(:,1)),'Location','northwest')

    if top_flange == true
        location = " outer top flange";
    else
        location = " outer bottom flange";
    end
    title(Beam_name+location)


    % Saves the data as txt files in the format wanted for the python
    % script
    if bot_flange == true
        % writematrix(output, 'Experimental_Bot_Flange_Strain.txt')
```

```
    else
        % writematrix(output, 'Experimental_Top_Flange_Strain.txt')
    end
end


% For Web and inner flanges
if left_web == true || right_web == true
    % Defining output file for web, 7 strands
    output_web = zeros(7, t_experiment+1);
    % Assigning coordinates of strands. 0 is in center of cross-section
    output_web(:,1) = [-60 -40 -20 0 20 40 60]';

    % Defining output file for lower flange, 4 strands
    output_lower_flange = zeros(4, t_experiment+1);
    % Assigning coordinates of strands. 0 is in center of cross-section
    if right_web == true
        output_lower_flange(:,1) = [90 70 50 30]';
    else
        output_lower_flange(:,1) = [-90 -70 -50 -30]';
    end

    % Defining output file for upper flange, 4 strands
    output_upper_flange = zeros(4, t_experiment+1);
    % Assigning coordinates of strands. 0 is in center of cross-section
    if right_web == true
        output_upper_flange(:,1) = [90 70 50 30]';
    else
        output_upper_flange(:,1) = [-90 -70 -50 -30]';
    end


    % Takes the median of the data from each strand to get an "average"
    % value while disregatding outliers
    % Change indexes in data values to match gage location index
    output_web(1,2:end) = median(data_values(:,776:823),2,"omitmissing")';
    output_web(2,2:end) = median(data_values(:,938:984),2,"omitmissing")';
    output_web(3,2:end) = median(data_values(:,1097:1143),2,"omitmissing")';
    output_web(4,2:end) = median(data_values(:,1261:1305),2,"omitmissing")';
    output_web(5,2:end) = median(data_values(:,862:907),2,"omitmissing")';
    output_web(6,2:end) = median(data_values(:,1024:1066),2,"omitmissing")';
    output_web(7,2:end) = median(data_values(:,1192:1230),2,"omitmissing")';

    output_lower_flange(1,2:end) = median(data_values(:,462:509),2,"omitmissing")';
    output_lower_flange(2,2:end) = median(data_values(:,607:658),2,"omitmissing")';
    output_lower_flange(3,2:end) = median(data_values(:,692:735),2,"omitmissing")';
```

```
output_lower_flange(4,2:end) = median(data_values(:,536:584),2,"omitmissing")';

output_upper_flange(1,2:end) = median(data_values(:,1433:1480),2,"omitmissing")
output_upper_flange(2,2:end) = median(data_values(:,1587:1639),2,"omitmissing")
output_upper_flange(3,2:end) = median(data_values(:,1510:1558),2,"omitmissing")
output_upper_flange(4,2:end) = median(data_values(:,1353:1395),2,"omitmissing")

% Changes the data from microstrains to strains
output_web(:,2:end) = output_web(:,2:end)/10^6;
output_lower_flange(:,2:end) = output_lower_flange(:,2:end)/10^6;
output_upper_flange(:,2:end) = output_upper_flange(:,2:end)/10^6;



% Saves the data as txt files in the format wanted for the python
% script
if right_web == true
    location = " right";
    writematrix(output_web, 'Experimental_Right_Web_Strain.txt')
    writematrix(output_upper_flange([4 3 2 1], :), 'Experimental_Right_Botside_
    writematrix(output_lower_flange([4 3 2 1], :), 'Experimental_Right_Topside_
else
    location = " left";
    writematrix(output_web, 'Experimental_Left_Web_Strain.txt')
    writematrix(output_upper_flange, 'Experimental_Left_Botside_Top_Flange_Stra
    writematrix(output_lower_flange, 'Experimental_Left_Topside_Bot_Flange_Stra
end

% Plots
close all
figure()
plot(plottime,output_web(:,2:end))
xlabel('Time[-]')
ylabel('Strain [-]')
legend(string(output_web(:,1)),'Location','northwest')
title(Beam_name+location+" web")

figure()
plot(plottime,output_lower_flange(:,2:end))
xlabel('Time[-]')
ylabel('Strain [-]')
legend(string(output_lower_flange(:,1)),'Location','best')
title(Beam_name+location+" inner bottom flange")

figure()
plot(plottime,output_upper_flange(:,2:end))
```

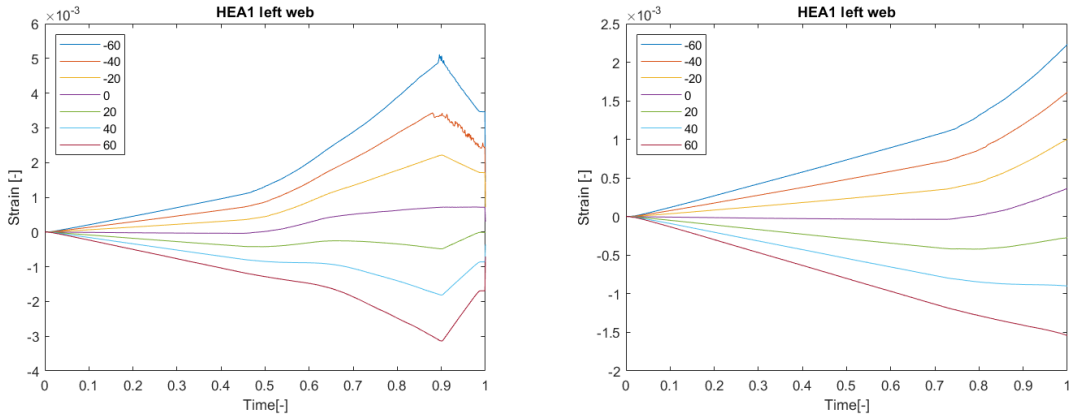
```
    xlabel('Time[-]')
    ylabel('Strain [-]')
    legend(string(output_upper_flange(:,1)),'Location','best')
    title(Beam_name+location+" inner top flange")


end


writematrix(plottime','Experimental_Time_Increments.txt')
```

B

Original and cut strain curves

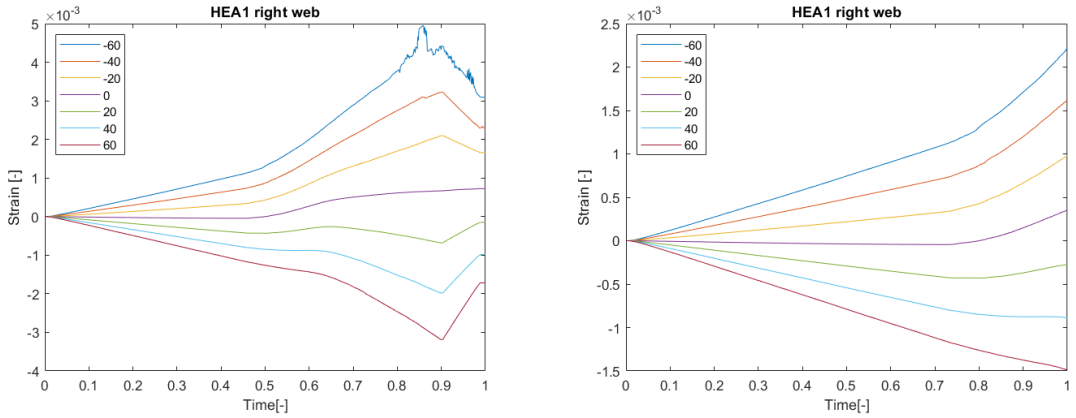
In the following pictures the full and cut strain data of each location on the six beams are shown. The numbers in the legends corresponds to the x- or y-coordinate of the strand depending on if the surface the strand is on is horizontal or vertical.

HEA1



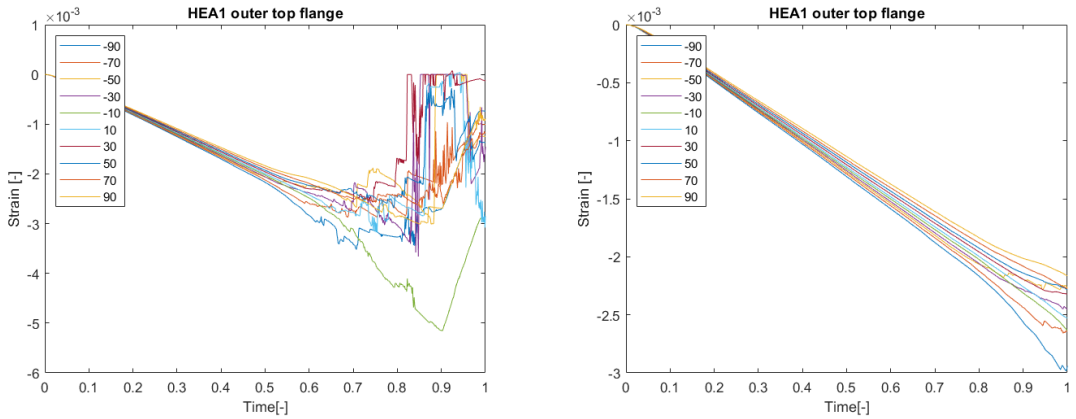
(a) Full data **(b)** Cut data

Figure B.1: HEA1 left web



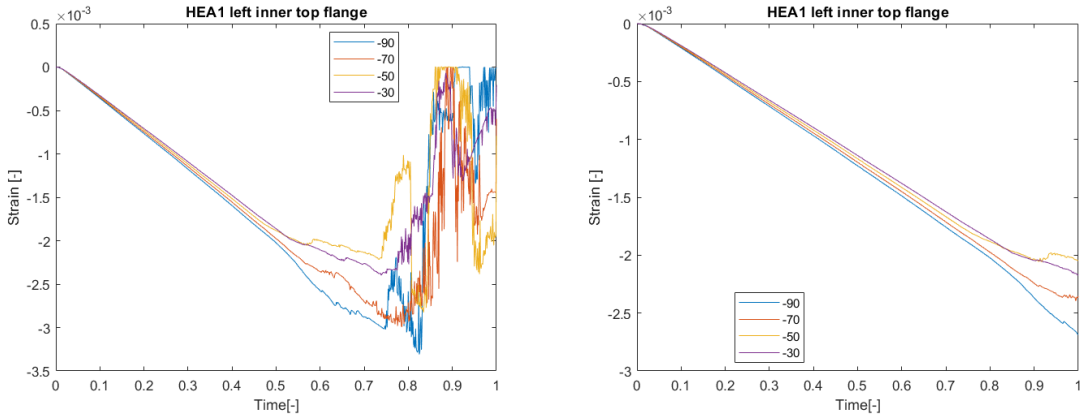
(a) Full data **(b)** Cut data

Figure B.2: HEA1 right web



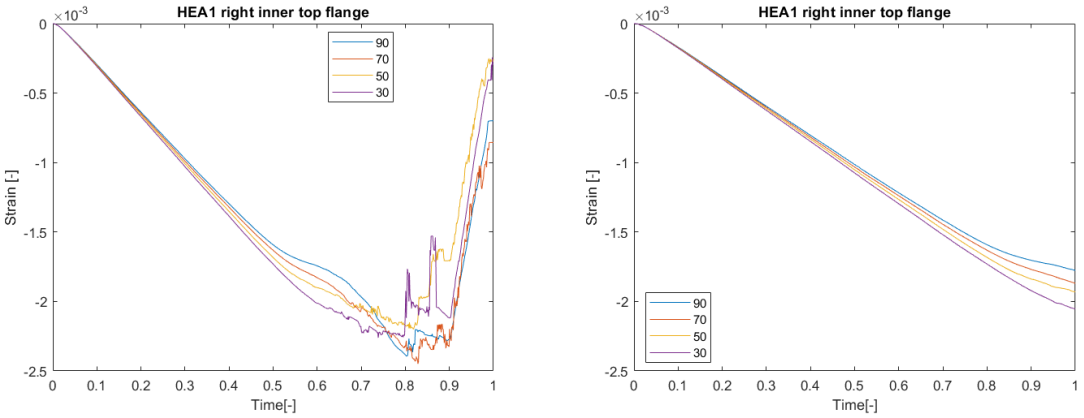
(a) Full data **(b)** Cut data

Figure B.3: HEA1 top flange



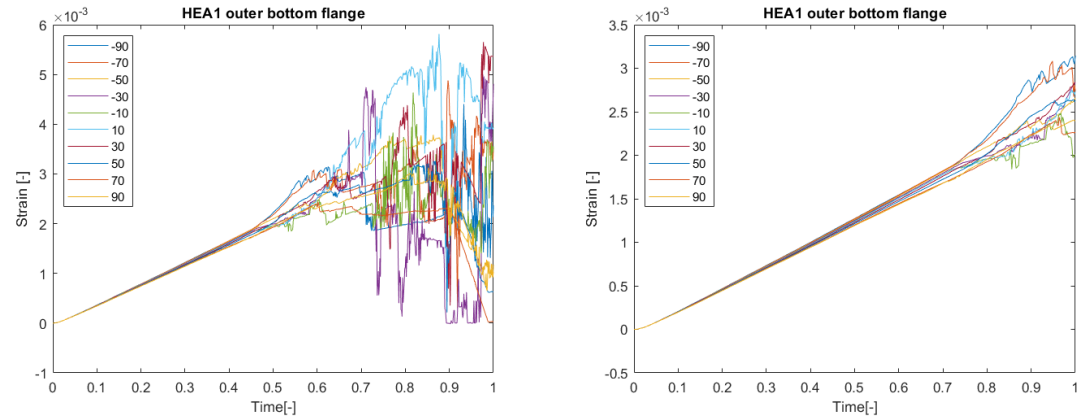
(a) Full data **(b)** Cut data

Figure B.4: HEA1 left inner top flange



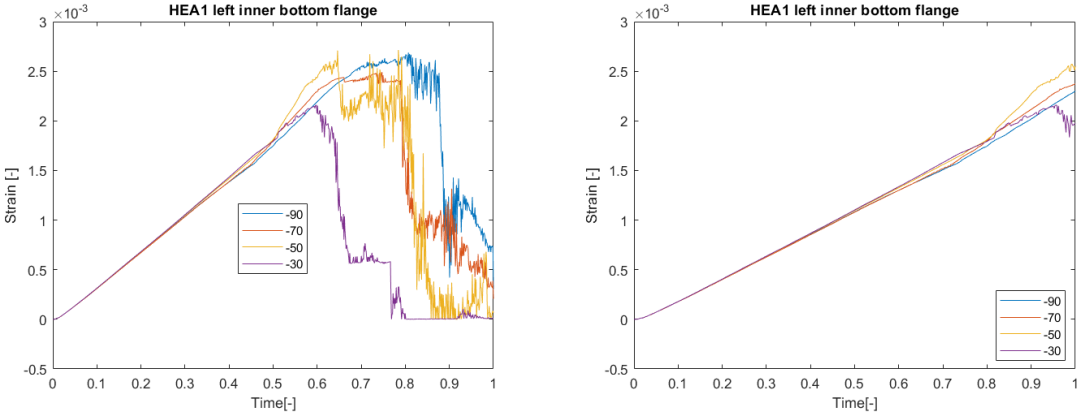
(a) Full data **(b)** Cut data

Figure B.5: HEA1 right inner top flange



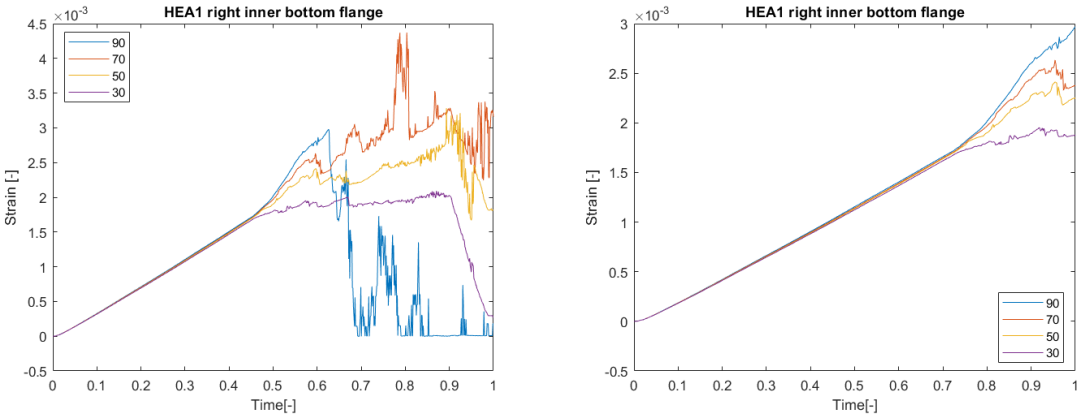
(a) Full data **(b)** Cut data

Figure B.6: HEA1 bottom flange



(a) Full data **(b)** Cut data

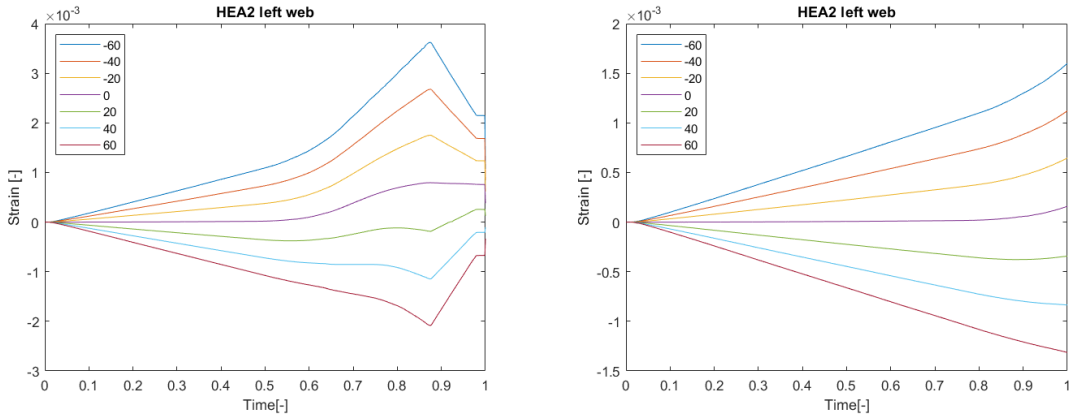
Figure B.7: HEA1 left inner bottom flange



(a) Full data **(b)** Cut data

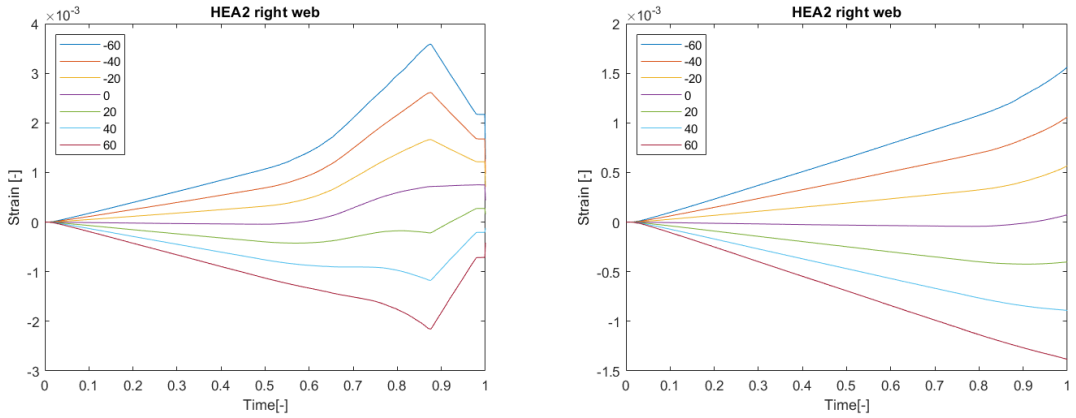
Figure B.8: HEA1 right inner top flange

HEA2



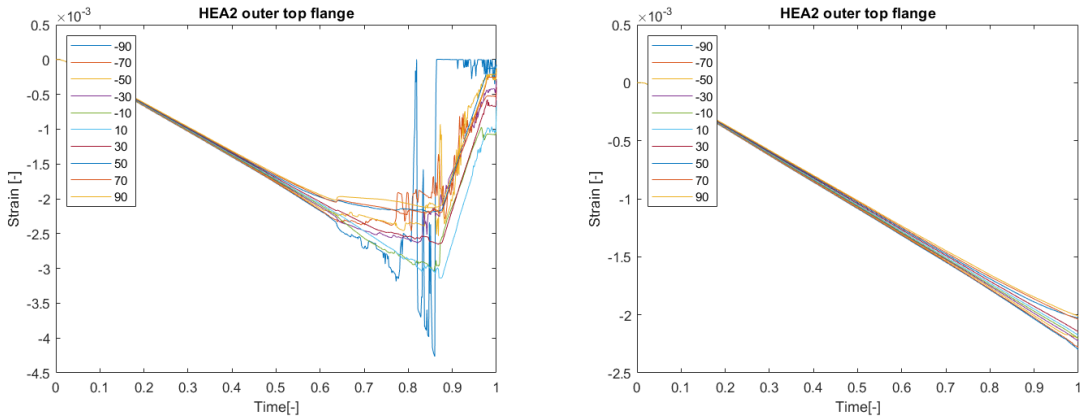
(a) Full data **(b)** Cut data

Figure B.9: HEA2 left web



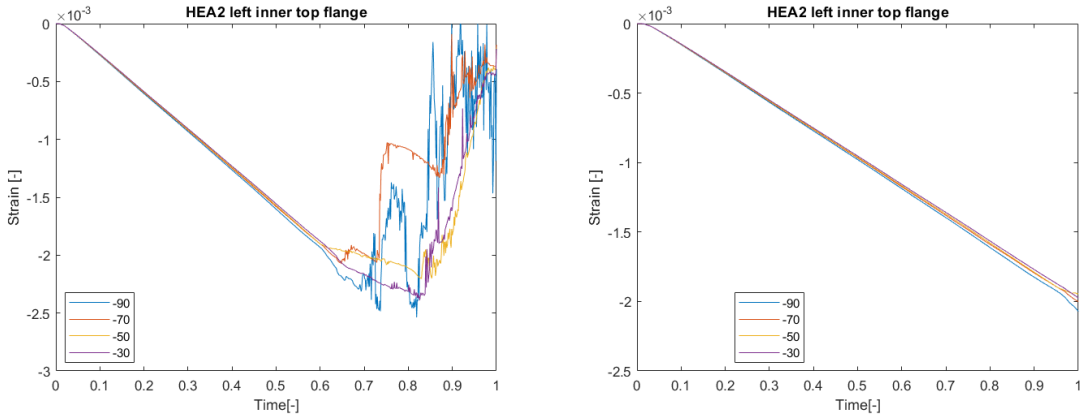
(a) Full data **(b)** Cut data

Figure B.10: HEA2 right web



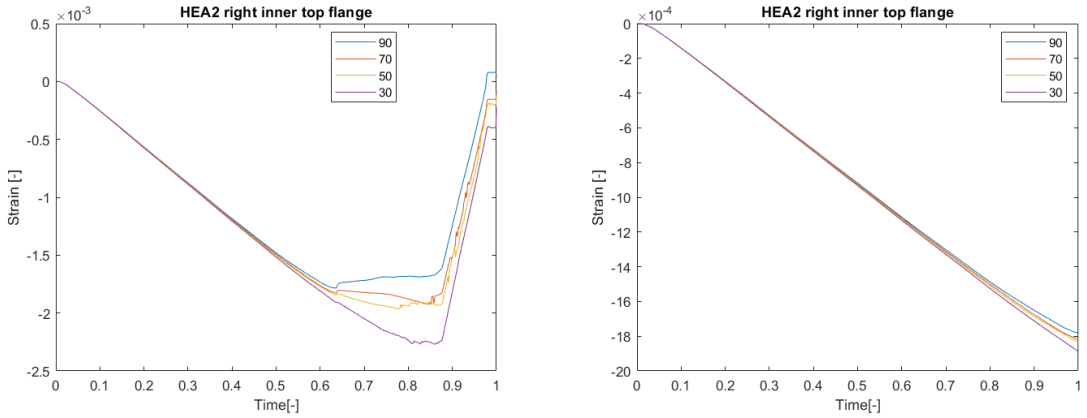
(a) Full data **(b)** Cut data

Figure B.11: HEA2 top flange



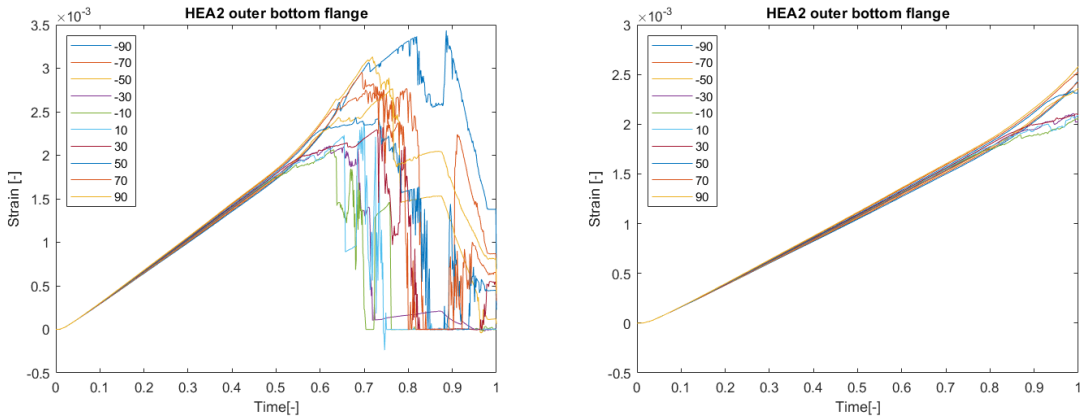
(a) Full data **(b)** Cut data

Figure B.12: HEA2 left inner top flange



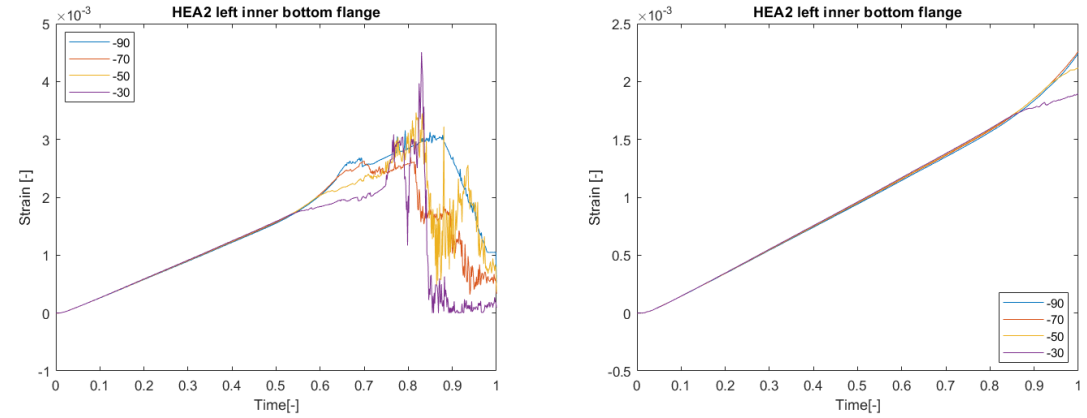
(a) Full data **(b)** Cut data

Figure B.13: HEA2 right inner top flange



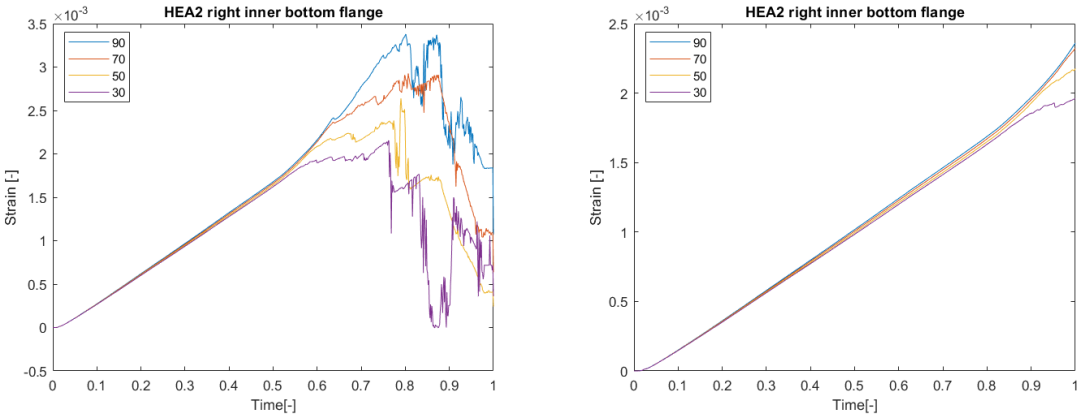
(a) Full data **(b)** Cut data

Figure B.14: HEA2 bottom flange



(a) Full data **(b)** Cut data

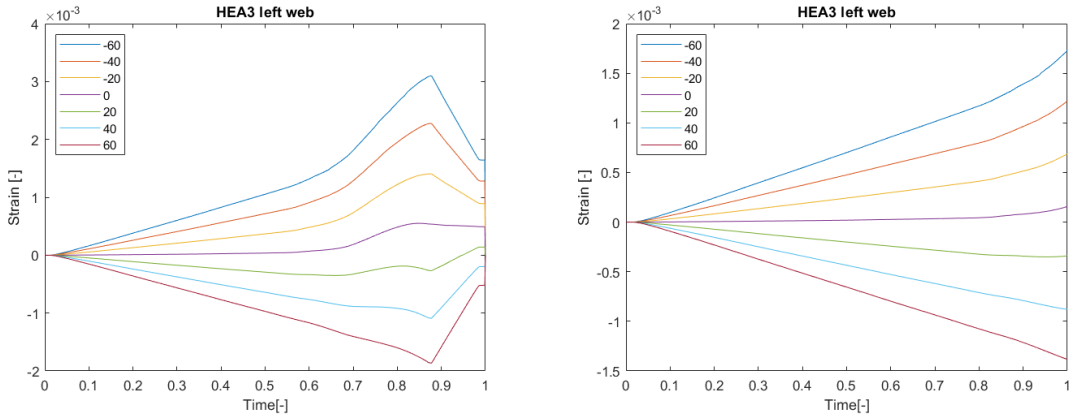
Figure B.15: HEA2 left inner bottom flange



(a) Full data **(b)** Cut data

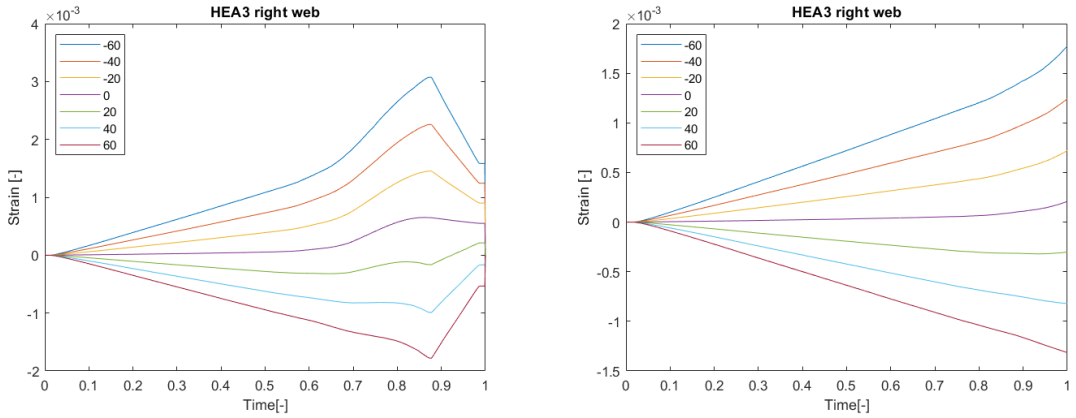
Figure B.16: HEA2 right inner top flange

HEA3



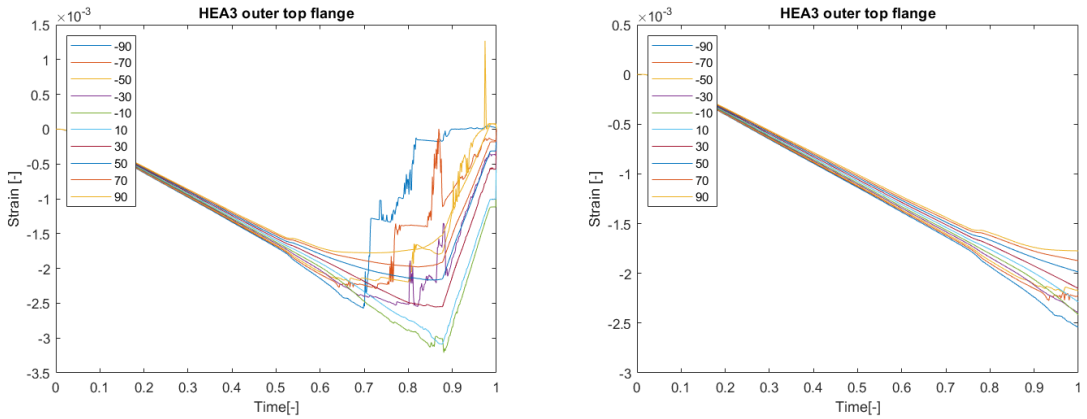
(a) Full data **(b)** Cut data

Figure B.17: HEA3 left web



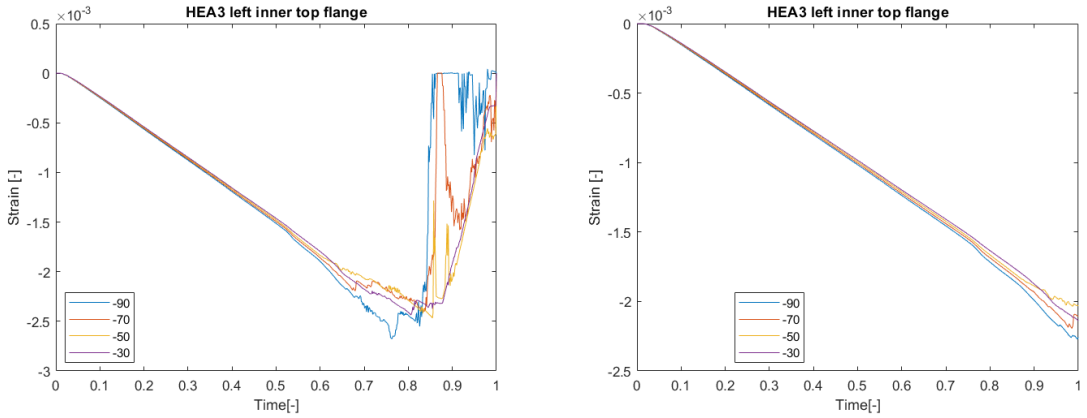
(a) Full data **(b)** Cut data

Figure B.18: HEA3 right web



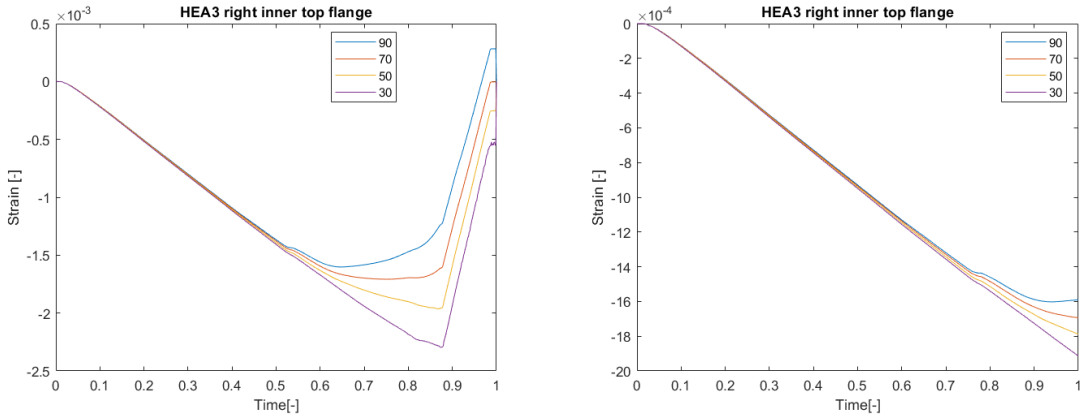
(a) Full data **(b)** Cut data

Figure B.19: HEA3 top flange



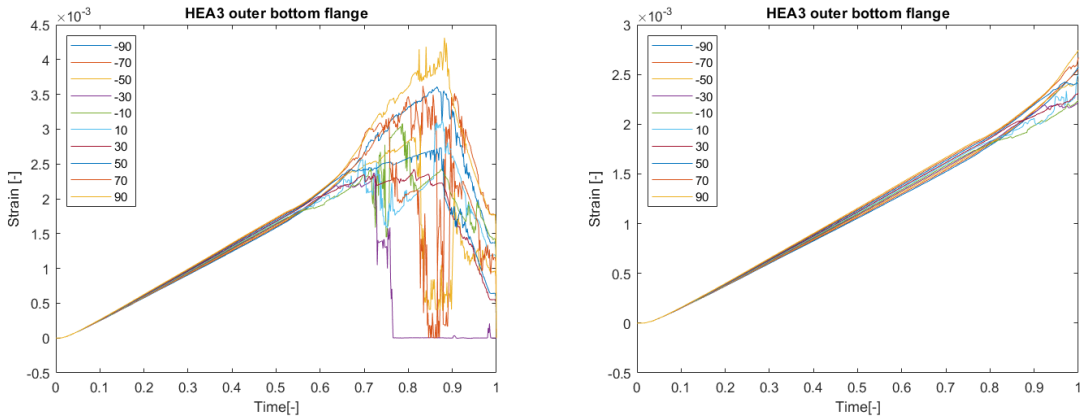
(a) Full data **(b)** Cut data

Figure B.20: HEA3 left inner top flange



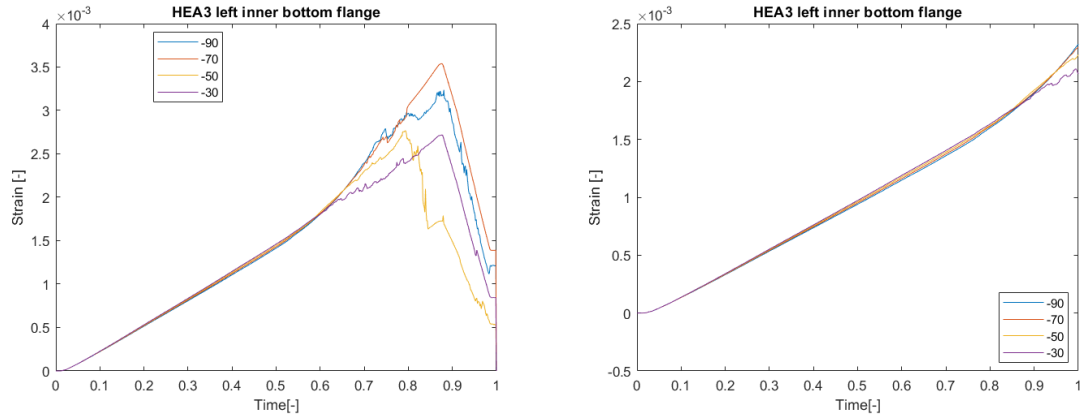
(a) Full data **(b)** Cut data

Figure B.21: HEA3 right inner top flange



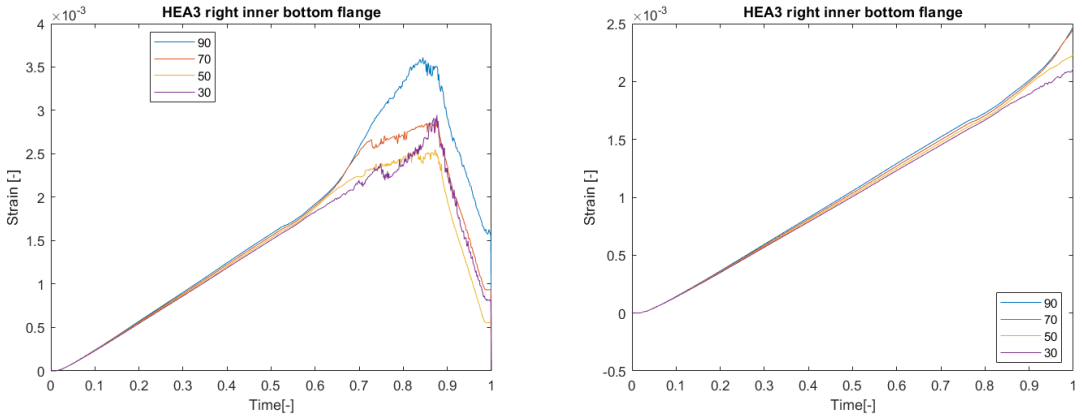
(a) Full data **(b)** Cut data

Figure B.22: HEA3 bottom flange



(a) Full data **(b)** Cut data

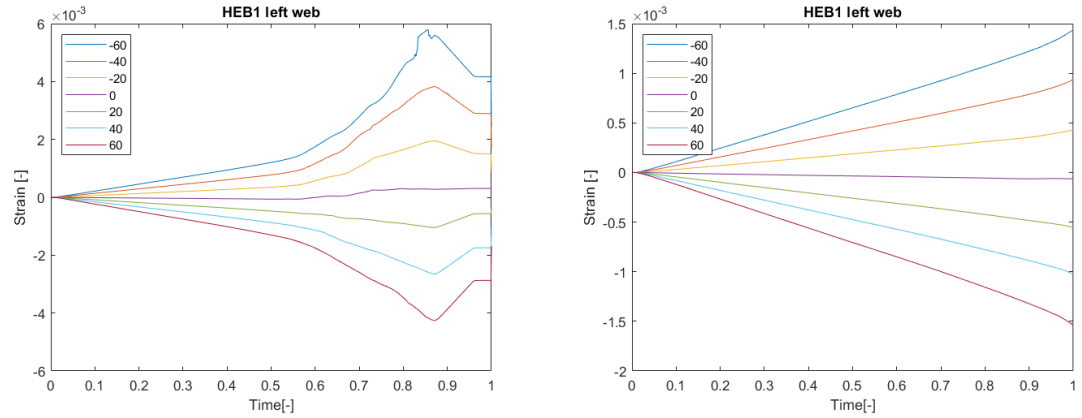
Figure B.23: HEA3 left inner bottom flange



(a) Full data **(b)** Cut data

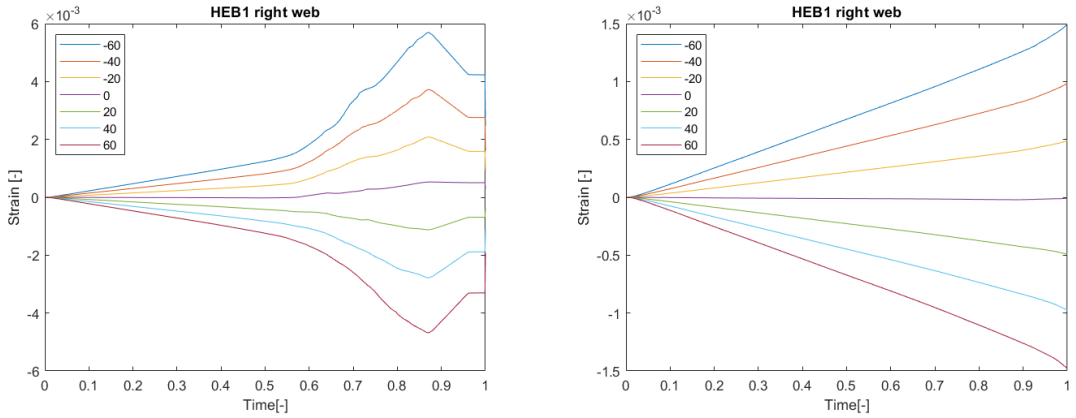
Figure B.24: HEA3 right inner top flange

HEB1



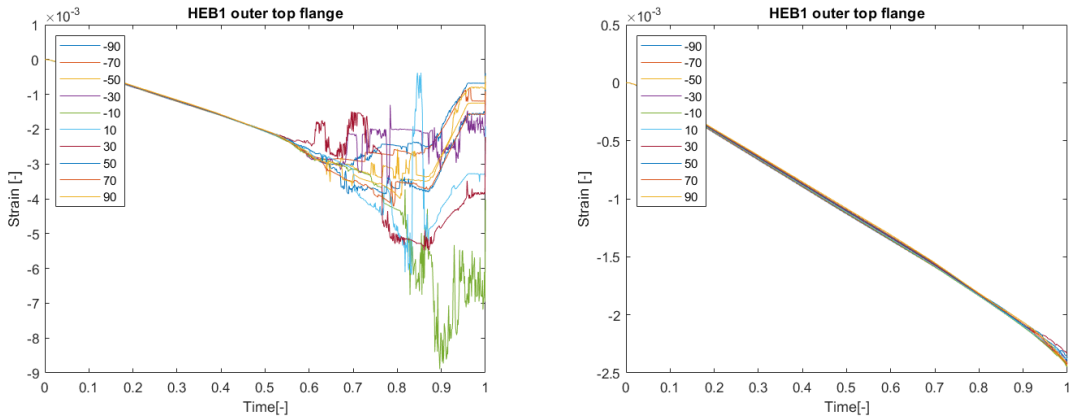
(a) Full data **(b)** Cut data

Figure B.25: HEB1 left web



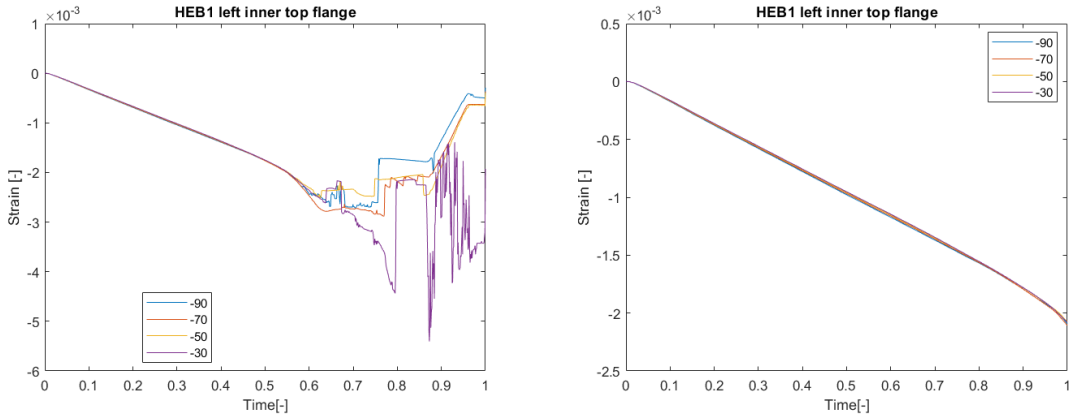
(a) Full data **(b)** Cut data

Figure B.26: HEB1 right web



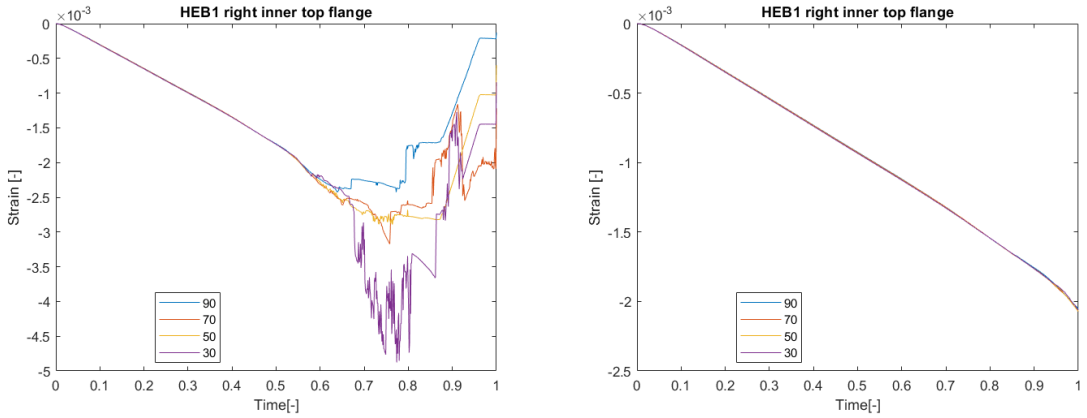
(a) Full data **(b)** Cut data

Figure B.27: HEB1 top flange



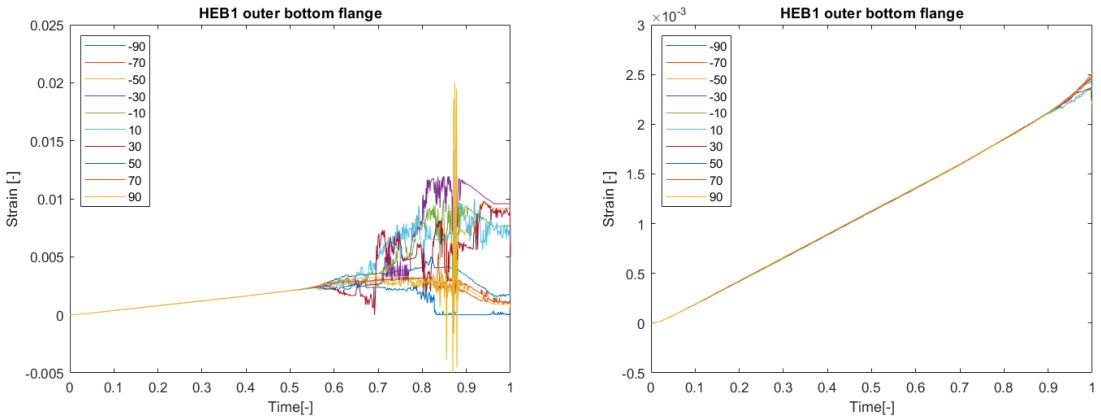
(a) Full data **(b)** Cut data

Figure B.28: HEB1 left inner top flange



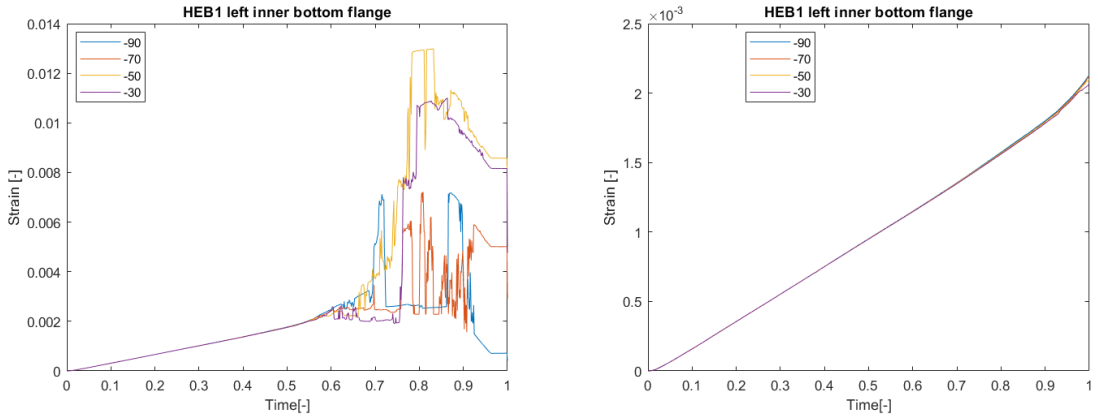
(a) Full data **(b)** Cut data

Figure B.29: HEB1 right inner top flange



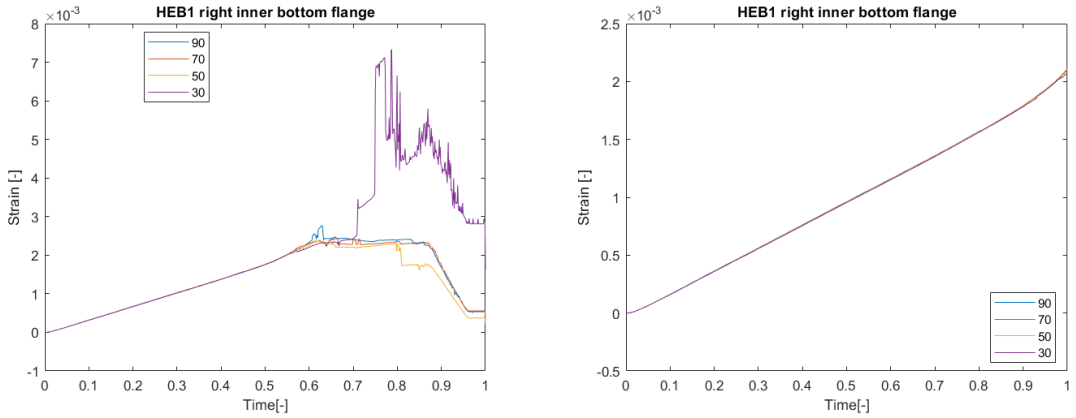
(a) Full data **(b)** Cut data

Figure B.30: HEB1 bottom flange



(a) Full data **(b)** Cut data

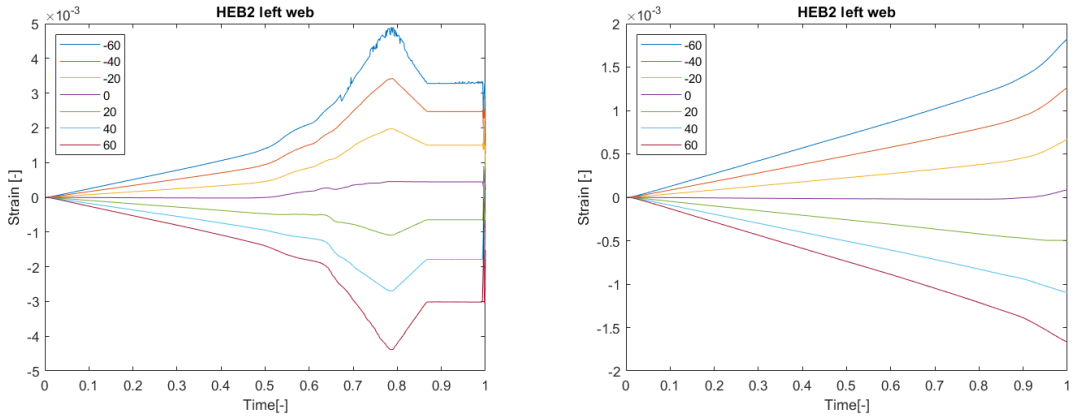
Figure B.31: HEB1 left inner bottom flange



(a) Full data **(b)** Cut data

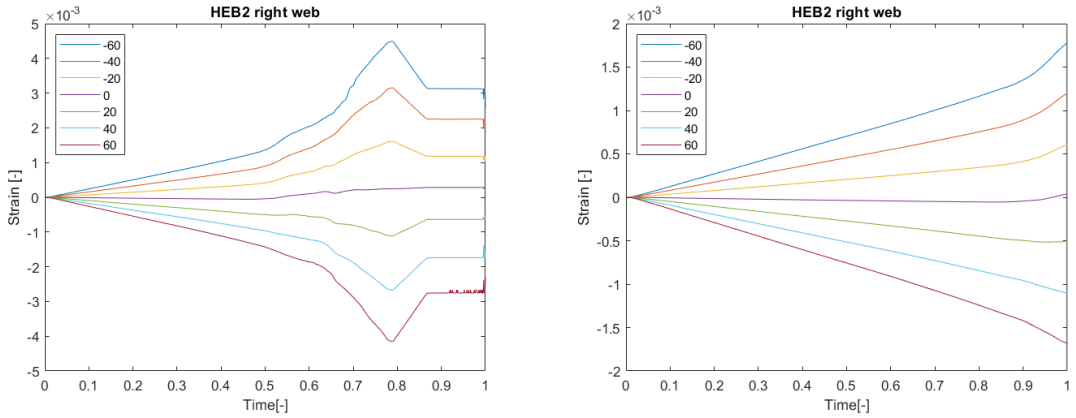
Figure B.32: HEB1 right inner top flange

HEB2



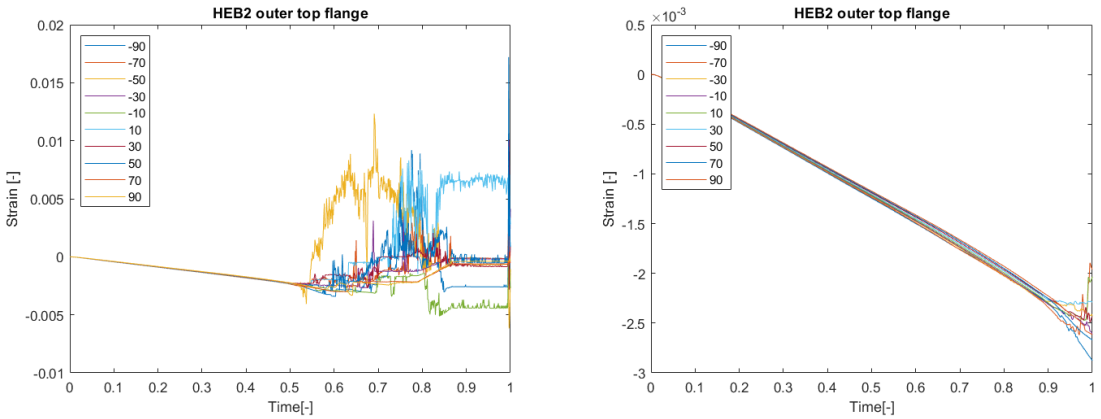
(a) Full data **(b)** Cut data

Figure B.33: HEB2 left web



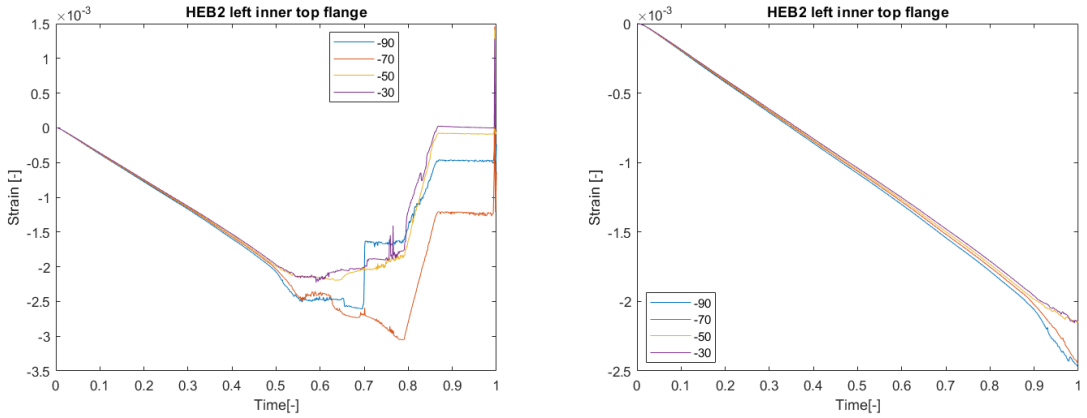
(a) Full data **(b)** Cut data

Figure B.34: HEB2 right web



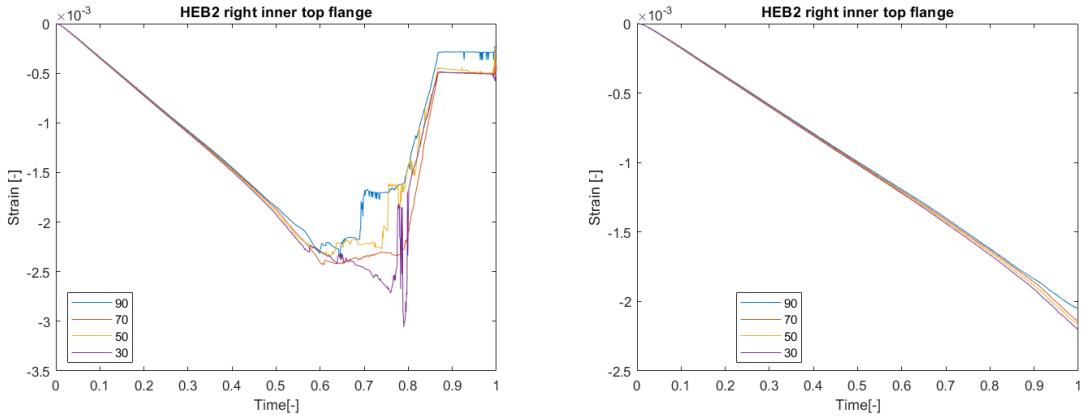
(a) Full data **(b)** Cut data

Figure B.35: HEB2 top flange



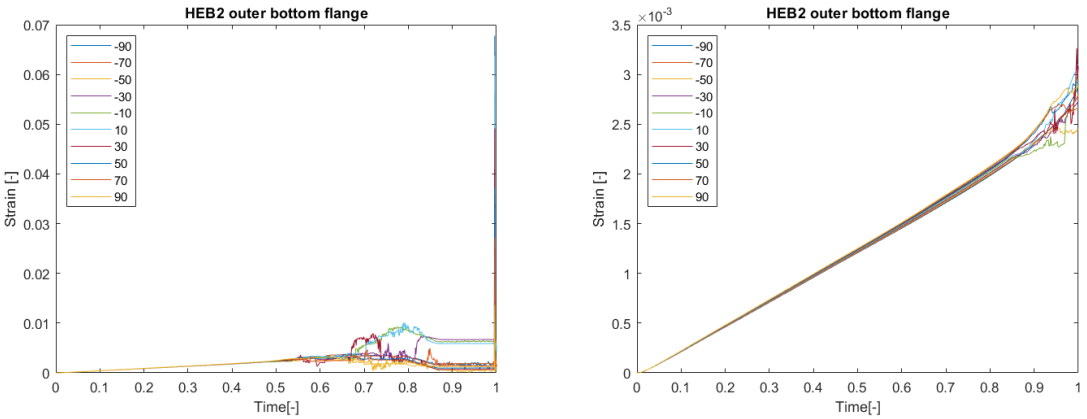
(a) Full data **(b)** Cut data

Figure B.36: HEB2 left inner top flange



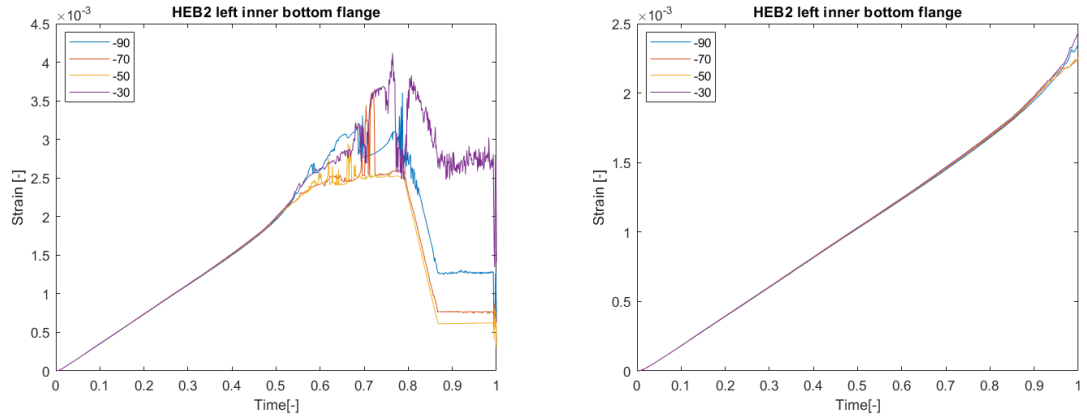
(a) Full data **(b)** Cut data

Figure B.37: HEB2 right inner top flange



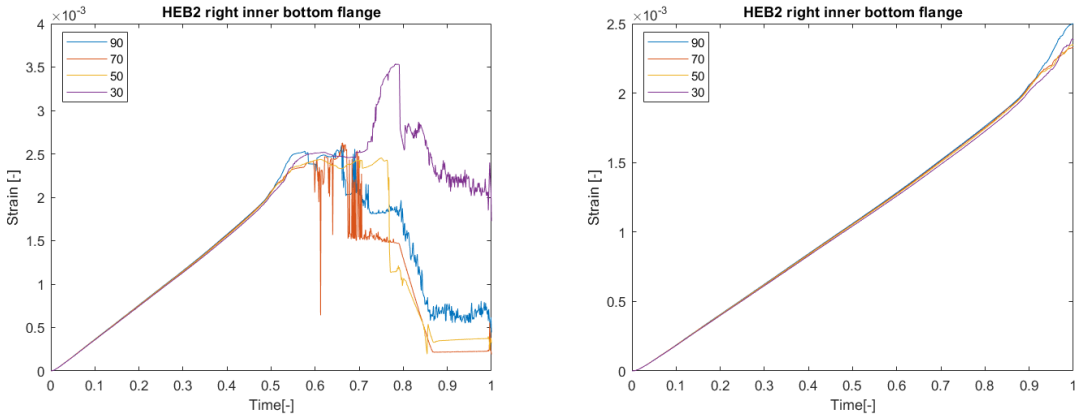
(a) Full data **(b)** Cut data

Figure B.38: HEB2 bottom flange



(a) Full data **(b)** Cut data

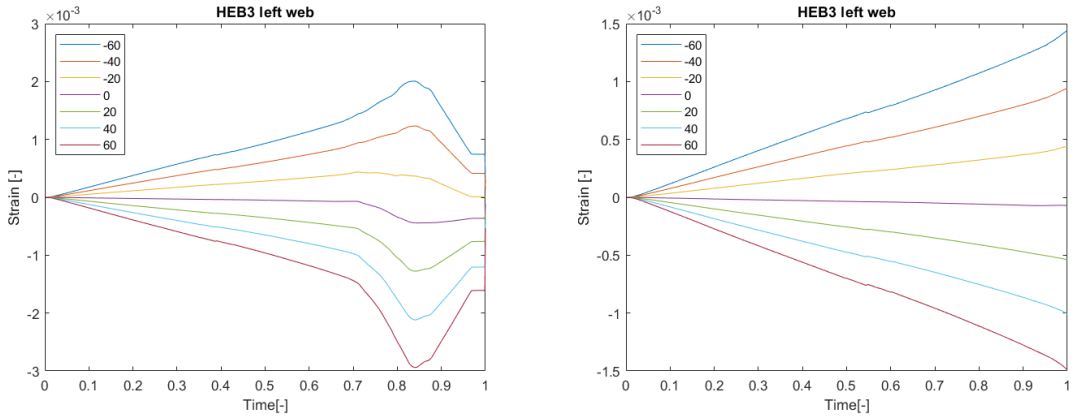
Figure B.39: HEB2 left inner bottom flange



(a) Full data **(b)** Cut data

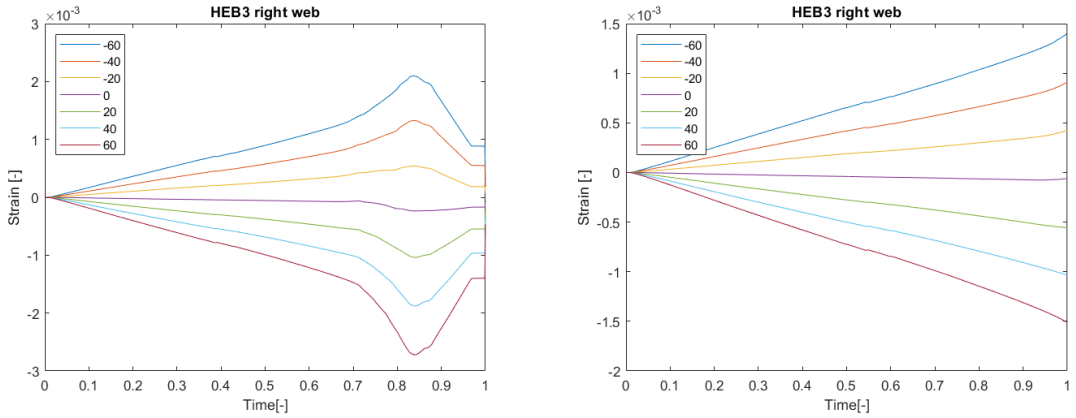
Figure B.40: HEB2 right inner top flange

HEB3



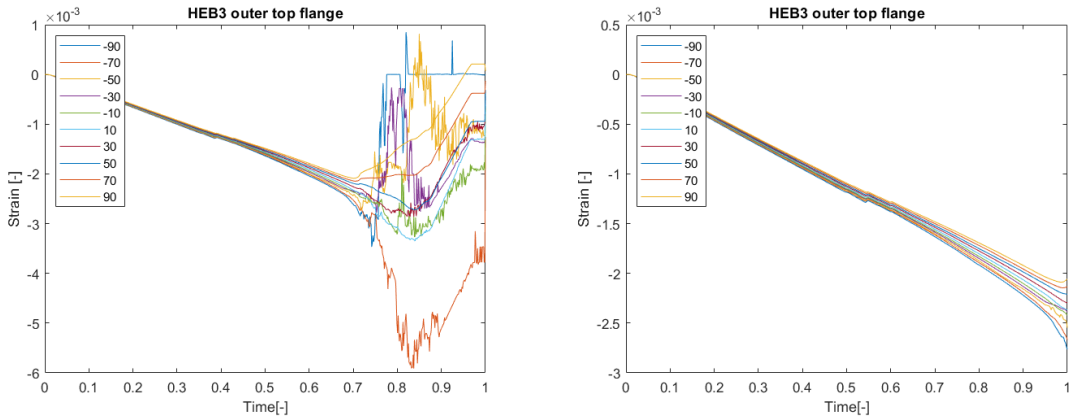
(a) Full data **(b)** Cut data

Figure B.41: HEB3 left web



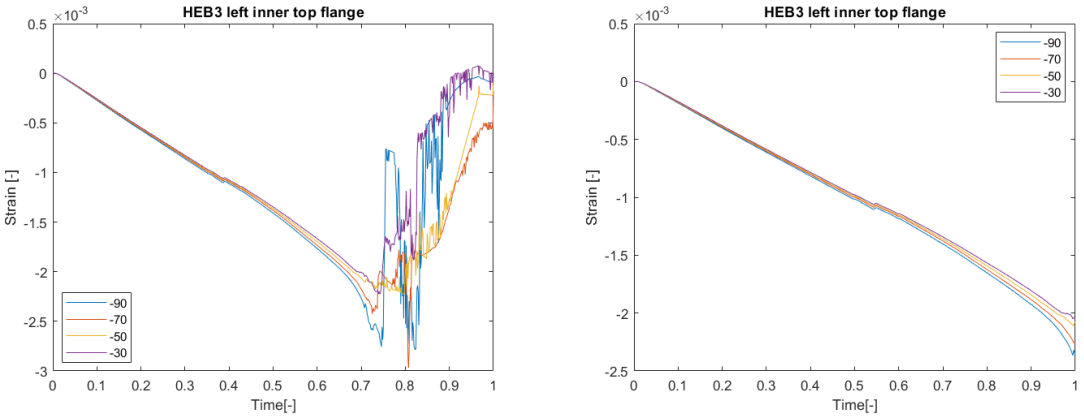
(a) Full data **(b)** Cut data

Figure B.42: HEB3 right web



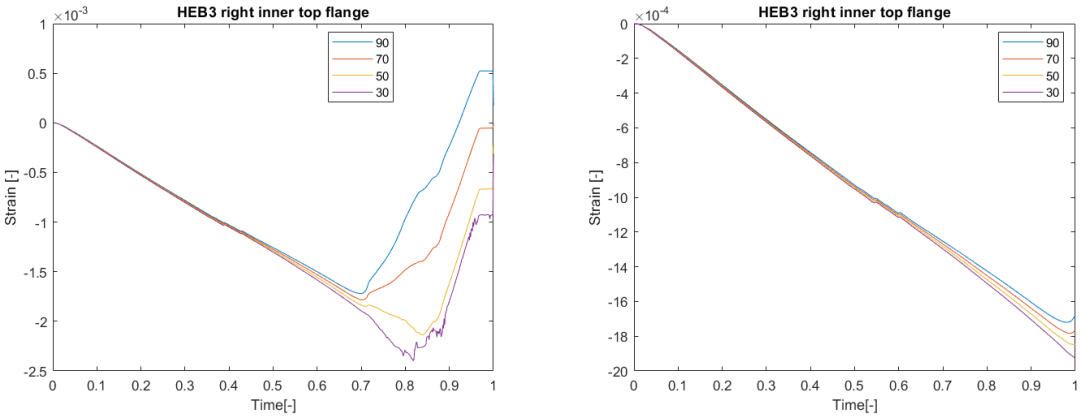
(a) Full data **(b)** Cut data

Figure B.43: HEB3 top flange



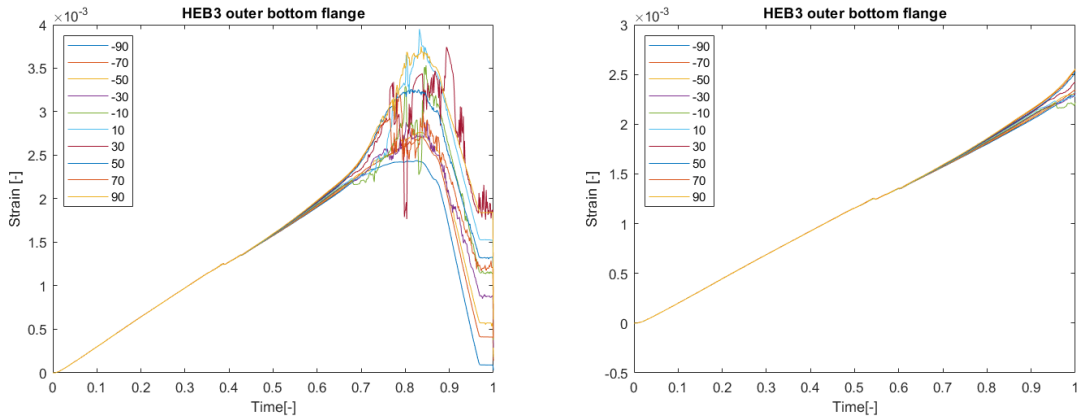
(a) Full data **(b)** Cut data

Figure B.44: HEB3 left inner top flange



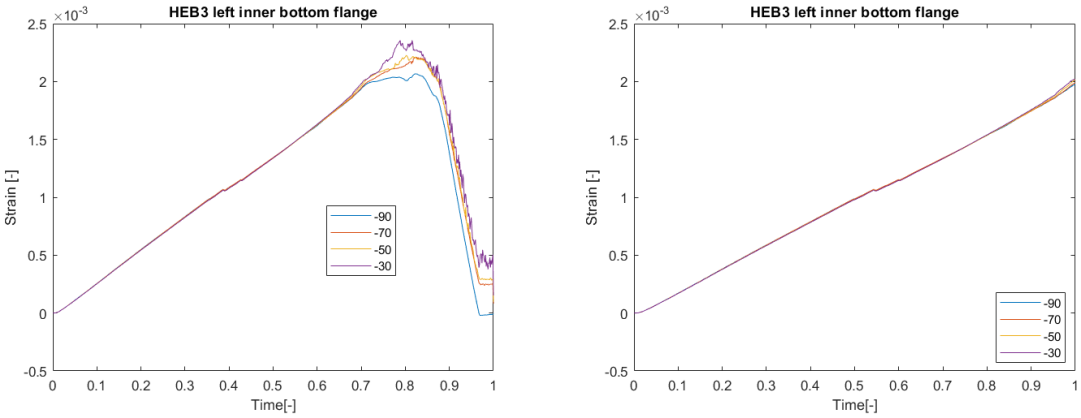
(a) Full data **(b)** Cut data

Figure B.45: HEB3 right inner top flange



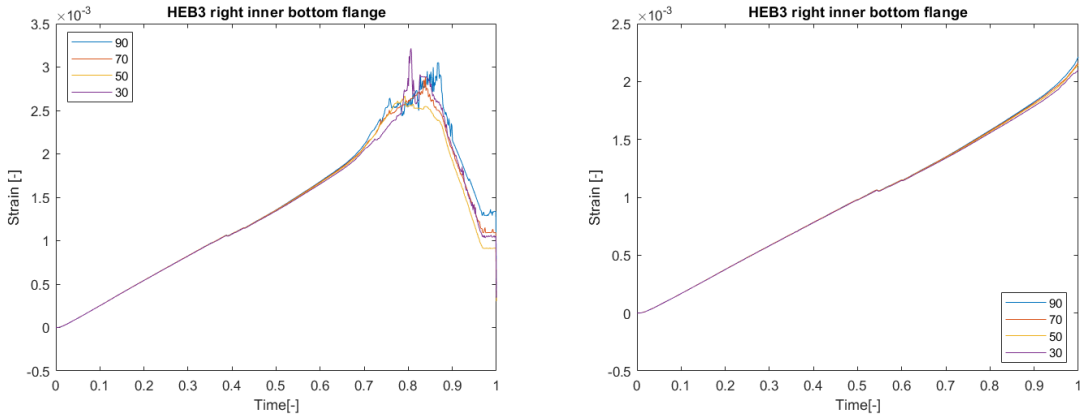
(a) Full data **(b)** Cut data

Figure B.46: HEB3 bottom flange



(a) Full data **(b)** Cut data

Figure B.47: HEB3 left inner bottom flange



(a) Full data **(b)** Cut data

Figure B.48: HEB3 right inner top flange

C

Tensile test calculations script

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d

pathname = 'C:/Users/qtv833/OneDrive - Chalmers/Master/Master thesis/Dogbones/Dragtest data/'
filename1 = 'Restspänning_exjobb_2024_hundben_HEA_flange_02.txt'
filename2 = 'Restspänning_exjobb_2024_hundben_HEA_flange_03.txt'
filename3 = 'Restspänning_exjobb_2024_hundben_HEA_flange_04.txt'
data1 = np.loadtxt(pathname+filename1, delimiter='\t', skiprows=5)
data2 = np.loadtxt(pathname+filename2, delimiter='\t', skiprows=5)
data3 = np.loadtxt(pathname+filename3, delimiter='\t', skiprows=5)

# Cross-sectional area of test specimen
area1 = 179.247925
area2 = 168.0569
area3 = 165.91575

# Change from force [kN] to stress [MPa]
data1[:,0] = data1[:,0]*1000/area1
data2[:,0] = data2[:,0]*1000/area2
data3[:,0] = data3[:,0]*1000/area3

# Change strain from [mm per 50 mm] to [-]
data1[:,1] = data1[:,1]/50
data2[:,1] = data2[:,1]/50
data3[:,1] = data3[:,1]/50


# Calculates the maximum strain where all three curves have data
common_strain_max = min([data1[-1,1],data2[-1,1],data3[-1,1]])-0.01

# Creates a strain vector
n = 2000
common_strain = []
for i in range(n+1):
    common_strain.append(i/n*common_strain_max)

```

```
38 # Creates interpolation functions based on the three specimens stress
   ↪ strain data
39 f1 = interp1d(data1[:,1], data1[:,0], kind='linear',
   ↪ fill_value="extrapolate")
40 f2 = interp1d(data2[:,1], data2[:,0], kind='linear',
   ↪ fill_value="extrapolate")
41 f3 = interp1d(data3[:,1], data3[:,0], kind='linear',
   ↪ fill_value="extrapolate")
42
43 stress_interp1 = f1(common_strain)
44 stress_interp2 = f2(common_strain)
45 stress_interp3 = f3(common_strain)
46
47 mean_stress = np.mean([stress_interp1, stress_interp2,
   ↪ stress_interp3], axis=0)
48 # print(mean_stress[0])
49
50
51 # Calculates Young's modulus in GPa
52 E_modulus =
   ↪ (mean_stress[5]-mean_stress[1])/(common_strain[5]-common_strain[1])
53 print(E_modulus)
54
55 # Calculates plastic strain
56 plastic_strain = common_strain - mean_stress/E_modulus
57 for i in range(len(plastic_strain)):
58     if plastic_strain[i] < 0.0001:
59         plastic_strain[i] = 0
60
61
62 k = 25
63 print(mean_stress[k])
64 print(plastic_strain[k])
65
66 plastic_table = np.transpose(np.array([mean_stress[k:],
   ↪ plastic_strain[k:]]))
67
68 print(plastic_table[:5,:])
69
70 # np.savetxt(pathname+'Plastic_Table_HEA_Web.txt', plastic_table)
71
72 plt.figure()
73 plt.plot(data1[:,1], data1[:,0], linestyle='dashed', linewidth = 0.7)
74 plt.plot(data2[:,1], data2[:,0], linestyle='dashed', linewidth = 0.7)
75 plt.plot(data3[:,1], data3[:,0], linestyle='dashed', linewidth = 0.7)
76 plt.plot(common_strain, mean_stress, 'k')
77 plt.legend(['1','2','3', 'Mean'])
78
```

```
79 plt.show()
```

D

Script for running abaqus and calculating error

```
from abaqus import *
from abaqusConstants import *
import regionToolset
import __main__
import section
import part
import material
import assembly
import step
import interaction
import load
import mesh
import job
import sketch
import visualization
from visualization import *
import xyPlot
import connectorBehavior
import odbAccess
from odbAccess import *
from operator import add


import matplotlib.pyplot as plt
import numpy as np
import scipy as sci
import os
import sys

# ----------------------------------------------------------------
### Function creating the geometry of a solid HEA/HEB-shaped beam ###
def Create_solid_beam(model, part, Width, Heigth, Flange, Web,
    Radius, Length):
    s = mdb.models[model].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
    g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
```

```
36      s.setPrimaryObject(option=STANDALONE)
37      ### Defining the outer geometry
38      s.rectangle(point1=(Width/2.0, Heigth/2.0), point2=(-Width/2.0,
39                  (Heigth/2.0-Flange)))
40      s.rectangle(point1=(-Width/2.0, -Heigth/2.0), point2=(Width/2.0,
41                  -(Heigth/2.0-Flange)))
42      s.rectangle(point1=(-Web/2.0, (Heigth/2.0-Flange)),
        ↪  point2=(Web/2.0,
43                  -(Heigth/2.0-Flange)))
44      ### Trimming excess lines
45      s.autoTrimCurve(curve1=g[3],  point1=(0,  (Heigth/2-Flange)))
46      s.autoTrimCurve(curve1=g[13], point1=(0,  (Heigth/2-Flange)))
47      s.autoTrimCurve(curve1=g[7],  point1=(0, -(Heigth/2-Flange)))
48      s.autoTrimCurve(curve1=g[11], point1=(0, -(Heigth/2-Flange)))
49      ### Adding a radius to the inward corners
50      s.FilletByRadius(radius=Radius, curve1=g[15],
        ↪  nearPoint1=(-Width/2.0,
51          (Heigth/2.0-Flange)), curve2=g[10], nearPoint2=(-Web/2.0, 0))
52      s.FilletByRadius(radius=Radius, curve1=g[10],
        ↪  nearPoint1=(-Web/2.0, 0),
53          curve2=g[16], nearPoint2=(-Width/2.0, -(Heigth/2.0-Flange)))
54      s.FilletByRadius(radius=Radius, curve1=g[17],
        ↪  nearPoint1=(Width/2.0,
55          -(Heigth/2-Flange)), curve2=g[12], nearPoint2=(Web/2.0, 0))
56      s.FilletByRadius(radius=Radius, curve1=g[12],
        ↪  nearPoint1=(Web/2.0, 0),
57          curve2=g[14], nearPoint2=(Width/2.0, (Heigth/2.0-Flange)))
58      ### Create the modell ###
59      p = mdb.models[model].Part(name=part, dimensionality=THREE_D,
60          type=DEFORMABLE_BODY)
61      p = mdb.models[model].parts[part]
62      p.BaseSolidExtrude(sketch=s, depth=Length)
63      s.unsetPrimaryObject()
64      del mdb.models[model].sketches['__profile__']
65
66  # ----------------------------------------------------------------
67  ### Function creating a shell HEA/HEB-shaped beam ###
68  def Create_shell_beam(model, part, Width, Heigth, Flange, Length):
69      s = mdb.models[model].ConstrainedSketch(name='__profile__',
70          sheetSize = 200.0)
71      g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
72      s.setPrimaryObject(option=STANDALONE)
73      s.Line(point1=(-Width/2.0,(Heigth-Flange)/2.0),
        ↪  point2=(Width/2.0,(Heigth-Flange)/2.0))
74      s.Line(point1=(-Width/2.0,-(Heigth-Flange)/2.0),
        ↪  point2=(Width/2.0,-(Heigth-Flange)/2.0))
75      s.Line(point1=(0.0,(Heigth-Flange)/2.0),
        ↪  point2=(0.0,-(Heigth-Flange)/2.0))
```

```
76      p = mdb.models[model].Part(name=part, dimensionality=THREE_D,
        ↪  type=DEFORMABLE_BODY)
77      p.BaseShellExtrude(sketch=s, depth=Length)
78      s.unsetPrimaryObject()
79      del mdb.models[model].sketches['__profile__']
80
81  # ------------------------------------------------------------------
82  # DATUM PLANE AND PARTITION OF THESE #
83
84  def Create_Datum_Plane_Part(model, part, plane, offset):
85      p = mdb.models[model].parts[part]
86      myPlane = p.DatumPlaneByPrincipalPlane(principalPlane=plane,
        ↪  offset=offset)
87      myID = myPlane.id
88      return myID
89
90  # ------------------------------------------------------------------
91  def Create_Partition_by_Plane_Part(model, part, plane_id):
92      p = mdb.models[model].parts[part]
93      c = p.cells[:]
94      d = p.datums
95      p.PartitionCellByDatumPlane(datumPlane=d[plane_id], cells=c)
96
97
98
99  # ------------------------------------------------------------------
100 # SET CREATION
101 def Create_Set_All_Cells(model, part, set_name):
102     p = mdb.models[model].parts[part]
103     c = p.cells[:]
104     p.Set(cells=c, name=set_name)
105
106 # ------------------------------------------------------------------
107 def Create_Set_Cells(model, part, set_name, xmin, xmax, ymin, ymax,
    ↪  zmin, zmax):
108     p = mdb.models[model].parts[part]
109     c = p.cells
110     myCell = c.getByBoundingBox(xMin=xmin, xMax=xmax, yMin=ymin,
        ↪  yMax=ymax, zMin=zmin, zMax=zmax)
111     p.Set(cells=myCell, name=set_name)
112
113 # ------------------------------------------------------------------
114 def Create_Set_Cells_Instance(model, instance, set_name, xmin, xmax,
    ↪  ymin, ymax, zmin, zmax):
115     a = mdb.models[model].rootAssembly
116     i = a.instances[instance]
117     c = i.cells
```

```
118     myCell = c.getByBoundingBox(xMin=xmin, xMax=xmax, yMin=ymin,
        ↪  yMax=ymax, zMin=zmin, zMax=zmax)
119     a.Set(cells=myCell, name=set_name)
120 
121 # -------------------------------------------------------------
122 def Create_Set_Edge(model, part, set_name, x, y, z):
123     edge = ()
124     p = mdb.models[model].parts[part]
125     e = p.edges
126     myEdge = e.findAt((x,y,z),)
127     edge = edge + (e[myEdge.index:myEdge.index+1], )
128     p.Set(edges=edge, name=set_name)
129     return myEdge
130 
131 # -------------------------------------------------------------
132 def Create_Set_Face(model, part, set_name, x, y, z):
133     face   = ()
134     p      = mdb.models[model].parts[part]
135     f      = p.faces
136     myFace = f.findAt((x,y,z),)
137     face   = face + (f[myFace.index:myFace.index+1],)
138     p.Set(faces=face, name=set_name)
139     return myFace
140 
141 
142 
143 # -------------------------------------------------------------
144 # MATERIAL CREATION
145 def Create_Material(model, material_name, density, youngs_modulus,
    ↪  poissons, plastic_table):
146     mdb.models[model].Material(name=material_name)
147 
        ↪  mdb.models[model].materials[material_name].Density(table=((density,
        ↪  ), ))
148 
        ↪  mdb.models[model].materials[material_name].Elastic(table=((youngs_modulus,
        ↪  poissons), ))
149 
        ↪  mdb.models[model].materials[material_name].Plastic(scaleStress=None,
        ↪  table=plastic_table)
150 
151 
152 
153 # -------------------------------------------------------------
154 # Section creation and assignmen
155 def Create_shell_section(model, section_name, material_name,
    ↪  shell_thickness):
156     mdb.models[model].HomogeneousShellSection(name=section_name,
```

```
157     preIntegrate=OFF, material=material_name, thicknessType=UNIFORM,
158     thickness=shell_thickness, thicknessField='',
        ↪  nodalThicknessField='',
159     idealization=NO_IDEALIZATION, poissonDefinition=DEFAULT,
160     thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
161     integrationRule=SIMPSON, numIntPts=5)
162 
163 def Create_solid_section(model, section_name, material_name):
164     mdb.models[model].HomogeneousSolidSection(name=section_name,
        ↪  material=material_name,thickness=None)
165 
166 
167 
168 # ------------------------------------------------------------
169 # Assign section
170 def Assign_section(model, part, set_name, section_name):
171     p = mdb.models[model].parts[part]
172     region = p.sets[set_name]
173     p.SectionAssignment(region=region, sectionName=section_name,
        ↪  offset=0.0, offsetType=MIDDLE_SURFACE, offsetField ='',
174                         thicknessAssignment=FROM_SECTION)
175 
176 
177 # ------------------------------------------------------------
178 
179 def Create_instance(model, part, instance):
180     a = mdb.models[model].rootAssembly
181     a.DatumCsysByDefault(CARTESIAN)
182     p = mdb.models[model].parts[part]
183     a.Instance(name=instance, part=p, dependent=OFF)
184 
185 # ------------------------------------------------------------
186 def Translate_instance(model,instance,vector):
187     a = mdb.models[model].rootAssembly
188     a.translate(instanceList=(instance, ), vector=vector)
189 
190 
191 # Instance sets for coupling
192 # ------------------------------------------------------------
193 # Create instance
194 # For edges (shell)
195 def Create_Set_Edge_instance(model, instance, set_name, x, y, z):
196     edge = ()
197     a = mdb.models[model].rootAssembly
198     e = a.instances[instance].edges
199     myEdge = e.findAt((x,y,z),)
200     edge = edge + (e[myEdge.index:myEdge.index+1], )
201     a.Set(edges=edge, name=set_name)
```

```
202     return myEdge
203 
204 # ------------------------------------------------------------
205 # For faces (solid)
206 def Create_Set_Face_instance(model, instance, set_name, x, y, z):
207     face = ()
208     a = mdb.models[model].rootAssembly
209     f = a.instances[instance].faces
210     myface = f.findAt((x,y,z),)
211     face = face + (f[myface.index:myface.index+1], )
212     a.Set(faces=face, name=set_name)
213     return myface
214 
215 # ------------------------------------------------------------
216 # Merge sets
217 def Merge_set_in_instance(model, instance, set_name, input_sets):
218     a = mdb.models[model].rootAssembly
219     sets = []
220     for i in range(len(input_sets)):
221         sets.append(a.sets[input_sets[i]])
222     newset = a.SetByBoolean(name=set_name, sets=sets,
      ↪  operation=UNION)
223     for i in range(len(input_sets)):
224         del a.sets[input_sets[i]]
225     return newset
226 
227 
228 # Coupling of shell to solid
229 # ------------------------------------------------------------
230 def Shell_to_solid(model, Shell_instance, Solid_instance, x, y, z,
  ↪  Coupling_name, Surf_edge, Surf_face):
231     # For shell region
232     a  = mdb.models[model].rootAssembly
233     s1 = a.instances[Shell_instance].edges
234     side1Edges1 = s1.findAt(((x,y,z),),
235                             ((x,-y,z),),
236                             ((-x,y,z),),
237                             ((-x,-y,z),),
238                             ((0.0,0.0,z),))
239     region1=a.Surface(side1Edges=side1Edges1, name=Surf_edge)
240     # For solid region
241     a = mdb.models[model].rootAssembly
242     s2 = a.instances[Solid_instance].faces
243     side1Faces1 = s2.findAt(((0.0,0.0,z),),
244                             ((0,y,z),),
245                             ((0,-y,z),))
246     region2=a.Surface(side1Faces=side1Faces1, name=Surf_face)
247     # Constraint coupling
```

```
248     mdb.models[model].ShellSolidCoupling(name=Coupling_name,
        ↪  shellEdge=region1,
249         solidFace=region2, positionToleranceMethod=COMPUTED)
250
251
252 # Datumplane and partition
253 # ------------------------------------------------------------------
254 def Create_Datum_Plane(model, plane, offset):
255     a = mdb.models[model].rootAssembly
256     myPlane = a.DatumPlaneByPrincipalPlane(principalPlane=plane,
        ↪  offset=offset)
257     myID = myPlane.id
258     return myID
259
260 # ------------------------------------------------------------------
261 def Create_Partition_by_Plane(model, instance, plane_id):
262     a = mdb.models[model].rootAssembly
263     c = a.instances[instance].cells[:]
264     d = a.datums
265     a.PartitionCellByDatumPlane(datumPlane=d[plane_id], cells=c)
266
267 # ------------------------------------------------------------------
268 #  Creation of surface
269 def Create_Shell_Surface(model, Shell_instance, Surf_name, x, y, z):
270     a  = mdb.models[model].rootAssembly
271     s1 = a.instances[Shell_instance].edges
272     side1Edges1 = s1.findAt(((x,y,z),),
273                             ((x,-y,z),),
274                             ((-x,y,z),),
275                             ((-x,-y,z),),
276                             ((0.0,0.0,z),))
277     region1=a.Surface(side1Edges=side1Edges1, name=Surf_name)
278     return region1
279
280 # ------------------------------------------------------------------
281 def Create_RP(model,x,y,z,set_name):
282     a = mdb.models[model].rootAssembly
283     myRP = a.ReferencePoint(point=(x,y,z))
284     r = a.ReferencePoints
285     myRP_pos = r.findAt((x,y,z),)
286     refPoint1 = (myRP_pos,)
287     a.Set(referencePoints = refPoint1, name=set_name)
288     return myRP,myRP_pos
289
290 # Time steps
291 # ------------------------------------------------------------------
292 def Create_Step(model, step_name, previous_step, maxNumInc,
    ↪  initialInc, minInc, maxInc, nlgeom):
```

```
293     mdb.models[model].StaticStep(name=step_name,
        ↪ previous=previous_step, maxNumInc=maxNumInc,
        ↪ initialInc=initialInc, minInc=minInc, maxInc=maxInc,
        ↪ nlgeom=nlgeom)
294
295
296 # Boundary conditions
297 # ------------------------------------------------------------
298 def Create_Initial_BC(model, set_name, BC_name, u1, u2, u3, ur1, ur2,
    ↪ ur3):
299     a = mdb.models[model].rootAssembly
300     region = a.sets[set_name]
301     mdb.models[model].DisplacementBC(name=BC_name,
        ↪ createStepName='Initial', region=region, u1=u1, u2=u2, u3=u3,
        ↪ ur1=ur1, ur2=ur2, ur3=ur3, amplitude=UNSET,
        ↪ distributionType=UNIFORM, fieldName='', localCsys=None)
302
303
304 # Loading conditions
305 # ------------------------------------------------------------
306 def Create_Gravity(model, load_name, step_name):
307     mdb.models[model].Gravity(name=load_name,
        ↪ createStepName=step_name, comp2=-9810.0,
        ↪ distributionType=UNIFORM, field='')
308
309 def Create_Displacment(model, set_name, BC_name, step_name, u1, u2,
    ↪ u3, ur1, ur2, ur3):
310     a = mdb.models[model].rootAssembly
311     region = a.sets[set_name]
312     mdb.models[model].DisplacementBC(name=BC_name,
        ↪ createStepName=step_name, region=region, u1=u1, u2=u2, u3=u3,
        ↪ ur1=ur1, ur2=ur2, ur3=ur3, amplitude=UNSET, fixed=OFF,
        ↪ distributionType=UNIFORM, fieldName='', localCsys=None)
313
314 # Mesh controlls
315 # ------------------------------------------------------------
316
317 def Set_Mesh_Type_Solid(model, instance, elemShape, reduced_on,
    ↪ quadratic_on):
318     a = mdb.models[model].rootAssembly
319     c = a.instances[instance].cells[:]
320     if elemShape == TET:
321         a.setMeshControls(regions=c, elemShape=elemShape,
            ↪ technique=FREE)
322     if elemShape == HEX_DOMINATED:
323         a.setMeshControls(regions=c, elemShape=elemShape)
324     if quadratic_on == True:
325         if reduced_on:
```

```
326             elemType1 = mesh.ElemType(elemCode=C3D20R)
327         else:
328             elemType1 = mesh.ElemType(elemCode=C3D20)
329         elemType2 = mesh.ElemType(elemCode=C3D15)
330         elemType3 = mesh.ElemType(elemCode=C3D10)
331     else:
332         if reduced_on == True:
333             elemType1 = mesh.ElemType(elemCode=C3D8R,
                ↪  elemLibrary=STANDARD, secondOrderAccuracy=OFF,
                ↪  distortionControl=DEFAULT)
334         else:
335             elemType1 = mesh.ElemType(elemCode=C3D8,
                ↪  elemLibrary=STANDARD, secondOrderAccuracy=OFF,
                ↪  distortionControl=DEFAULT)
336         elemType2 = mesh.ElemType(elemCode=C3D6,
            ↪  elemLibrary=STANDARD)
337         elemType3 = mesh.ElemType(elemCode=C3D4,
            ↪  elemLibrary=STANDARD)
338     pickedRegions =(c, )
339     a.setElementType(regions=pickedRegions, elemTypes=(elemType1,
        ↪  elemType2, elemType3))
340
341 def Set_Mesh_Type_Shell(model,instance,Elem_shape): # TRI or QUAD
342     a = mdb.models[model].rootAssembly
343     f = a.instances[instance].faces[:]
344     a.setMeshControls(regions=f, elemShape=Elem_shape)
345
346
347 # Seed controls
348 # ----------------------------------------------------------------
349 def Seed_edge(model,instance,seed_size,x,y,z):
350     a = mdb.models[model].rootAssembly
351     e = a.instances[instance].edges
352     Edge = e.findAt(((x,y,z),),
353                     ((x,-y,z),),
354                     ((-x,y,z),),
355                     ((-x,-y,z),),
356                     ((0.0,0.0,z),))
357     a.seedEdgeBySize(edges=Edge, size=seed_size, deviationFactor=0.1,
        ↪  constraint=FINER)
358
359 # Residual stress calculation
360 # ----------------------------------------------------------------
361
362 # Calculating RS coefficents A and B
363 def Calculate_RS_Coefficients(height, width, flange_thickness,
    ↪  web_thickness, c, d):
364     Aa = 2*flange_thickness*width
```

```
365     Bb = 2*flange_thickness*width**3/12
366     Cc = web_thickness*(height-2*flange_thickness)
367     Dd = web_thickness*(height-2*flange_thickness)**3/12
368     a = c + d*(height-flange_thickness)**2/4
369     b = -(Aa*a + Cc*c + Dd*d)/Bb
370     return [a,b]
371 
372 # ------------------------------------------------------------
373 # Function to calculate RS in an element sett
374 def Calculate_RS_In_Element(model, instance, elem_num, height,
    ↪  flange_thickness, a, b, c, d):
375     coords = Get_Element_Coords(model, instance, elem_num)
376     x = coords[0]
377     y = coords[1]
378     if -(height/2-flange_thickness) < y <height/2-flange_thickness:
379         RS = c + d*y**2
380     else:
381         RS = a + b*x**2
382     return RS
383 
384 # Residual stress application
385 # ------------------------------------------------------------
386 
387 # Element coordinates
388 def Get_Element_Coords(model,instance , elem_num):
389     a = mdb.models[model].rootAssembly
390     elements = a.instances[instance].elements
391     region =
        ↪  regionToolset.Region(elements=elements[elem_num-1:elem_num])
392     properties = a.getMassProperties(regions=region)
393     centroid_list = properties['volumeCentroid']
394     return centroid_list
395 
396 # ------------------------------------------------------------
397 # Number of elements of a instance
398 def Get_Num_Elems(model, instance):
399     a = mdb.models[model].rootAssembly
400     i = a.instances[instance]
401     A = a.getMeshStats((i,))
402     sum = A.numHexElems + A.numWedgeElems + A.numTetElems
403     return sum
404 
405 # ------------------------------------------------------------
406 def Get_Num_Elems_Set(model,set_name):
407     a = mdb.models[model].rootAssembly
408     s = a.sets[set_name]
409     num_elements = len(s.elements)
410     return num_elements
```

```
411
412 # -------------------------------------------------------------
413 # Number of elements along a section of the instance
414 def Get_Num_Elements_Section(model, instance, beam_length,
    ↪ elem_length):    # Works aslong as the elements are hex elements
415     N = Get_Num_Elems(model, instance)
416     elements_along_length = beam_length/elem_length
417     elements_section = N/elements_along_length
418     print(elements_along_length,elements_section)
419     return int(elements_section)
420
421 # -------------------------------------------------------------
422
423 # Set of elements along a section of the instance
424 def Create_Element_Set(model, instance, element_numbers, set_name):
425     a = mdb.models[model].rootAssembly #.instances[instance]
426
        ↪ a.SetFromElementLabels(elementLabels=((instance,element_numbers),),
        ↪ name=set_name)
427
428 # -------------------------------------------------------------
429
430 # Combining previous functions to create set for assigning a residual
    ↪ stress
431 def Create_Set_For_Residual_Stress(model, instance, beam_length,
    ↪ elem_length, first_element, set_name):
432     N = Get_Num_Elems(model, instance)
433     n_sec = Get_Num_Elements_Section(model, instance, beam_length,
        ↪ elem_length)
434     element_numbers = []
435     for i in range(int(N/n_sec)):
436         element_numbers.append(first_element + i*n_sec)
437     Create_Element_Set(model, instance, element_numbers, set_name)
438
439 # -------------------------------------------------------------
440
441 # Assigning residual stress to a set cretated
442 def Create_Residual_Stress(model, instance, set_name,
    ↪ residual_stress_name, stress_value):
443     a = mdb.models[model].rootAssembly
444     region = a.sets[set_name]
445     mdb.models[model].Stress(name=residual_stress_name,
        ↪ region=region, distributionType=UNIFORM, sigma11=0.0,
        ↪ sigma22=0.0, sigma33=stress_value, sigma12=0.0, sigma13=0.0,
        ↪ sigma23=0.0)
446
447 # -------------------------------------------------------------
448
```

```
449 # Creates sets and residual stresses in a subsection
450 def Create_Residual_Stress_In_Subsection(model, instance,
    ↪ subsection_set_name, subsection_length, elem_length, CS_height,
    ↪ flange_thickness, a, b, c, d):
451     assem = mdb.models[model].rootAssembly
452     s = assem.sets[subsection_set_name]
453     N = Get_Num_Elems_Set(model,subsection_set_name)
454     n_sec = N/(subsection_length/elem_length)
455     for i in range(n_sec):
456         first_element = s.elements[i].label
457         element_numbers = []
458         for j in range(int(N/n_sec)):
459             element_numbers.append(first_element + j*n_sec)
460         Create_Element_Set(model, instance, element_numbers,
            ↪ subsection_set_name+' RS-set_'+str(i+1))
461         stress_value = Calculate_RS_In_Element(model, instance,
            ↪ element_numbers[0], CS_height, flange_thickness, a, b, c,
            ↪ d)
462         Create_Residual_Stress(model, instance, subsection_set_name+'
            ↪ RS-set_'+str(i+1), subsection_set_name+' RS-'+str(i+1),
            ↪ stress_value)
463
464
465
466 # ----------------------------------------------------------------
467
468
469 # Creating job
470 def Create_Job(model, job_name, cpus):
471     job = mdb.Job(name=job_name, model=model, description='',
472                   type=ANALYSIS, atTime=None, waitMinutes=0,
473                   waitHours=0, queue=None, memory=90,
                      ↪ memoryUnits=PERCENTAGE,
474                   getMemoryFromAnalysis=True, explicitPrecision=SINGLE,
475                   nodalOutputPrecision=SINGLE, echoPrint=OFF,
                      ↪ modelPrint=OFF,
476                   contactPrint=OFF, historyPrint=OFF,
                      ↪ userSubroutine='', scratch='',
477                   resultsFormat=ODB, numThreadsPerMpiProcess=1,
                      ↪ multiprocessingMode=DEFAULT,
478                   numCpus=cpus, numDomains=cpus, numGPUs=0)
479     return job
480 # ----------------------------------------------------------------
481
482
483 def Create_Path(path_name, point_list):         # Format ((0,1,2),
    ↪ (0,2,3))
484     session.Path(name=path_name, type=2, expression=point_list)
```

```
485
486
487 # ------------------------------------------------------------
488
489 # Aquires strain data from path
490 def Create_XYData_From_Path(path_name, path_direction, data_name,
    ↪  step_number, frame_number):
491     pth = session.paths[path_name]
492     session.XYDataFromPath(name=data_name, path=pth,
        ↪  includeIntersections=True, projectOntoMesh=True,
        ↪  pathStyle=PATH_POINTS, numIntervals=10,
        ↪  projectionTolerance=0, shape=UNDEFORMED,
        ↪  labelType=path_direction, removeDuplicateXYPairs=True,
        ↪  includeAllElements=False, step=step_number,
        ↪  frame=frame_number) #variable=('LE', INTEGRATION_POINT,
        ↪  ((COMPONENT, 'LE33'),),)
493
494
495 # ------------------------------------------------------------
496 def Create_Array_Of_Current_Variable_From_Path(path_name, point_list,
    ↪  path_direction, n_frames):
497     Create_Path(path_name, point_list)
498     for i in range(n_frames):
499         Create_XYData_From_Path(path_name, path_direction,
            ↪  path_name+' frame: '+str(i), 2, i)
500         new_strain = np.array(session.xyDataObjects[path_name+'
            ↪  frame: '+str(i)].data)
501         if i == 0:
502             gravity_strain = new_strain
503             strain = new_strain
504             strain[:,1] = np.zeros(np.shape(strain[:,1]))
505         else:
506             strain = np.concatenate((strain,
                ↪  new_strain[:,1,None]-gravity_strain[:,1,None]),
                ↪  axis=1)     # Adds only the element's strain at each
                ↪  time step as a column in the strain array.
507     return strain
508
509 # ------------------------------------------------------------
510
511 ### Abaqus run code
512 def FEM_Main(myDeflection, c, d, myPath, myPathExperimentalData):
513     os.chdir(myPath)
514
515     myModel = 'Beam'
516     mdb.Model(name=myModel)
517
518     myShell = 'Shell_beam'
```

```
519     mySolid = 'Solid_beam'
520
521     myInstanceSolid = 'Solid'
522     myInstanceShell = 'Shell'
523
524     myWidth  = 200
525     myHeight = 190
526     myFlange = 9.529
527     myWeb    = 6.708
528     myRadius = 18
529
530     myLengthShell = 1200
531     myLengthSolid = 600
532
533     del mdb.models['Model-1']
534
535     ### Create parts ###
536     Create_solid_beam(myModel, mySolid, myWidth, myHeight, myFlange,
        ↪  myWeb, myRadius, myLengthSolid)
537     Create_shell_beam(myModel, myShell, myWidth, myHeight, myFlange,
        ↪  myLengthShell)
538
539     ### Partion solid beam into 3 parts ###
540     Cut_1 = Create_Datum_Plane_Part(myModel, mySolid, XZPLANE,
        ↪  (myHeight/2.0 - myFlange - myRadius))
541     Cut_2 = Create_Datum_Plane_Part(myModel, mySolid, XZPLANE,
        ↪  -(myHeight/2.0 - myFlange - myRadius))
542
543     Create_Partition_by_Plane_Part(myModel, mySolid, Cut_1)
544     Create_Partition_by_Plane_Part(myModel, mySolid, Cut_2)
545
546     ### Create material ###
547     myMaterialName = 'S355_web'
548     myMaterialName2 = 'S355_flange'
549     myDensity = 7800e-12    # Tonne/mm^3
550
551     myYoungs_flange = 203580        # MPa
552     myYoungs_web = 201132
553     myPoissons = 0.3
554
555     myPlastic_table_flange =
        ↪  np.loadtxt(myPathExperimentalData+'Plastic_Table_Flange.txt')
556     myPlastic_table_web =
        ↪  np.loadtxt(myPathExperimentalData+'Plastic_Table_Web.txt')
557
558     Create_Material(myModel, myMaterialName, myDensity,
        ↪  myYoungs_flange, myPoissons, myPlastic_table_flange)
```

```
559     Create_Material(myModel, myMaterialName2, myDensity,
        ↪  myYoungs_web, myPoissons, myPlastic_table_web)
560
561     ### Create sections ###
562     Create_shell_section(myModel, 'Shell_flange', myMaterialName2,
        ↪  myFlange)
563     Create_shell_section(myModel, 'Shell_web', myMaterialName, myWeb)
564     Create_solid_section(myModel, 'Solid_flange', myMaterialName2)
565     Create_solid_section(myModel, 'Solid_web', myMaterialName)
566
567     ### Create sections sets for shell part ###
568
569     # For the four flange "parts"
570     Create_Set_Face(myModel, myShell, 'Shell_set_flange_1',
        ↪  -(myWidth-myFlange)/2.0, -(myHeight-myFlange)/2.0,
        ↪  myLengthShell/2.0)
571     Create_Set_Face(myModel, myShell, 'Shell_set_flange_2',
        ↪  (myWidth-myFlange)/2.0, -(myHeight-myFlange)/2.0,
        ↪  myLengthShell/2.0)
572     Create_Set_Face(myModel, myShell, 'Shell_set_flange_3',
        ↪  -(myWidth-myFlange)/2.0, (myHeight-myFlange)/2.0,
        ↪  myLengthShell/2.0)
573     Create_Set_Face(myModel, myShell, 'Shell_set_flange_4',
        ↪  (myWidth-myFlange)/2.0, (myHeight-myFlange)/2.0,
        ↪  myLengthShell/2.0)
574
575     # For the web "part"
576     Create_Set_Face(myModel, myShell, 'Shell_set_web', 0.0, 0.0, 0.0)
577
578     ### Assign sections ###
579     Assign_section(myModel, myShell, 'Shell_set_flange_1',
        ↪  'Shell_flange')
580     Assign_section(myModel, myShell, 'Shell_set_flange_2',
        ↪  'Shell_flange')
581     Assign_section(myModel, myShell, 'Shell_set_flange_3',
        ↪  'Shell_flange')
582     Assign_section(myModel, myShell, 'Shell_set_flange_4',
        ↪  'Shell_flange')
583
584     Assign_section(myModel, myShell, 'Shell_set_web', 'Shell_web')
585
586     ### Assign solid section
587     Create_Set_Cells(myModel, mySolid, 'Solid_set_web', -myWeb,
        ↪  myWeb, -(myHeight/2.0-myFlange-myRadius),
        ↪  (myHeight/2.0-myFlange-myRadius),
588                      -myLengthSolid, myLengthSolid)
589
```

```
590     Create_Set_Cells(myModel, mySolid, 'Solid_set_bottom_flange',
        ↪  -myWidth, myWidth,
591                      -(myHeight/2.0),
                         ↪  -(myHeight/2.0-myFlange-myRadius),
                         ↪  -myLengthSolid, myLengthSolid)
592
593     Create_Set_Cells(myModel, mySolid, 'Solid_set_top_flange',
        ↪  -myWidth, myWidth,
594                      (myHeight/2.0-myFlange-myRadius), (myHeight/2.0),
                         ↪  -myLengthSolid, myLengthSolid)
595
596     Assign_section(myModel, mySolid, 'Solid_set_web', 'Solid_web')
597     Assign_section(myModel, mySolid, 'Solid_set_bottom_flange',
        ↪  'Solid_flange')
598     Assign_section(myModel, mySolid, 'Solid_set_top_flange',
        ↪  'Solid_flange')
599
600     ### Create instance ###
601     Create_instance(myModel, mySolid, myInstanceSolid)
602     Create_instance(myModel, myShell, myInstanceShell)
603
604     ### Translate instance ###
605     Translate_instance(myModel, myInstanceShell, (0.0, 0.0,
        ↪  1.0*myLengthSolid))
606
607     ### Interaction between solid and shell parts ###
608     Shell_to_solid(myModel, myInstanceShell, myInstanceSolid,
        ↪  (myWidth-myFlange)/2.0, (myHeight-myFlange)/2.0,
        ↪  1.0*myLengthSolid, 'Coupling_1', 'Surf_e_1a', 'Surf_f_1')
609
610     ### Datumplane and partition ###
611     Load_plane1 = Create_Datum_Plane(myModel, XYPLANE, 300.0)
612     Create_Partition_by_Plane(myModel, myInstanceSolid, Load_plane1)
613
614     ### Time steps ###
615     Create_Step(myModel, 'RS_equlibrium', 'Initial', 200, 1, 1E-5, 1,
        ↪  ON)
616     Create_Step(myModel, 'Gravity', 'RS_equlibrium', 200, 0.5, 1E-5,
        ↪  0.5, ON)
617     Create_Step(myModel, 'Loading', 'Gravity', 2000, 0.05, 1E-6,
        ↪  0.05, ON)
618
619     ### Vertical support ###
620     # myWidth is multiplied by 0.95 to ensure that the short edge is
        ↪  chosen
621     Create_Set_Edge_instance(myModel, myInstanceShell, 'Support_1a',
        ↪  (0.95*myWidth)/2.0, -(myHeight-myFlange)/2.0,
        ↪  1.0*(myLengthShell+myLengthSolid))
```

```
622     Create_Set_Edge_instance(myModel, myInstanceShell, 'Support_1b',
        ↪  -(0.95*myWidth)/2.0, -(myHeight-myFlange)/2.0,
        ↪  1.0*(myLengthShell+myLengthSolid))
623     Merge_set_in_instance(myModel, myInstanceShell, 'Support_1',
        ↪  ['Support_1a','Support_1b'])
624     Create_Initial_BC(myModel, 'Support_1', 'BC1', UNSET, SET, UNSET,
        ↪  UNSET, UNSET, UNSET)
625
626     ### Horizontal support ###
627     Create_Set_Edge_instance(myModel, myInstanceSolid,
        ↪  'Horizontal_support_a', myWidth/2.0, (myHeight-myFlange)/2,
        ↪  300)
628     Create_Set_Edge_instance(myModel, myInstanceSolid,
        ↪  'Horizontal_support_b', -myWidth/2.0, (myHeight-myFlange)/2,
        ↪  300)
629     Merge_set_in_instance(myModel, myInstanceSolid,
        ↪  'Horizontal_support',
        ↪  ['Horizontal_support_a','Horizontal_support_b'])
630     Create_Initial_BC(myModel, 'Horizontal_support', 'BC_Horizontal',
        ↪  SET, UNSET, UNSET, UNSET, UNSET, UNSET)
631
632     ### Symmetry condition ###
633     Create_Set_Face_instance(myModel, myInstanceSolid, 'Symm_web',
        ↪  0.0, 0.0, 0.0)
634     Create_Set_Face_instance(myModel, myInstanceSolid,
        ↪  'Symm_top_flange', 0.0, (myHeight-myFlange)/2.0, 0.0)
635     Create_Set_Face_instance(myModel, myInstanceSolid,
        ↪  'Symm_bot_flange', 0.0, -(myHeight-myFlange)/2.0, 0.0)
636     Merge_set_in_instance(myModel, myInstanceSolid,'Symm',
        ↪  ['Symm_web','Symm_top_flange','Symm_bot_flange'])
637     assem = mdb.models[myModel].rootAssembly
638     Sym_region = assem.sets['Symm']
639     mdb.models[myModel].ZsymmBC(name='Symmetry condition',
        ↪  createStepName='Initial', region=Sym_region, localCsys=None)
640
641     ### Displacements loads ###
642
643     # Self weigth
644     Create_Gravity(myModel, 'Gravity', 'Gravity')
645
646     # Applied load
647     Create_Set_Edge_instance(myModel, myInstanceSolid, 'Load_1',
        ↪  (myWidth-myFlange)/2.0, (myHeight)/2.0, 300)
648     Create_Displacment(myModel, 'Load_1', 'Displacement', 'Loading',
        ↪  UNSET, myDeflection, UNSET, UNSET, UNSET, UNSET)
649
650     ### Seed and mesh ###
651
```

```
652     # Mesh types
653     Set_Mesh_Type_Solid(myModel, myInstanceSolid, HEX, False, False)
654     Set_Mesh_Type_Shell(myModel, myInstanceShell, TRI)
655 
656     Seed_solid = 5
657     Seed_shell_inner = 5.0
658     Seed_shell_outer = 50.0
659 
660     # Seed edges of shell
661     # For shell instance
662     Seed_edge(myModel, myInstanceShell, Seed_shell_inner,
        ↪  (myWidth-myFlange)/2.0, (myHeight-myFlange)/2.0,
        ↪  1.0*myLengthSolid)
663     Seed_edge(myModel,myInstanceShell, Seed_shell_outer,
        ↪  (myWidth-myFlange)/2.0, (myHeight-myFlange)/2.0,
        ↪  1.0*(myLengthShell+myLengthSolid))
664 
665     # Seed solid instance
666     assem = mdb.models[myModel].rootAssembly
667     partInstances =(assem.instances[myInstanceSolid], )
668     assem.seedPartInstance(regions=partInstances, size=Seed_solid,
        ↪  deviationFactor=0.1, minSizeFactor=0.1)
669 
670     # Generate mesh
671     assem = mdb.models[myModel].rootAssembly
672     partInstances =(assem.instances[myInstanceSolid],
        ↪  assem.instances[myInstanceShell])
673     assem.generateMesh(regions=partInstances)
674 
675     # Create residual stess field
676     myC0 = c
677     myD0 = d
678     [myA0, myB0] = Calculate_RS_Coefficients(myHeight, myWidth,
        ↪  myFlange, myWeb, myC0, myD0)
679     print(myA0,myB0)
680 
681 
682     # RS for topflange
683     Create_Set_Cells_Instance(myModel, myInstanceSolid, 'Top flange
        ↪  inside load', -myWidth/2.0, myWidth/2.0,
        ↪  myHeight/2.0-(myFlange+myRadius), myHeight/2.0, 0,
        ↪  myLengthSolid/2.0)
684     Create_Residual_Stress_In_Subsection(myModel,myInstanceSolid,
        ↪  'Top flange inside load', myLengthSolid/2, Seed_solid,
        ↪  myHeight, myFlange, myA0, myB0, myC0, myD0)
685 
```

```
686     Create_Set_Cells_Instance(myModel, myInstanceSolid, 'Top flange
        ↪  outside load', -myWidth/2.0, myWidth/2.0,
        ↪  myHeight/2.0-(myFlange+myRadius), myHeight/2.0,
        ↪  myLengthSolid/2.0, myLengthSolid)
687     Create_Residual_Stress_In_Subsection(myModel,myInstanceSolid,
        ↪  'Top flange outside load', myLengthSolid/2, Seed_solid,
        ↪  myHeight, myFlange, myA0, myB0, myC0, myD0)
688
689     # RS for bottom flange
690     Create_Set_Cells_Instance(myModel, myInstanceSolid, 'Bottom
        ↪  flange inside load', -myWidth/2.0, myWidth/2.0,
        ↪  -myHeight/2.0, -(myHeight/2.0-(myFlange+myRadius)),  0,
        ↪  myLengthSolid/2.0)
691     Create_Residual_Stress_In_Subsection(myModel,myInstanceSolid,
        ↪  'Bottom flange inside load', myLengthSolid/2, Seed_solid,
        ↪  myHeight, myFlange, myA0, myB0, myC0, myD0)
692
693     Create_Set_Cells_Instance(myModel, myInstanceSolid, 'Bottom
        ↪  flange outside load', -myWidth/2.0, myWidth/2.0,
        ↪  -myHeight/2.0, -(myHeight/2.0-(myFlange+myRadius)),
        ↪  myLengthSolid/2.0, myLengthSolid)
694     Create_Residual_Stress_In_Subsection(myModel,myInstanceSolid,
        ↪  'Bottom flange outside load', myLengthSolid/2, Seed_solid,
        ↪  myHeight, myFlange, myA0, myB0, myC0, myD0)
695
696     # RS for web
697     Create_Set_Cells_Instance(myModel, myInstanceSolid, 'Web inside
        ↪  load', -myWidth/2.0,
        ↪  myWidth/2.0,-(myHeight/2.0-(myFlange+myRadius)),
        ↪  myHeight/2.0-(myFlange+myRadius),   0, myLengthSolid/2.0)
698     Create_Residual_Stress_In_Subsection(myModel,myInstanceSolid,
        ↪  'Web inside load', myLengthSolid/2, Seed_solid, myHeight,
        ↪  myFlange, myA0, myB0, myC0, myD0)
699
700     Create_Set_Cells_Instance(myModel, myInstanceSolid, 'Web outside
        ↪  load', -myWidth/2.0, myWidth/2.0,
        ↪  -(myHeight/2.0-(myFlange+myRadius)),
        ↪  myHeight/2.0-(myFlange+myRadius),  myLengthSolid/2.0,
        ↪  myLengthSolid)
701     Create_Residual_Stress_In_Subsection(myModel,myInstanceSolid,
        ↪  'Web outside load', myLengthSolid/2, Seed_solid, myHeight,
        ↪  myFlange, myA0, myB0, myC0, myD0)
702
703     # Create a job
704     myJobName = 'Seed_'+str(Seed_solid)+'mm'
705     myJob = Create_Job(myModel,myJobName,20)
706     mdb.jobs[myJobName].submit(consistencyChecking=OFF)
707     myJob.waitForCompletion()
```

```
708
709
710
711     # Post processing --------------
712
713     myOdbPath = myPath+myJobName+'.odb'
714     myOdb = session.openOdb(myOdbPath)
715     session.viewports['Viewport: 1'].setValues(displayedObject=myOdb)
716
717     # Sets the current variable to longitudinal strain
718     session.viewports['Viewport:
        ↪ 1'].odbDisplay.setPrimaryVariable(variableLabel='LE',
        ↪ outputPosition=INTEGRATION_POINT, refinement=(COMPONENT,
        ↪ 'LE33'), )
719     session.viewports['Viewport:
        ↪ 1'].odbDisplay.display.setValues(plotState=(CONTOURS_ON_DEF,
        ↪ ))
720     n_frames = len(myOdb.steps['Loading'].frames)
721
722     increment_time = []
723     for i in range(n_frames):
724
            ↪ increment_time.append(myOdb.steps['Loading'].frames[i].frameValue)
725
726     # Strain on web and on the outer side of flanges
727     topsideTopFlangeStrain =
        ↪ Create_Array_Of_Current_Variable_From_Path('Topside top
        ↪ flange', ((-myWidth/2.0, myHeight/2.0, 0), (myWidth/2.0,
        ↪ myHeight/2.0, 0)), X_COORDINATE, n_frames)
728     botsideBotFlangeStrain =
        ↪ Create_Array_Of_Current_Variable_From_Path('Botside bot
        ↪ flange', ((-myWidth/2.0, -myHeight/2.0, 0), (myWidth/2.0,
        ↪ -myHeight/2.0, 0)), X_COORDINATE, n_frames)
729     rightWebStrain =
        ↪ Create_Array_Of_Current_Variable_From_Path('Rightside Web',
        ↪ ((myWeb/2.0, -(myHeight/2.0-myFlange-myRadius), 0),
        ↪ (myWeb/2.0, (myHeight/2.0-myFlange-myRadius), 0)),
        ↪ Y_COORDINATE, n_frames)
730     leftWebStrain =
        ↪ Create_Array_Of_Current_Variable_From_Path('Leftside Web',
        ↪ ((-myWeb/2.0, -(myHeight/2.0-myFlange-myRadius), 0),
        ↪ (-myWeb/2.0, (myHeight/2.0-myFlange-myRadius), 0)),
        ↪ Y_COORDINATE, n_frames)
731
732     # Strains on the inner side of the flanges
```

```
733     leftTopsideBotFlangeStrain =
        ↪  Create_Array_Of_Current_Variable_From_Path('Left topside bot
        ↪  flange', ((-myWidth/2.0, -myHeight/2.0+myFlange, 0),
        ↪  (-(myWeb/2.0+myRadius), -myHeight/2.0+myFlange, 0)),
        ↪  X_COORDINATE, n_frames)
734     leftBotsideTopFlangeStrain =
        ↪  Create_Array_Of_Current_Variable_From_Path('Left botside top
        ↪  flange', ((-myWidth/2.0, myHeight/2.0-myFlange, 0),
        ↪  (-(myWeb/2.0+myRadius), myHeight/2.0-myFlange, 0)),
        ↪  X_COORDINATE, n_frames)
735     rightTopsideBotFlangeStrain =
        ↪  Create_Array_Of_Current_Variable_From_Path('Right topside bot
        ↪  flange', ((myWeb/2.0+myRadius, -myHeight/2.0+myFlange, 0),
        ↪  (myWidth/2.0, -myHeight/2.0+myFlange, 0)), X_COORDINATE,
        ↪  n_frames)
736     rightBotsideTopFlangeStrain =
        ↪  Create_Array_Of_Current_Variable_From_Path('Right botside top
        ↪  flange', ((myWeb/2.0+myRadius, myHeight/2.0-myFlange, 0),
        ↪  (myWidth/2.0, myHeight/2.0-myFlange, 0)), X_COORDINATE,
        ↪  n_frames)
737
738     output_format = '%2.7e'
739
740     # Saves the strain arrays obtained from abaqus
741     # np.savetxt(myPath + 'Botside_Bot_Flange_Strain.txt',
        ↪  botsideBotFlangeStrain, delimiter=',', fmt=output_format)
742     # np.savetxt(myPath + 'Topside_Top_Flange_Strain.txt',
        ↪  topsideTopFlangeStrain, delimiter=',', fmt=output_format)
743     # np.savetxt(myPath + 'Right_Web_Strain.txt', rightWebStrain,
        ↪  delimiter=',', fmt=output_format)
744     # np.savetxt(myPath + 'Left_Web_Strain.txt', leftWebStrain,
        ↪  delimiter=',', fmt=output_format)
745
746     # np.savetxt(myPath + 'Left_Topside_Bot_Flange_Strain.txt',
        ↪  leftTopsideBotFlangeStrain, delimiter=',', fmt=output_format)
747     # np.savetxt(myPath + 'Left_Botside_Top_Flange_Strain.txt',
        ↪  leftBotsideTopFlangeStrain, delimiter=',', fmt=output_format)
748     # np.savetxt(myPath + 'Right_Topside_Bot_Flange_Strain.txt',
        ↪  rightTopsideBotFlangeStrain, delimiter=',',
        ↪  fmt=output_format)
749     # np.savetxt(myPath + 'Right_Botside_Top_Flange_Strain.txt',
        ↪  rightBotsideTopFlangeStrain, delimiter=',',
        ↪  fmt=output_format)
750
751     np.savetxt(myPath + 'Time_Increments.txt', increment_time,
        ↪  delimiter=',', fmt=output_format)
752     np.savetxt(myPath + 'RS_Coefficients.txt', [myA0, myB0, myC0,
        ↪  myD0], delimiter=',')
```

```
753     return topsideTopFlangeStrain, botsideBotFlangeStrain,
        ↪ rightWebStrain, leftWebStrain, leftTopsideBotFlangeStrain,
        ↪ leftBotsideTopFlangeStrain, rightTopsideBotFlangeStrain,
        ↪ rightBotsideTopFlangeStrain, increment_time
754
755
756 # ------------------------------------------------------------
757
758
759 def Create_Interpolated_Strain_At_Right_Coord(FEM_data, test_coord):
760     compare_list = []
761     compare_list.append(test_coord)
762     k = 0
763     # Looks through the FEM strain data array to find the two
        ↪ coordinates that are closest to the test coordinate
764     # Then it interpolates the strain data between the two
        ↪ coordinates to get data that should be similar to the
        ↪ coordinate inbetween
765     # If the exact coordinate is available in the FEM_data, it
        ↪ directly picks that
766     for x in FEM_data[:,0]:
767         cur_sign = np.sign(x-test_coord)
768         if k == 0:
769             initial_sign = cur_sign
770         if cur_sign == 0:
771             compare_list += list(FEM_data[k,1:])
772             return compare_list
773         elif cur_sign != initial_sign:
774             for i in range(1,len(FEM_data[k,:])):
775                 compare_list.append(np.interp(test_coord,
                    ↪ FEM_data[k-1:k+1,0], FEM_data[k-1:k+1,i]))
776             return compare_list
777         k += 1
778
779 # ------------------------------------------------------------
780
781 def Create_Interpolated_Strain_To_Right_Time(test_time, FEM_time,
    ↪ test_Strain_List):
782     compare_list = []
783     compare_list.append(test_Strain_List[0])
784
785     # Looks through the test strain data array to find the two
        ↪ timesteps that are closest to the FEM timestep
786     # Then it interpolates the strain data between the two times to
        ↪ get data that should be similar to the FEM timestep
787     # If the exact time is available in the FEM_data, it directly
        ↪ picks that value instead
788     for F_t in FEM_time:
```

```
789         k = 0
790         for t in test_time:
791             cur_sign = np.sign(t-F_t)
792             if k == 0:
793                 intial_sign = cur_sign
794             if cur_sign == 0:
795                 compare_list.append(test_Strain_List[k+1])
796                 break
797             elif cur_sign != intial_sign:
798                 compare_list.append(np.interp(F_t,
                    ↪  test_time[k-1:k+1], test_Strain_List[k:k+2]))
799                 break
800             k +=1
801     return(compare_list)
802 # ------------------------------------------------------------
803 # Calculates error by comparing data from physical test with abaqus
    ↪  data
804 # The FEM data is interpolated to fit the experiments coordinates and
805 # the experimental data is interpolated to fit the FEM datas
    ↪  timesteps
806 #
807 def Calculate_Error(experiment_data_file_name, file_folder_path,
    ↪  exp_time, FEM_data, FEM_time):
808     exp_data = np.loadtxt(file_folder_path+experiment_data_file_name,
        ↪  delimiter=',', dtype=float)
809
810     shape_of_exp_data = exp_data.shape
811     shape_of_FEM_data = FEM_data.shape
812
813     FEM_Data_For_Comparison = np.zeros((shape_of_exp_data[0],
        ↪  shape_of_FEM_data[1]))   # Shape = (Number of coords from
        ↪  experiments, number of time steps from FEM+1)
814     for i in range(shape_of_exp_data[0]):
815         FEM_Data_For_Comparison[i,:] =
            ↪  Create_Interpolated_Strain_At_Right_Coord(FEM_data,
            ↪  exp_data[i,0])
816
817     #
        ↪  np.savetxt('C:/Users/qtv833/Test_run_local/FEM_for_compare.txt',FEM_Data_For_
        ↪  delimiter=',',fmt='%4.4f')
818
819     exp_data_For_Comparison =
        ↪  np.zeros((shape_of_exp_data[0],shape_of_FEM_data[1]))
820     for i in range(shape_of_exp_data[0]):
821         exp_data_For_Comparison[i,:] =
            ↪  Create_Interpolated_Strain_To_Right_Time(exp_time,
            ↪  FEM_time, exp_data[i,:])
822
```

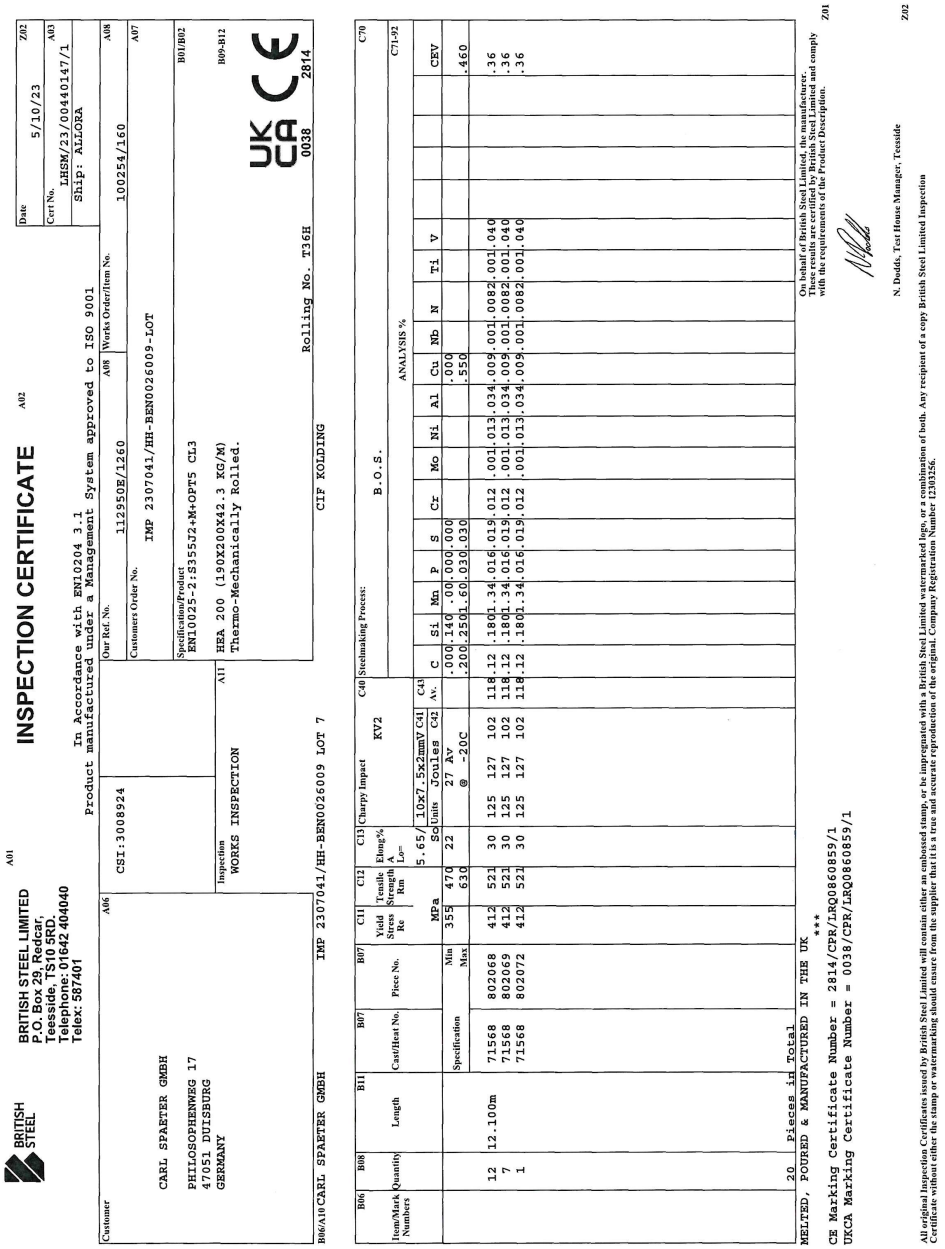
```
823     #
        ↪ np.savetxt('C:/Users/qtv833/Test_run_local/Test_for_compare.txt',exp_data_For
        ↪ delimiter=',',fmt='%4.4f')
824
825     diff =
        ↪ abs(FEM_Data_For_Comparison[:,1:]-exp_data_For_Comparison[:,1:])
826     # diff =
        ↪ abs(FEM_Data_For_Comparison[:,-1]-exp_data_For_Comparison[:,-1])
827     error = np.sum(diff)
828     return error
829
830
831 #
    ↪ ------------------------------------------------------------------------------
832
833
834
835
836
837
838 # Run FEM_Main with input c and d
839 c = float(sys.argv[-2])
840 d = float(sys.argv[-1])
841 myDeflection = float(sys.argv[-3])
842 myPath = str(sys.argv[-5])
843 myPathExperimentalData = str(sys.argv[-4])
844
845 topsideTopFlangeStrain, botsideBotFlangeStrain, rightWebStrain,
    ↪ leftWebStrain, leftTopsideBotFlangeStrain,
    ↪ leftBotsideTopFlangeStrain, rightTopsideBotFlangeStrain,
    ↪ rightBotsideTopFlangeStrain, FEM_increments =
    ↪ FEM_Main(myDeflection,c,d, myPath, myPathExperimentalData)
846
847
848 exp_time =
    ↪ np.loadtxt(myPathExperimentalData+'Experimental_Time_Increments.txt',
    ↪ delimiter=',', dtype=float)
849
850 error = Calculate_Error('Experimental_Bot_Flange_Strain.txt',
    ↪ myPathExperimentalData, exp_time, botsideBotFlangeStrain,
    ↪ FEM_increments)
851 error += Calculate_Error('Experimental_Top_Flange_Strain.txt',
    ↪ myPathExperimentalData, exp_time, topsideTopFlangeStrain,
    ↪ FEM_increments)
852 error += Calculate_Error('Experimental_Left_Web_Strain.txt',
    ↪ myPathExperimentalData, exp_time, leftWebStrain, FEM_increments)
853 error += Calculate_Error('Experimental_Right_Web_Strain.txt',
    ↪ myPathExperimentalData, exp_time, rightWebStrain, FEM_increments)
```

```
854
855 error +=
    ↪ Calculate_Error('Experimental_Left_Topside_Bot_Flange_Strain.txt',
    ↪ myPathExperimentalData, exp_time, leftTopsideBotFlangeStrain,
    ↪ FEM_increments)
856 error +=
    ↪ Calculate_Error('Experimental_Left_Botside_Top_Flange_Strain.txt',
    ↪ myPathExperimentalData, exp_time, leftBotsideTopFlangeStrain,
    ↪ FEM_increments)
857 error +=
    ↪ Calculate_Error('Experimental_Right_Topside_Bot_Flange_Strain.txt',
    ↪ myPathExperimentalData, exp_time, rightTopsideBotFlangeStrain,
    ↪ FEM_increments)
858 error +=
    ↪ Calculate_Error('Experimental_Right_Botside_Top_Flange_Strain.txt',
    ↪ myPathExperimentalData, exp_time, rightBotsideTopFlangeStrain,
    ↪ FEM_increments)
859
860
861 np.savetxt(myPath+'Error.txt', [error], delimiter=',', fmt='%3.8f')
```

E

Steel specification from supplier

HEA200 specification



BRITISH STEEL

BRITISH STEEL LIMITED
P.O. Box 29, Redcar,
Teesside, TS10 5RD.
Telephone: 01642 404040
Telex: 587401

A01

INSPECTION CERTIFICATE A02

In Accordance with EN10204 3.1
Product manufactured under a Management System approved to ISO 9001

Date 5/10/23 Z02
Cert No. LHSM/23/00440147/1 A03
Ship: ALLORA

Customer A06
CARL SPAETER GMBH

PHILOSOPHENWEG 17
47051 DUISBURG
GERMANY

CSI:3008924

Our Ref. No. 112950E/1260 A08
Works Order/Item No. 100254/160 A08
Customers Order No. IMP 2307041/HH-BEN0026009-LOT A07

Specification/Product B01/B02
EN10025-2:S355J2+M+OPT5 CL3

HEA 200 (190X200X42.3 KG/M) B09-B12
Thermo-Mechanically Rolled.

Inspection A11
WORKS INSPECTION

UK CA CE 0038 2814

Rolling No. T36H

B06/A10 CARL SPAETER GMBH IMP 2307041/HH-BEN0026009 LOT 7 CIF KOLDING

Item/Mark Numbers	B06	Quantity B08	Length B11	Cast/Heat No. B07	Piece No. B07	Yield Stress Re MPa	Tensile Strength Rm MPa	Elong% A Lo= 5.65/So	Charpy Impact Units	10x7.5x2mmV Joules 27 Av @ -20C	C41 C42	KV2	C43 Av.	C40 Steelmaking Process: B.O.S.	C	Si	Mn	P	S	Cr	Mo	Ni	Al	Cu	Nb	N	Ti	V			C70 C71-92 CEV
				Specification		Min	355	470	22	30	125	127	102	118	ANALYSIS %	.000	.140	.00	.000	.000					.000							
						Max		630								.200	.250	1.60	.030	.030					.550							.460
		12	12.100m	71568	802068		412	521	30	125	127	102	118	12		.180	1.34	.016	.019	.012		.001	.013	.034	.009	.001	.0082	.001	.040			.36
		7		71568	802069		412	521	30	125	127	102	118	12		.180	1.34	.016	.019	.012		.001	.013	.034	.009	.001	.0082	.001	.040			.36
		1		71568	802072		412	521	30	125	127	102	118	12		.180	1.34	.016	.019	.012		.001	.013	.034	.009	.001	.0082	.001	.040			.36
		20	Pieces in Total																												

MELTED, POURED & MANUFACTURED IN THE UK ***

CE Marking Certificate Number = 2814/CPR/LRQ0860859/1
UKCA Marking Certificate Number = 0038/CPR/LRQ0860859/1

On behalf of British Steel Limited, the manufacturer.
These results are certified by British Steel Limited and comply with the requirements of the Product Description. Z01

N. Dodds, Test House Manager, Teesside Z02

All original Inspection Certificates issued by British Steel Limited will contain either an embossed stamp, or be impregnated with a British Steel Limited watermarked logo, or a combination of both. Any recipient of a copy British Steel Limited Inspection Certificate without either the stamp or watermarking should ensure from the supplier that it is a true and accurate reproduction of the original. Company Registration Number 1230256.

HEB200 specification

SWT
STAHLWERK THÜRINGEN

A01 Stahlwerk Thüringen GmbH
A05 Quality Assurance Department
Kronacher Straße 6
07333 Unterwellenborn
Germany

A08 Our Order No.: 20028830
A07 Your Order No.: 300428-2

B02 Quality: S355J2+M
according to: EN 10025-2/2019

A03 Certificate No. 314/10-2023 1/3
Advice No. 800157892

A02 Inspection Certificate 3.1

according to EN 10204:2004/3.1

SWT

A04
A06

BE GROUP Sverige AB
Krusegatan 19b
SE-212 25 Malmoe

Pos. A10	Heat No. B07	Section B01	According to	Length B09	Pieces B08	Theoretical Weight B12
5	72981	HE 200B	EN 10365	10.100 mm	8	4.953 kg
5	72982	HE 200B	EN 10365	10.100 mm	8	4.953 kg
6	73047	HE 180B	EN 10365	10.100 mm	18	9.308 kg
15	72981	HE 200B	EN 10365	12.100 mm	22	16.317 kg
16	73003	HE 200A	EN 10365	10.100 mm	36	15.380 kg

Heat Analysis [%]

Heat No. B07	C C71	Si C72	Mn C73	P C74	S C75	N C76	Al C77	Nb C78	V C79	Cr C80
max	0,20	0,25	1,60	0,030	0,030					0,29
min		0,14								
72981	0,08	0,19	1,42	0,019	0,022	0,010	0,013	0,040	0,008	0,08
72982	0,08	0,18	1,41	0,022	0,019	0,010	0,015	0,038	0,007	0,08
73003	0,08	0,18	1,44	0,019	0,020	0,010	0,015	0,040	0,008	0,08
73047	0,08	0,18	1,45	0,018	0,019	0,012	0,014	0,041	0,008	0,06

Heat Analysis [%]

Heat No. B07	Cu C81	Ni C82	Mo C83	Ti C84	CEV C93
max	0,55	0,42	0,11		0,46
min					
72981	0,34	0,12	0,03	0,021	0,37
72982	0,23	0,12	0,02	0,024	0,36
73003	0,29	0,18	0,04	0,021	0,38
73047	0,23	0,12	0,02	0,022	0,36

Z03 Works inspector
René Merbach

Z02 05.10.2023

Z04

CE
0769
23
032 CPR 2020-10-01 SWT

SWT
STAHLWERK THÜRINGEN

A01 Stahlwerk Thüringen GmbH
A05 Quality Assurance Department
Kronacher Straße 6
07333 Unterwellenborn
Germany

A08 Our Order No.: 20028830
A07 Your Order No.: 300428-2

B02 Quality: S355J2+M
according to: EN 10025-2/2019

A03 Certificate No. 314/10-2023 2/3
Advice No. 800157892
A02 Inspection Certificate 3.1

according to EN 10204:2004/3.1

SWT

A04
A06

BE GROUP Sverige AB
Krusegatan 19b
SE-212 25 Malmoe

Tensile test (ISO 6892-1)

Heat No. B07	Yield stress [N/mm2] C11	Tensile strength [N/mm2] C12	Elongation $5.65\sqrt{S_o}$ [%] C13	C14
max		630		
min	355	470	22,0	
72981	431	520	32,0	
72982	422	513	33,0	
73003	393	543	27,0	
73047	396	513	31,0	

Impact test (ISO 148-1)

Heat No. B07	Type/Shape C40/C41	Direction C02	Temperature C03	Requirement 1,2,3/M	1 [J] C42	2 C42	3 C42	M C43
72981	KV 450	L	-20 °C	19 / 27	143	116	129	129
72982	KV 450	L	-20 °C	19 / 27	137	133	146	138
73003	KV 450/7.5	L	-20 °C	14 / 20	99	44	63	68
73047	KV 450	L	-20 °C	19 / 27	116	140	136	130

Z01

- electric arc melting
- ladle refined, fine grained and fully killed

Material number: 1.0577

Surface condition according to DIN EN 10163-3, class C, subclass 1

Material for galvanization. Si: 0.14% - 0.25%

General technical terms of delivery for hot rolled structural steel products: EN 10025-1
Intended uses: Welded, bolted and riveted structures
Expressed as indicated in the DoP:
Tolerances / Tensile strength / Yield strength / Elongation
Impact strength / Weldability / Durability

Z03 Works inspector
René Merbach

Z02 05.10.2023

Z04

CE

0769
23
032 CPR 2020-10-01 SWT

F

Optimisation script

```python
import os
import numpy as np
from scipy.optimize import minimize
import datetime

# To change before running script
# # In OS_test
# - Deflection
# - Folder
# # In Shell_to_solid
# - Youngs modulus
# - Beam dimensions


c0 = -100
d0 = 0.02
myDeflection = -29.360
x0 = np.array([c0, d0])

# simplex0 = np.array([[0, 0], [1*c0, 0], [0, 1.5*d0]])
# simplex0 = np.array([[0, 0], [1*c0, 1*d0], [1*c0, 0]])
simplex0 = np.array([[0, 0], [1*c0, 1*d0], [0, 1.5*d0]])

myPath = 'C:/Users/qtv833/Test_run_local/'
myPathExperimentalData = 'C:/Users/qtv833/Test_run_local/HEA2/'

input_tolerance = 0.01
error_tolerance = 0.0001




def minimize_this(x, info):
    c = x[0]
    d = x[1]

    # Runs abaqus script with values for c and d
```

```
38      os.system('abaqus cae noGUI=Shell_to_solid_test_v5.py --
        ↪  '+myPath+' '+myPathExperimentalData+' '+str(myDeflection)+'
        ↪  '+str(c)+' '+ str(d))
39      error = np.loadtxt(myPath+'Error.txt', dtype=float,
        ↪  delimiter=',')
40      RS_coefficients = np.loadtxt(myPath+'RS_Coefficients.txt',
        ↪  dtype=float, delimiter=',')
41
42      # Prints current values and saves them
43      print('Time, Iter, a, b, c, d, error')
44      print(datetime.datetime.now().time(), info['N_iter'],
        ↪  RS_coefficients[0], RS_coefficients[1], RS_coefficients[2],
        ↪  RS_coefficients[3], float(error))
45      info['iter_list'].append(info['N_iter'])
46      info['a'].append(RS_coefficients[0])
47      info['b'].append(RS_coefficients[1])
48      info['c'].append(RS_coefficients[2])
49      info['d'].append(RS_coefficients[3])
50      info['error'].append(error)
51      info['N_iter']+=1
52      np.savetxt(myPath+'Iteration_History.txt',
        ↪  np.transpose([info['iter_list'], info['a'], info['b'],
        ↪  info['c'], info['d'], info['error']]), delimiter=',',
        ↪  header='Iteration, a, b, c, d, error')
53      return error
54
55
56
57  res = minimize(minimize_this, x0, args={'N_iter':0, 'iter_list':[],
    ↪  'a': [], 'b': [], 'c': [], 'd': [], 'error': []},),
    ↪  method='Nelder-Mead', options = {'xatol': input_tolerance,
    ↪  'fatol': error_tolerance, 'disp':True,
    ↪  'initial_simplex':simplex0})
58  print(res.x)
59
60
```

G

Beam dimensions

HEA200-1

Top flange

Position [mm]	1900	2000	2100	Average
Left thickness at 1cm [mm]	8.7	8.7	8.7	8.7
Left thickness at 4cm [mm]	9.15	9.1	9.2	9.15
Left thickness at 7cm [mm]	9.7	9.6	9.7	9.666667
Right thickness at 7cm [mm]	9.7	9.8	9.8	9.766667
Right thickness at 4cm [mm]	9.7	9.7	9.7	9.7
Right thickness at 1cm [mm]	9.95	10	9.8	9.916667
Average	9.483333	9.483333	9.483333	**9.483333**

Bottom flange

Position [mm]	1900	2000	2100	Average
Left thickness at 1cm [mm]	9	9	9	9
Left thickness at 4cm [mm]	9.4	9.3	9.4	9.366667
Left thickness at 7cm [mm]	9.6	9.6	9.65	9.616667
Right thickness at 7cm [mm]	9.9	9.9	9.95	9.916667
Right thickness at 4cm [mm]	9.7	9.8	9.8	9.766667
Right thickness at 1cm [mm]	9.7	9.7	9.7	9.7
Average	9.55	9.55	9.583333	**9.561111**

Web

Position [mm]	-40	center	40	Average
Thickness Left [mm]	6.8	6.9	6.7	6.8
Thickness Right [mm]	6.7	6.8	6.75	6.75
Average	6.75	6.85	6.725	**6.775**

HEA200-2

Top flange

Position [mm]	1900	2000	2100	Average
Left thickness at 1mm [mm]	9.7	9.8	9.8	9.766667
Left thickness at 4mm [mm]	9.8	9.8	9.8	9.8
Left thickness at 7mm [mm]	10	10	10	10
Right thickness at 7mm [mm]	9.8	9.8	9.75	9.783333
Right thickness at 4mm [mm]	9.35	9.5	9.4	9.416667
Right thickness at 1mm [mm]	9	9	9.1	9.033333
Average	9.608333	9.65	9.641667	**9.633333**

Bottom flange

Position [mm]	1900	2000	2100	Average
Left thickness at 1.5mm [mm]	9.8	9.9	9.85	9.85
Left thickness at 4mm [mm]	9.7	9.75	9.7	9.716667
Left thickness at 6.5mm [mm]	9.8	9.75	9.75	9.766667
Right thickness at 6.5mm [mm]	9.5	9.4	9.5	9.466667
Right thickness at 4mm [mm]	9.1	9.1	9	9.066667
Right thickness at 1.5mm [mm]	8.7	8.65	8.7	8.683333
Average	9.433333	9.425	9.416667	**9.425**

Web

Position [mm]	-40	center	40	Average
Thickness Left [mm]	6.7	6.8	6.8	6.766667
Thickness Right [mm]	6.7	6.7	6.55	6.65
Average	6.7	6.75	6.675	**6.708333**

HEA200-3

Top flange

Position [mm]	1900	2000	2100	Average
Left thickness at 1mm [mm]	9.6	9.65	9.7	9.65
Left thickness at 4mm [mm]	9.8	9.8	9.85	9.816667
Left thickness at 7mm [mm]	9.9	9.9	9.9	9.9
Right thickness at 7mm [mm]	9.7	9.7	9.7	9.7
Right thickness at 4mm [mm]	9.4	9.4	9.4	9.4
Right thickness at 1mm [mm]	9	9	9.1	9.033333
Average	9.566667	9.575	9.608333	**9.583333**

Bottom flange

Position [mm]	1900	2000	2100	Average
Left thickness at 1mm [mm]	9.9	9.9	9.85	9.883333
Left thickness at 4mm [mm]	9.6	9.7	9.7	9.666667
Left thickness at 7mm [mm]	9.7	9.75	9.7	9.716667
Right thickness at 7mm [mm]	9.5	9.6	9.6	9.566667
Right thickness at 4mm [mm]	9.1	9.1	9.1	9.1
Right thickness at 1mm [mm]	8.5	8.6	8.6	8.566667
Average	9.383333	9.441667	9.425	**9.416667**

Web

Position [mm]	-40	center	40	Average
Thickness Left [mm]	6.6	6.8	6.7	6.7
Thickness Right [mm]	6.8	6.8	6.8	6.8
Average	6.7	6.8	6.75	**6.75**

HEB200-1

Top flange

Position [mm]	1900	2000	2100	Average
Left thickness at 1mm [mm]	14.4	14.5	14.55	14.48333
Left thickness at 4mm [mm]	14.7	14.8	14.6	14.7
Left thickness at 7mm [mm]	15.1	15.1	15.1	15.1
Right thickness at 7mm [mm]	15.35	15.3	15.3	15.31667
Right thickness at 4mm [mm]	15.2	15.5	15	15.23333
Right thickness at 1mm [mm]	15	14.9	14.85	14.91667
Average	14.95833	15.01667	14.9	**14.95833**

Bottom flange

Position [mm]	1900	2000	2100	Average
Left thickness at 1mm [mm]	14.6	14.6	14.6	14.6
Left thickness at 4mm [mm]	14.75	14.9	14.9	14.85
Left thickness at 7mm [mm]	15.3	15.3	15.3	15.3
Right thickness at 7mm [mm]	15.1	15.15	15.15	15.13333
Right thickness at 4mm [mm]	14.9	14.9	14.9	14.9
Right thickness at 1mm [mm]	14.8	14.8	14.75	14.78333
Average	14.90833	14.94167	14.93333	**14.92778**

Web

Position [mm]	-40	center	40	Average
Thickness Left [mm]	9.1	9	9.1	9.066667
Thickness Right [mm]	9	8.95	9	8.983333
Average	9.05	8.975	9.05	**9.025**

HEB200-2

Top flange

Position [mm]	1900	2000	2100	Average
Left thickness at 1mm [mm]	15	14.95	15	14.98333
Left thickness at 4mm [mm]	15.1	15.1	15.1	15.1
Left thickness at 7mm [mm]	15.4	15.3	15.3	15.33333
Right thickness at 7mm [mm]	15.1	15.1	15.1	15.1
Right thickness at 4mm [mm]	14.7	14.7	14.7	14.7
Right thickness at 1mm [mm]	14.6	14.5	14.5	14.53333
Average	14.98333	14.94167	14.95	**14.95833**

Bottom flange

Position [mm]	1900	2000	2100	Average
Left thickness at 1mm [mm]	14.85	14.8	14.8	14.81667
Left thickness at 4mm [mm]	14.95	14.9	14.9	14.91667
Left thickness at 7mm [mm]	15.2	15.1	15.2	15.16667
Right thickness at 7mm [mm]	15.3	15.4	15.3	15.33333
Right thickness at 4mm [mm]	14.9	14.8	14.85	14.85
Right thickness at 1mm [mm]	14.6	14.6	14.6	14.6
Average	14.96667	14.93333	14.94167	**14.94722**

Web

Position [mm]	-40	center	40	Average
Thickness Left [mm]	9.1	9	9.1	9.066667
Thickness Right [mm]	9.1	9	9.2	9.1
Average	9.1	9	9.15	**9.083333**

HEB200-3

Top flange

Position [mm]	1900	2000	2100	Average
Left thickness at 1mm [mm]	15	15	15	15
Left thickness at 4mm [mm]	15.1	15.1	15.1	15.1
Left thickness at 7mm [mm]	15.3	15.4	15.4	15.36667
Right thickness at 7mm [mm]	15.2	15.1	15.1	15.13333
Right thickness at 4mm [mm]	14.7	14.8	14.75	14.75
Right thickness at 1mm [mm]	14.5	14.55	14.5	14.51667
Average	14.96667	14.99167	14.975	**14.97778**

Bottom flange

Position [mm]	1900	2000	2100	Average
Left thickness at 1mm [mm]	14.8	14.75	14.8	14.78333
Left thickness at 4mm [mm]	14.95	14.9	14.95	14.93333
Left thickness at 7mm [mm]	15.2	15.2	15.15	15.18333
Right thickness at 7mm [mm]	15.4	15.35	15.3	15.35
Right thickness at 4mm [mm]	14.8	14.9	14.9	14.86667
Right thickness at 1mm [mm]	14.6	14.6	14.6	14.6
Average	14.95833	14.95	14.95	**14.95278**

Web

Position [mm]	-40	center	40	Average
Thickness Left [mm]	9	9	9.1	9.033333
Thickness Right [mm]	9.1	9	9.1	9.066667
Average	9.05	9	9.1	**9.05**

H

Comparison of final strain curves

In the following pictures the strain curves obtained from Abaqus with the final values of residual stress coefficients as input are compared to the strain curves from the physical tests. The difference between the curve pairs are used to calculate the final error. In addition, figures showing the difference in deflection over time for Abaqus and the physical tests.

HEA1

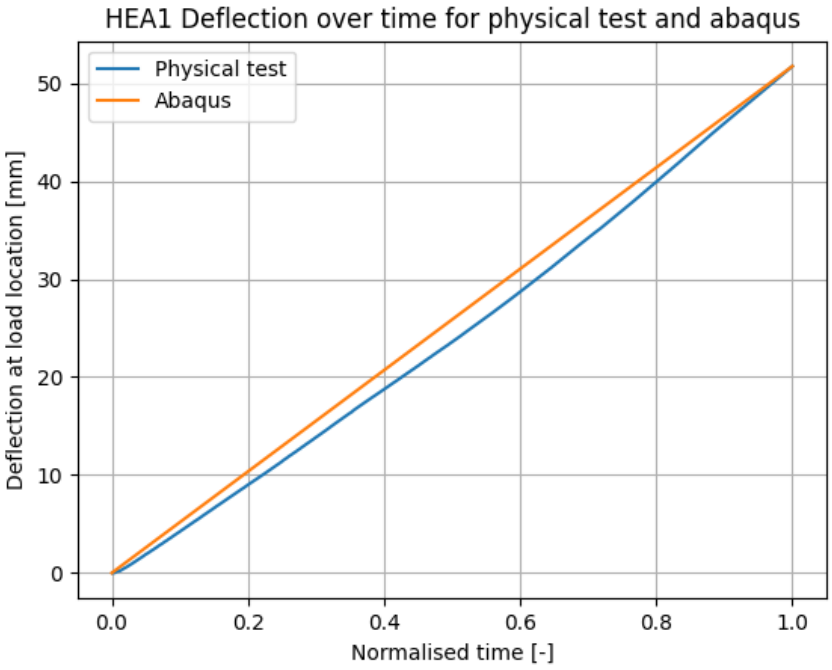


Figure H.1: HEA1 deflection in abaqus and physical test

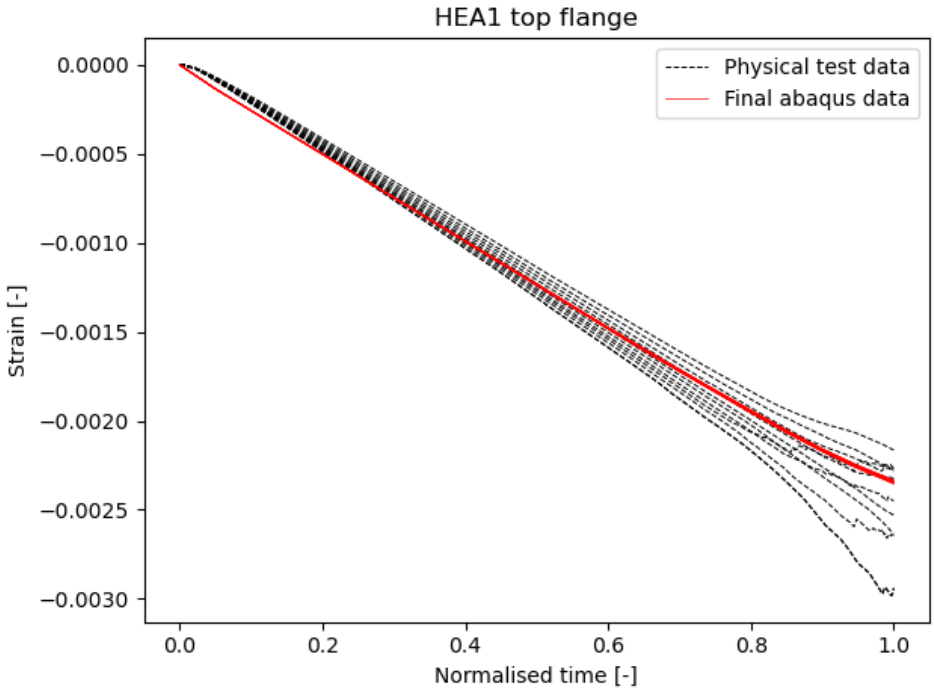


Figure H.2: HEA1 top flange

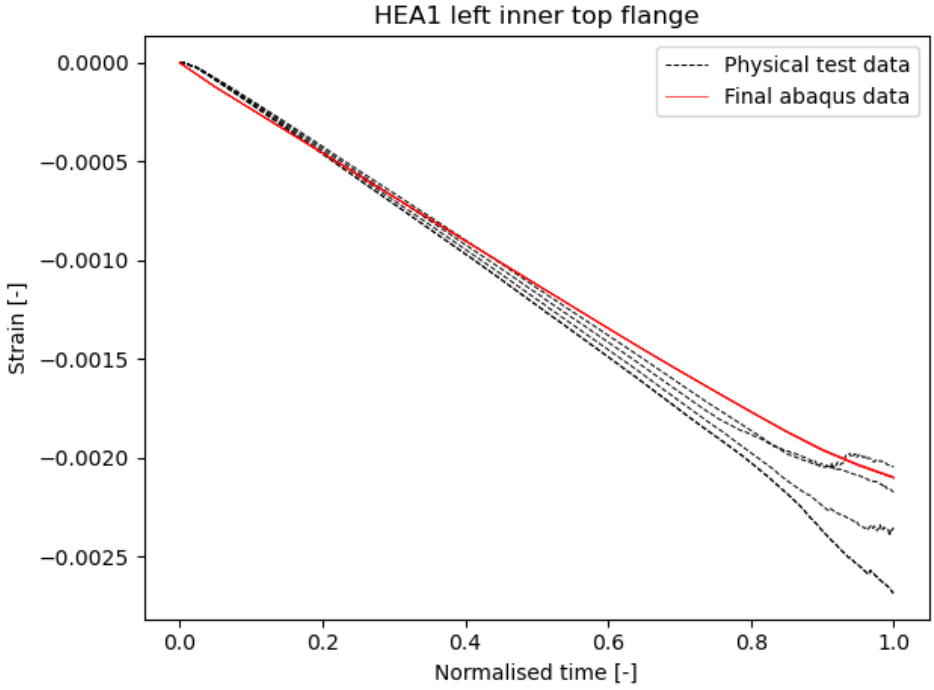


Figure H.3: HEA1 left inner top flange

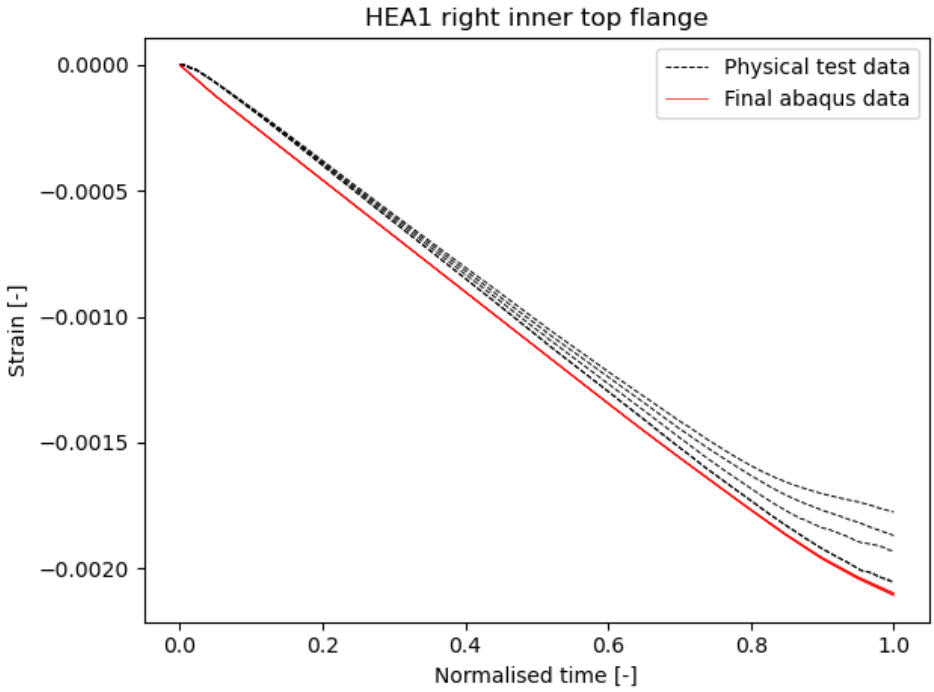


Figure H.4: HEA1 right inner top flange

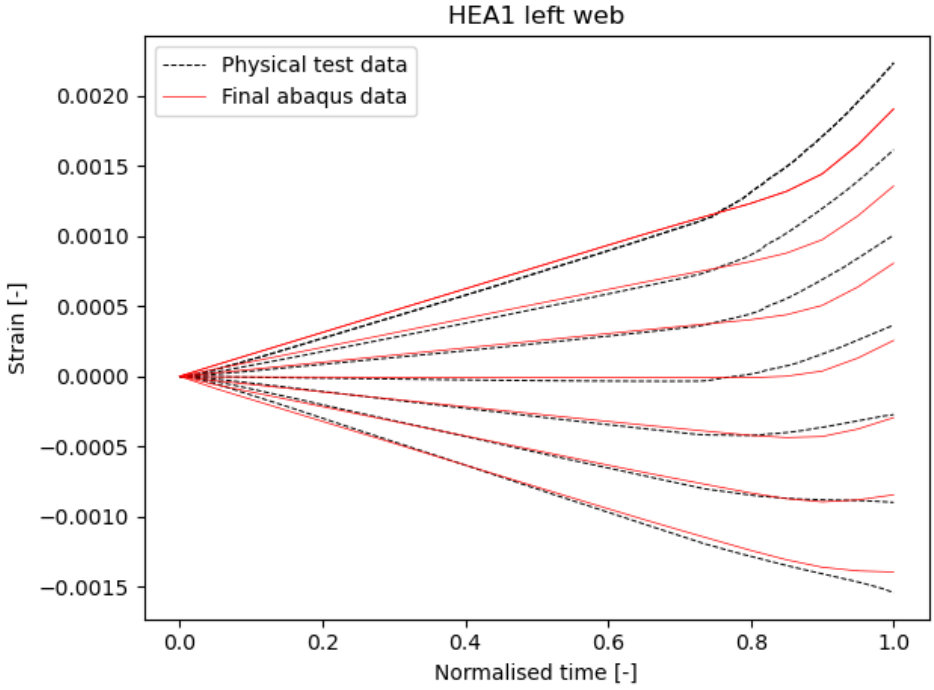


Figure H.5: HEA1 left web

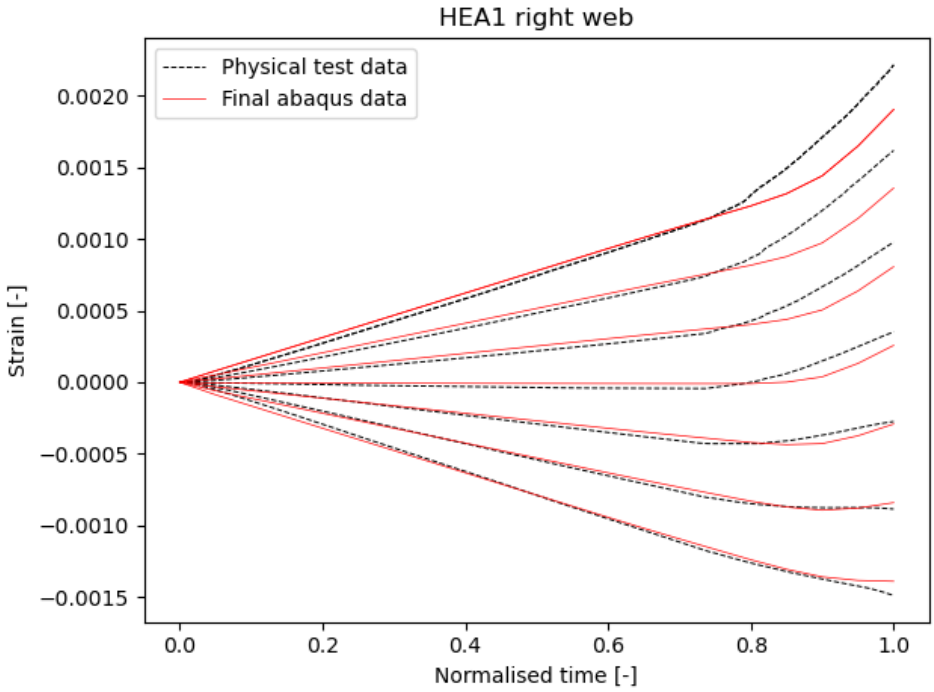


Figure H.6: HEA1 right web

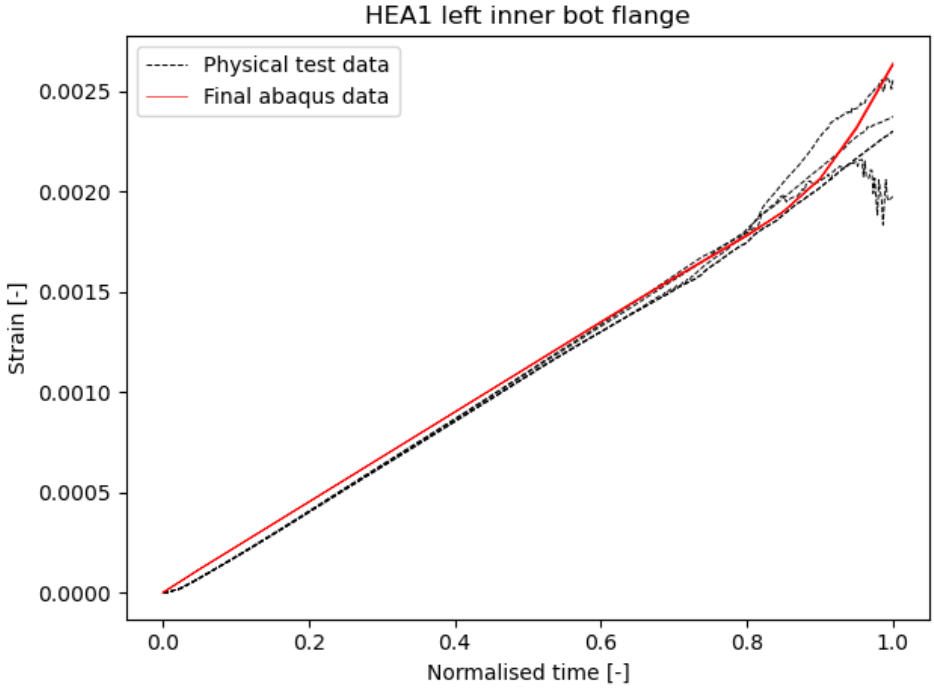


Figure H.7: HEA1 left inner bot flange

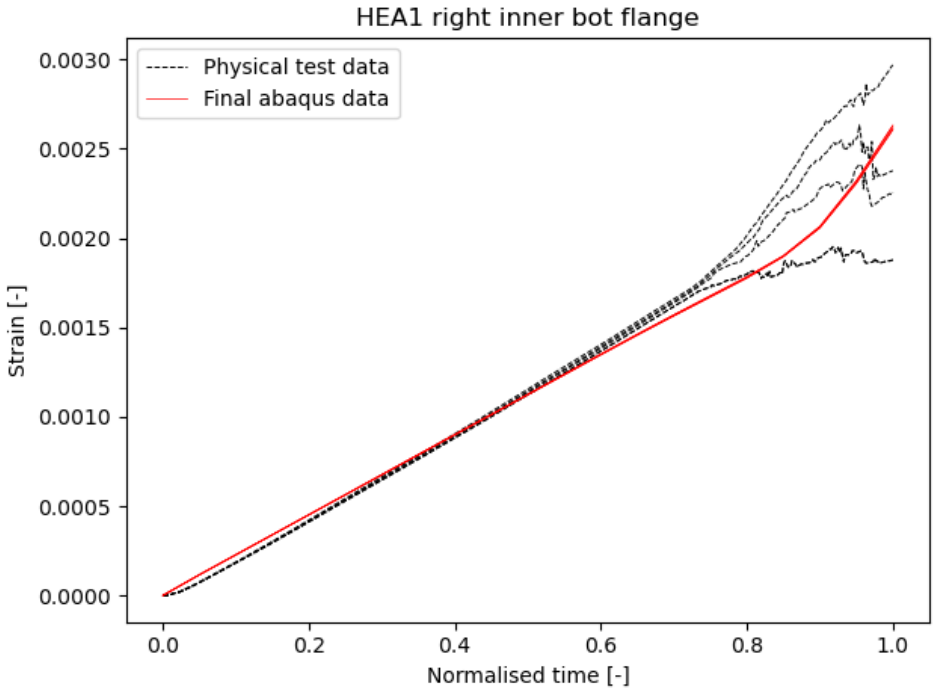


Figure H.8: HEA1 right inner bot flange

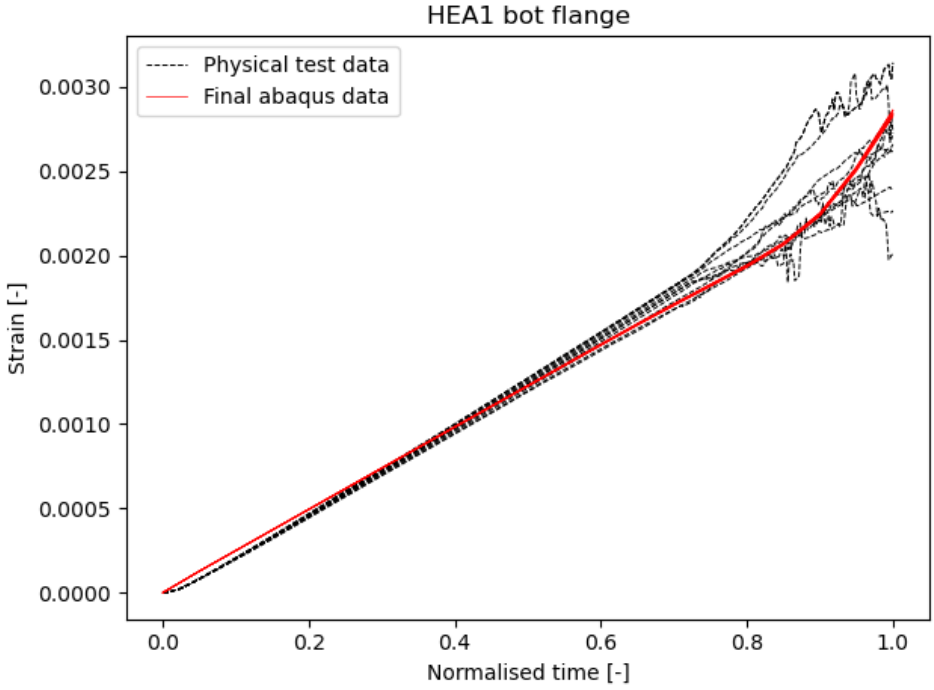


Figure H.9: HEA1 bot flange

HEA2

Figure H.10: HEA2 deflection in abaqus and physical test

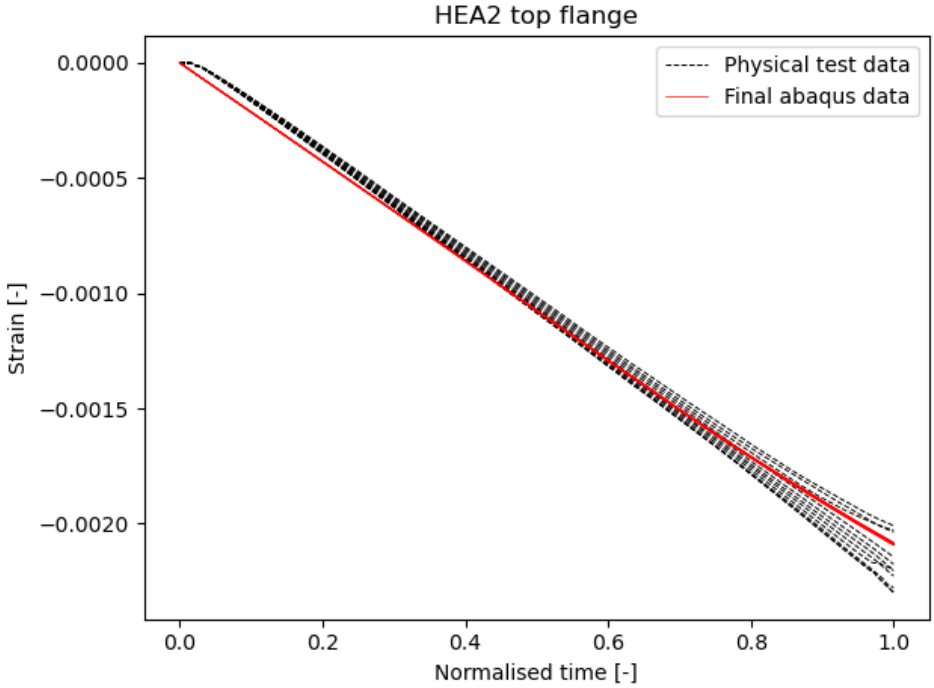


Figure H.11: HEA2 top flange

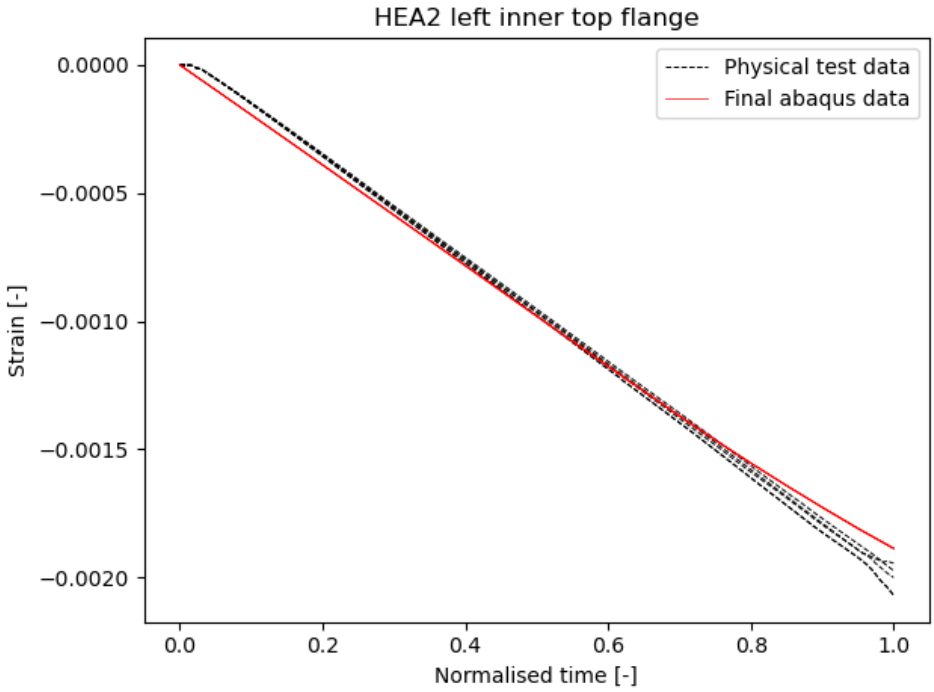


Figure H.12: HEA2 left inner top flange

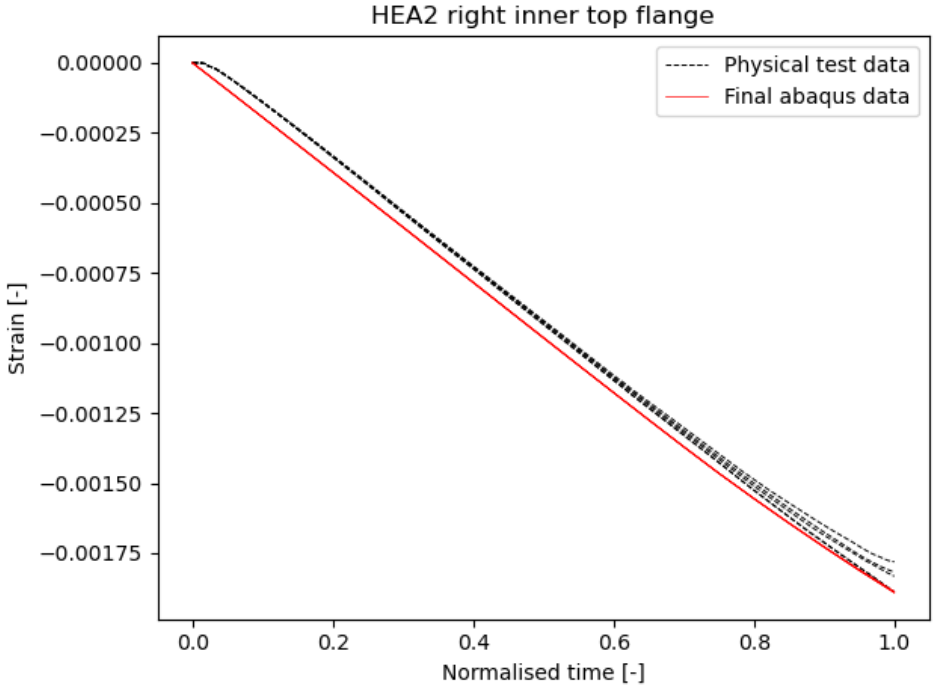


Figure H.13: HEA2 right inner top flange

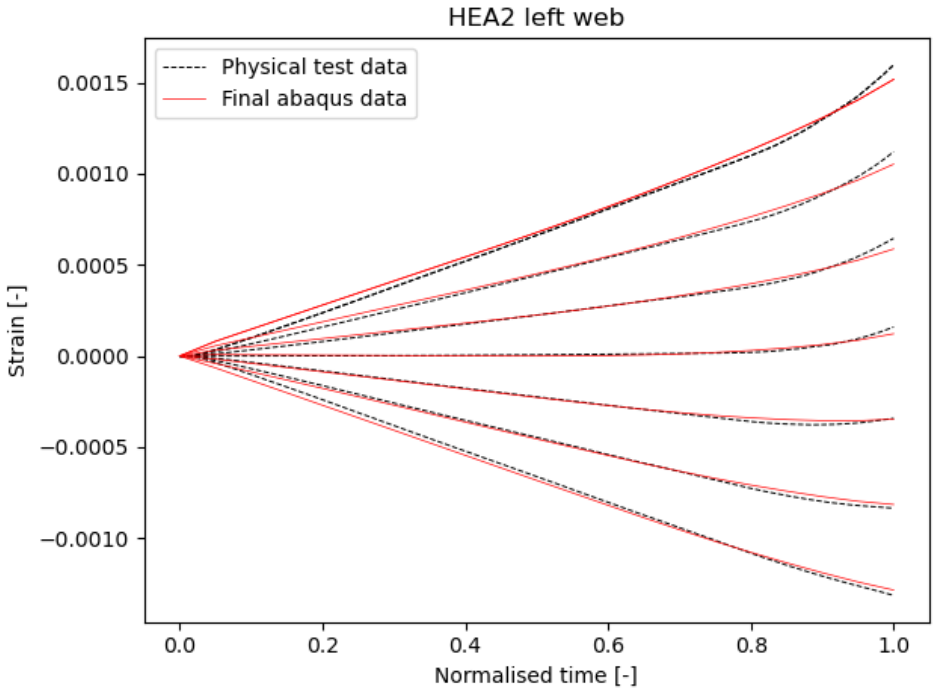


Figure H.14: HEA2 left web

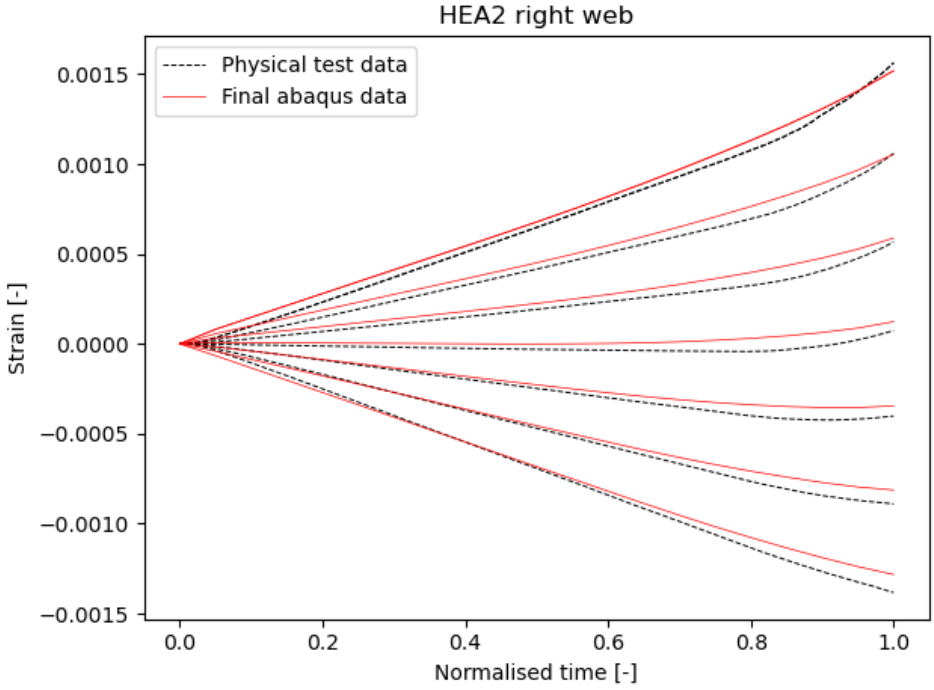


Figure H.15: HEA2 right web

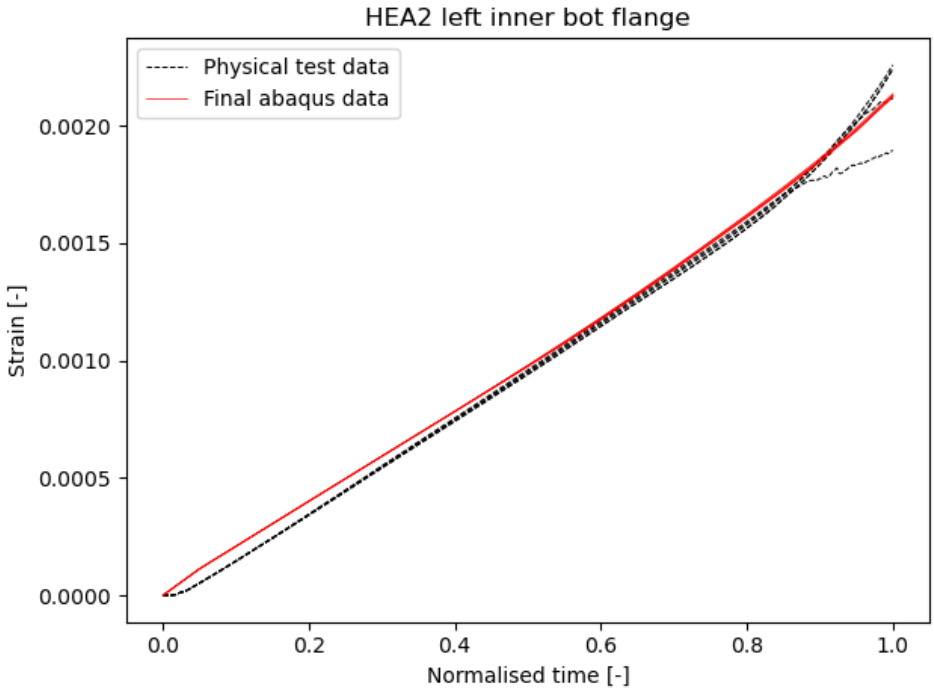


Figure H.16: HEA2 left inner bot flange

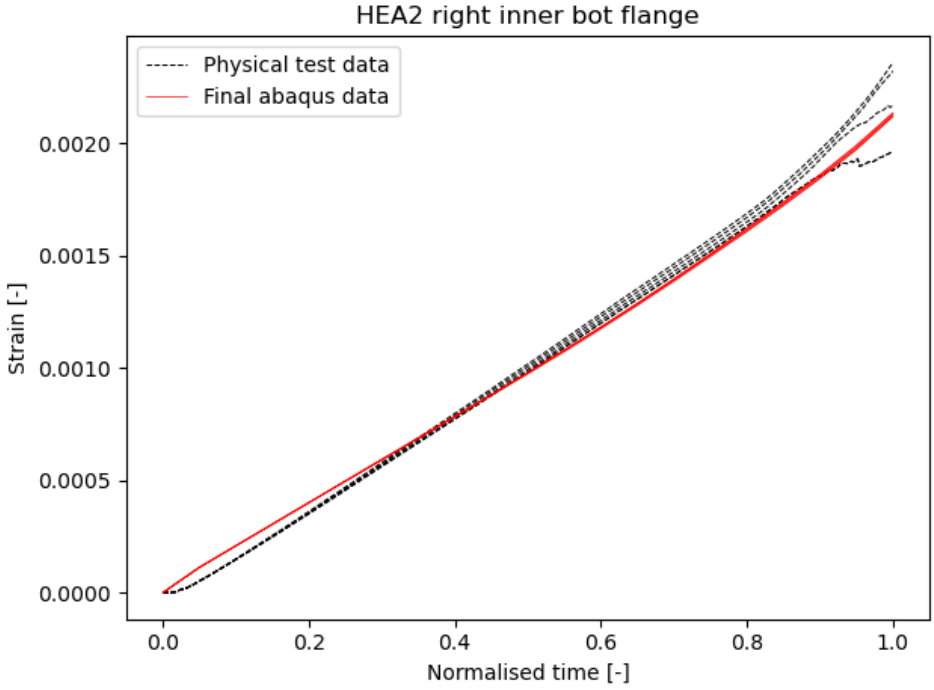


Figure H.17: HEA2 right inner bot flange

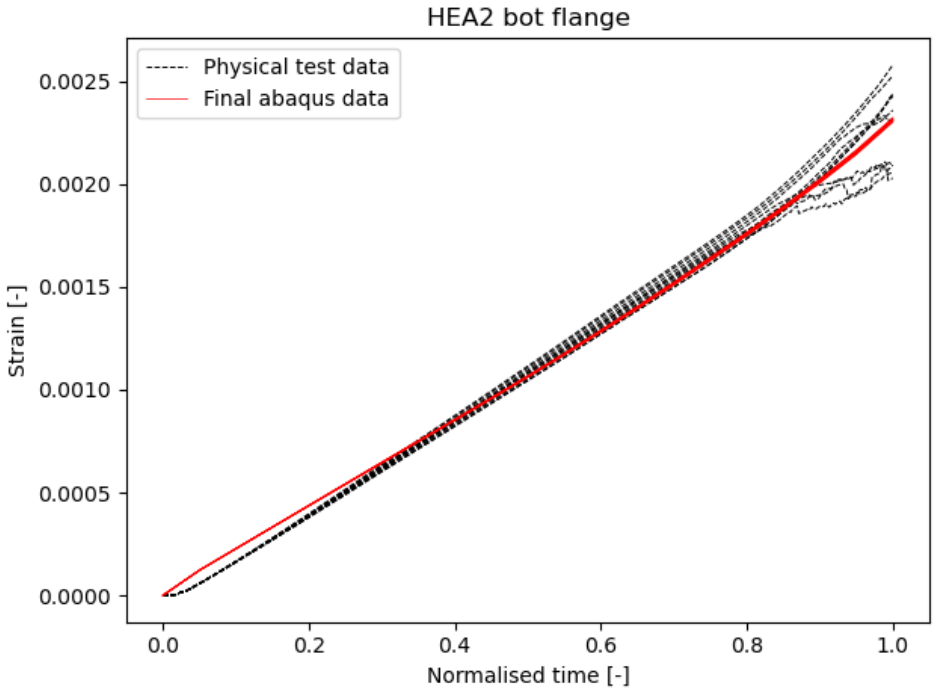


Figure H.18: HEA2 bot flange

HEA3

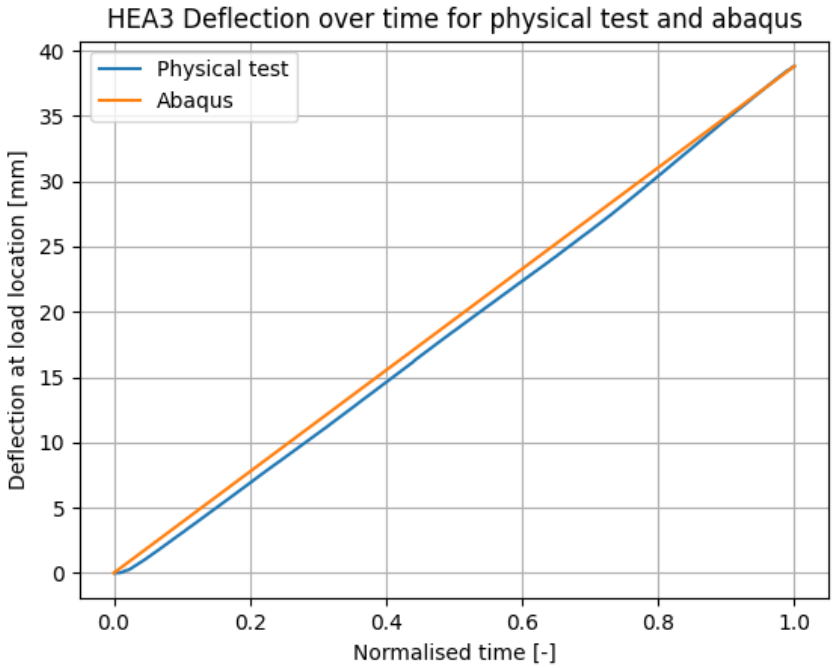


Figure H.19: HEA3 deflection in abaqus and physical test

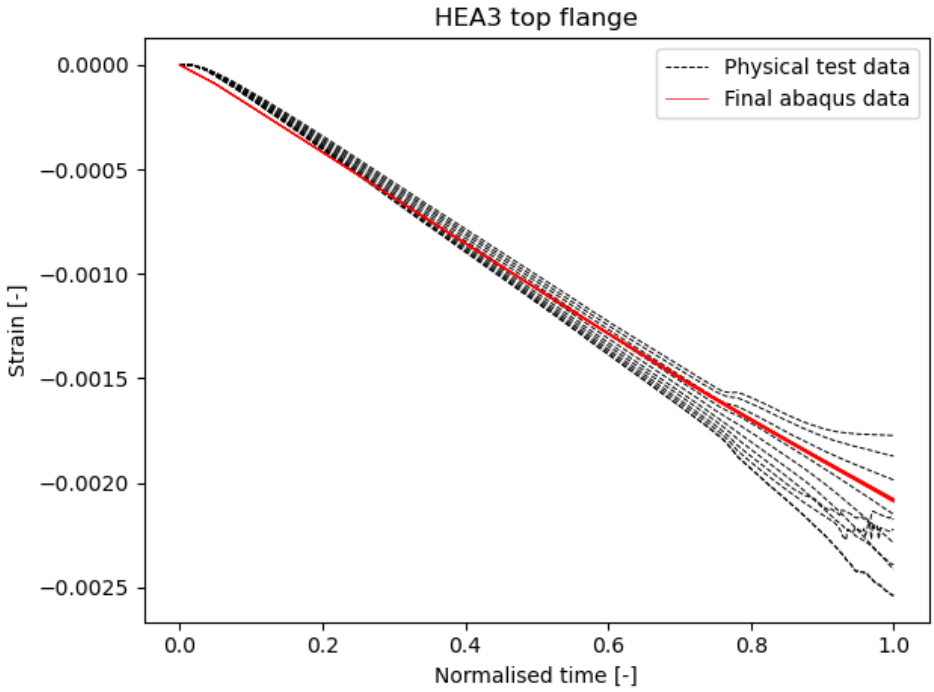


Figure H.20: HEA3 top flange

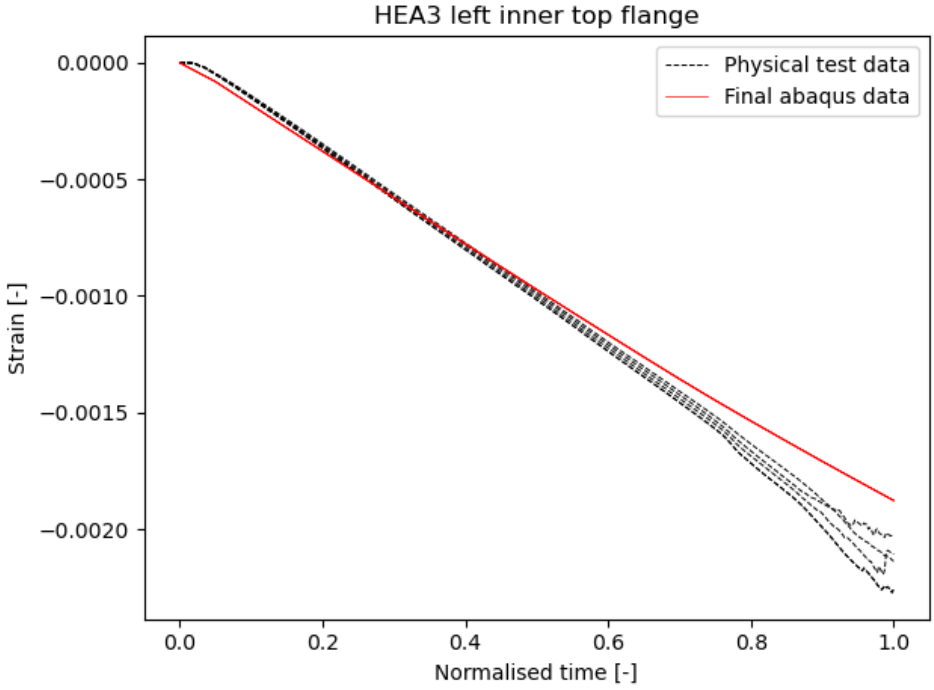


Figure H.21: HEA3 left inner top flange

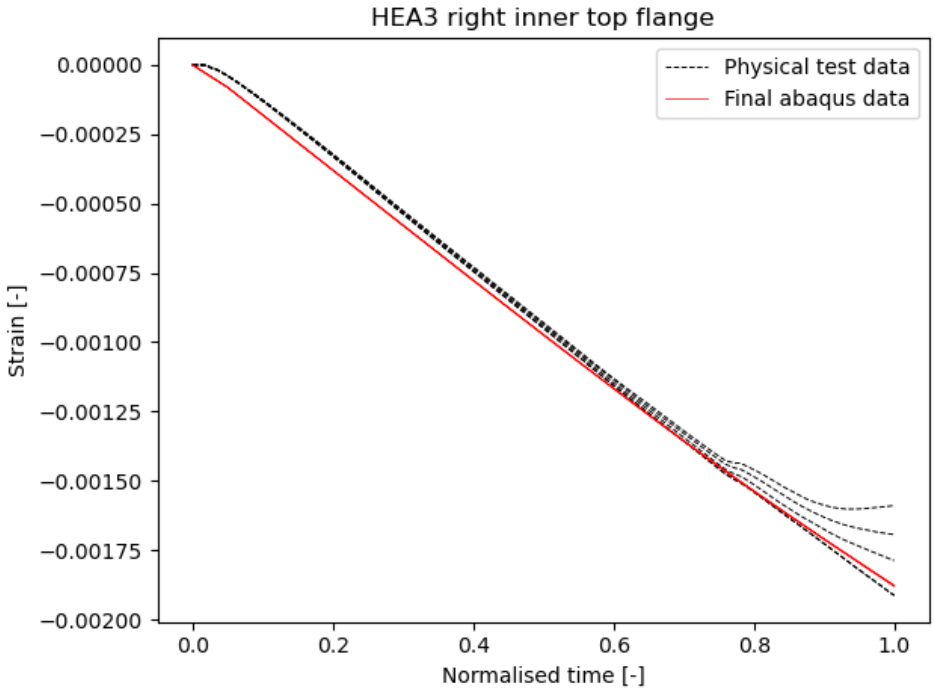


Figure H.22: HEA3 right inner top flange

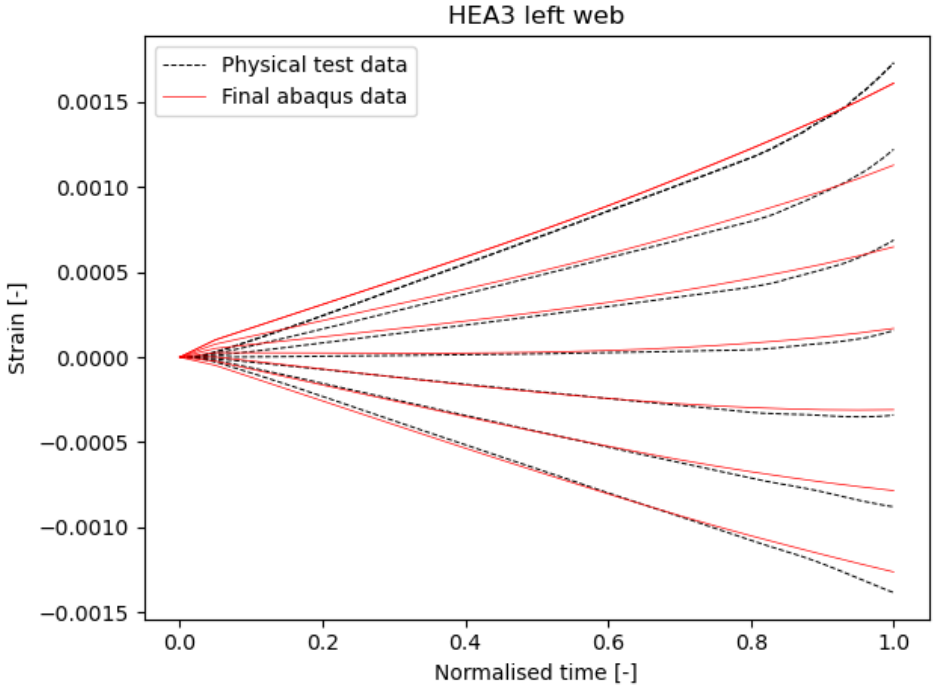


Figure H.23: HEA3 left web

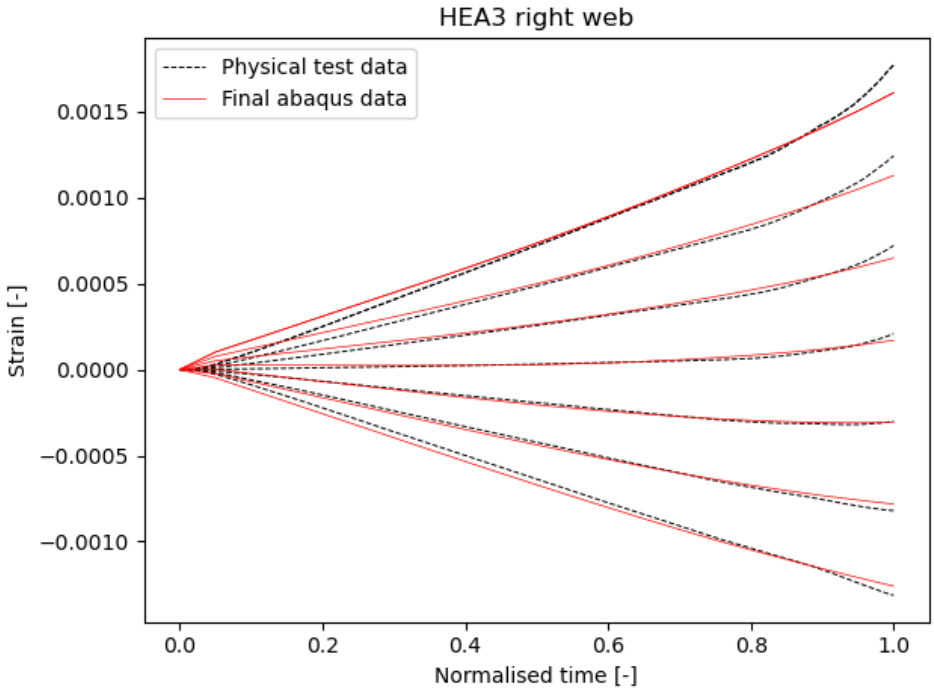


Figure H.24: HEA3 right web

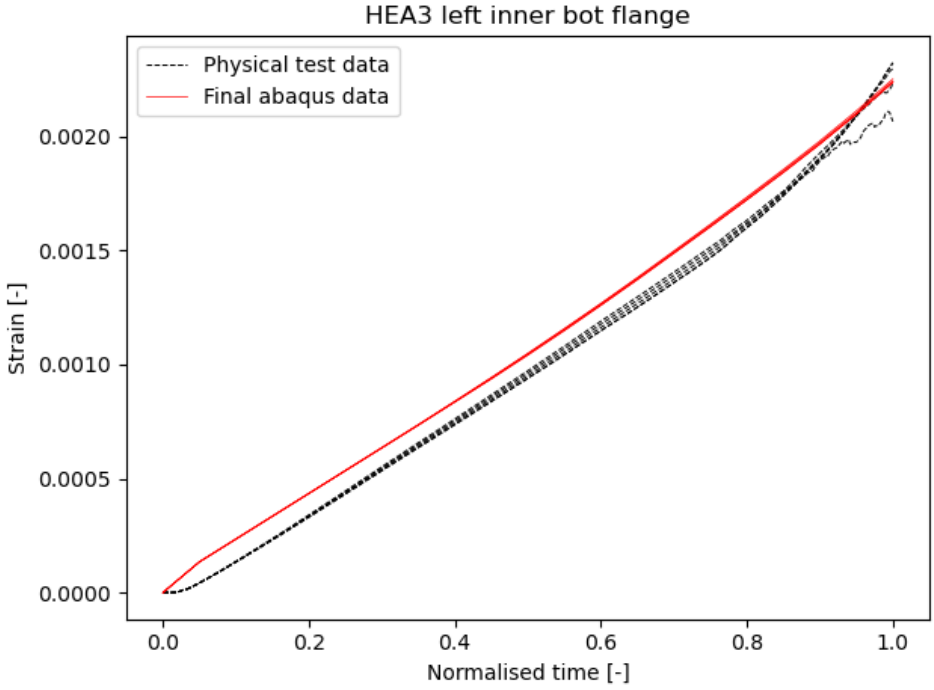


Figure H.25: HEA3 left inner bot flange

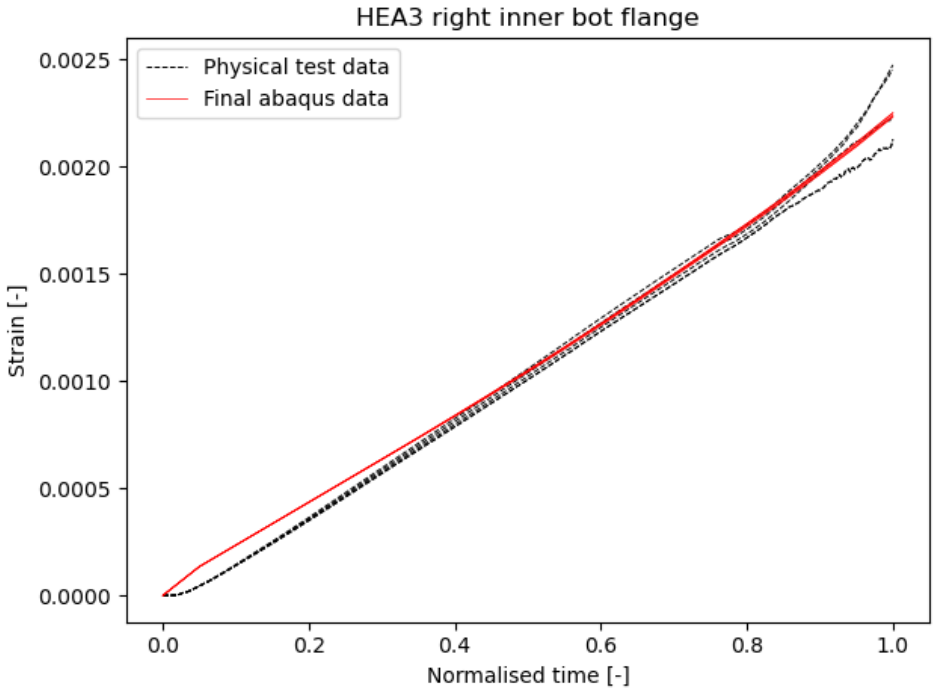


Figure H.26: HEA3 right inner bot flange

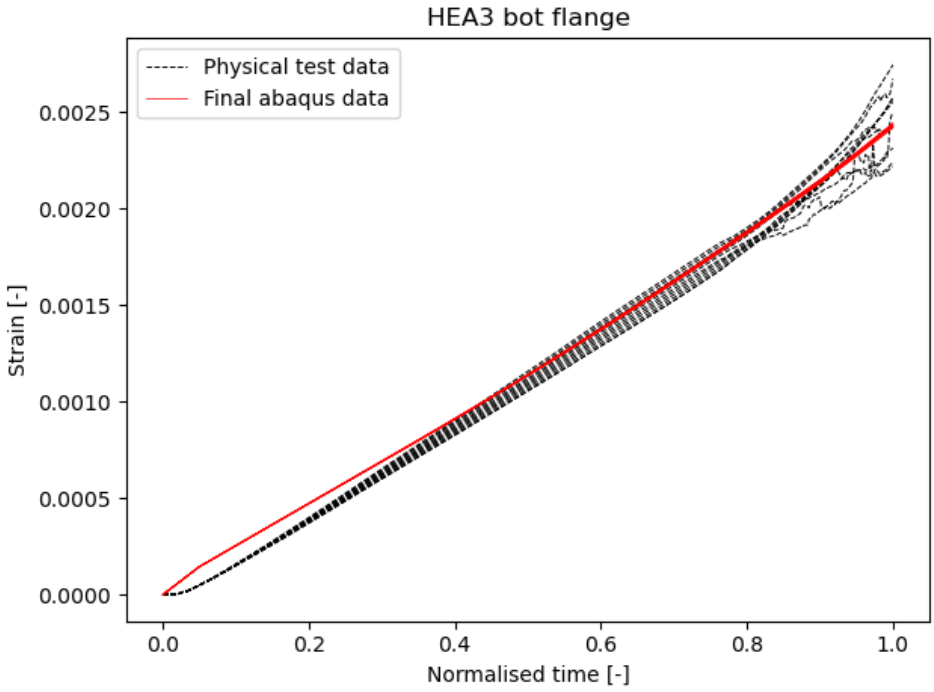


Figure H.27: HEA3 bot flange

HEB1

Figure H.28: HEB1 deflection in abaqus and physical test

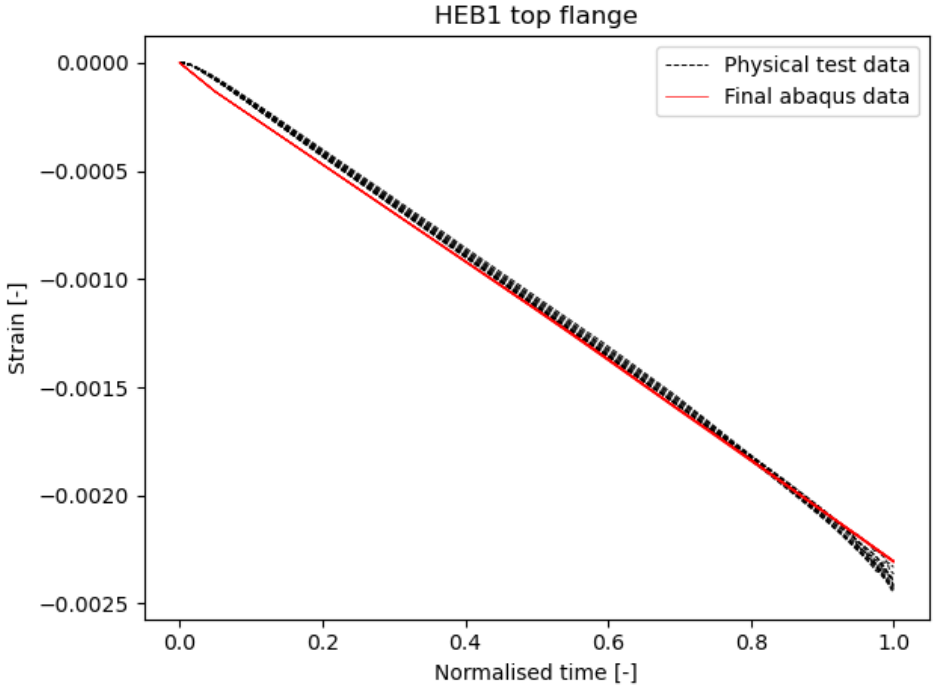


Figure H.29: HEB1 top flange

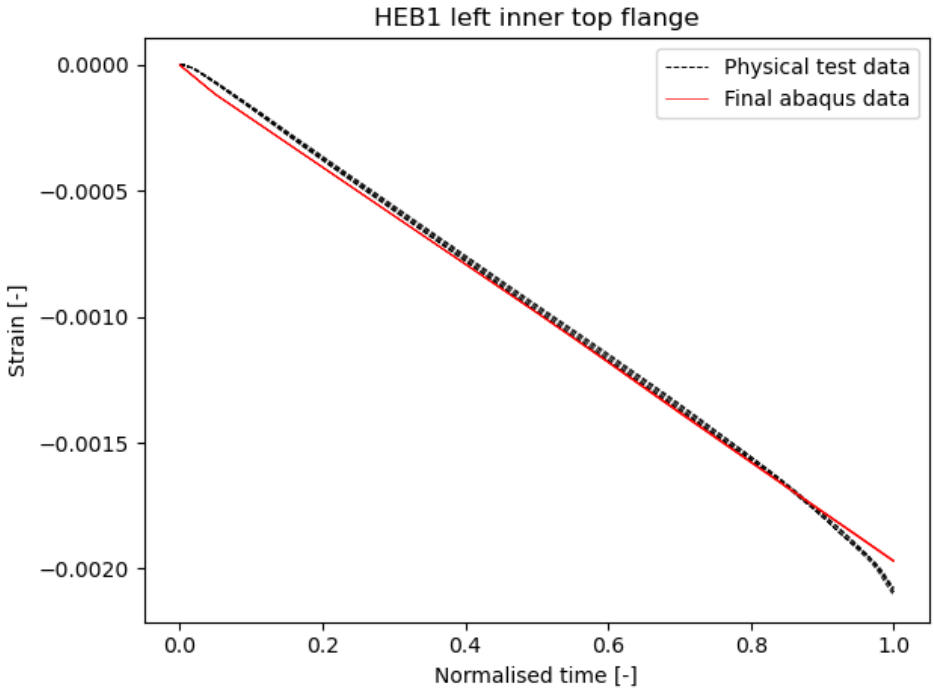


Figure H.30: HEB1 left inner top flange

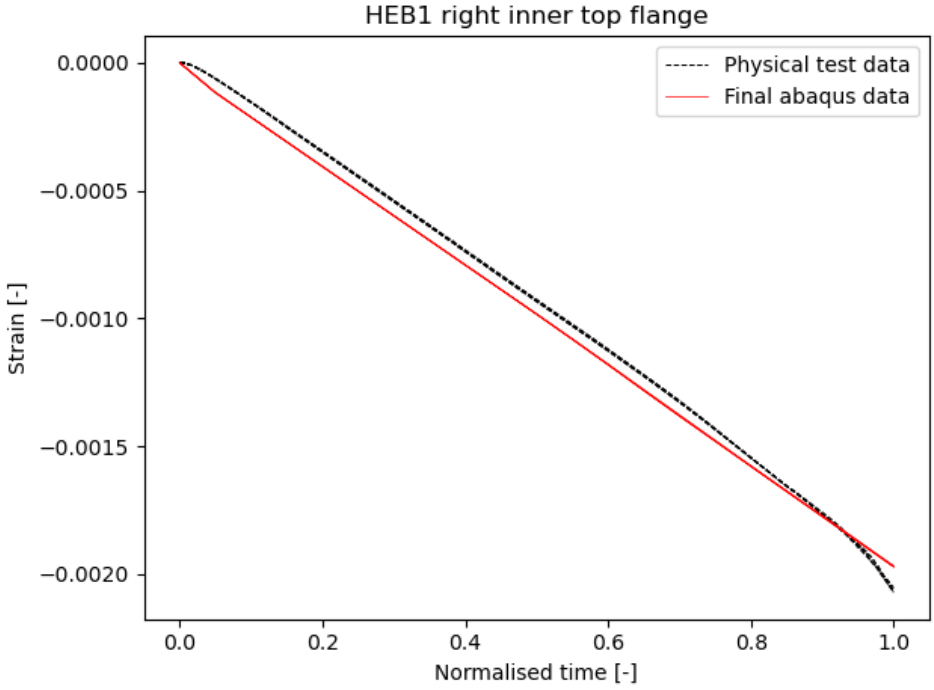


Figure H.31: HEB1 right inner top flange

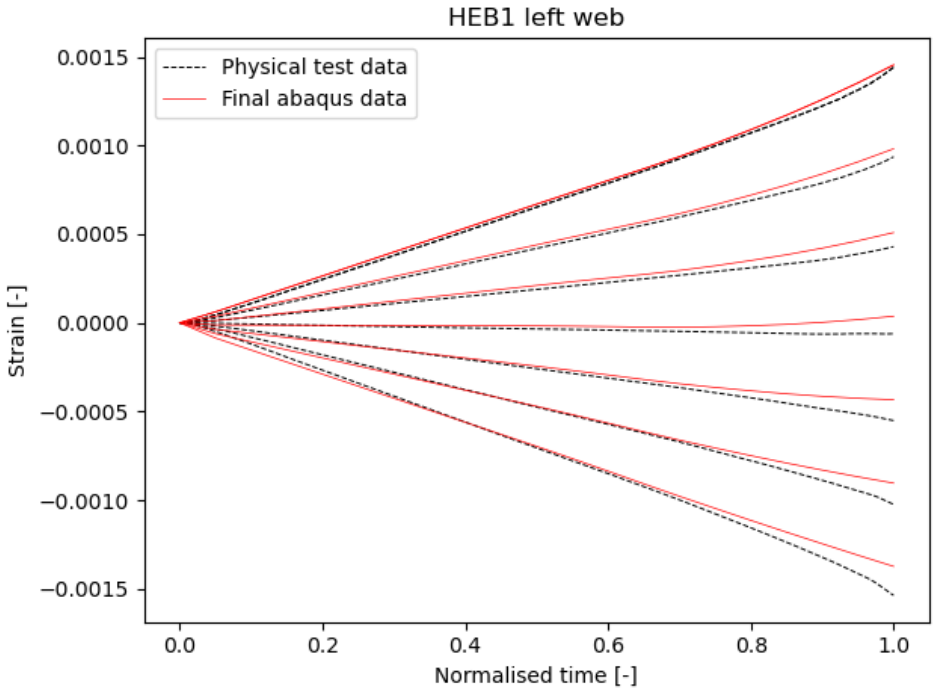


Figure H.32: HEB1 left web

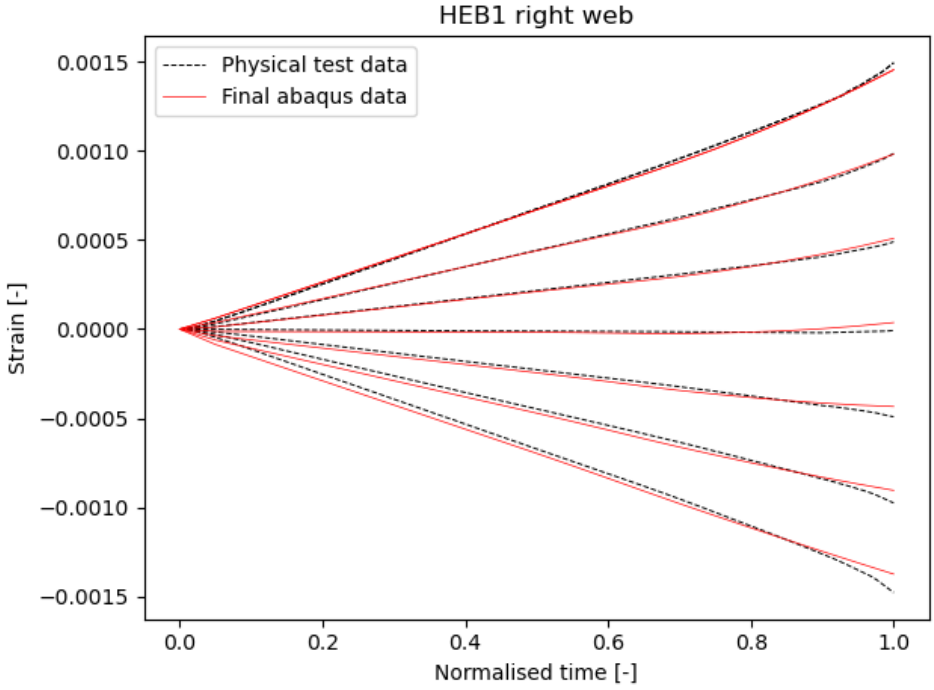


Figure H.33: HEB1 right web

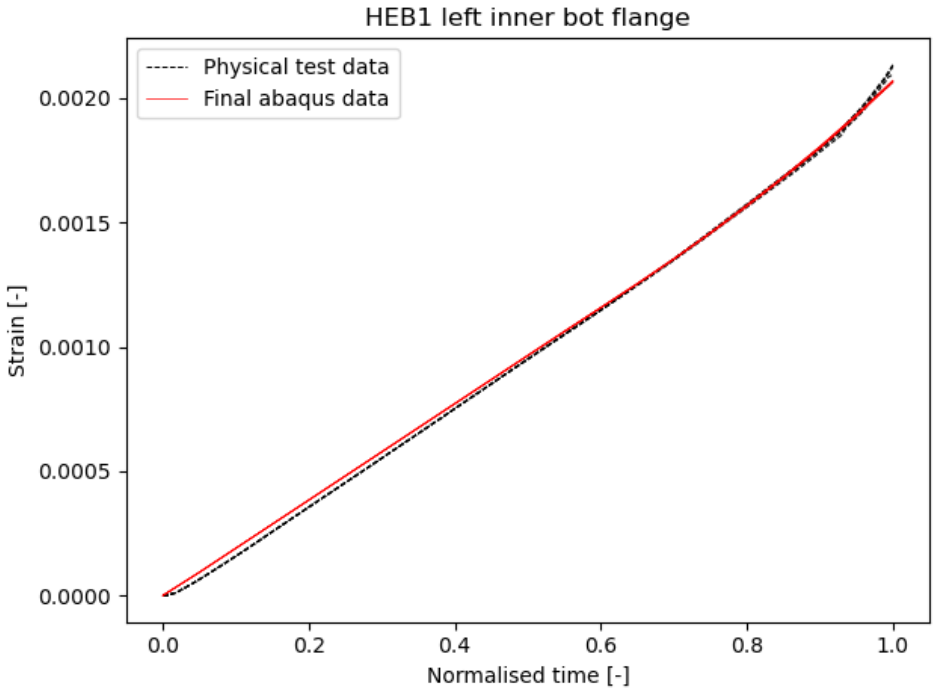


Figure H.34: HEB1 left inner bot flange

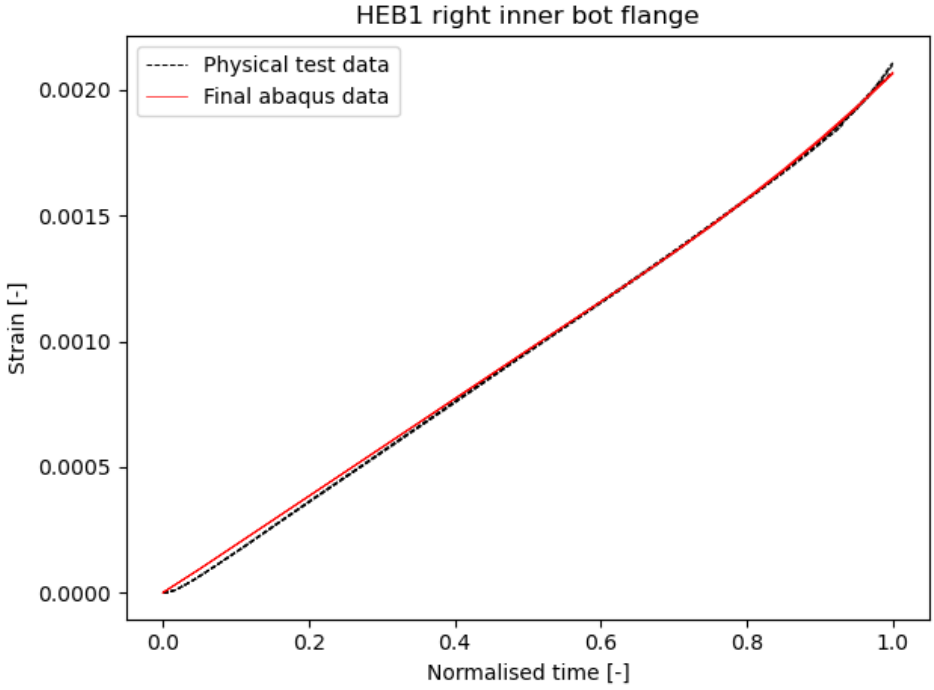


Figure H.35: HEB1 right inner bot flange

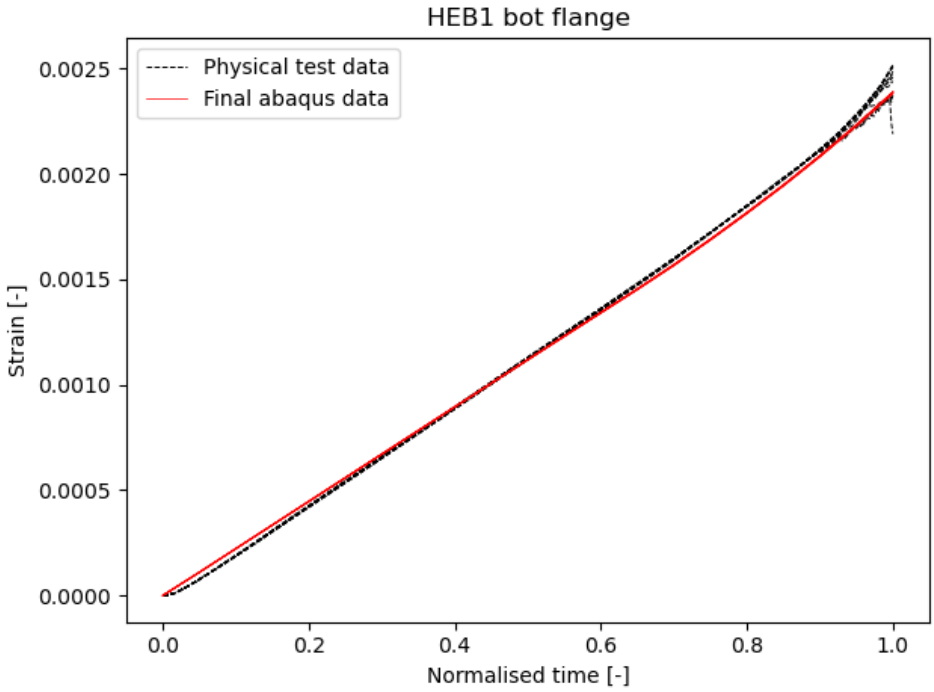


Figure H.36: HEB1 bot flange

HEB2

Figure H.37: HEB2 deflection in abaqus and physical test

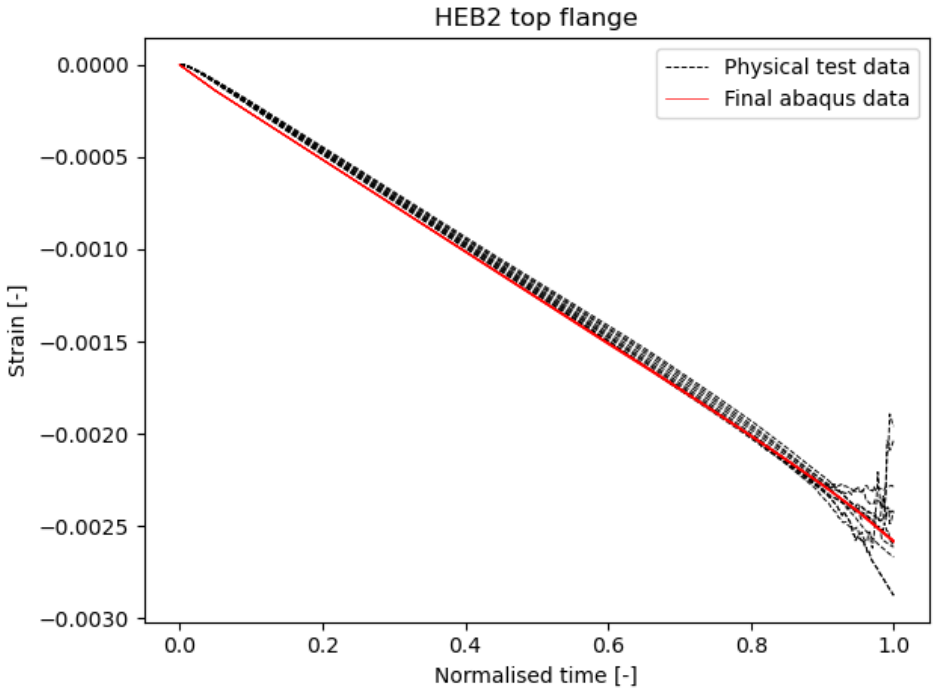


Figure H.38: HEB2 top flange

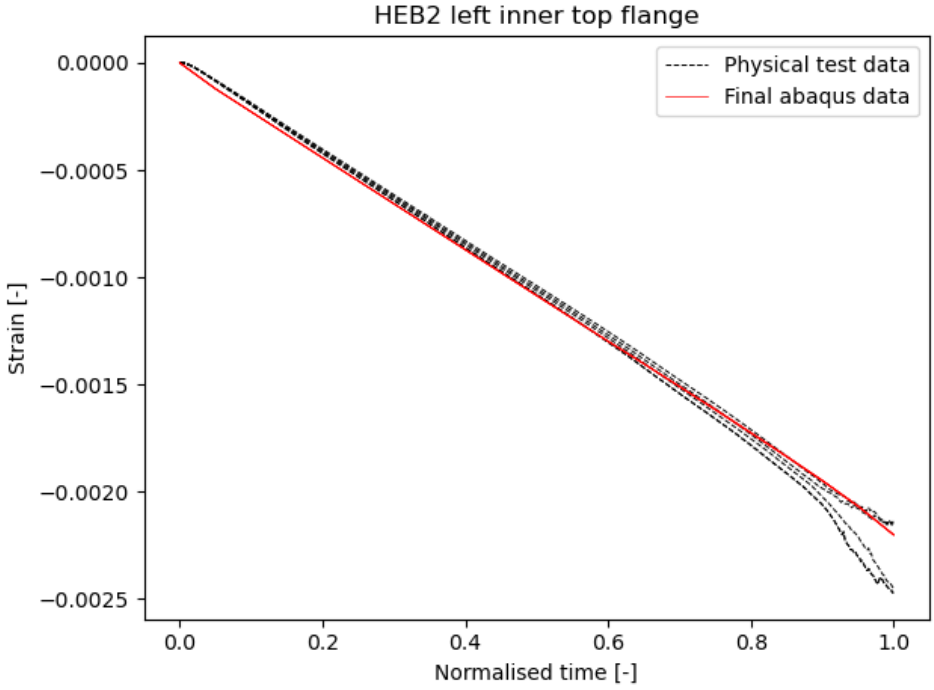


Figure H.39: HEB2 left inner top flange

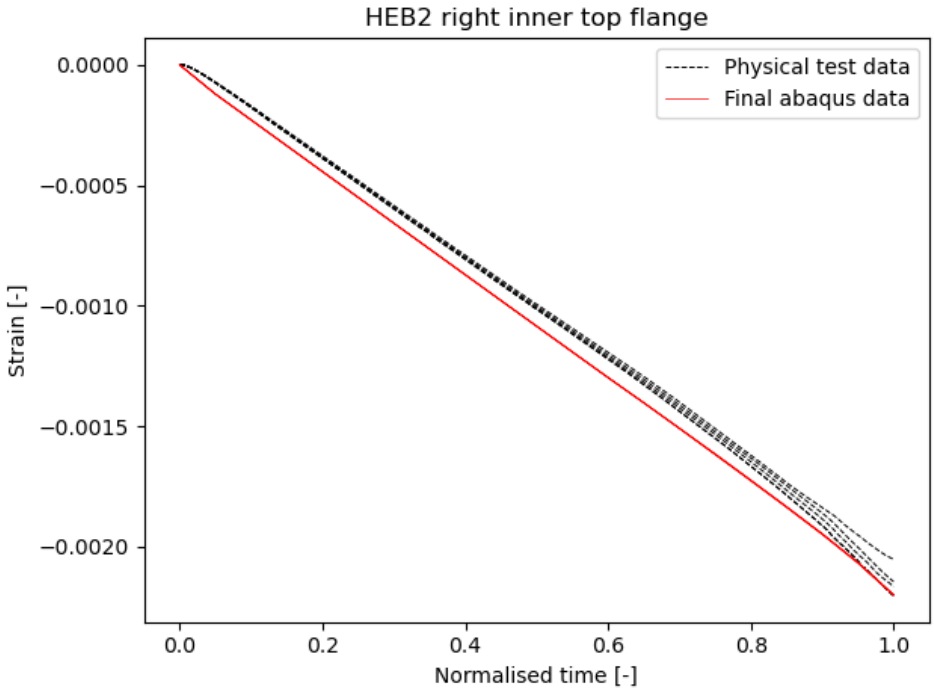


Figure H.40: HEB2 right inner top flange

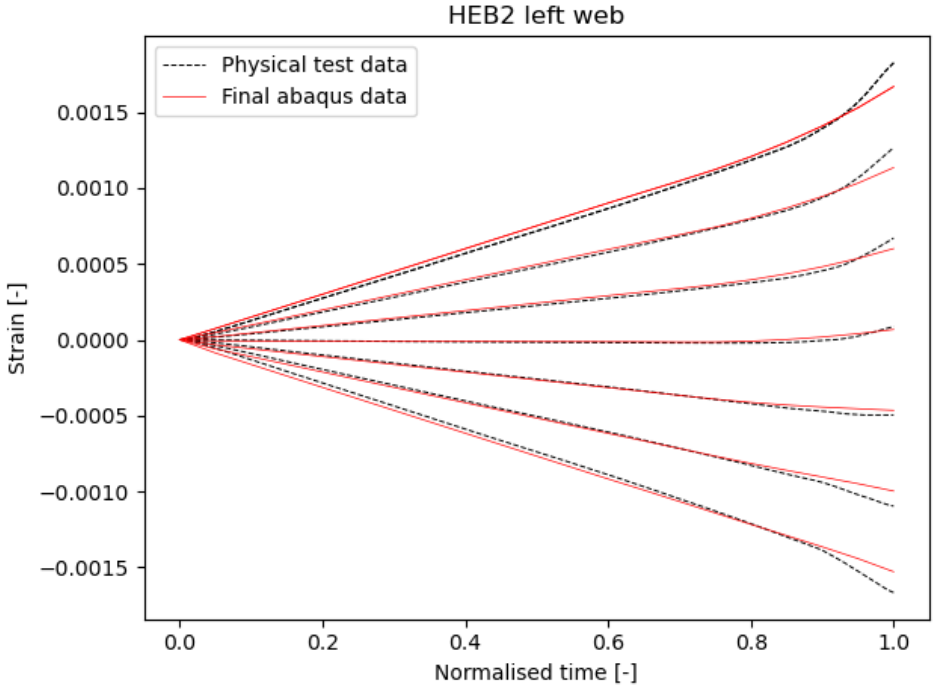


Figure H.41: HEB2 left web

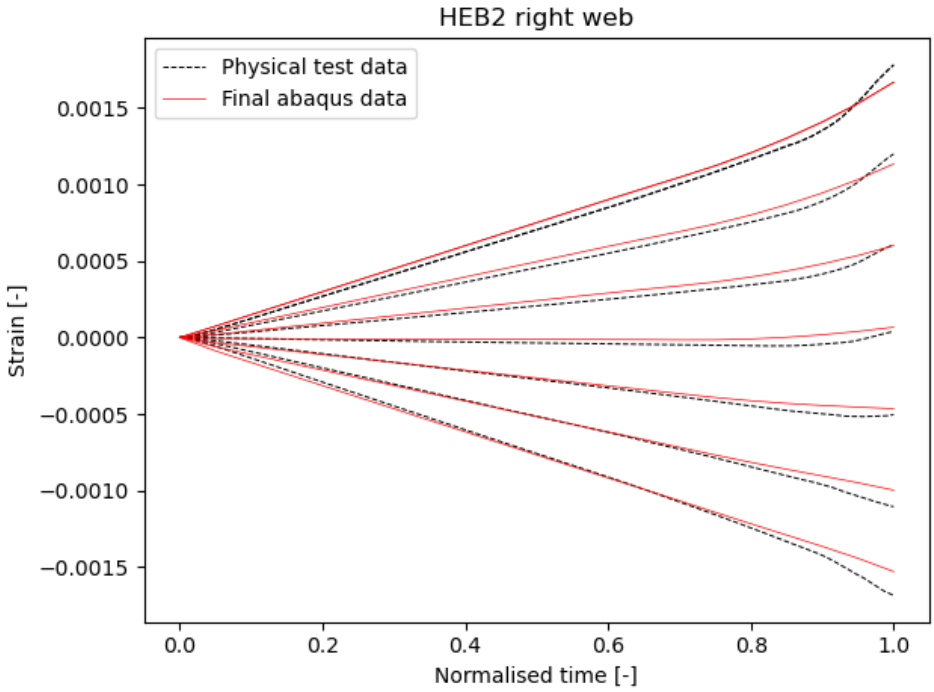


Figure H.42: HEB2 right web

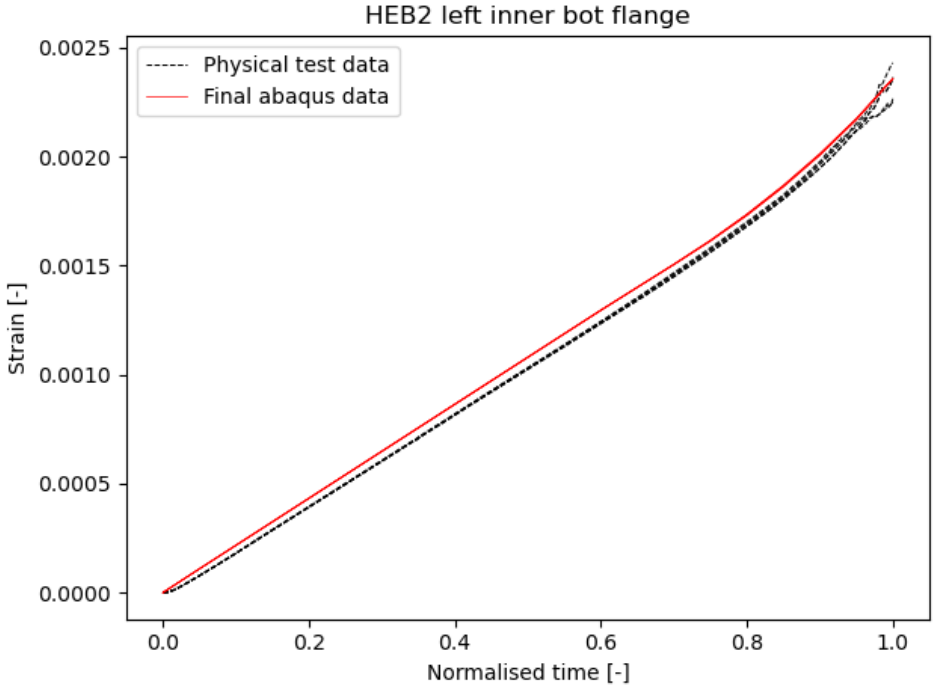


Figure H.43: HEB2 left inner bot flange

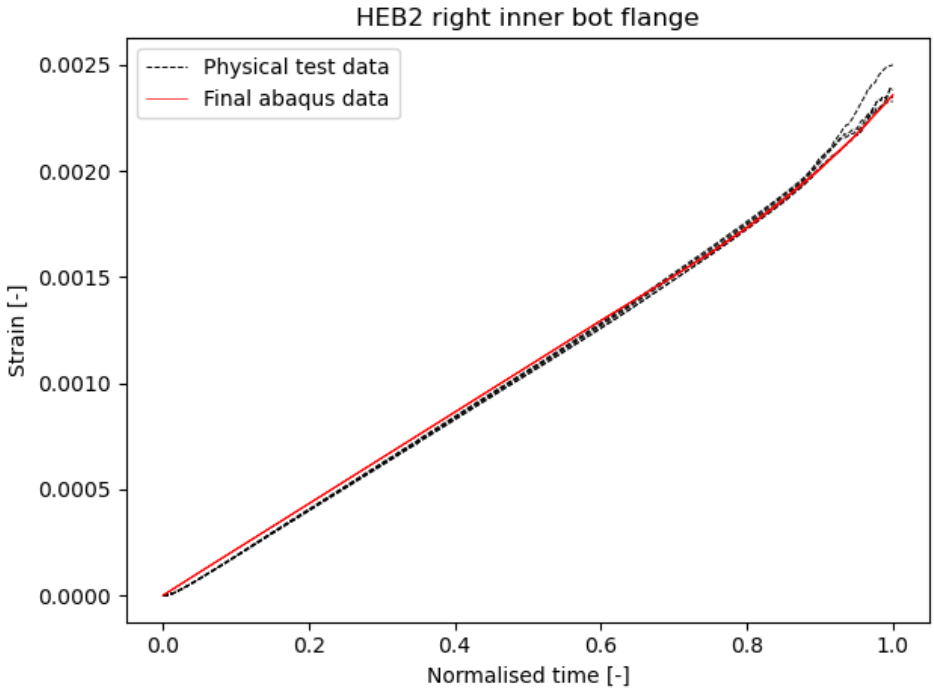


Figure H.44: HEB2 right inner bot flange

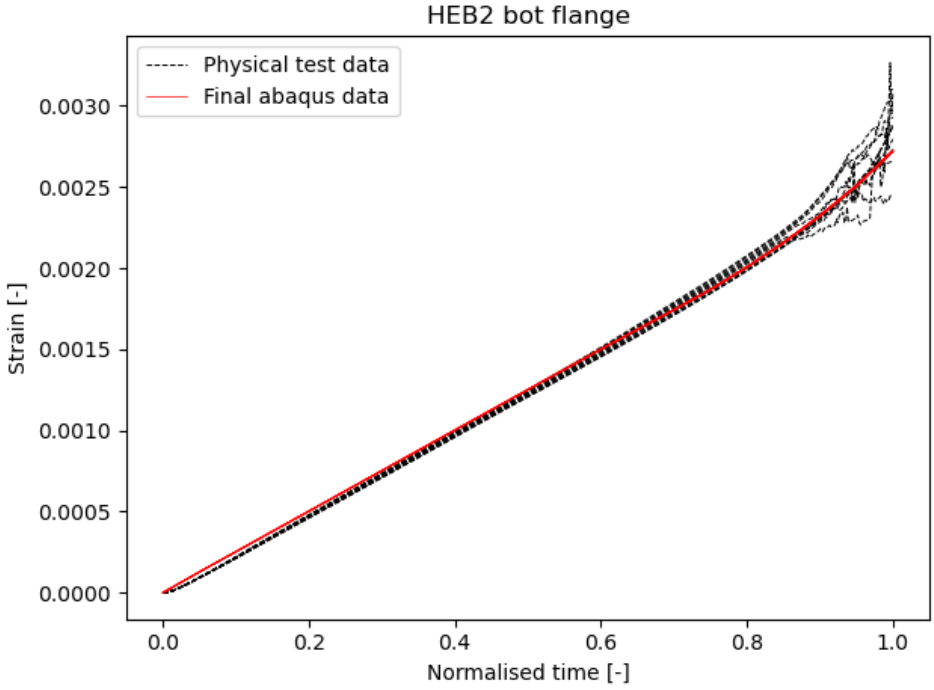


Figure H.45: HEB2 bot flange

HEB3

Figure H.46: HEB3 deflection in abaqus and physical test

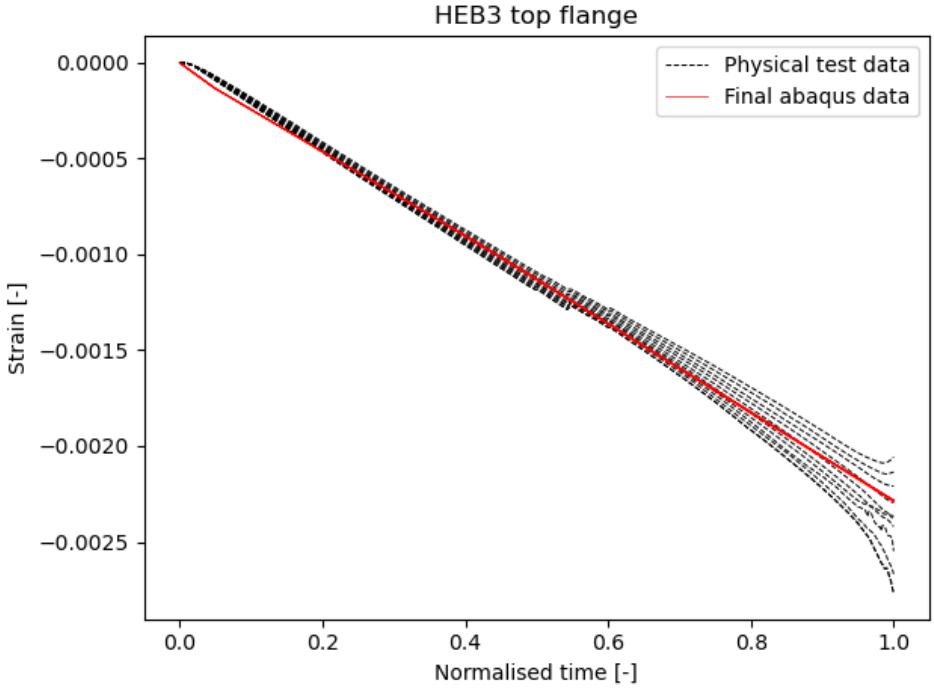


Figure H.47: HEB3 top flange

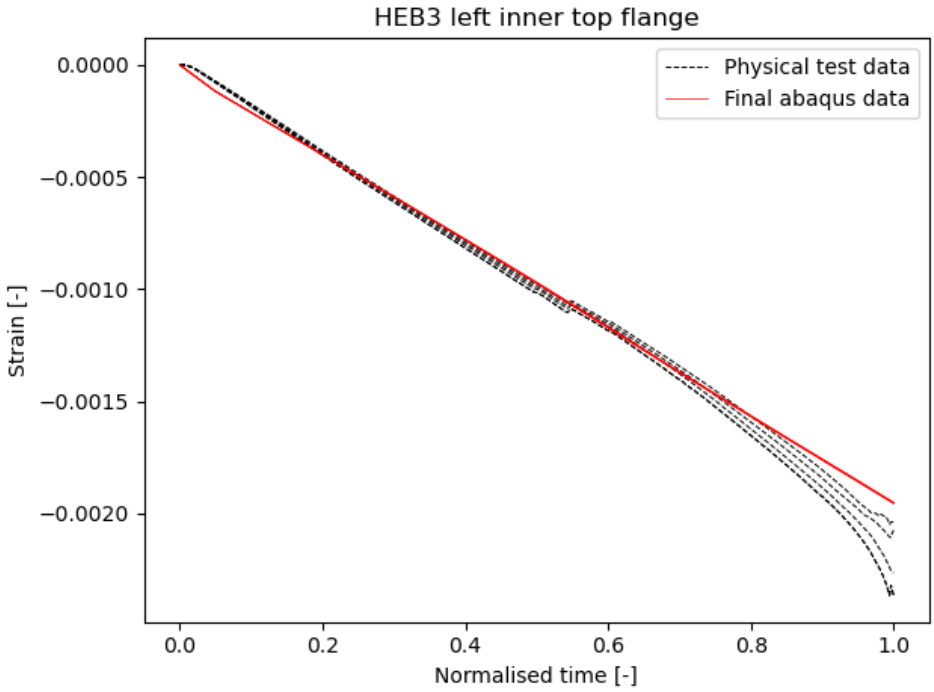


Figure H.48: HEB3 left inner top flange

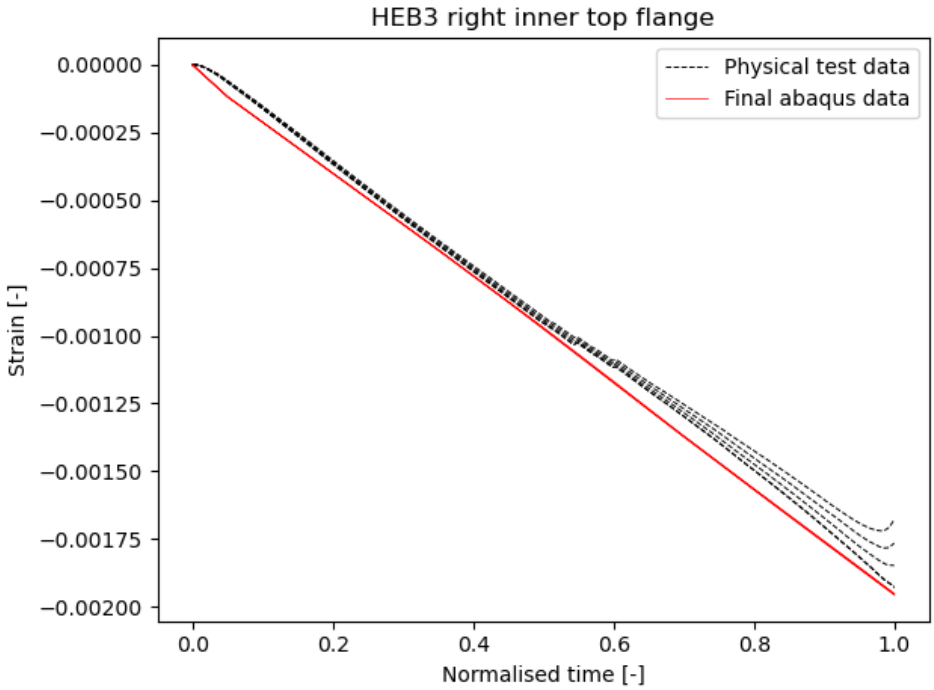


Figure H.49: HEB3 right inner top flange

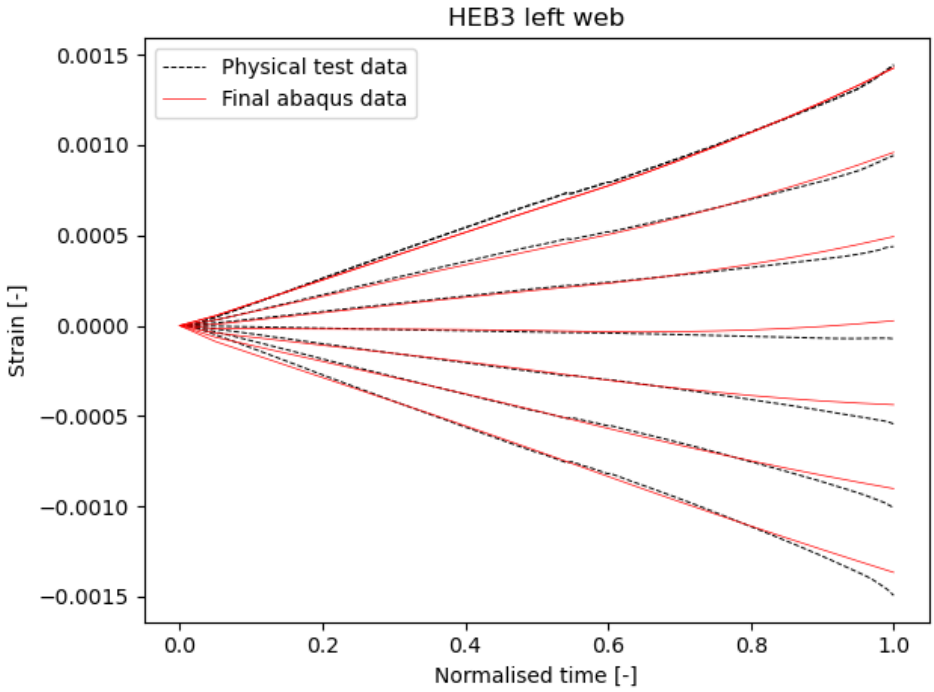


Figure H.50: HEB3 left web

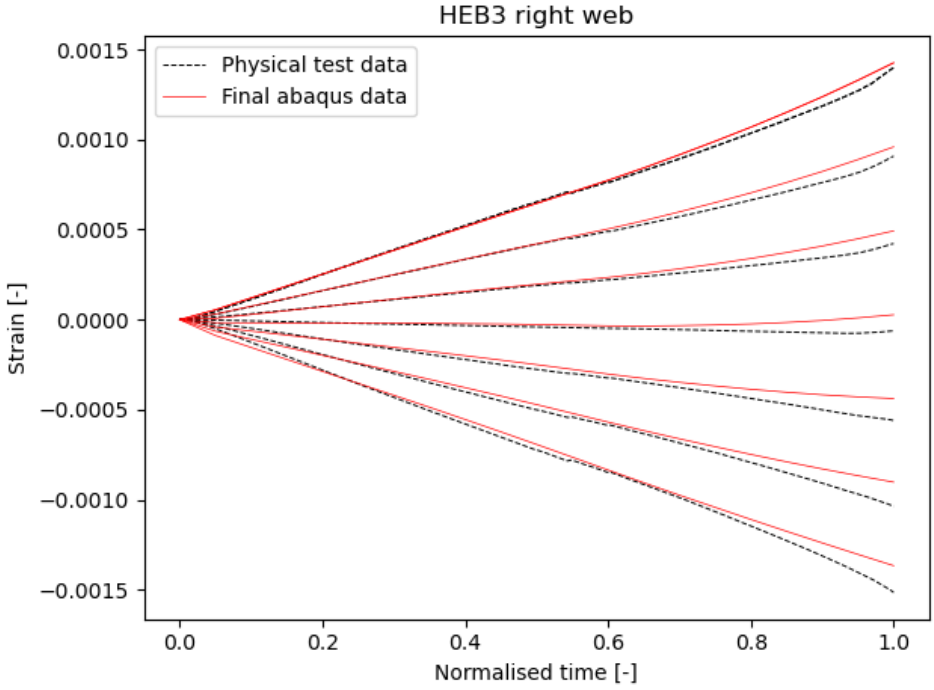


Figure H.51: HEB3 right web

Figure H.52: HEB3 left inner bot flange

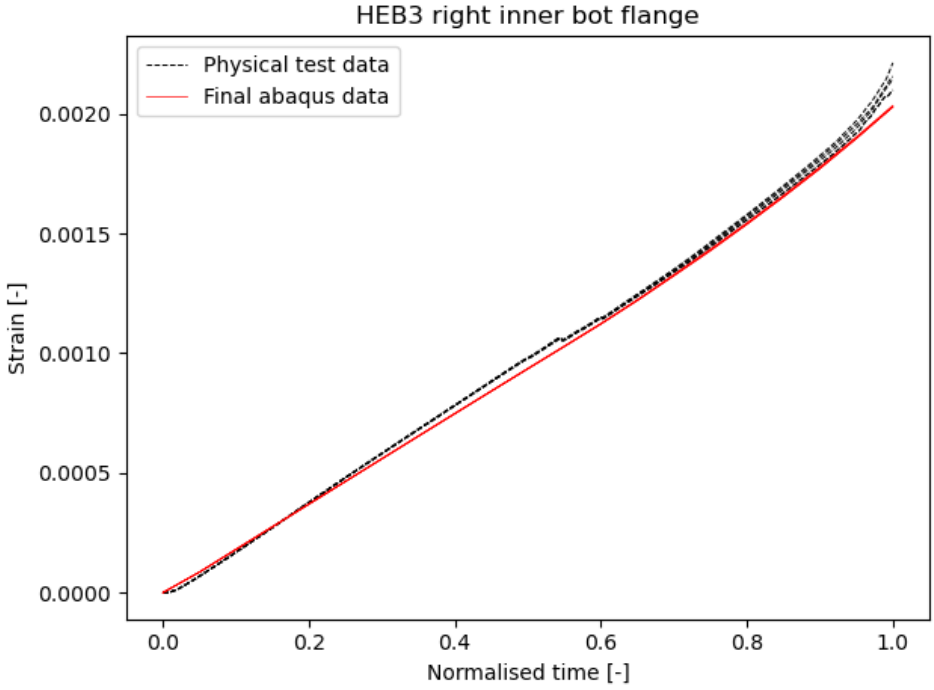


Figure H.53: HEB3 right inner bot flange

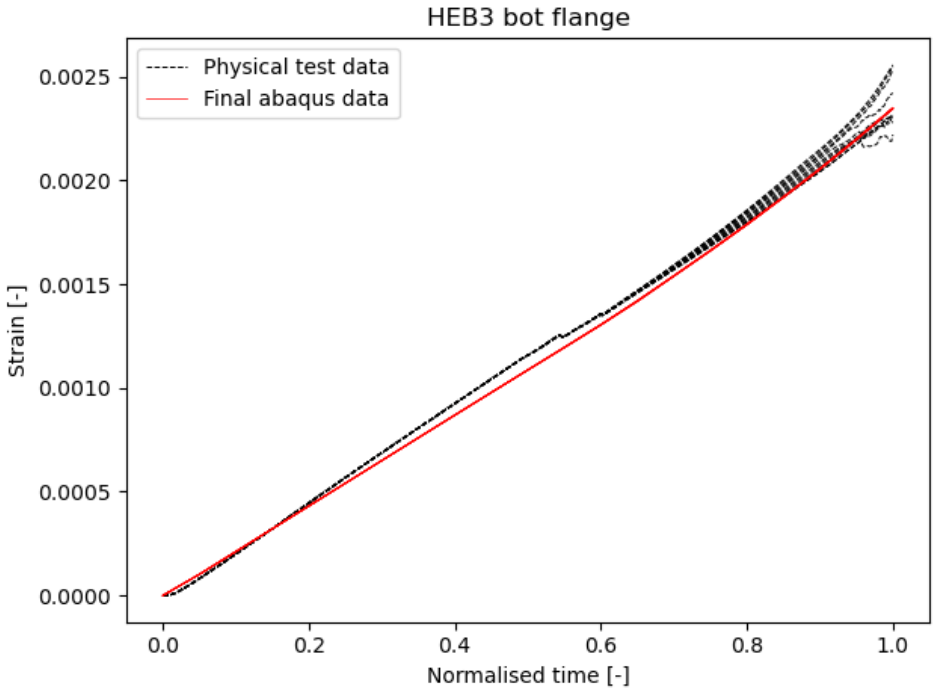


Figure H.54: HEB3 bot flange

I

Full run strain comparison of HEB1

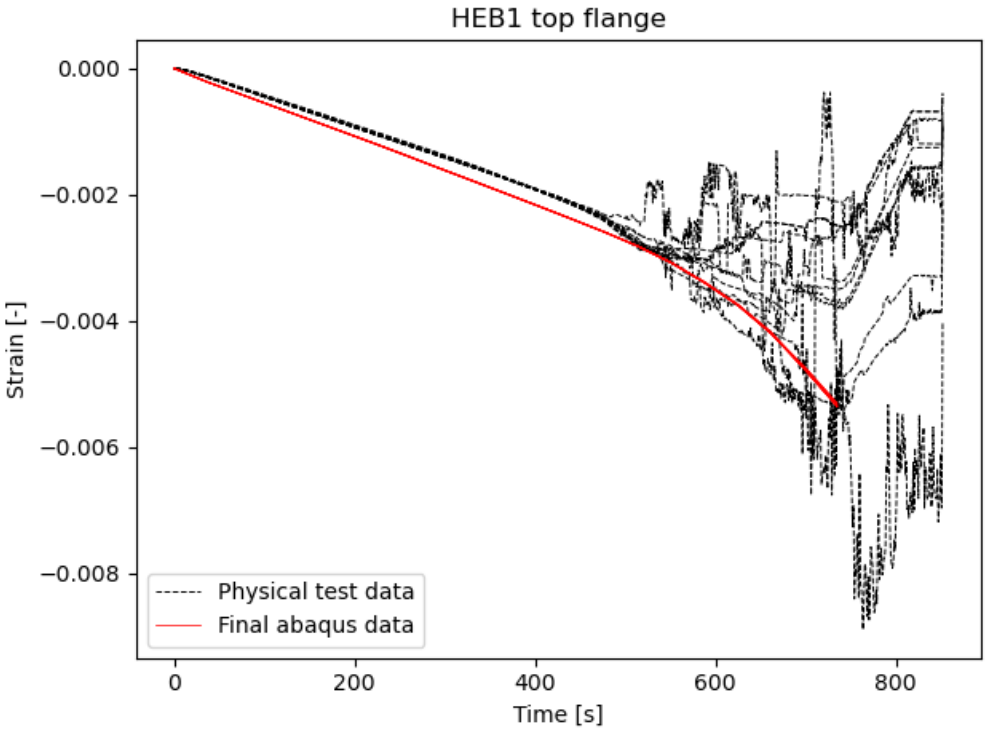


Figure I.1: HEB1 Extended run top flange

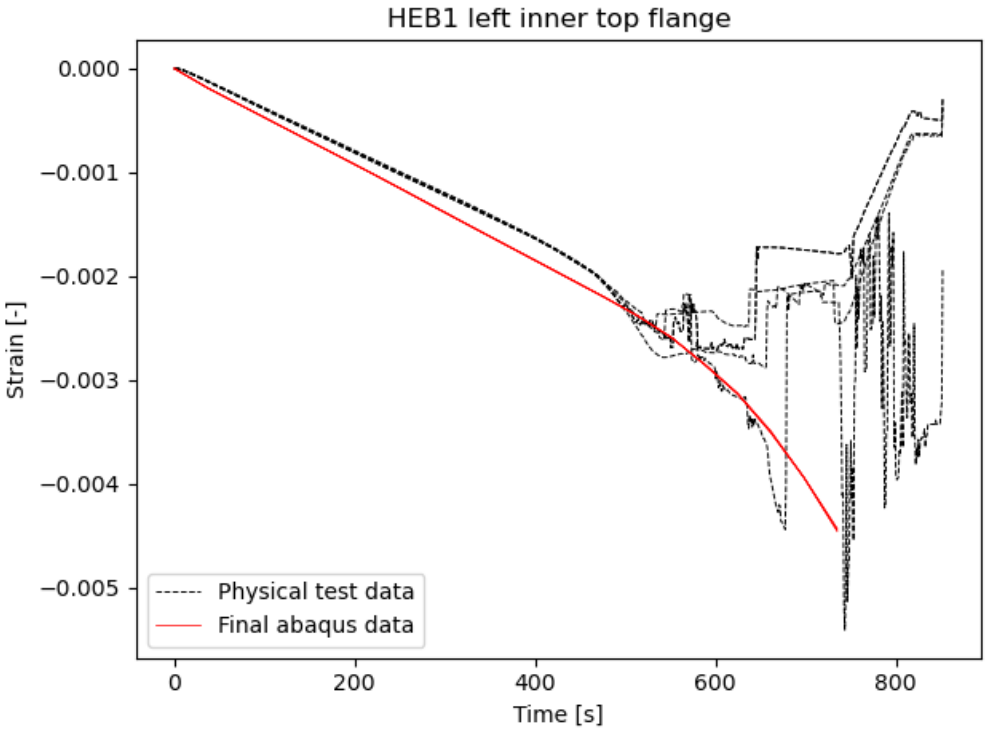


Figure I.2: HEB1 Extended run left inner top flange

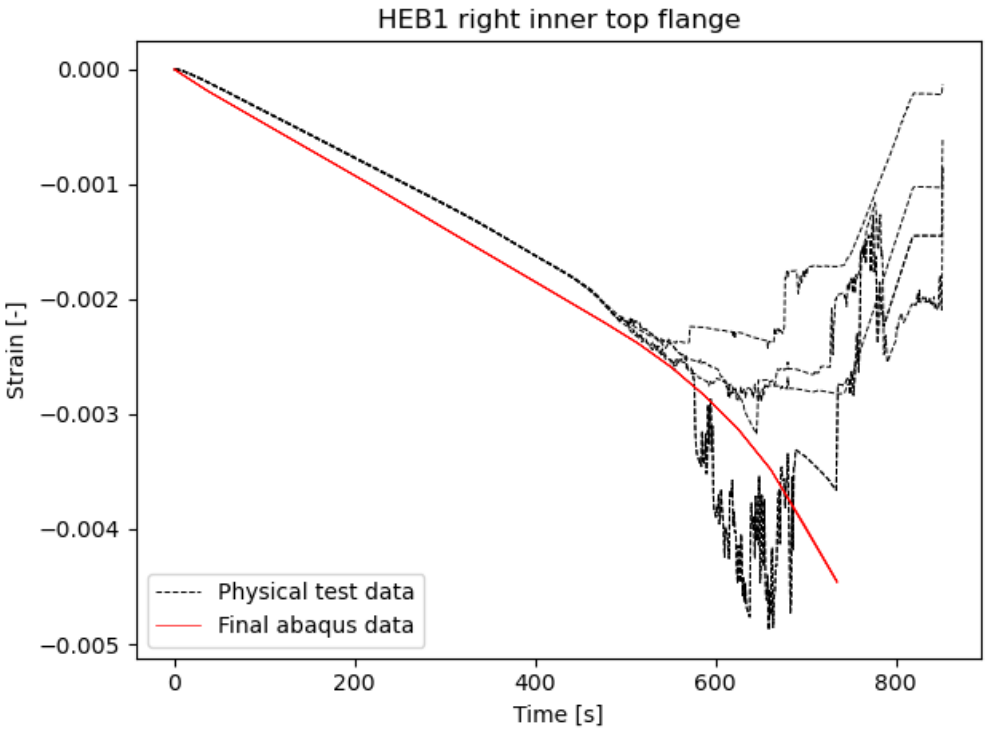


Figure I.3: HEB1 Extended run right inner top flange

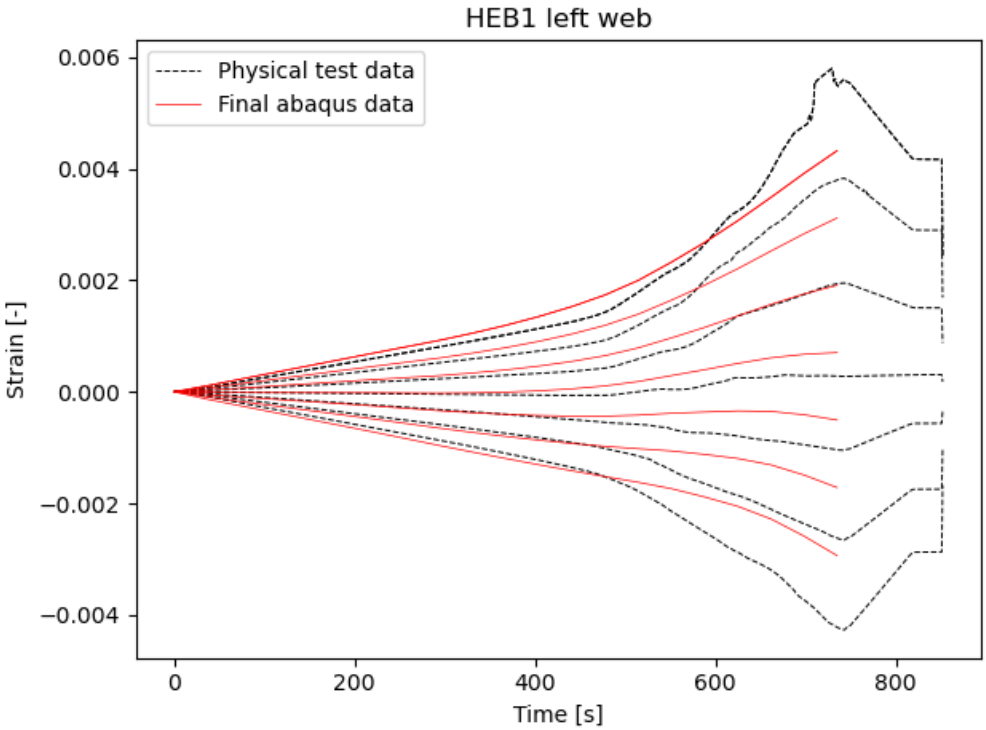


Figure I.4: HEB1 Extended run left web

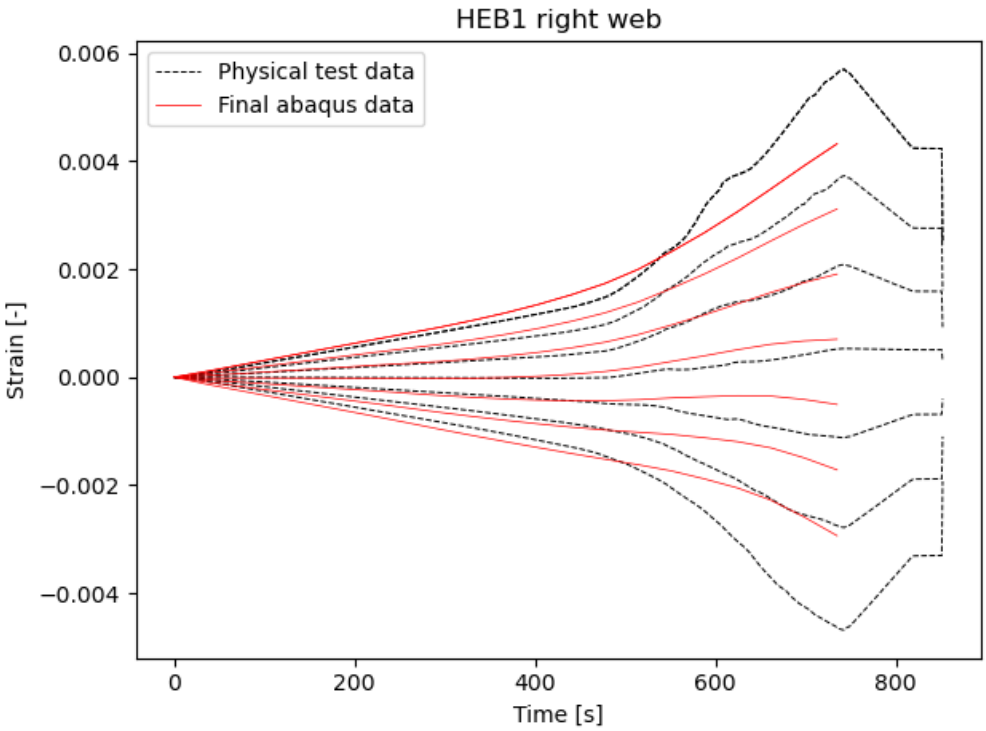


Figure I.5: HEB1 Extended run right web

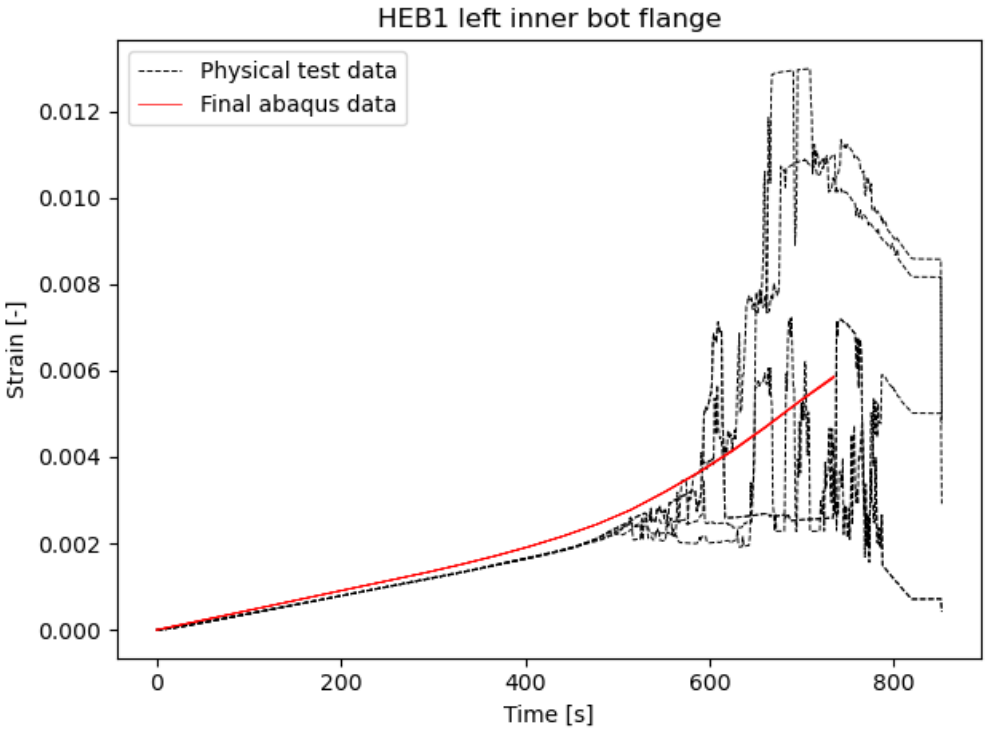


Figure I.6: HEB1 Extended run left inner bot flange

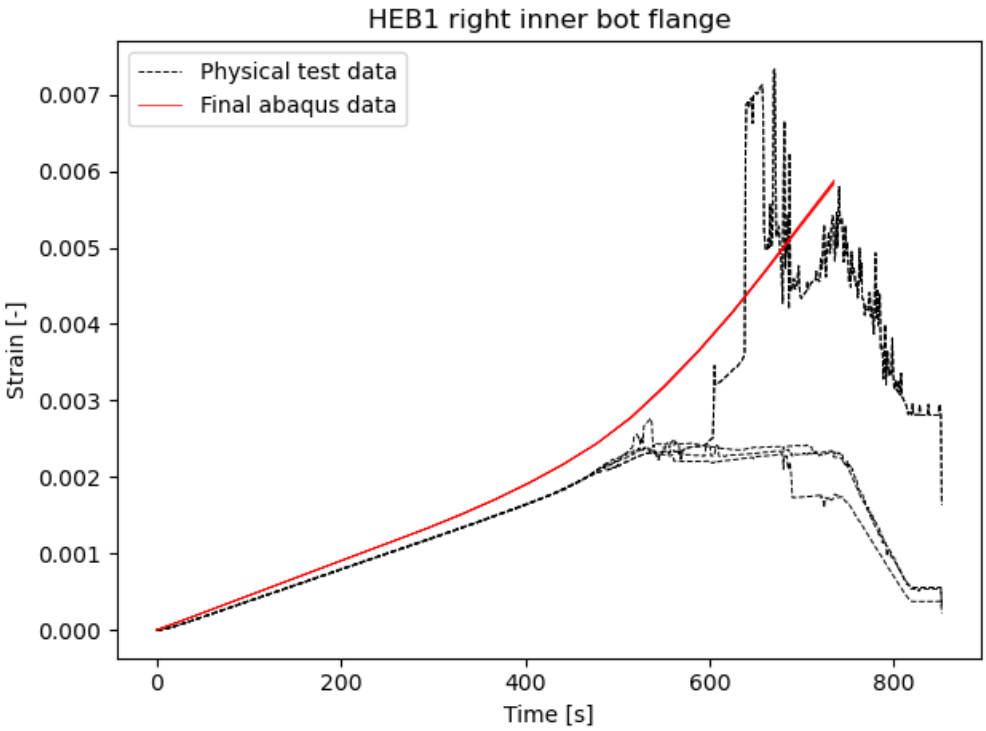


Figure I.7: HEB1 Extended run right inner bot flange

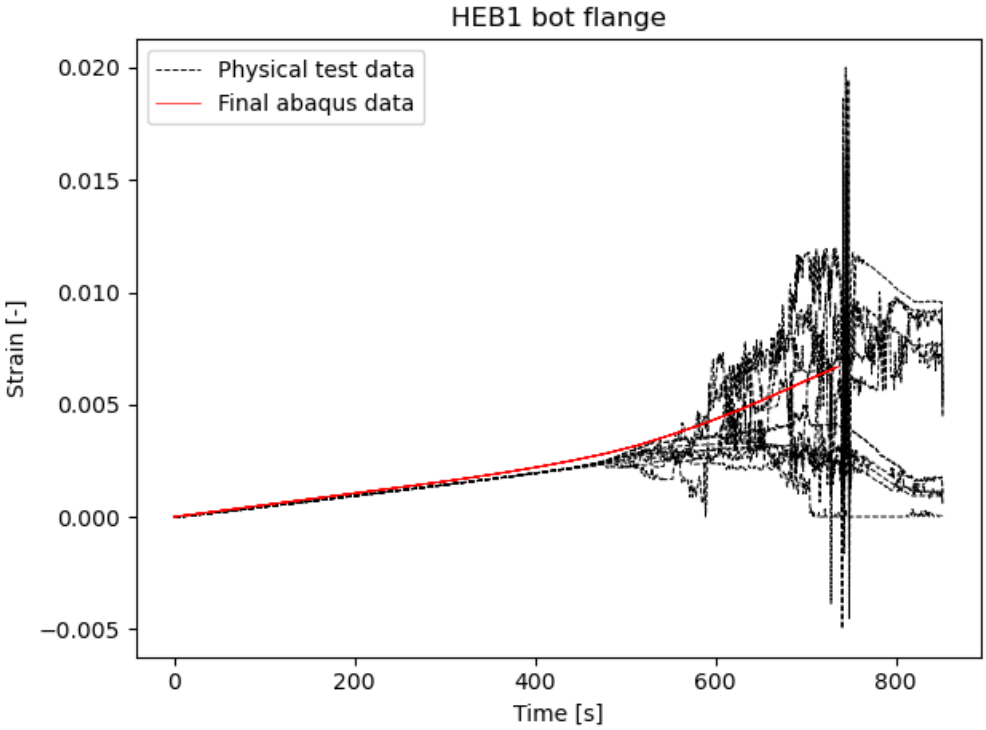


Figure I.8: HEB1 Extended run bot flange

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

CHALMERS
UNIVERSITY OF TECHNOLOGY