HORVÁTH, ANDRÁS LEVENTE
MSC THESIS

Budapest University of Technology and Economics
Faculty of Mechanical Engineering
Department of Applied Mechanics

Budapest University of Technology and Economics
Faculty of Mechanical Engineering
Department of Applied Mechanics

Parameter calibration software for the two-layer viscoplastic model

MSc THESIS

Author: HORVÁTH, ANDRÁS LEVENTE

Supervisor:
Dr. Kossa, Attila
Associate professor

Budapest, 2019

Budapest University of Technology and Economics
Faculty of Mechanical Engineering
Department of Applied Mechanics
1111 Budapest, Műegyetem rkp. 5., Building MM
www.mm.bme.hu

FINAL PROJECT A ASSIGNMENT (MSc)

Identification			
	Name: **András Levente HORVÁTH**		Student ID: **76444630881**
	Code of the Curriculum: **2N-MW0**	Code of the Specialisation: **2N-MW0-SM**	Document ref. number: **FPA-MM-19t / BHI0A4**
	Curriculum: **Mechanical Engineering Modelling**		
	Final Project issued by **Department of Applied Mechanics**	Final exam organised by **Department of Applied Mechanics**	
	Supervisor: **Dr. Attila KOSSA**, Neptun: CX3ZQO, ID: 71525329052 Associate Professor, kossa@mm.bme.hu, +36-1/463-1368		

PROJECT DESCRIPTION		
	Title	**PARAMETER CALIBRATION SOFTWARE FOR THE TWO-LAYER VISCOPLASTIC MODEL** *(Paraméterillesztő szofver a "Two-Layer Viscoplastic" anyagmodellhez)*
	Details	Detailed tasks of Final Project A and B: 1. Give a detailed description of the Two-Layer Viscoplastic model available in the commercial FE software ABAQUS! 2. Construct a numerical algorithm to solve the model response for incremental strain input! 3. Build a calibration software for the parameter optimization task in Python environment! 4. Demonstrate the applicability of the software using real experimental data! 5. Summarize the results in English and in Hungarian. Prepare a poster presentation of the work.
	Advisor	Advisor's Affiliation: --- Advisor: ---

AUTHENTICATION		
	Handed out: 4th February 2019	Deadline: 17th May 2019
	Compiled by …… Supervisor	Controlled and Approved by L.S. …… Head of Department
	The undersigned declares that all prerequisites of the Final Project have been fully accomplished. Otherwise, the present assignment for the Final Project is to be considered invalid. Budapest, 4th February 2019 ……. Student	

Declacations

Declaration of acceptance

This thesis meets all of the content and format requirements specified for Diploma Work and Thesis Work by the Faculty of Mechanical Engineering of Budapest University of Technology and Economics and also entirely meets all of the requirements of the project announcement. This thesis is considered to be suitable for public assessment and public presentation.

Date of submission:

supervisor

Declaration of individual work

I, the undersigned Horváth, András Levente (BHI0A4), a student at Budapest University of Technology and Economics, in awareness of my legal responsibilities, hereby duly declare and confirm with my handwritten signature below, that this thesis, and every part of it, has not been plagiarized, and that this work is solely of my own intellectual property and initiative, and that any literary works, tools, etc., that have been referenced or used in this work, have been referenced and used according to proper codes of conduct.

Budapest, 2019.

student

Abstract (Hungarian)

A különféle mechanikai számítások során az anyagok mechanikai viselkedését paraméteres egyenletekkel - anyagmodellekkel - írhatjuk le. Ezen számítások (pl. VEM szimulációk) pontosságához a megfelelő modell kiválasztásán túl elengedhetetlen az anyagparaméterek pontos ismerete, ezért ez a paraméteridentifikáció gyakori probléma a mérnöki gyakorlatban.

Diplomatervemben az ABAQUS [12] kereskedelmi végeselemes szoftverben elérhető "Two-layer viscoplastic" modell paramétereinek meghatározásával (avagy illesztésével) foglalkozom. A modell segítségével hatékonyan tudjuk modellezni a hőre lágyuló polimerek anyagi viselkedését, de fémekhez is egyaránt használható. Ez az anyagmodell két fő részből áll: az egyik rész írja le a rugalmas-képlékeny viselkedést, míg a vele párhuzamosan kapcsolt másik rész felelős a viszkoelasztikus viselkedés modellezéséért. A modell anyagparamétereinek száma a felhasznált nemlineáris elemektől függ. Az általam vizsgált modellnek összesen 7 paramétere van.

Az anyagparaméterek meghatározása egy optimalizálási problémaként írható le, melynek során a mért adatok és az anyagmodell által előrejelzett viselkedés közötti eltérését minimalizáljuk a modellparaméterek megfelelő megválasztásával. Összetett viselkedésű anyagok (mint a "Two-layer viscoplastic" modell) esetén ez egy sokdimenziós problémát jelent, ami már manuálisan vagy a legegyszerűbb algoritmusokkal sem kezelhető könnyedén.

A fenti modell paraméterillesztése elvégezhető kereskedelmi szoftverek használatával. Az irodalomban több helyen javasolt módszer az ABAQUS (vagy más végeselem program) összekötése egy külső optimalizálóval. Bár ez a módszer működőképes, egyetlen mérés kiértékelése sok órát vehet igénybe.

A munkám során egy olyan szoftvert fejlesztettem a Python környezetben, ami a paraméterillesztést sokkal hatékonyabban képes elvégezni. A program a Pythonhoz elérhető számos modult és saját fejlesztésű kódot egyaránt felhasznál. A programhoz grafikus felhasználói felület (GUI) is készült, ami könnyebben megtanulhatóvá és kényelmesebbé teszi a használatát. A paraméterek illesztése teljesen elvégezhető a GUI használatával, ahol megtalálhatóak az optimalizálási folyamat fontos beállításai is. Munkám során számos algoritmust és beállítást vizsgáltam meg, hogy a leghatékonyabb módszert megtaláljam.

Az elkészített program teljesíti a kitűzött célokat. A paraméterek meghatározása valós mérési adatokon néhány percbe telik, adott esetben 300-400x gyorsabban, mint a külső optimalizálót használó módszer esetén. Ezen kívül az így kapott anyagparaméterek által adott viselkedés jobban követi a mérési adatokat. A gyorsabb kiértékelés kutatási célokra is hasznos, például a paraméterek hőmérsékletfüggésének vizsgálata során.

Bár a dolgozatom a "Two-layer viscoplastic" modellel foglalkozik, az elkészített szoftverhez - a moduláris felépítés miatt - hozzáadhatók más anyagmodellek is, így a paraméter identifikáció azokon is elvégezhető.

Abstract

In the field of mechanics, we can describe the behavior of materials with parametric equations (material models) during our calculations. For the accuracy of our calculations (e.g., FEM simulations), we must use an appropriate model. Furthermore, it is crucial to know the precise value of the material parameters. For this reason, the identification of these parameters is a common task in engineering practice.

In this thesis, the parameter calibration (or identification, fitting) of the "Two-layer viscoplastic" model found in the commercial finite element software ABAQUS [12] is examined. This model can efficiently model thermoplastic polymers, but it can be used for metals as well. This model consists of two main parts, which are in parallel connection. One describes the elastic-plastic behavior, while the other is responsible for the modeling of the viscoelastic effect. The total number of parameters depends on the nonlinear elements used in the model. The particular model I examine has 7 parameters.

The calibration of the material parameters can be described as an optimization problem where the difference between the measured data and the behavior predicted by the model must be minimized via the appropriate choice of the material parameters. In case of complex material behavior (like two-layer viscoplasticity), this is a multi-dimensional problem, which cannot be treated easily manually or by the simplest algorithms.

To obtain the material parameters of the model mentioned above, commercial products can be used. The approach recommended by the literature is to connect ABAQUS (or another finite element software) with an external optimizer. While this method does work, it can take several hours to fit the parameters on a single measurement file.

My task was to develop a software in the Python environment, which is able to perform the parameter calibration much more efficiently. The software relies on the vast array of modules available for Python, as well as custom code. The program was given a graphical user interface (GUI) to make its use easier to learn and more convenient. The calibration process can be done entirely using the GUI, as it contains the important settings for the optimization process as well. During my work, I have investigated several algorithms and settings to determine the most effective way of the parameter fitting.

The created software achieved its goal. When tested on real measurement data, it gave results in a few minutes, sometimes 300-400x faster than the method using an external optimizer. Furthermore, the obtained parameter values modeled the behavior more closely. The faster evaluation of measurements is useful for research purposes as well, e.g., during the investigation of the temperature dependence of the parameters.

While my thesis focuses on the "Two-layer viscoplastic" model, other material models can be added to my software due to its modular design. The parameter identification can be carried out on these added models as well.

Keywords: two-layer viscoplasticity, parameter fitting, optimization, Python

Contents

Acknowledgments

At the beginning of my thesis, I thank Attila Kossa for introducing me to this topic, providing guidance in the design of the software, and helping the making of this thesis. Special thanks to him for this topic could be presented on the TDK conference.

I also thank András Horváth for both his professional and parental support. Finally, I thank the support and patience of each family member and friend.

This research was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences and Hungarian National Research, Development and Innovation Office (NKFI FK 128662).

Budapest, 2019. december

Horváth A. Levente

Symbols, notations

During my thesis these notation will be used:

Engineering strain in a specimen with L_0 initial length and ΔL length change under the uniaxial load:

$$\varepsilon_{eng} = \frac{\Delta L}{L_0} \text{ [-]}. \tag{1}$$

True or logarithmic or Hencky strain in a specimen with L_0 initial length and ΔL length change under the uniaxial load:

$$\varepsilon_{true} = \ln\left(\frac{\Delta L + L_0}{L_0}\right) \text{ [-]}. \tag{2}$$

Conversion from engineering strain to true strain:

$$\varepsilon_{true} = \ln(1 + \varepsilon_{eng}). \tag{3}$$

Conversion from engineering strain rate to true strain rate:

$$\dot{\varepsilon}_{true} = \frac{\partial \ln(1 + \varepsilon_{eng})}{\partial \varepsilon_{eng}} \dot{\varepsilon}_{eng} = \frac{\dot{\varepsilon}_{eng}}{1 + \varepsilon_{eng}}. \tag{4}$$

Engineering stress in a specimen with A_0 initial cross section and under F uniaxial loading force:

$$\sigma_{eng} = \frac{F}{A_0} \text{ [Pa]}. \tag{5}$$

True stress in a specimen under F uniaxial loading force with A_c current cross section:

$$\sigma_{true} = \frac{F}{A_c} \text{ [Pa]}. \tag{6}$$

Conversion from true stress to engineering stress in one dimensional incompressible case:

$$\sigma_{true} = \sigma_{eng}(1 + \varepsilon_{eng}). \tag{7}$$

In small strain theory ($\varepsilon_{eng} \leq 0.05$) the two strain measures are approximately the same: $\varepsilon_{eng} \approx \varepsilon_{true}$. In this case, the engineering quantities can be used. In finite strain ($\varepsilon_{eng} > 0.05$), it is necessary to use the true strain and stress.

A **load case** is the force or displacement load on a material, given as a function of time. In my thesis, usually, $\varepsilon_{eng}(t)$ will be given.

Note

In my thesis, in some equations, I did not specify if they are in the small or the finite strain region. In this case, I have used ε and σ. If ε_{eng} and σ_{eng} is substituted respectively, then the equation is true for small strain theory. Substituting ε_{true} and σ_{true} gives the equation for finite strain.

Chapter 1

Introduction

1.1 Overview

In the last decades, the significance of computer simulations has increased year-by-year in many fields of engineering practice. An essential factor is the available computational power, which increased by magnitudes. This makes it possible to use highly accurate numerical approximations in complex design cases instead of analytical formulae for simplified cases or empirical formulae.

As the investigated phenomena became more complex, more complex equations had to be used to describe them accurately. The analytical solution of these equations is often unknown. However, numerical methods can provide good enough approximations to use these equations during engineering tasks. One of the most famous examples is the Navier-Stokes equations. Their analytical solution is still unknown after well over 100 years, but numerical solutions provide good enough approximations to solve many problems in practice (e.g., design airplanes).

In the field of mechanics, some material models cannot be solved in closed form as well. In this thesis, I will be investigating the 'Two-layer viscoplastic' model, which does not have a known analytical solution either.

It is essential to determine the precise value of the parameters in the material model equations o approximate the real phenomena accurately. It is possible to carry out several measurements, each focusing on a different type of behavior and determine the parameters this way. However, this method has multiple limitations.

Another method is to use data from a single measurement and fit the material parameters by software designed for this task. During the parameter fitting, we compare the calculated material behavior to measurement data for a specific load case. The fitting process can be treated as an optimization problem, where the difference between the calculated and measured behavior must be minimized via the appropriate choice of the material parameters.

1.2 Goal

The previously mentioned optimization task is often too complicated to do manually or handle it by trivial algorithms. Such software do exist, but they are not freely available. (Source: [10], [11].) The goal of my thesis is to create a software, which is able to perform the parameter fitting.

This software must perform this task efficiently, meaning under as short time as possible. Efficiency will be measured against an other method using commercial software.

The software must have a graphical user interface (GUI) so it can be used effectively

without any programming background. The important settings of the material fitting process should be available on the GUI.

Finally, this software should have a modular structure. While my thesis focuses on the two-layer viscoplastic model, most of the program code is not specific for this particular model. Having a modular structure allows the software to be easily expanded by new material models, numerical methods, optimization algorithms, etc. These expansions should work with the existing GUI.

1.3 Structure

In the second chapter of my thesis, I am going to give a detailed description of the 'Two-layer viscoplastic model' in general. I will introduce the particular configuration I used during my work. The notations specific for this model will be introduced. This will be followed by the description of the parameter fitting process.

The third chapter includes the details of the developed software. First, the programming language and used modules are introduced. Then I will write about the developed graphical user interface, its design, and functionality. After this, I describe a few test cases which I will use later to compare different methods. Then the numerical solution of the material equations is shown, including the comparison of different numerical methods. This is followed by the introduction of the optimizer algorithms and their settings. This part includes some comparisons between different algorithms and settings. Finally, I briefly describe the internal organization of the code.

The fourth chapter includes the results obtained on real measurement data. First, the measurements are described. Then I show the results of the parameter fitting on the shown data. The process of the optimization is analyzed on a selected case. Based on the results, the temperature dependence of parameters is investigated. Finally, I compare the performance of my software to another parameter calibration method, which uses commercial software. I show the efficiency of my method.

In the final main chapter, I evaluate the developed software by investigating the goals set at the start of my thesis. I will briefly write about potential ways to improve the program. After this, the results are summarized.

The references and sources are included in chapter six. This is followed by the Appendix. Appendix A contains the plots of the fitted curves on the measurement data. Appendix B contains a step-by-step user guide.

Chapter 2

The two-layer viscoplastic model

The two-layer viscoplastic model (denoted as TLVP from now on) was proposed by Kichen in 1992. (Source: [6], [1]) While the original model was developed for polymers, the implementation in the finite element software ABAQUS is recommended for metals at elevated temperatures as well. In general, this implementation is meant to model materials, where plasticity, as well as time-dependent behavior, is observed. (Source: [12], [13])

The TLVP model, in general, consists of two 'branches', which are in parallel connection. One branch models the elastic-plastic behavior, the other is responsible for the viscoelastic behavior. This generic configuration in 1D can be seen in Figure 2.1. The specific elements are not determined in the branches, only the type of behavior they show. The particular elements in the model can be changed to e.g., some nonlinear elements. For this reason, this model has numerous particular versions.

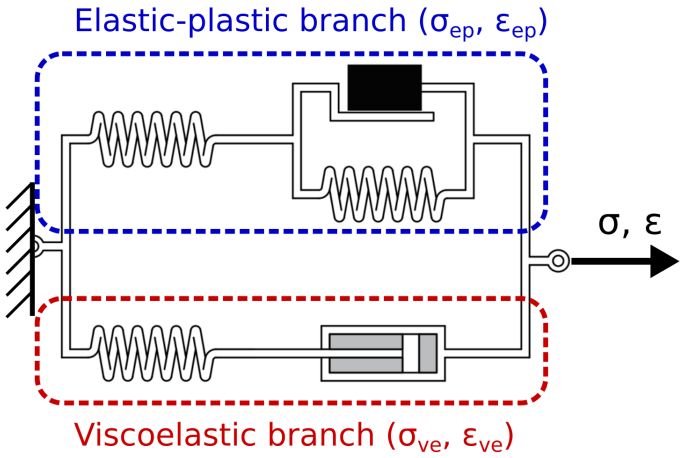

Figure 2.1. The branches of the generic 1D model

2.1 Particular configuration

The version discussed in this thesis is identical to the one presented in Figure 2.1. The equations were written for the 1-dimensional case, as this represents uniaxial loading. 1D equations are also much easier to solve numerically.

2.1.1 Elastic-plastic branch

Parameters

This branch shows linear isotropic hardening, which can be described with 3 material parameters.

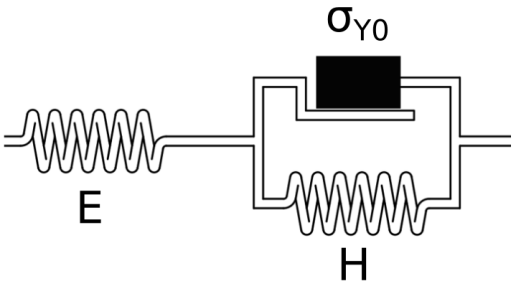

Figure 2.2. The elastic-plastic network

- E - Elastic modulus [Pa]
- H - Plastic hardening modulus [Pa]
- σ_{Y0} - Initial yields stress [Pa]

All of these material parameters (E, H, σ_{Y0}) must be positive and usually $H < E$.

Behavior

From parameters above the elastic-plastic modulus E_{ep} can be calculated as:

$$E_{ep} = \frac{EH}{E+H}. \tag{2.1}$$

During loading, the material has linear elastic behavior with elastic modulus E as long as the stress is lower than the current yield stress σ_Y. If the stress exceeds the yield stress σ_Y, the material enters the plastic region. The further stress changes will be calculated according to the elastic-plastic modulus E_{ep} as long as the material is not unloaded.

If the stress exceeds the yield stress σ_Y, the current yield stress is updated to the new maximum absolute stress. This behavior can be seen in Figure 2.3.

Equations

The yield function F defines the elastic and elastic-plastic region as:

$$\text{yield function: } F = |\sigma| - \sigma_Y \leq 0, \tag{2.2}$$

$$\text{elastic region } (F < 0)\text{: } \dot{\sigma} = E\dot{\varepsilon}, \tag{2.3}$$

$$\text{elastic-plastic region } (F = 0)\text{: } \dot{\sigma} = E_{ep}\dot{\varepsilon}. \tag{2.4}$$

$F > 0$ has no meaning by definition.

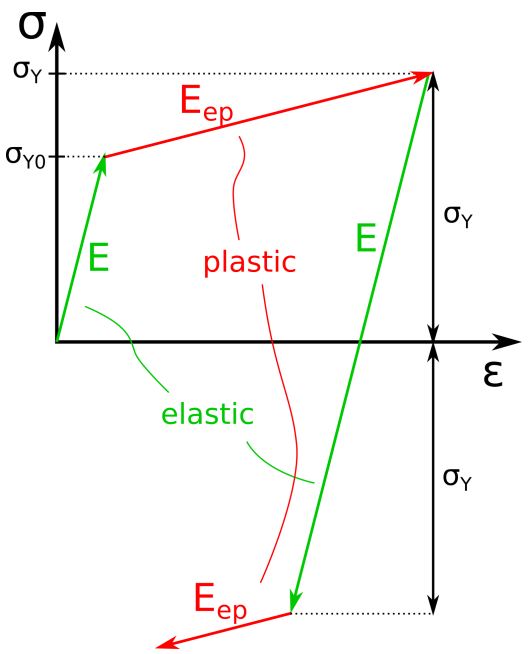

Figure 2.3. Isotropic hardening behavior

2.1.2 Viscoelastic branch

Parameters

This branch contains a spring-like and a dashpot-like element. In my case, the spring is linear, and the damping is nonlinear with time hardening. In total, this branch is characterized by 4 material parameters.

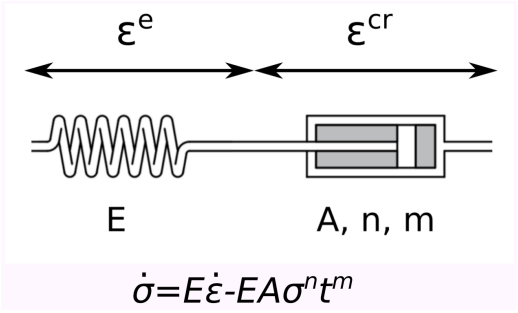

Figure 2.4. The viscoelastic network

- E - Elastic modulus [Pa]
- A - Nonlinear member multiplier [Pa^{-n}s^{-m-1}]
- n - Exponent of σ [-]
- m - Exponent of time [-]

From these parameters E, A and n must be positive and $-1 < m < 0$. A is typically several orders of magnitude lower than E. n is typically between 1 and 5.

These equations are true for both small-strain and finite strain. In case of small-strain, we interpret ε as ε_{eng} and σ as σ_{eng}. In case of finite strain we use ε_{true} and σ_{true} respectively.

Note: as indicated before, the unit of A is [Pa^{-n}s^{-m-1}]. However, to simplify notations I will use [-] for the rest of this thesis.

Behavior

During loading, this material builds up stress. When the deformation amount is kept constant, relaxation happens, meaning that the stress will eventually reach zero in the deformed state. A typical example can be seen in Figure 2.5.

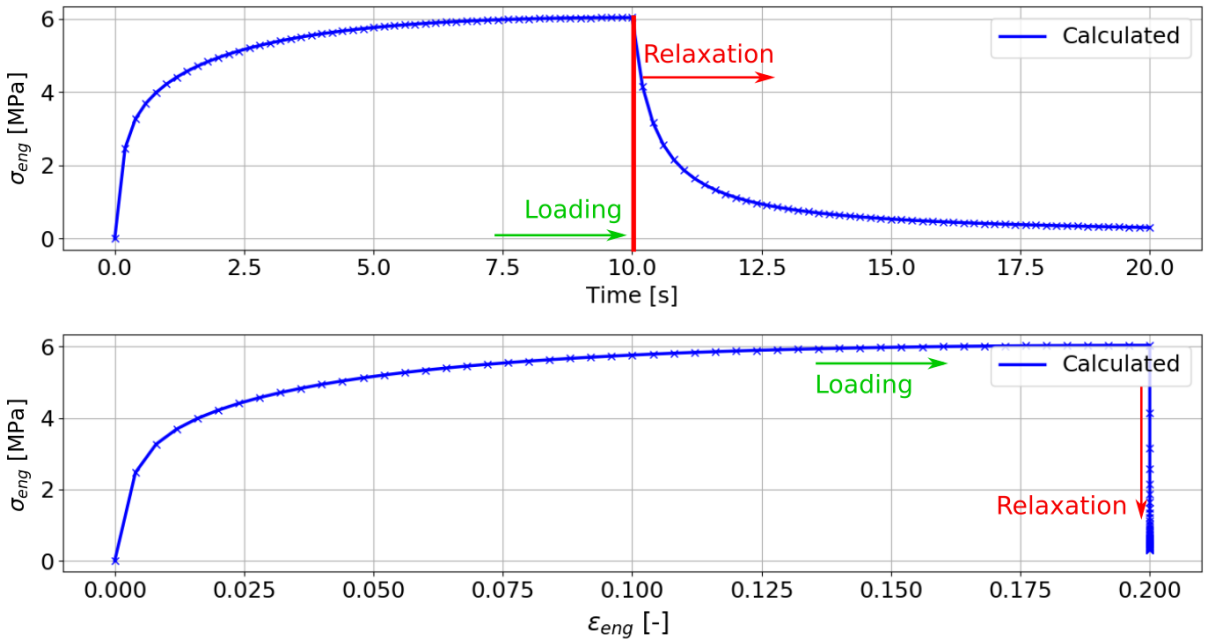

Figure 2.5. Typical viscoelastic behavior

Equations

Let us denote the elastic deformation of the spring with ε^e and the deformation of the dashpot - which is responsible for the creep - with ε^{cr}. The total deformation of the viscoelastic network is:

$$\varepsilon = \varepsilon^e + \varepsilon^{cr}. \tag{2.5}$$

We can write the following equation for the spring:

$$\varepsilon^e = \frac{\sigma}{E}. \tag{2.6}$$

The same equation in differential form:

$$\dot{\varepsilon^e} = \frac{\dot{\sigma}}{E}. \tag{2.7}$$

For the dashpot we can write the equation in differential form:

$$\dot{\varepsilon^{cr}} = A\sigma^n t^m, \tag{2.8}$$

where t denotes time.

If we substitute (2.7) and (2.8) into the time derivative of (2.5) we get:

$$\dot{\varepsilon} = \frac{\dot{\sigma}}{E} + A\sigma^n t^m. \tag{2.9}$$

We can express σ in differential form:

$$\dot{\sigma} = E\dot{\varepsilon} - EA\sigma^n t^m. \tag{2.10}$$

Unfortunately, the general analytical solution of equation 2.10 is unknown. However, it can be handled by numerical methods.

These equations are again true for both small-strain and finite strain. In case of small-strain, we interpret ε as ε_{eng} and σ as σ_{eng}. In case of finite strain we use ε_{true} and σ_{true} respectively.

2.1.3 Full network

In total this particular configuration has 7 material parameters.

As the two branches are in parallel connection, the deformation is identical in both. The stresses can be added to obtain the stress in the complete system. Using the notations presented on Figure 2.1:

$$\sigma = \sigma_{ep} + \sigma_{ve}, \tag{2.11}$$

$$\varepsilon = \varepsilon_{ep} = \varepsilon_{ve}. \tag{2.12}$$

2.2 Notations based on ABAQUS

The notations used in the ABAQUS finite element software can be seen in Figure 2.6. (Source: [13])

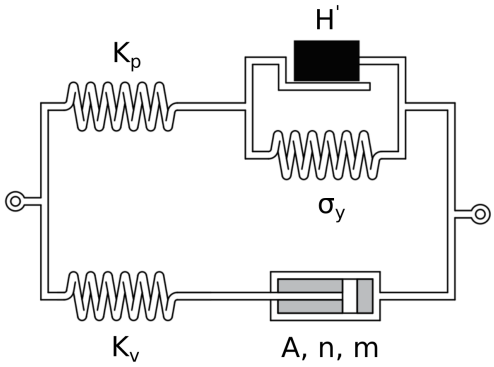

Figure 2.6. The notations used by ABAQUS

However, in the software, K_p and K_v are not given directly. E and f is given instead, where E is the 'total elastic modulus', and f is the proportion of the viscoelastic part in it. In this case E is positive and $0 \leq f \leq 1$. My software also uses this kind of notation instead of giving K_p and K_v.

$$E = K_p + K_v. \tag{2.13}$$

$$f = \frac{K_v}{K_v + K_p}. \tag{2.14}$$

The notations used in ABAQUS and in my thesis and software differ only very slightly. However, the table below lists all of them to avoid any confusion.

Table 2.1. Notations

Quantity	ABAQUS notation	My notation
Total elastic modulus	E	E
Viscoelastic proportion	f	f
Hardening modulus	H'	H
Initial yield stress	σ_y	σ_{Y0} or $sY0$
Nonlinear member multiplier	A	A
Stress exponent	n	n
Time exponent	m	m

2.3 Parameter fitting

Parameter fitting can be done in multiple ways, as mentioned in the Overview section.

2.3.1 Suggested method by ABAQUS

The ABAQUS manual does describe a way to do the calibration of the material parameters. (Source: [13]) This method requires multiple measurements, each focusing on different properties of the material. During this process, $m = 0$ is assumed, and time hardening is ignored. This leaves 6 parameters to calibrate: K_v, K_p, σ_y, H', A, and n.

The experiments need to be performed in uniaxial tension at different constant strain rates. Using a very low strain rate (effectively static load), the elastic-plastic branch dominates the behavior. The stress in the viscoelastic branch is 'relaxed out' too quickly in this case. This way the parameters of the elastic-plastic branch (K_p, σ_y, H') can be determined.

$E = K_p + K_v$ can be determined by using the initial section of the measurement at a relatively high strain rate. It is recommended to repeat this experiment several times with different strain rates to obtain a result independent from the strain rate. From this, K_v can be calculated as $K_v = E - K_p$.

At constant applied strain rate ($\dot{\varepsilon}_0$) the steady-state behavior of the viscoelastic branch yields constant stress:

$$\sigma_v = A^{-\frac{1}{n}} \dot{\varepsilon}_0^{\frac{1}{n}}. \tag{2.15}$$

Assuming that $K_p \gg H$ the behavior of the entire material in steady-state:

$$\sigma = A^{-\frac{1}{n}} \dot{\varepsilon}_0^{\frac{1}{n}} + \sigma_{Y0} + H\dot{\varepsilon}_0. \tag{2.16}$$

To determine if the steady-state was reached, one can plot $\overline{\sigma} = \sigma - \sigma_{Y0} - H\dot{\varepsilon}_0$ and see where it becomes constant. This constant value of $\overline{\sigma}$ will be equal to σ_v. By conducting several experiments like this, the value of A and n can be determined.

Issues

The reason I described this method in such detail is to demonstrate that it is possible to calibrate the parameters without the heavy use of computer algorithms. However, this method has several drawbacks.

First, during the entire process, $m = 0$ is assumed. This means that it is not possible to calibrate m with this method. The other assumption is $K_p \gg H$. We will be able to see from the results, that this is not necessarily true either, sometimes $K_p \approx 2...5H$. These assumptions might affect the accuracy of the parameter fits negatively.

It is also clear from this description that several measurements are needed to determine all 6 material parameters. (Unfortunately, the guide does not give an approximate number for the number of experiments needed.) During these experiments, the material specimen might get permanently damaged or destroyed. For example, from Equation (2.16), we can see that the material is expected to enter the elastic-plastic region - this means permanent deformation.

Overall, this can be a time consuming and expensive process. It is also possible that we don't have enough specimens from the material, especially if investigating the thermal dependence of the parameters. For examining thermal dependence, this series of experiments need to be repeated on multiple temperature levels, further amplifying the disadvantages of this method.

2.3.2 General method

For the reasons mentioned above, it would be beneficial to find a method that can calibrate the material parameters based on a single (or only a few) measurements.

Parameter fitting via computer optimization offers a solution to this problem. With this method, the material equations can be solved many (thousands of) times for the load applied during the measurement. The simulated behavior can be compared to measurement data. The difference between the calculated and measured behavior can be expressed by a scalar. This scalar should get smaller the closer the curves get to each other.

Overall the scalar measuring the difference is a function of the material parameters. This can be treated as a multi-dimensional optimization problem where this function should be minimized. There are numerous algorithms available for minimizing such scalar-valued vector functions. The optimizer calculates new parameters to check based on the previous attempts. When the difference becomes low enough (or a certain number of attempts is exceeded), the optimization stops. The 'calculation-evaluation-new parameters' cycle is called the optimization loop.

As no assumptions are made, the results obtained with this method might be more accurate than the results from the previous method. Determining the material parameters from a single measurement saves work hours and material specimens as well. Instead of human work hours, computational time is used.

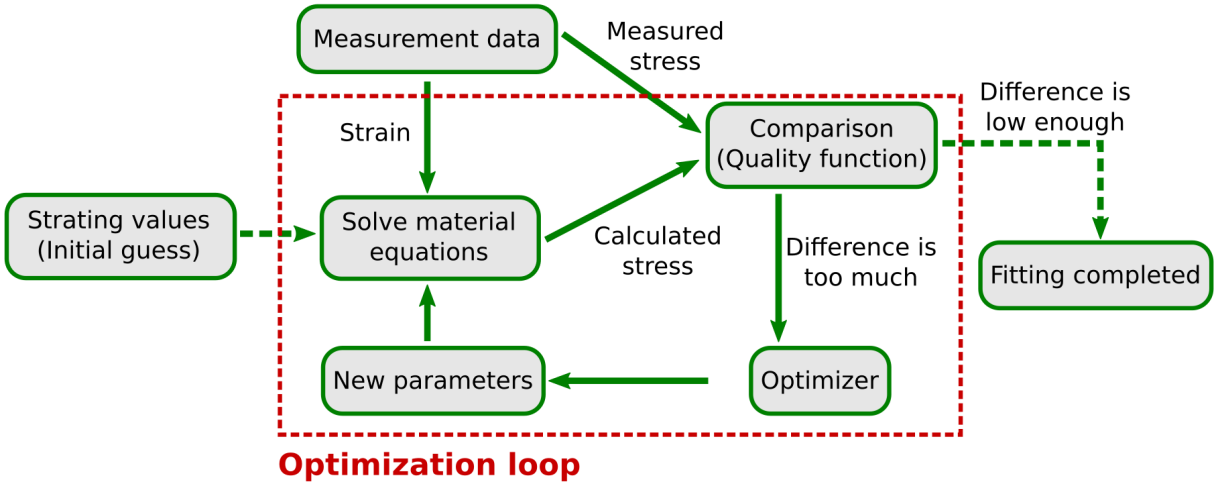

Figure 2.7. The flowchart of the genaral method

Chapter 3

The calibration software

As mentioned in the Introduction chapter, fitting or calibration the material parameters can be a complicated process which cannot be treated with trivial methods. This configuration of the TLVP model has 7 parameters, meaning that the simplest approaches will not be effective. A complex optimization algorithm is needed to obtain a good solution. Currently, there is no freely available software that would handle this task.

The commercial material parameter fitting software I found do not handle the TLPV model either. The calibration can be solved by connecting a FEM software (which has the TLPV model, e.g., ABAQUS) and an external optimizer. However, this method is magnitudes slower than a single software that can solve the material model equations and do the optimization as well. Naturally, it would be hard to compete with commercial FEM software in case of analyzing complex 3D geometry. However, here we only consider the material equations in a 1D case. As optimization is an iterative process, the FEM software loses a significant amount of time in each iteration because e.g., it has to start up for each iteration. As we will see later in this thesis, using FEM software with an external optimizer does provide a solution for parameter fitting. However, this process can take several hours.

The main goal of my software is to provide an efficient method for parameter fitting. Also, the user interface can be designed according to the task, so it is easy for new users to learn it.

Most of the programming effort (or lines of code) are not specific to the TLPV model. For this reason, I wrote the code in a way that new materials or numerical methods can be added to the existing software. These new models will appear on the interface (after a program restart). They will also work with the optimization methods, etc.

There are numerous programming languages, which could be used to create such software. For applications where performance is important, C/C++ is generally considered a good solution. However, developing software in C/C++ is considerably slower and less productive for someone without particular knowledge of this field.

For this reason, Python was used for the development of the parameter fitting software. I have used previously, it is suitable for quick development, and it has satisfactory performance when the right modules are used.

3.1 The Python programming language

Python is a high level, object-oriented, interpreted, general-purpose programming language. The first version was published by Guido van Rossum in 1991. Its design focuses on code readability and making software development easier over performance. Since 2001, it is developed by Python Software Foundation. (Source: [15], [14])

Over the past 5 years, Python was one of the most rapidly growing languages. While it is difficult to measure how 'important' and 'good' a programming language is, Python is definitely one of the most used programming languages today. It is used scientific applications, artificial intelligence, web development, etc. (Source: [16], [17])

3.1.1 Advantages of Python

There are several reasons why Python became so popular. For my work, these were the most important ones:

- It is free. Python is open source and in active development, meaning the current and all future versions can be freely downloaded and used for any purpose freely. This is a major benefit compared to e.g., MatLab or Wolfram Mathematica, as there is no licensing needed. Python can be installed and kept up-to-date on any number of machines.
- It is platform-independent. Python is available on all major operating systems. After installing the interpreter, the code can run without any modifications. Python code can also be 'packed' into self-contained executables, so an interpreter is not needed. (However, these executables are no longer platform independent.)
- Well-made third-party modules. The Python community created several powerful modules that are easy to install thanks to the Python developers supporting external modules. These modules are easy to learn, as they are logically built similarly to Python. These modules are a fundamental part of why Python is so popular. They make development as well as runtimes magnitudes faster. Most modules are free to use for any purpose.
- Well-structured code. As the language has strict syntax, the structure of the code is more consistent. This helps code to remain readable and understandable even in larger programs. Naturally, a lot depends on the programmer, but the syntax helps to keep the code organized.

(Source: [15], [18])

3.1.2 The most important used modules

3.1.2.1 NumPy

Its name comes from 'Numerical Python'. This is a fundamental module for scientific use and numerical computations. It contains data structures (e.g., C-like arrays) and functions essential for quick numerical applications. It often utilizes highly optimized C/C++ or Fortran code in the background. C/C++ and Fortran still provide one of the fastest running codes in numerical applications. This implementation allows Python to use the speed of C/C++ and Fortran while keeping the benefits of Python. (Source: [19])

3.1.2.2 SciPy

The name comes from 'Scientific Python'. This module contains a wide array of numerical methods e.g., for solving differential equations and other tools for interpolation, statistics, etc. It also contains the optimization algorithms I used for parameter fitting. It is used most commonly combined with NumPy and utilizes C/C++ and Fortran code as well. (Source: [20])

3.1.2.3 Matplotlib

The main purpose of this module is to create high-quality 2D plots. However, it is also able to create animations and 3D plots, as well. It is often used for static figures, but interactive plots can be made as well. (Source: [21])

3.1.2.4 PyQt5

Qt is a C++ library for creating platform-independent graphical user interfaces. The PyQt5 module makes it possible to use version 5 of Qt from Python. Unlike the other listed modules, this module is not completely free. License fee has to be paid for selling software made with it. However, as my program is not made for commercial distribution, I chose this graphical interface module because it is platform-independent, powerful, and easy to use. (Source: [22])

In Qt, the elements (labels, input fields, buttons, etc.) are called 'widgets'. The positioning of widgets relative to each other is done with 'layouts'. One of the most useful abilities of Qt is that layouts can be 'nested', so a layout can position other sub-layouts (which contain their own widgets). This allows for great flexibility when designing the interface, as an entire set of widgets can be repositioned easily without breaking their relative positions to each other.

Moreover, Qt and the layouts handle the basic GUI functions without having to program them, e.g., closing, minimizing, or resizing the window.

3.2 Graphical user interface (GUI)

The GUI was made using the previously presented PyQt5 module. As the focus of my thesis is on the parameter fitting, I will not go into great detail regarding the internal structure of the GUI.

The most important design principle was that the layout of the GUI should logically follow the steps of the optimization workflow. The other principle was to keep the area of the plot as large as possible. The user input area is on the upper part, while the bottom is only for plotting the results.

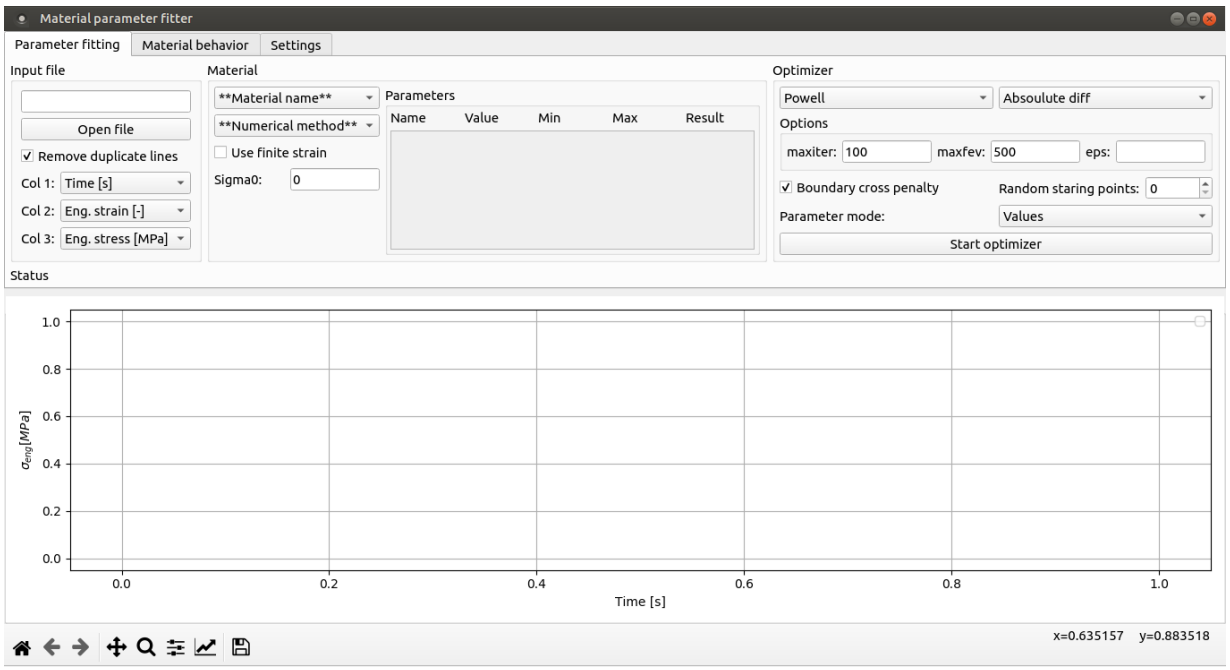

Figure 3.1. The GUI of the program after a fresh start

The user input area is organized into three tabs based on the type of functionality it is responsible for. These are visible on the top-left part of the GUI.

3.2.1 Parameter fitting tab

This is the most important part of the GUI, as it contains what is needed to do material parameter calibration. The widgets are organized into three groups.

3.2.1.1 Input file

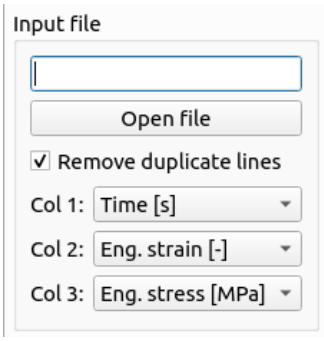

Figure 3.2. The file input group

In the 'input file' group, the user can pick the file containing the measurement data used for fitting the material parameters. The input file must be in .csv format. The first 3 columns will be considered only. These columns should contain the time data in $[s]$, the engineering strain data, and the engineering stress data in [MPa]. The order of these columns can be defined by the user. By default, the order is time, engineering strain, engineering stress. Alternatively, the user can select true strain instead of engineering strain. Every type of

data (time, strain, stress) must be defined exactly once. The fitting process cannot be started otherwise. The data in the time column must strictly monotonously increasing for the file to be considered valid.

The input file may have a header or other comment rows. These must have a '#' as a first character.

As many measurement data files had a few duplicated rows for unknown reasons, the user can allow the program to remove these automatically.

3.2.1.2 Material selection and settings

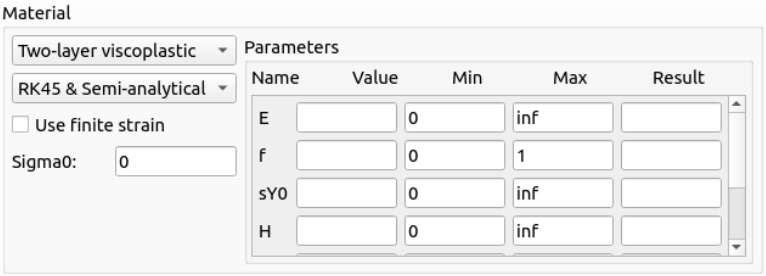

Figure 3.3. The material settings group

Here the user can select the material model which should be fitted on the measurement data. The numerical method used for calculating the material equations can be selected as well. If no numerical method is selected, the default will be used. The numerical implementation of the material equations will be discussed in their own section of this thesis.

The finite strain calculations can be enabled via a checkbox, and small strain theory is used otherwise. The results will be returned in engineering strain and stress, regardless of the selected strain theory. The initial stress in the material can be given in the 'Sigma0' input field.

On the right side, the user can set the material parameters. The 'Value' will be used by the optimizer as an initial guess. As these parameters have some constraints (e.g., most of them should be positive), the user can give their lower and upper limits in the 'Min' and 'Max' columns, respectively. By default, the theoretical limits are applied, 'inf' denotes ∞. The handling of these limits will be discussed later. The 'Result' column cannot be edited by the user; the result of the optimization will appear there.

3.2.1.3 Optimizer settings

Here the user can select the optimizer algorithm, the objective function, and other settings. The parameter fitting process can be started from here when everything has been set. As these settings have a great impact on the parameter fitting process, they will be discussed in detail in their own section of this thesis.

3.2.1.4 Status bar

A status bar is located on the bottom of this tab, where the program can display various messages to the user. These include error messages in case of incorrect or missing inputs.

During parameter fitting the iteration counter, the elapsed time and the R^2 value of the fitted curve is also displayed here.

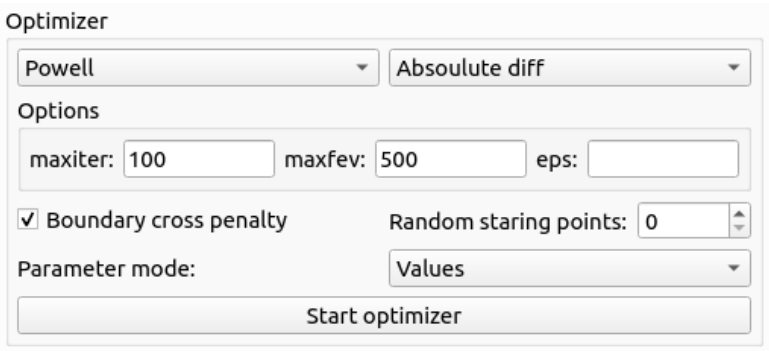

Figure 3.4. The optimizer settings group

3.2.2 Material behavior tab

In this tab, the user can solve the material model equations for a specific load case with specific parameters. This is useful for testing how the material would perform in these specific cases, comparing the speed of different numerical methods, the effect of parameter combinations on performance, etc.

Moreover, the results for specific cases can be compared to results obtained from other calculations (e.g., ABAQUS).

3.2.2.1 Material selection and settings

The material settings are mostly the same as described in the 'Parameter fitting' tab. The parameter limits are displayed as a reminder for the user, but they have no effect on the calculations. The 'Results' column is also not displayed on this tab, as the parameters are given by the user.

3.2.2.2 Load case

Defining a load case is not necessary for the parameter fitting process, but it can be used to study the material behavior under the specified load. The load case is defined by time and engineering strain data. The user can give up to 10 data points. Between these data points, linear interpolation is applied, the 'Splits' column defines the number of intervals defined between the two points. (This is defined at the endpoint of the interval.)

For example, if the user inputs are:

Figure 3.5. Defining a load case

then the resulting load case can be seen on Figure 3.6 (the 'x' markers denote the resulting loading data).

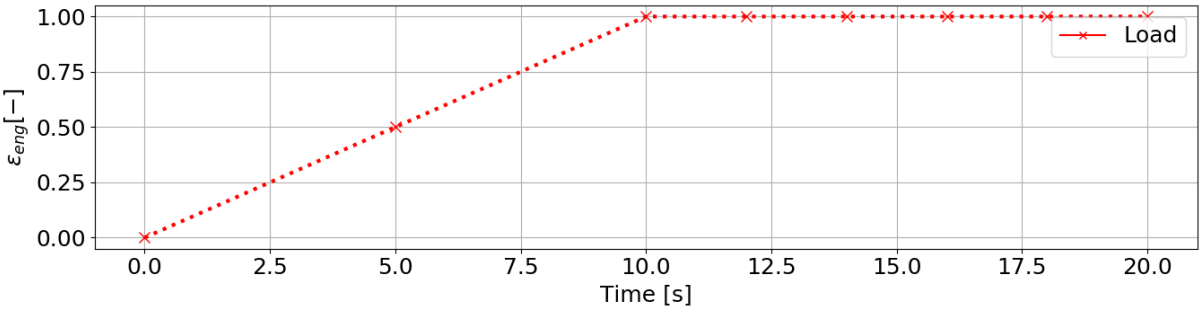

Figure 3.6. The defined load case

If the load case table is left empty, then the program will look for load data from the measurement file loaded in the 'Parameter fitting' tab and calculate using that data. If no load data is found, then the calculation cannot be started.

3.2.2.3 Status bar

Similarly to the parameter fitting tab, this tabs has it's own status bar. It will display error messages if needed, and it will display the calculation time of the material model in milliseconds.

3.2.3 Settings tab

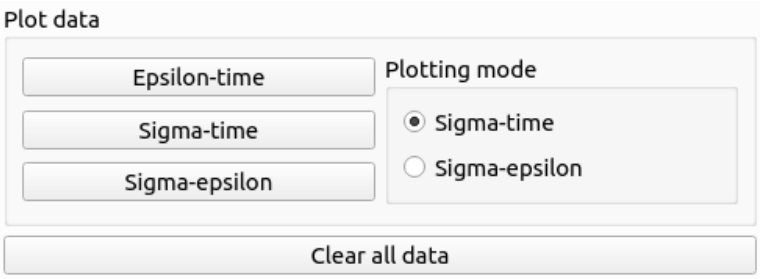

Figure 3.7. The available options

This tab contains the plotting settings. The user can switch between plotting the stress against time or strain. The plot is updated after switching the mode. Here the user can plot the measurement data in various formats.

All input measurement data and calculated and stored data can be cleared here as well. This can be useful if the user wants to check the behavior of a material, but he has a measurement file loaded. If the data is not cleared, the program will automatically draw the 'Difference' curve - even if the studied material is completely unrelated to the measurement file.

As this tab contains mostly 'convenience' functionality (i.e., this functionality is not needed directly for the parameter fitting), it received the least amount of attention during development. The number of available settings will be increased as the code is further developed.

3.2.4 Behavior during the parameter fitting process

While the parameter fitting process is running, the interface remains responsive, but the 'Start optimizer' button is disabled.

At the end of each iteration, the plotting area shows the latest result of the optimizer. At the same time, the 'Results' column is updated with the current material parameters. The status bar is also refreshed with the elapsed time and R^2 value of the current fit.

The optimizer algorithms are supposed to return the current best result after the iterations. This is used to visualize the improvements made in almost real-time. However, one of the algorithms (SLSQP) does not send the best result for unknown reasons. (The code is the same for all optimizer algorithms. It works as expected for every other algorithm.) This only affects the displayed and plotted values during the optimization, but not the final result. The type of plot ($\sigma_{eng} - t$ or $\sigma_{eng} - \varepsilon_{eng}$) can be changed during the process from the 'Settings' tab.

During the parameter fitting process, three logfiles are saved. One contains the material parameters after every function evaluation. The second is quite similar, but it contains the parameters handled by the optimizer. The third logfile records the optimizer parameters after every iteration, using the callback function.

These logs can be used to visualize the evolution of the solution (see section 4.3). They can be useful for debugging purposes as well during the development of the code.

3.3 Test cases for comparisons and benchmarks

In order to be able to compare numerical methods and measure performance in general, I have defined a few load cases as well as material parameter combinations. The load cases are similar to the later presented measurements, and they have increasing complexity. The parameters have the same order of magnitude as expected from previous parameter fittings.

The test cases do not have their temporal resolution pre-defined. The load case ($\varepsilon(t)$) is only given by a few points, and it is assumed to be linear between these points.

The material parameters are in [MPa] instead of [Pa]. This will be true for the stress values shown later in the thesis as well.

3.3.1 Test case 1

Table 3.1. Load case and material parameters for Test case 1

t [s]	ε_{eng} [-]
0	0
20	1
40	1
60	0.5
80	0.5

Parameter	Value
E [MPa]	800
f [-]	0.5
σ_{Y0} [MPa]	5
H [MPa]	20
A [-]	0.001
n [-]	2
m [-]	-0.5

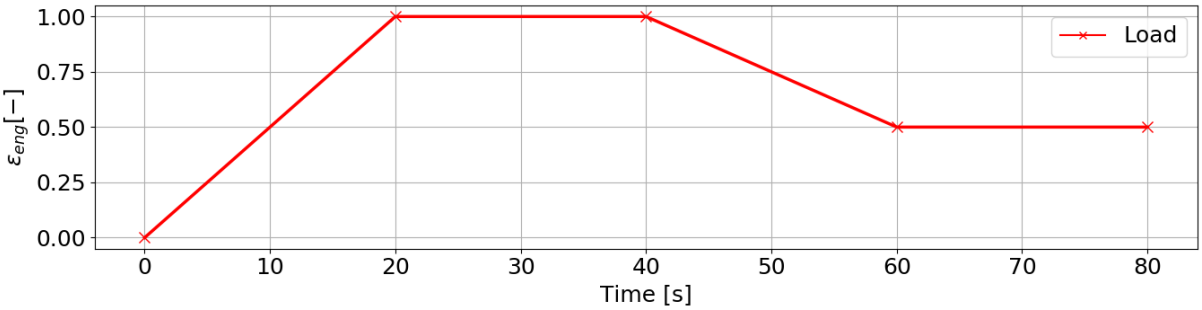

Figure 3.8. Test case 1

3.3.1.1 Test case 1/b and 1/c

These test cases are identical to Test case 1, with the exception of the parameter f. In Test case 1/b $f = 0.001$. This setup primarily examines the elastic-plastic behavior. In Test case 1/c $f = 0.999$. This setup primarily examines the viscoelastic behavior.

These cases were defined, so their numerical solution can be compared to the solution obtained by ABAQUS. This way, it can be examined, which branch is responsible for any potential difference between the two methods.

3.3.2 Test case 2

Table 3.2. Load case and material parameters for Test case 2

t [s]	ε_{eng} [-]
0	0
20	0.5
50	0.5
80	-0.5
100	-0.5

Parameter	Value
E [MPa]	500
f [-]	0.7
σ_{Y0} [MPa]	10
H [MPa]	30
A [-]	0.002
n [-]	2.5
m [-]	−0.3

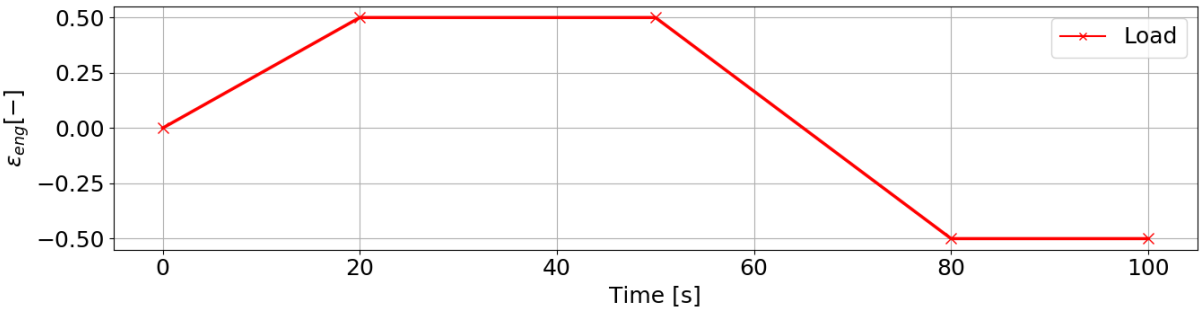

Figure 3.9. Test case 2

3.3.3 Test case 3

Table 3.3. Load case and material parameters for Test case 3

t [s]	ε_{eng} [-]
0	0
20	0.3
40	0.3
70	-0.2
90	-0.2
120	0.4
150	0.4

Parameter	Value
E [MPa]	400
f [-]	0.4
σ_{Y0} [MPa]	3
H [MPa]	50
A [-]	0.002
n [-]	3.5
m [-]	-0.6

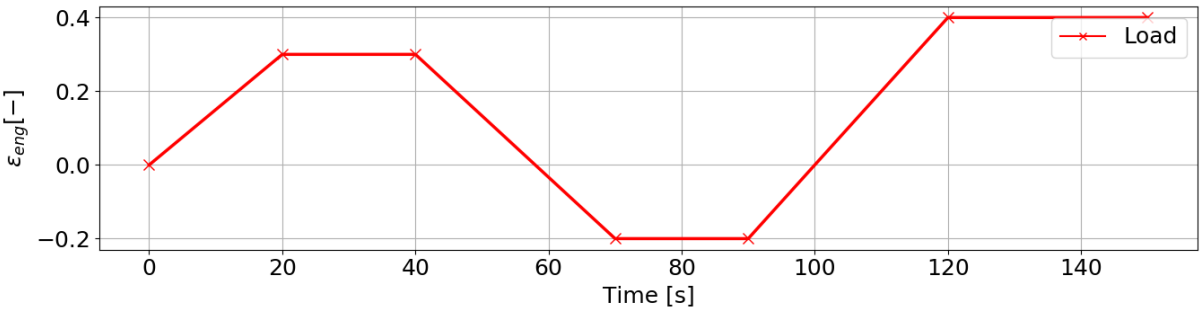

Figure 3.10. Test case 3

3.3.4 Test case 4 - for optimizer benchmarks

In order to be able to compare different optimizer settings, I have defined a problem based on the previous test cases. It is important to note that this is a single test case, and the optimizer algorithms can be sensitive to the exact settings and tasks. However, the most important trends will hopefully be demonstrated.

The load case is the same as in Test case 1 because it is similar to the measurements. The material behavior was simulated with 200 ms temporal resolution, then imported into the software as a measurement result file.

The test starts the optimizer from a 'good' initial guess. The quality of the guess is reasonable for real cases. This task is meant to test the speed of different algorithms.

3.3.5 A note on benchmarks

The presented results were measured on a laptop with the following main specifications:

CPU: Intel Core i7-4500U @ 1.80GHz (4 cores)

RAM: 8 GB (DDR3)

Operating system: Ubuntu 18.04.3 LTS 64-bit

Naturally, the runtimes would be different on other hardware.

It is important to note that my program mostly uses 1 core for the calculations. The individual measured runtimes have a noticeable deviation from the average (especially at

Table 3.4. Optimizer test case 4

Parameter	Accurate value	Initial guess	Lower limit	Upper limit
E[MPa]	500	800	300	1200
f[-]	0.7	0.5	0.1	0.9
σ_{Y0}[MPa]	10	3	1	30
H[MPa]	30	50	10	100
A[-]	0.002	0.001	0	0.1
n[-]	2.5	1.5	1	5
m[-]	-0.3	-0.5	-0.9	-0.1

very short times). For this reason, I have tried to measure the runtime several times and average the results.

3.4 Numerical solution of the material model

We have to solve the material model equations with the specific material parameters in order to be able to compare the results with the measured data.

During the calculation, the material parameters and the load case (ε, $\dot{\varepsilon}$) are always known.

In small strain, the obtained results are in engineering stress. However, in finite strain theory solving the equations gives true stress. As the measurement data is expected to be in engineering stress, the calculated stress for finite strain needs to be converted into engineering stress as well. This is done after the calculations, using Equation (7).

3.4.1 Elastic-plastic branch

The equations describing the behavior are given in subsection 2.1.1. Two methods are implemented in the program to solve the equations numerically.

We have 3 material parameters (E, σ_{Y0}, H) and E_{ep} is calculated according to Equation (2.1):

$$E_{ep} = \frac{EH}{E+H}.$$

3.4.1.1 Incremental method

This method is solving the equations incrementally. In each step, the solver does a trial step. If calculating the $(n+1)$th state form the known nth state:

$$\Delta\sigma_{trial} = \sigma_n + E \cdot (\varepsilon_{n+1} - \varepsilon_n). \tag{3.1}$$

Then we must check the following inequality:

$$\Delta\sigma_{trial} \leq \sigma_{Y,n}. \tag{3.2}$$

If (3.2) is fulfilled, then the yield function (see Equation (2.2)) has a negative value ($F < 0$), meaning that the material is in the elastic region. In this case $\sigma_{n+1} := \Delta\sigma_{trial}$ and the solver moves on to the next increment.

If (3.2) is violated, then the material has entered the elastic-plastic region. In this case we calculate σ_{n+1} as:

$$\sigma_{n+1} = \sigma_n + E_{ep} \cdot (\varepsilon_{n+1} - \varepsilon_n). \tag{3.3}$$

To satisfy the condition that the yield function cannot be positive, we must update the yield stress:

$$\sigma_{Y,n+1} = |\sigma_{n+1}|, \tag{3.4}$$

then the solver can move on to the next increment.

Issues with the incremental method

This method is easy to implement, but it does have some issues. Most importantly, the method relies on sufficiently small increments. If this condition is not met, then it loses accuracy. The most extreme case is where the first increment already falls into the plastic region - in this case, the elastic region is entirely missing, yielding very inaccurate results.

Figure 3.11 illustrates another source of inaccuracy. If a trial step is just barely over the yield stress, then plastic behavior is assumed for the entire increment. This 'shifts' the entire $\sigma - \varepsilon$ curve down significantly as the $\sigma - \varepsilon$ curve is usually much more steep in the elastic

region. However, if the trial step is just below (on equal to) the yield stress, then the obtained solution will be much more accurate, as the next increment will be in the plastic region almost completely.

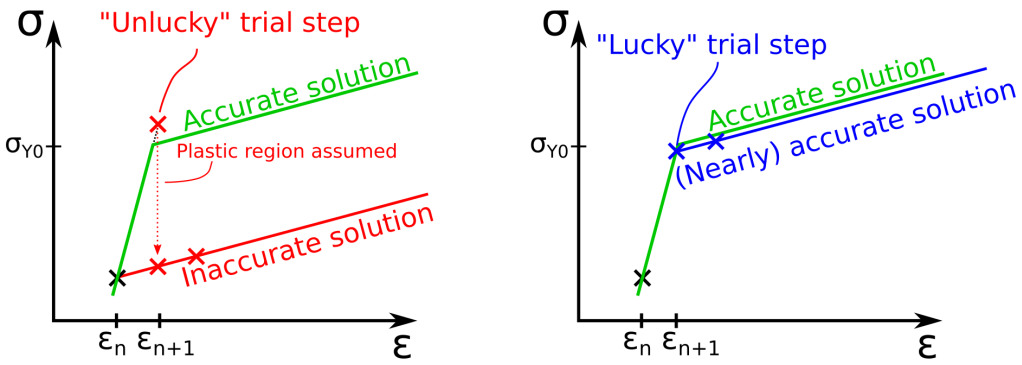

Figure 3.11. 'Unlucky' and 'lucky' trial steps

This introduces some uncertainty in the results as the demonstrated effect depends not only on the ε increments, but the material parameters as well. This uncertainty can be mitigated by using much smaller increments. However, this increases computational costs as well. As the method is incremental, any error from the accurate solution will be carried over to the next increments, and accuracy is never regained.

Figure 3.12 shows the effect of the number of increments used on the results. (The material parameters were $E = 1000$ [MPa], $\sigma_{Y0} = 10$ [MPa], $H = 100$ [MPa].) As we can see, using only 9 increments leads to very inaccurate results, as the elastic region is skipped. With 10 increments, we get fully accurate results, as the first increment ended precisely at the end of the elastic region. Using 11 increments creates a slight error, and using 19 produces quite inaccurate results again. As we can see, increasing the number of increments does not necessarily increase accuracy.

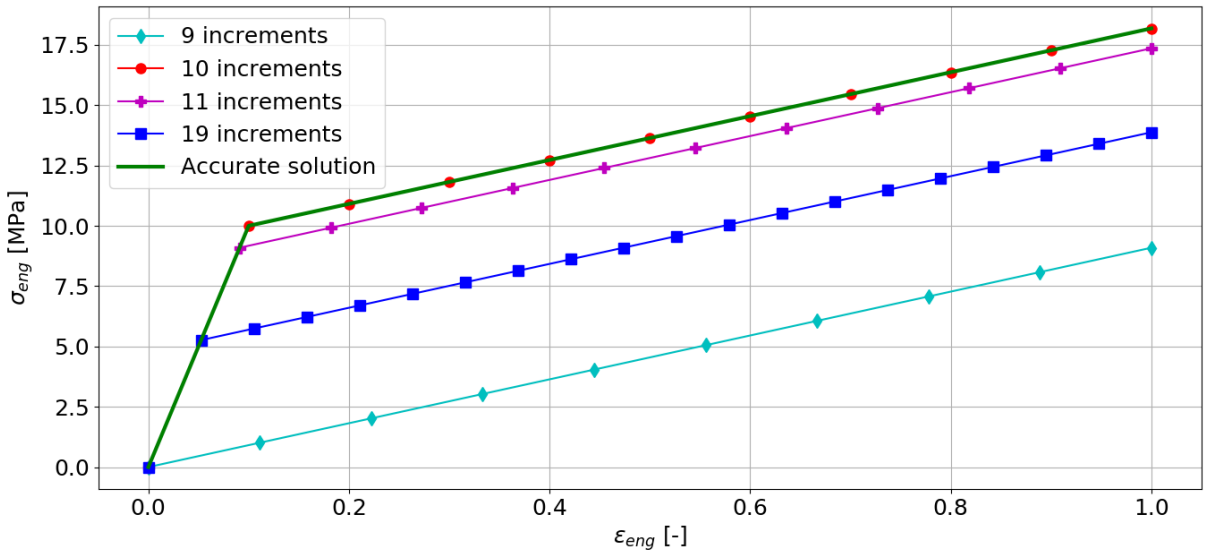

Figure 3.12. The effect of the number of increments

These uncertainties affect parameter fitting as well. The performance of some optimizer algorithms can get hindered by the non-continuous behavior of parameter changes.

There are ways to mitigate the described drawbacks (interval halving near the plastic limit, etc.). However, these would increase computational costs considerably and would never completely remove the inaccuracies.

3.4.1.2 'Elastic-Plastic Transition' method

The main idea of this method is to calculate the strain value analytically, where the transition happens from the elastic region to the elastic-plastic region.

This algorithm splits the load case based on whether the material is exposed to tension on compression. This is done based on the sign of $\dot{\varepsilon}$. Positive $\dot{\varepsilon}$ means tension, negative $\dot{\varepsilon}$ means compression. (If $\dot{\varepsilon} = 0$ the stress is constant as well.)

The program does the following calculations on each load segment. At the start of the ith load segment, the value of the strain and stress in known, denoted by σ_i and ε_i. The yield stress at the start of the segment is known as well, denoted by $\sigma_{Y,i}$. From this, we can calculate the strain change needed to enter the plastic region in this given load segment.

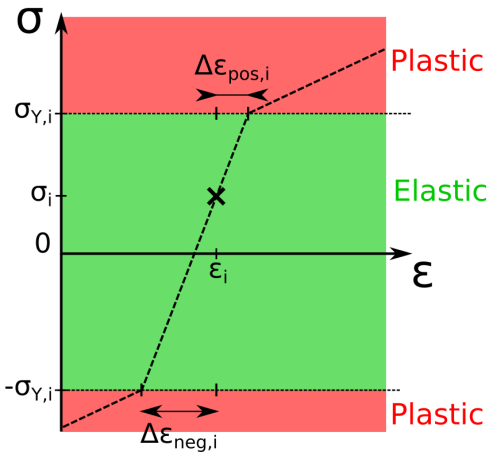

Figure 3.13. Limit strain changes

For tensile load:

$$\Delta\varepsilon_{pos,i} = \frac{\sigma_{Y,i} - \sigma_i}{E} \geq 0. \tag{3.5}$$

For compressing load:

$$\Delta\varepsilon_{neg,i} = -\frac{\sigma_{Y,i} + \sigma_i}{E} \leq 0. \tag{3.6}$$

The material is in the plastic region at calculation point ε_n if (in the ith load segment):

$$\varepsilon_n > \Delta\varepsilon_{pos,i} + \varepsilon_i, \tag{3.7}$$

or

$$\varepsilon_n < \Delta\varepsilon_{neg,i} + \varepsilon_i. \tag{3.8}$$

If point ε_n is in the elastic zone then:

$$\sigma_n = \sigma_i + E(\varepsilon_n - \varepsilon_i). \tag{3.9}$$

If point ε_n is in the plastic zone during tension then:

$$\sigma_n = \sigma_{Y,i} + E_{ep}(\varepsilon_n - (\Delta\varepsilon_{pos,i} + \varepsilon_i)). \tag{3.10}$$

If point ε_n is in the plastic zone during compression then:

$$\sigma_n = \sigma_{Y,i} + E_{ep}(\varepsilon_n - (\Delta\varepsilon_{neg,i} + \varepsilon_i)). \tag{3.11}$$

At the end of each load segment, the yield stress is updated if the maximum of the absolute value of the stress is higher than the current yield stress.

This method keeps its accuracy regardless of the density of the strain data. This is beneficial for the optimizer algorithms as well, as the results will be a continuous function of the parameters. Despite having a more complex computational algorithm than the iterative method, it also runs faster.

As the Elastic-Plastic Transition approach is superior to the iterative method both in accuracy and performance, my program always uses the Elastic-Plastic Transition method by default. The iterative approach is available for the sake of being able to compare the two methods, but it is quite impractical to use during 'real' calculations.

3.4.1.3 Performance comparison of the two methods

I have compared the performance of the two methods as a function of the resolution. I have used the previously introduced Test case 3. (From the material parameters, only E, σ_{Y0}, and H were used.) As the strain goes over 5% significantly, finite strain theory was used.

The resolution is given as the total number of $\varepsilon(t)$ points, given at uniform increments in time. Runtimes were measured as the average of at least 100 runs.

Table 3.5. Performance comparison: elastic-plastic branch

Resolution	Incremental runtime [ms]	Elastic-Plastic Transition runtime [ms]
7	0.041	0.360
61	0.238	0.418
601	2.111	0.494
6001	25.042	0.852
60001	187.949	4.984

As we can see from this data, the Elastic-Plastic Transition method is slower than the incremental method at *very* low resolutions. Measurements typically created a few hundred to a few thousand measurement points. In this range, the Elastic-Plastic Transition approach is much faster.

3.4.2 Viscoelastic branch

The differential equation has the following form (Equation (2.10)):

$$\dot{\sigma} = E\dot{\varepsilon} - EA\sigma^n t^m.$$

For small strain we can write:

$$\dot{\sigma}_{eng} = E\dot{\varepsilon}_{eng} - EA\sigma_{eng}^n t^m. \tag{3.12}$$

For finite strain:

$$\dot{\sigma}_{true} = E\dot{\varepsilon}_{true} - EA\sigma_{true}^n t^m. \tag{3.13}$$

The conversion formula (Equation (4)) between the true and engineering strain rates:

$$\dot{\varepsilon}_{true} = \frac{\dot{\varepsilon}_{eng}}{1 + \varepsilon_{eng}}.$$

Substituting (4) into (3.13):

$$\dot{\sigma}_{true} = E\frac{\dot{\varepsilon}_{eng}}{1 + \varepsilon_{eng}} - EA\sigma_{true}^{n} t^{m}. \tag{3.14}$$

With this formula, we can calculate the true stress directly from the engineering strain.

The equation was solved at the points where ε_{eng} the time was known. $\dot{\varepsilon}_{eng}$ at the ith point was calculated with a simple numerical formula:

$$\dot{\varepsilon}_{eng,i} = \frac{\varepsilon_{eng,i} - \varepsilon_{eng,i-1}}{t_i - t_{i-1}}. \tag{3.15}$$

As the initial stress is an input parameter (usually 0), it is not needed to solve the equation at the starting point. Thus the value of $\dot{\varepsilon}_{eng}$ at the initial point is not necessary to calculate.

These equations have a singularity at $t = 0$. For this reason a numerical 'trick' is used. If the value of t is below a certain threshold (e.g. 10^{-6}) a small number err is added to it. In my code $err = 10^{-50}$. Note, that usually $\sigma = 0$ at $t = 0$, which cancels this term.

The σ^n term in Equation (2.10) is correct for positive stress. It can be extended to handle negative stresses using the usual method (like ABAQUS does internally):

$$\mathrm{sign}(\sigma) \cdot |\sigma|^{n}. \tag{3.16}$$

There are several numerical methods available to solve the Equations (3.12) and (3.14) with the previously described numerical 'tricks'.

3.4.2.1 The Runge-Kutta method (RK45)

The principal idea of the Runge-Kutta method was proposed by C. Runge and developed later by W. Kutta and others around 1900. (Source: [23])

'Runge-Kutta' denotes an entire family of numerical methods. Their detailed description can be found in [2].

The most commonly used version is the fourth-order Runge-Kutta method - also known as 'the Runge-Kutta method', 'RK4', 'RK45', or 'RK4(5)'. It is a common numerical solver as it is relatively simple, robust, and many efficient implementations are freely available in many programming languages.

My program uses the implementation in SciPy available in the `scipy.integrate.ode` function, named `'dopri5'`. This implementation sets its time steps automatically, and the results are returned where the input data is given in time.

In the expected range of the material parameters, this method performs well, usually solving the equation within 50-500 ms. However, with some parameter combinations, the equation becomes stiff. In this case, performance is significantly impacted. (See subsection 3.4.2.3 for details.)

3.4.2.2 The LSODA method

LSODE (Livermore Solver for Ordinary Differential Equations) is a solver for both stiff and non-stiff problems described around 1990. A detailed description of the method can be found at: [3].

The SciPy implementation is available in the `scipy.integrate.ode` function, named `'lsoda'`. This method also sets its time steps automatically, and the results are returned where the input data is given in time.

In the expected range of the material parameters, this solver is considerably slower than the Runge-Kutta method. However, it handles stiff parameter combinations much better. It loses some performance in stiff cases, but not nearly as much as the RK45 method. (See subsection 3.4.2.3 for details.)

3.4.2.3 Performance comparison

The performance was compared on the previously presented Test case 1. From the material parameters, only E, A, n, and m were used.

As the strain goes over 5% significantly, finite strain theory was used. The resolution is given as the total number of $\varepsilon(t)$ points given at uniform increments in time. Runtimes were measured as the average of at least 20 runs.

Non-stiff case

Table 3.6. Performance comparison: viscoelastic branch (non-stiff case)

Resolution	RK45 runtime [ms]	LSODA runtime [ms]
5	8.973	25.422
41	15.712	56.904
401	36.646	216.991
4001	248.890	374.771
40001	2419.490	582.125

Stiff case

The equation gets stiff if the $E \cdot A$ product is high. To create the stiff test, Test case 1 was modified to have $E = 800000$ (this is 1000 times higher than the original value).

Table 3.7. Performance comparison: viscoelastic branch (stiff case)

Resolution	RK45 runtime [ms]	LSODA runtime [ms]
5	47.311*	58.221
41	437.093*	128.349
401	1866.942*	607.500
4001	2596.646	2896.363

*: in these cases, the solver gave warnings, that the maximum number of allowed steps was reached. It is likely that the solutions are not accurate here.

As we can see, the LSODA solver is indeed faster with stiff problems. More importantly, it keeps its accuracy as well.

However, in the non-stiff case, the RK45 method performed better. As the material parameters are expected to give a non-stiff equation, the RK45 method is the default setting in the program. The LSODA solver is available as well.

It is important to note that even in the non-stiff case, both methods are much slower than the calculations done in the elastic-plastic branch. The difference according to Tables 3.5 and 3.6 and additional tests is about 50-100x times. So to increase the performance of the full material model, the viscoelastic branch should be improved first.

3.4.3 Comparing to ABAQUS

The strains are the same in the two branches, and the stresses can be added together according to subsection 2.1.3.

It is important to investigate how the material behavior in my software compares to the material behavior calculated by ABAQUS. For this, I have used to previously described test cases.

The ABAQUS results were imported as measurement results (hence the label 'Measurement' on the plot legend). The temporal resolution is fixed at 10 ms. For this reason, the individual points are not marked on the plot (the markers would be too close and blend into each other). During the ABAQUS solution, the Creep, swelling, and viscoelastic strain error tolerances were set to 10^-5 based on the recommendations.

The 'Difference' is calculated by subtracting the ABAQUS results from the result obtained from my code. The maximum absolute difference is marked by a green dot.

It is important to note that my software got the ABAQUS results as a measurement result file with 10 ms time increments. For this reason, the stress response is obtained with 10 ms increments as well. However, the RK45 (or LSODA) method uses smaller time steps to obtain a better solution. These timesteps are chosen automatically by the solver.

The relative difference is calculated as:

$$\text{Relative difference} = \frac{\text{my calculation - ABAQUS result}}{\text{ABAQUS result}}. \tag{3.17}$$

3.4.3.1 Test case 1

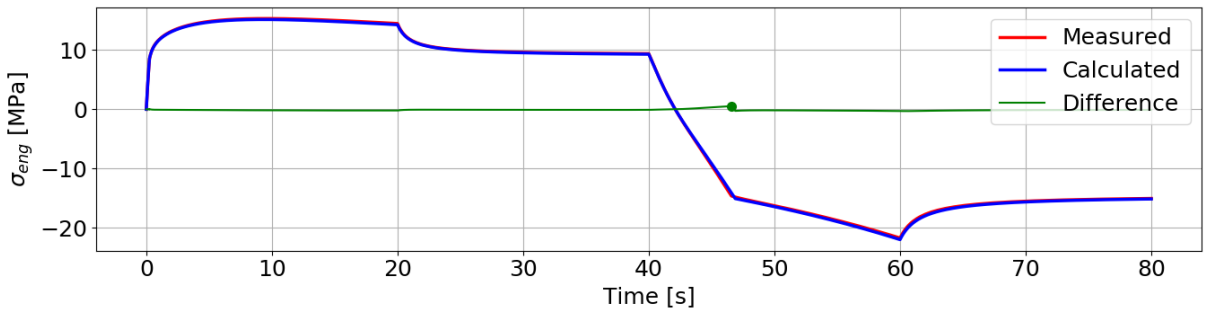

Figure 3.14. Comparison with ABAQUS - Test case 1

Maximum absolute difference: 0.537 [MPa]

Relative difference at the maximum absolute difference: -3.676%

3.4.3.2 Test case 1/b

This case primarily examines the elastic-plastic behavior.

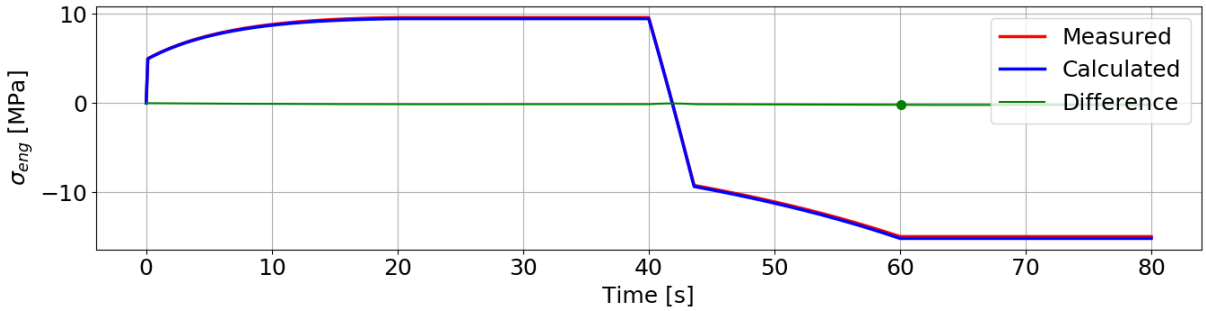

Figure 3.15. Comparison with ABAQUS - Test case 1/b

Maximum absolute difference: 0.208 [MPa]
Relative difference at the maximum absolute difference: 1.388%

3.4.3.3 Test case 1/c

This case primarily examines the viscoelastic behavior.

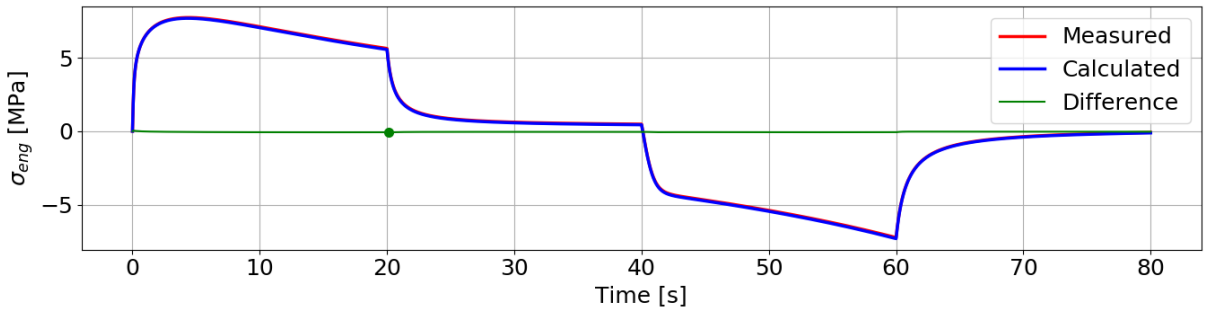

Figure 3.16. Comparison with ABAQUS - Test case 1/c

Maximum absolute difference: 0.076 [MPa]
Relative difference at the maximum absolute difference: -1.634%

3.4.3.4 Test case 2

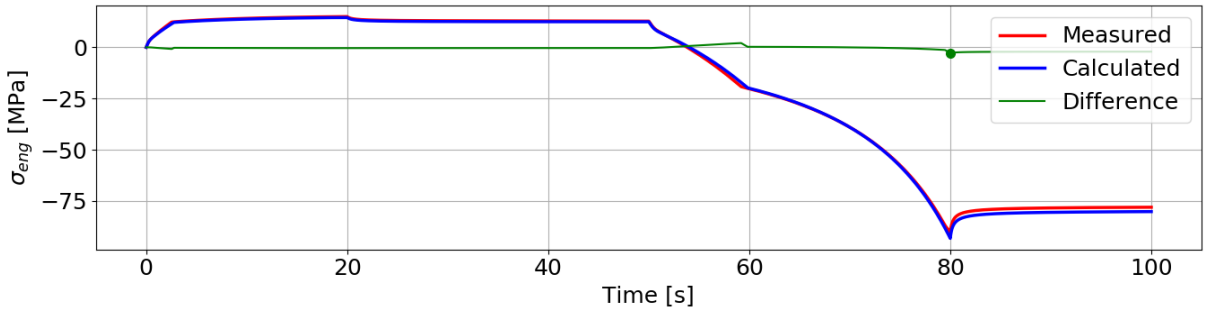

Figure 3.17. Comparison with ABAQUS - Test case 2

Maximum absolute difference: 3.003 [MPa]
Relative difference at the maximum absolute difference: 3.327%

3.4.3.5 Test case 3

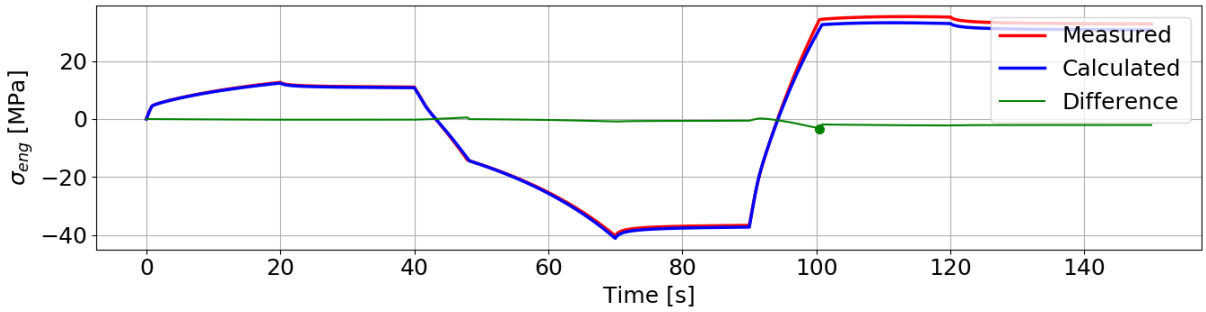

Figure 3.18. Comparison with ABAQUS - Test case 3

Maximum absolute difference: 3.266 [MPa]
Relative difference at the maximum absolute difference: -9.551%

3.4.3.6 Summary

The relative differences at the maximal absolute differences are well below 5% in most cases. In Test case 3, it almost reaches 10%. However, this test case is considerably more complex than the measurements.

Overall the results obtained from my code follow the ABAQUS results quite well. Thus the material parameters obtained at the end of the fitting process will be usable in ABAQUS.

3.5 Optimizer algorithms and settings

The interface of the optimizer settings can be seen on Figure 3.4. In this section I will go over these settings and their impact on the optimization process.

Terminology

An **optimizer method or algorithm** is a computer algorithm designed to find the local minimums (or maximums) of a multi-variable scalar valued function by changing the value of the variables.

The **quality or object(ive) function** is the function what is minimized (or maximized). In my case this function expresses the difference between the measured and simulated data as a scalar. The closer are the measured and simulated behaviors, the smaller is the value of this function. For this reason my software is minimizing the quality function.

A **function evaluation** means one calculation, where the material behavior is simulated for a specific load case and parameters, then the difference from the measured behavior is calculated by the quality function.

An **(optimizer) iteration** means one internal cycle of the optimizer algorithm. One iteration may involve several (sometimes 50-100) function evaluations.

Bounded optimization means that the variables can have lower and/or upper limits. Not all algorithms are designed for bounded optimization.

3.5.1 Investigated algorithms

The algorithms listed below can be selected from the interface. My program uses the implementation available in SciPy, in the `scipy.optimize.minimize` function.

3.5.1.1 Powell

The algorithm was published by M. J. D. Powell in the referenced article: [7].

A modified version of this method is available in SciPy. It does not calculate or estimate the derivative of functions. For this reason, it is well suited for handling complicated but continuous functions.

This was one of the more reliable and effective algorithms. It was not too sensitive for the starting point. A detailed comparison to other methods will be shown later in this thesis.

3.5.1.2 BFGS, L-BFGS-B

The BFGS (Broyden-Fletcher-Goldfarb-Shanno) method was described in 1970 by Charles George Broyden, Roger Fletcher, Donald Goldfarb, and David Shanno. (Source: [4])

In L-BFGS-B, the 'L-' denotes the limited memory implementation, which is achieved by simplifying the stored inverse of the Hessian matrix. '-B' denotes that this method is able to do bounded optimization.

As this is a hill-climbing algorithm, it looks for local optima. For this reason, this method is more sensitive to the starting point than the Powell method. A detailed comparison to other methods will be shown later in this thesis.

3.5.1.3 SLSQP

Its name comes from Sequential Least Squares Programming. SciPy uses a version implemented by Dieter Kraft in the background. Like L-BFGS-B, this method can also handle bounded optimization. (Source: [8])

For my problem, this method gave solutions faster than other methods. However, it was quite sensitive to the starting point and other settings. A detailed comparison to other methods will be shown later in this thesis.

3.5.1.4 Neldel-Mead

This algorithm is a simplex method, meaning that it uses geometric concepts for optimization. (Source: [9])

This algorithm is robust, as no derivatives are taken during the optimization. It usually needs more iterations than other methods but does fewer function evaluations per iteration. However, in my cases, it's performance was considerably worse than the previously mentioned methods. For this reason, I decided not to do detailed tests with it.

3.5.1.5 CG

This method uses a nonlinear conjugate gradient algorithm described by Polak and Ribiere, which only uses the first derivatives. (Source: [5])

In my cases, CG's performance was considerably worse than the previously mentioned methods. In many cases, it failed to find the optimum. For this reason, I decided not to do detailed tests with it.

3.5.2 Quality functions

As described previously, the quality function quantifies the difference between the measured and simulated data. The value of the quality function should get lower, as the measured and simulated data gets closer; it should reach 0 when there is no difference between the two. There is an infinite number of functions, which could be used as quality functions.

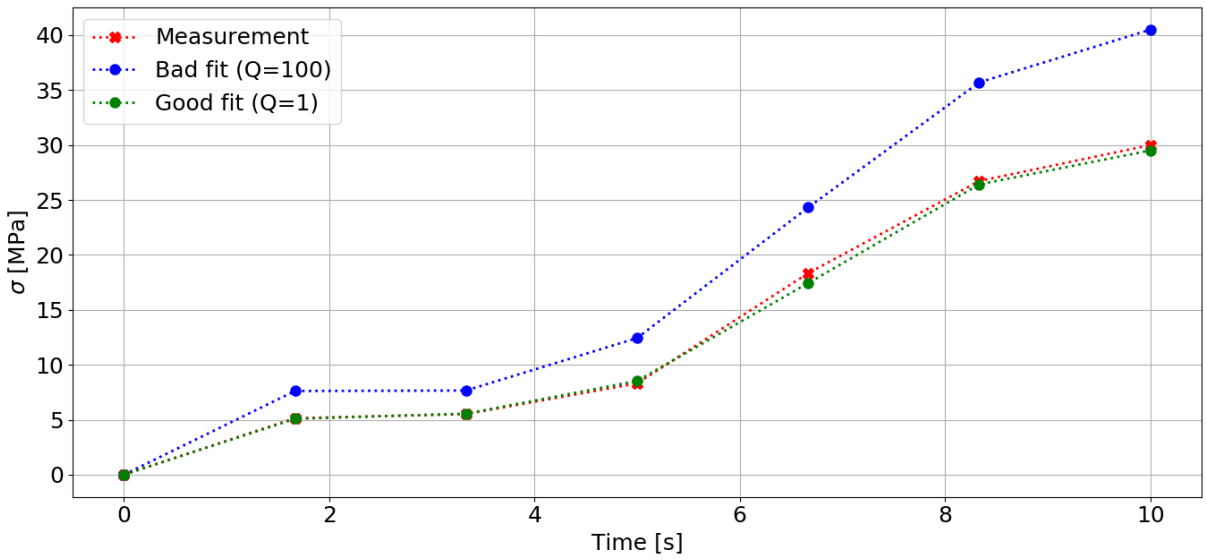

Figure 3.19. Good and bad fit (illustration)

The value of the quality function is denoted by Q, the measured and simulated data is compared at the ε_{eng} values indexed by $i = 1 \dots N$. The measured engineering stress at the ith point is denoted by σ_i^{meas}. The simulated engineering stress at the ith point is denoted by σ_i^{sim}.

Sum of absolute differences:

$$Q_{abs} = \sum_{i=1}^{N} |\sigma_i^{meas} - \sigma_i^{sim}|. \tag{3.18}$$

Sum of absolute differences weighted by $\Delta\varepsilon$:

$$Q_{abs,\Delta\varepsilon} = \sum_{i=2}^{N} |\sigma_i^{meas} - \sigma_i^{sim}|(\varepsilon_i - \varepsilon_{i-1}). \tag{3.19}$$

Sum of absolute differences, normalized:

$$Q_{abs,norm} = \frac{Q_{abs}}{N}. \tag{3.20}$$

Sum of squared differences:

$$Q_{sq} = \sum_{i=1}^{N} (\sigma_i^{meas} - \sigma_i^{sim})^2. \tag{3.21}$$

Sum of squared differences weighted by $\Delta\varepsilon$:

$$Q_{sq,\Delta\varepsilon} = \sum_{i=2}^{N} (\sigma_i^{meas} - \sigma_i^{sim})^2(\varepsilon_i - \varepsilon_{i-1}). \tag{3.22}$$

Sum of squared differences, normalized:

$$Q_{sq,norm} = \sqrt{\frac{Q_{sq}}{N}}. \tag{3.23}$$

Using the formulas weighted by $\Delta\varepsilon$ does not have a significant impact of Q if the measurement points are in equal ε distance from each other. (In this case, the weighting just applies a constant multiplier.) However, if the distribution of the measurement points is far from uniform, this option can help to obtain an overall better fit, as it will prevent a few points close together 'pulling' the curve away from the far away point.

Normalizing the Q value is beneficial in order to get a result independent from N. This way, comparing the quality of two fits on different sets of data becomes possible.

3.5.3 Internal settings of the optimizer algorithms

Each method available in the `scipy.optimize.minimize` function has different settings. While tweaking the internal settings 'properly' might lead to improvements in performance, changing them 'incorrectly' might make them not find the optimum as well. One of the design goals with this software was that it should be easy to use. For these reasons, the GUI only includes the 3 most important options.

- 'maxiter': Maximum number of allowed optimizer iterations.
- 'maxfev': Maximum number of allowed function evaluations. After going over this limit, the optimizer stops at the end of the iteration. This setting is ignored by BFGS, SLSQP, and CG.
- 'eps': Step size used for the estimation of the Jacobian matrix. This setting only affects the BFGS, L-BFGS-B, CG, and SLSQP methods, as the other methods do not take derivatives. (The other methods ignore this setting.)

3.5.4 Penalty for crossing the parameter limits

From the described methods, only L-BFGS-B and SLSQP can handle bounded optimization problems. The rest of the methods may go outside the parameter ranges defined by the user. In order to 'discourage' the algorithms from going out of bounds, a penalty function is used. This makes the value of the quality function significantly worse if the boundaries are violated.

3.5.4.1 The 'cutoff' method

One possible way to handle boundary crossing is not to evaluate the quality function outside the designated domain. In this case, a very large number can be returned as the value of Q (e.g., 10^{100}).

The advantage of this method is that no computational power is used to evaluate the quality function at this invalid position. Invalid parameter values might mean physically impossible behavior as well (e.g., if the optimizer tries $E < 0$). The numerical methods still try to solve the equations with these parameter values. However, as the behavior might be unstable, the solution may take 10-100 times longer than with valid parameters.

However, this method has significant disadvantages, as well. Cutting off at the boundary means that the quality function is not continuous at the parameter boundaries. All of the previously mentioned algorithms were made to handle continuous problems. They run on non-continuous functions as well, but their behavior might be unpredictable. (For example, a gradient-based method might estimate very high derivative near the boundary.)

Another drawback is that by cutting off the optimizer receives false information, and all real information is lost. There is no difference between slight and major boundary violations. The quality function forms a 'horizontal' surface in the parameter space, with a tiny 'hole' in it as the valid domain. As the value of the gradient on the 'horizontal' surface is 0, the optimizer cannot determine which direction to go towards.

For the reasons mentioned above, I have decided against the cutoff method.

3.5.4.2 Based on the number of boundary crossings

My code uses a penalty factor described by the following equation:

$$p = 2^c. \tag{3.24}$$

Here p denotes the penalty factor and c the number of boundaries crossed. The penalty factor is applied as:

$$Q_{penalized} = pQ. \tag{3.25}$$

This method does not differentiate between slight and significant boundary crossings either. However, if the boundaries are set correctly, the value of Q expected to grow, the further away are the parameters from the valid domain. This penalty function amplifies this effect.

Q is again non-continuous at the boundary, but this effect is not nearly as severe as with the cutoff method so that the algorithms can handle it better.

Using this method allows the optimizer to get some information from outside the valid zone. The gradient is not 0, as should point towards the valid domain.

This option is turned on by default in the program, but it can be turned off by the user.

3.5.5 Random starting points

In the case of new materials or models, it might be challenging to give a good initial guess for the optimizer. In this case, it is common practice to give the optimizer some random starting points as well. These are evaluated before the optimizer is started. The one with the best quality function value is picked to be the actual starting point for the optimizer.

The functionality described above is available in my program is well. The user can set the number of random points generated, up to 1000. However, this functionality should be used with caution, as each point needs to be evaluated. In the case of a large amount of points, this might take a significant amount of time.

For each random starting point, the material parameters are generated between their respective bounds with uniform distribution. If the upper or lower boundary is not given (or is $\pm\infty$), then $\pm 10^{200}$ is used instead. As this value is unrealistically high for any parameter using any reasonable unit, it is strongly recommended to set the upper and lower limit for each parameter.

In general, the usage of this setting is not recommended if the initial guess is expected to be good, e.g., because of a previous parameter fit under similar conditions. A more detailed explanation is given in subsection 3.5.7.3.

3.5.6 Parameter handling mode

As we can see from the test cases, the expected value of the parameters is not in the same order of magnitude. For the material the measurements were made on $E \approx 1000[MPa]$, and $A \approx 0.001[-]$. This means the 'largest' parameter is about 6 orders of magnitude larger (in absolute value) than the 'smallest'.

This difference can affect the performance of some optimizer algorithms negatively. This can be avoided by applying a linear transformation on the parameters. This way, the optimizer 'sees' the transformed values only, and these are in the same order of magnitude if the correct transformation is applied. Naturally inverse transformation is applied before passing the parameters to the material model. For this reason, my software has three options for parameter handling.

'Values' mode

With this setting, no transformation is applied. The optimizer handles the material parameters directly.

'Multipliers' mode

The initial material parameter values given by the user (x_0) will be interpreted as multipliers for the parameters, as:

$$x_{material} = x_0 \cdot x_{optimizer}, \tag{3.26}$$

where $x_{material}$ denotes the material parameter passed to the model and $x_{optimizer}$ the parameter 'seen' by the optimizer.

In this case, the optimizer uses 1 as the initial value for each parameter. This way, the x_0 values given by the user will be equal to the actual material parameters of the initial guess.

The parameter boundaries need to be transformed accordingly as well. As they mean the limits for the values seen by the optimizer:

$$B_{optimizer} = \frac{B_{GUI}}{x_0}, \tag{3.27}$$

where $B_{optimizer}$ is the transformed boundary passed to the optimizer, and B_{GUI} is the boundary given by the user on the GUI.

In general, this method improved performance with all optimizer algorithms it was tried with. For this, the user needs to give correct estimations for the order of magnitude of the parameters, so the optimizer works with similar values for all parameters.

'Normalized' mode

This method transforms all parameters to be between 0 and 1 based on the parameter limits. The user has to define the parameter boundaries as finite values for this to work. The initial values passed to the optimizer:

$$x_{optimizer} = \frac{x_0 - B_l}{B_u - B_l}, \tag{3.28}$$

where B_l and B_u are the lower and upper boundaries given on the GUI, respectively.

In this mode, the lower limit for the optimizer is 0, and the upper limit is 1 for each parameter.

The inverse transformation to obtain the material parameters for stress calculation:

$$x_{material} = x_{optimizer}(B_u - B_l) + B_l. \tag{3.29}$$

This method does help performance as well, and it guarantees that the parameters are in the same order of magnitude.

3.5.7 Test results for different algorithms and settings

This section will contain various tests performed on the optimizer. It is impossible to test and include every possible combination of settings. However, the general trends will be demonstrated.

3.5.7.1 Algorithm performance test

Performance was tested using Test case 4 (see Table 3.4). The quality function was set to the normalized sum of squared differences. Maxiter and maxfev were both set to 500. Parameter handling mode was 'Normalized'. Other settings were left on the default.

As we can see by the number of asterisks needed, comparing these algorithms is no trivial task. However, the primary trend is visible: the best performing algorithms are Powell, L-BFGS-B, and SLSQP. I will only test these algorithms for the rest of this thesis.

3.5.7.2 Effect of parameter handling mode

Performance was tested using Test case 4 (see Table 3.4). The quality function was set to the normalized sum of squared differences. Maxiter was set to 250, maxfev to 2000. Other settings were left on the default. The time required for the optimizer, and the quality of the fit was investigated.

The optimizers did not always find the accurate values of the parameters. This is visible on the value of $Q_{sq,norm}$. $Q_{sq,norm} < 0.01$ indicates that the actual optimum was certainly found.

The Powell algorithm got stuck in a local optimum, which gave a good fit, but it was not exactly at the accurate values.

Table 3.8. Performance test of different algorithms

Algorithm	Time needed [s]
Powell	27.5*
BFGS	240.1**
L-BFGS-B	68.2*
SLSQP	69.4**
Neldel-Mead	99.8***
CG	690.1**

*: reached the maxfev limit before fully converged, but good fit
**: ignores maxfev
***: acceptable fit ($Q = 0.33$) but not exactly at the optimum, may have converged if given more time

Table 3.9. Effect of the parameter handling mode - performance and quality

Time [s]	Values	Multipliers	Normalized
Powell	152.8*	105.5	118.6
L-BFGS-B	260.0*	199.4*	184.5
SLSQP	N/A	217.2	84.4

$Q_{sq,norm}$	Values	Multipliers	Normalized
Powell	0.100*	0.100	0.101
L-BFGS-B	0.0037*	0.0217*	$2.27 \cdot 10^{-5}$
SLSQP	N/A	0.229	$4.33 \cdot 10^{-5}$

*: maxfev reached
N/A: the optimizer stopped after a few iterations, without any good results

L-BFGS-B performed quite well; it found the optimum in 'Values' mode but did not have enough function evaluations to get the precise values. In 'Normalized' mode, the optimum was found with great precision. In 'Multiplies' mode, it got close, and with more function evaluations, it most likely would have found the optimum.

SLSQP is a sensitive algorithm, as mentioned previously as well. With the 'Values' mode, it failed to give any reasonable result. In 'Multipliers' mode, it failed to find the optimum. However, in 'Normalized' mode, it found the optimum with great precision and over twice as fast as L-BFGS-B.

Overall, both the 'Multipliers' and 'Normalized' mode seems to perform better than the 'Values' mode. In this test, the 'Normalized' mode was the best, but there are some cases where 'Multipliers' mode performs better.

About the algorithms, we can say that SLSQP is the fastest by a significant margin if the settings are right. However, it is quite sensitive and can get unstable with the wrong settings. Powell gives acceptable but not great results reliably. L-BFGS-B tends to find the optimum with more precision than Powell, but it is slower than SLSQP.

3.5.7.3 Effect of the random starting points

The short and long term effect of using additional random starting points was examined using Test case 4. The SLSQP method was used with 'Normalized' parameter handling.

When investigating the short term effect, 'maxiter' was set to 50. A control run was made with no random points. Then 10 runs were made with 100 additional starting points each. The value of the quality function was examined after the 50 iterations.

Table 3.10. The short term effect of random starting points

	$Q_{sq,norm}$		
Control run	0.01163	Test run #5	2.10231 (!)
Test run #1	0.02412	Test run #6	0.04849
Test run #2	0.02317	Test run #7	0.07499
Test run #3	0.005229	Test run #8	0.07363
Test run #4	0.07279	Test run #9	0.005188
		Test run #10	0.06835

In all 10 test runs, one of the 100 random points was better than the provided initial guess. However, after 50 iterations, the value of $Q_{sq,norm}$ was only better than the control run two times. Seven times it was slightly worse, and once it was much worse. The runtime of the control run was $60s$. The test runs varied between $70 - 120s$.

The long term effect was investigated the same way, but 'maxiter' was set to 200. The optimizer always stopped before reaching 200 iterations.

Table 3.11. The long term effect of random starting points

	$Q_{sq,norm}$		
Control run	$3.31107 \cdot 10^{-5}$	Test run #5	$2.79169 \cdot 10^{-5}$
Test run #1	$2.20085 \cdot 10^{-5}$	Test run #6	0.04680 (!)
Test run #2	$2.70342 \cdot 10^{-5}$	Test run #7	$5.32332 \cdot 10^{-5}$
Test run #3	$3.11567 \cdot 10^{-5}$	Test run #8	$4.05218 \cdot 10^{-5}$
Test run #4	$2.98621 \cdot 10^{-5}$	Test run #9	$3.97268 \cdot 10^{-5}$
		Test run #10	$3.48100 \cdot 10^{-5}$

It is important to note that the parameters form the $Q_{sq,norm} \approx 10^{-5}$ magnitude are practically identical, the difference is $\ll 1\%$. 9 out of 10 times, the test runs found the optimum, just like the control run did. However, one of the test runs showed noticeably worse results than the rest. The runtime of the control run was $100s$. The test runs varied between $110 - 180s$.

Overall the usage of the random starting points has increased runtimes. Results were rarely improved; they got worse on average in the short-term tests. Long term results were not affected by much. However, in each test, there was a case where the results became much worse. Similar trends were observed using only 10 random points.

For the reasons mentioned above, it is not recommended to use the random starting points if the user can give a good initial guess. Runtimes will be increased, and results tend to be worse when using random points with a good enough initial guess.

The nature of optimization algorithms can explain this observation. As this quality function is complex, it probably has many local minimums. It is unlikely that the initial guess

by the user it close to one of them. However, the probability of 'hitting' a local minimum increases significantly, if we use several random points as well. However, this local optimum is not likely to be the global optimum, but it will be better than the guess given by the user. Thus the optimizer is started from the vicinity of a local optimum point. In this case, the algorithm might never realize (or only later realize) that there exists a better minimum place in the parameter space.

The probability of hitting a local minimum goes down, as the number of random points is decreased. However, this overall questions the usefulness of this technique for a function like the quality function of this problem.

3.6 Structure of the code

Before the current interface and structure were designed, the program had a previous version. This version only had some very basic functionality. The user could only select the material model and the optimizer algorithm. To change the rest of the settings, the source code needed to be modified. However, this is quite impractical, as several settings need to be changed regularly. Another drawback was that adding new items to the menus was not automated.

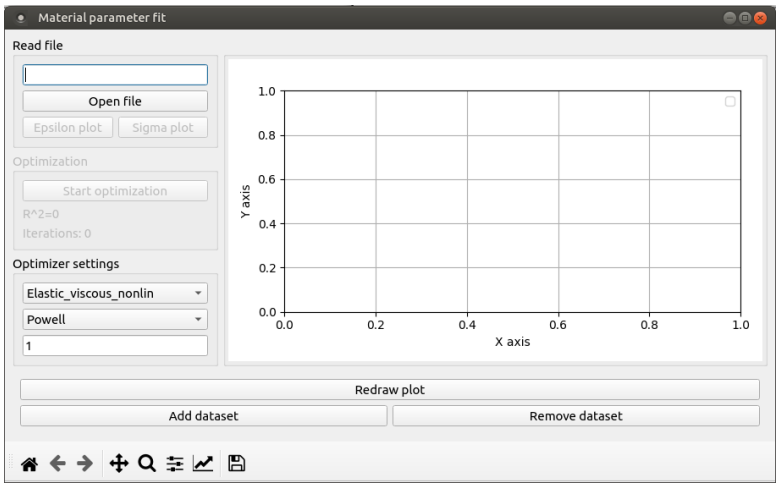

Figure 3.20. The old interface

Later the program was rewritten almost completely. Only a few small parts of this code were kept. However, it was a useful prototype and helped me design the current version of the software.

3.6.1 Files building up the code

In Python, the content of another file (functions, classes, variables, etc.) can be imported like any module with the `import file` command. The imported code will run just as quickly as code written in the same file. Importing the file does take some time at the start of the code, but this is barely noticeable. (According to my measurements, it is 100-150 ms for each import.)

This allows the code to be split into multiple files without any significant performance impact. This helps to keeps the code organized. Modifications are easier to make, as less time is spent looking for the right piece of code to be modified. For these reasons I have split my code into the following files:

gui.py

This file contains everything related to the graphical user interface. This is the largest of all files. It contains approximately 1200 lines of code. As the parts of this code are heavily interconnected, I did not split it into smaller files.

This code builds the GUI layout and defines the function of each element. These include: reading the measurement file and checking its format, passing the settings to the optimizer algorithm, plotting the results, etc.

This is the main file, running this file in Python will launch the program with all functionality.

mater_class.py

This file contains the definition of the 'material model' and 'material parameter' classes in general. It does not contain any specific material models. It only provides a general framework to define them. This is a short file with less than 100 lines of code.

The material parameters have the following important properties: `value`, `minimum` and `maximum`. The latter two store the upper and lower boundary of the value.

The material models can have material parameters defined, but their most important property is the `calcstress` function. When defined for a specific material, this contains all the numerical methods for calculating the stress for a load case using the material parameters. The material model class also contains a list of all available numerical methods.

mater_defs.py

This file contains specific material definitions and the definition of their numerical methods. In my case, the isotropic hardening behavior and the nonlinear viscoelastic behavior are defined here. The TLPV model uses the sum of the stresses of the two branches.

When building the GUI, `gui.py` will look for material models in this file and add them to the interface with their respective numerical methods. This means that when a new material model is defined (or an existing model gets a new numerical method), it will automatically appear on and work with the interface.

This file contains about 500-600 lines of code.

optimizerclass.py

The code in this file provides a framework to pass the settings to the optimizer algorithm more easily from the GUI. It handles the different parameter interpretations. The random starting points are generated and evaluated here as well. The 'callback' function communicates the current material parameters to the user interface.

This file contains about 200 lines of code.

quality_fns.py

This file contains the quality functions, which calculate the difference between the measured and simulated results. To make the code more flexible linear or cubic interpolation can be used on both the measured and simulated data to evaluate the quality function at different positions.

Similarly to the material models, building the GUI, `gui.py` will look for quality functions in this file and add them to the interface automatically.

This file contains about 100 lines of code.

3.6.2 Multi-threading

The behavior of the program during the optimization process id described in subsection 3.2.4. In order to achieve this, the program has to use two treads during optimization. (Threads are the structures used by the operating systems to manage different tasks at the same time.)

The optimization process is started and ran by a single function call (`scipy.optimize.minimize`) with the appropriate arguments. If this function was using the same thread as the GUI, the interface would become unresponsive until the plot of the result, the current value of the parameters, and changing the plotting mode.

For this reason, the GUI runs on the main thread. When the optimization is started, a new thread is made to do the calculations. The main thread receives updates at the end of each iteration using the callback function. PyQt5 does the creation and handling of the calculation thread. Once the thread is started, it cannot be stopped by my code due to the way how PyQt5 handles the thread.

The callback function only allows passing the current values used by the optimizer. To plot the current material behavior on the GUI, these are converted to material parameters based on the parameter handling mode. Then the material behavior is calculated on the main thread as well. After this, plotting the behavior is possible.

Chapter 4

Results on measurement data

4.1 Measurement data

The measurements were made on a microcellular polyethylene-terephthalate (MC-PET) material. This foam material is currently under development. The material consists of a foamed core layer and an external solid skin layer. However, we can assume that it can be modeled as a continuum, so the behavior is homogeneous.

In order to investigate both the elastic-plastic and viscoelastic behavior, the material was put under uniaxial load on a Zwick-Roell Testing System. The uniaxial load cycles were performed on multiple temperature levels, so the temperature dependence of the material parameters can be investigated. (Source: [6])

One load cycle consisted of a strain-controlled uploading, relaxation, then force-controlled unloading and relaxation. With this experiment, the elastic, plastic, and viscous behavior can be examined as well. An example from the measurement data (at 21°C) can be seen on Figure 4.1 and 4.2.

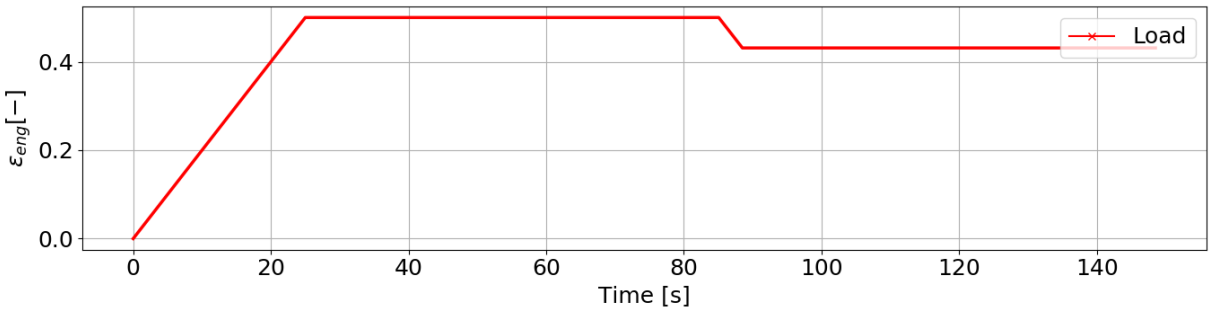

Figure 4.1. Strain load on the specimen

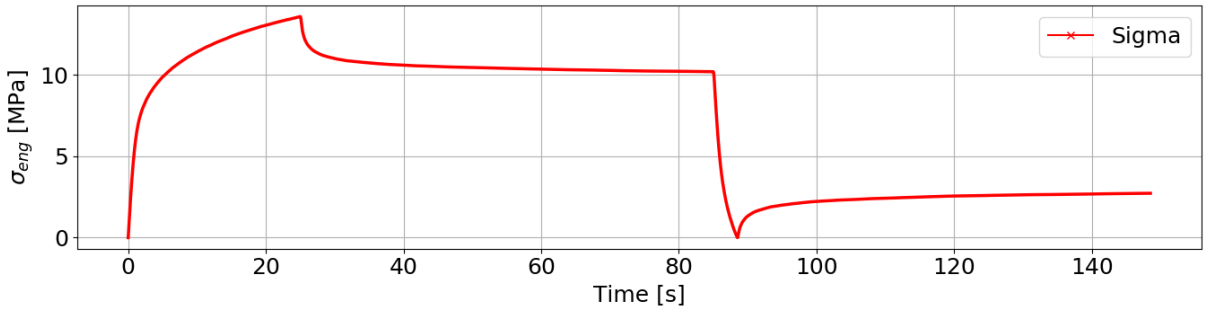

Figure 4.2. Stress response of the specimen

Measurements were made on the following temperatures: $21°C$, $60°C$, $75°C$, $83°C$, $90°C$, $97°C$, $106°C$, $120°C$, $160°C$, $210°C$.

4.2 Fitted parameters

I did the material fitting process for this set of measurement data as well. The efficiency of the software allowed the use of the full measurement files with all measurement points.

I have used the SLSQP algorithm because it was the most efficient in previous test cases. The quality function was the normalized sum of the squared differences, as this makes it possible to compare the quality of different fits. 'maxiter' was set to 200, 'maxfev' is ignored by SLSQP. The parameter handling mode was set to 'Multipliers', because in this particular case, it gave practically the same results as 'Normalized', but faster. Finite strain theory was used. All other settings were left on the default.

As the material parameters depend on the temperature, I have split the cases at $95°C$. The optimizer got different bounds and initial guess below and above this limit. These can be seen in Tables 4.1 and 4.2.

Table 4.1. Settings for $T < 95°C$

Parameter	Initial guess	Lower limit	Upper limit
E[MPa]	600	300	1200
f[-]	0.5	0	1
σ_{Y0}[MPa]	5	0	25
H[MPa]	40	1	100
A[-]	0.001	0	0.1
n[-]	2	1	5
m[-]	-0.5	-1	0

Table 4.2. Settings for $T > 95°C$

Parameter	Initial guess	Lower limit	Upper limit
E[MPa]	250	100	500
f[-]	0.5	0	1
σ_{Y0}[MPa]	3	0	10
H[MPa]	40	1	100
A[-]	0.005	0	1
n[-]	2	1	5
m[-]	-0.5	-1	0

During the optimization I recorded the final value of $Q_{sq,norm}$, the required runtime (denoted by t_{run}) and the number of function evaluations (denoted by N_{eval}) as well. The results can be seen in the tables below. (The value of A is multiplied by 1000 for better readability.) Appendix A contains the $\sigma_{eng} - t$ and $\sigma_{eng} - \varepsilon_{eng}$ plots for all fits.

Table 4.3. Results for $T < 95°C$

Temperature	21 [°C]	60 [°C]	75 [°C]	83 [°C]	90 [°C]
$Q_{sq,norm}$ [-]	0.1811	0.1567	0.1163	0.1209	0.1323
t_{run} [s]	63	97	200	121	90
N_{eval}	239	370	766	466	340
E [MPa]	807.302	669.398	501.825	468.175	416.924
f [-]	0.733	0.753	0.743	0.772	0.773
σ_{Y0} [MPa]	5.231	3.268	0.0	0.0	0.0
H [MPa]	28.726	26.953	32.343	30.904	30.035
$1000 * A$[-]	0.763	0.455	0.106	0.242	0.641
n [-]	3.357	3.654	3.519	3.469	3.235
m [-]	-0.782	-0.723	-0.23	-0.331	-0.389

Table 4.4. Results for $T > 95$°C

Temperature	97 [°C]	106 [°C]	120 [°C]*	160 [°C]*,**	210 [°C]
$Q_{sq,norm}$ [-]	0.1531	0.1606	0.1478	0.1078	0.09345
t_{run} [s]	88	107	124	99	150
N_{eval}	315	387	447	589	560
E [MPa]	375.541	364.491	300.143	149.996	131.422
f [-]	0.798	0.828	0.793	0.891	0.824
σ_{Y0} [MPa]	0.0	0.0	0.0	0.0	0.0
H [MPa]	30.01	29.806	24.51	99.991	12.658
$1000 * A$ [-]	1.524	2.721	12.747	75.842	562.011
n [-]	3.231	3.494	3.112	2.401	5.0
m [-]	-0.477	-0.587	-0.641	-1.0	-0.944

*: parameter handling was changed to 'Normalized' as 'Multipliers' failed to yield acceptable results
**: slightly changed boundaries

4.3 Details of a fitting

As the software saves detailed logs during the optimization process, the evolution of the material parameters can be plotted. I have chosen the fitting for the $T = 75$ [°C] case for these plots.

The parameters are plotted similarly to how the 'Normalized' parameter handling works. The plot uses linear transformation, where the largest recorded value of the parameter gets transformed to 1 and the lowest recorded value to 0.

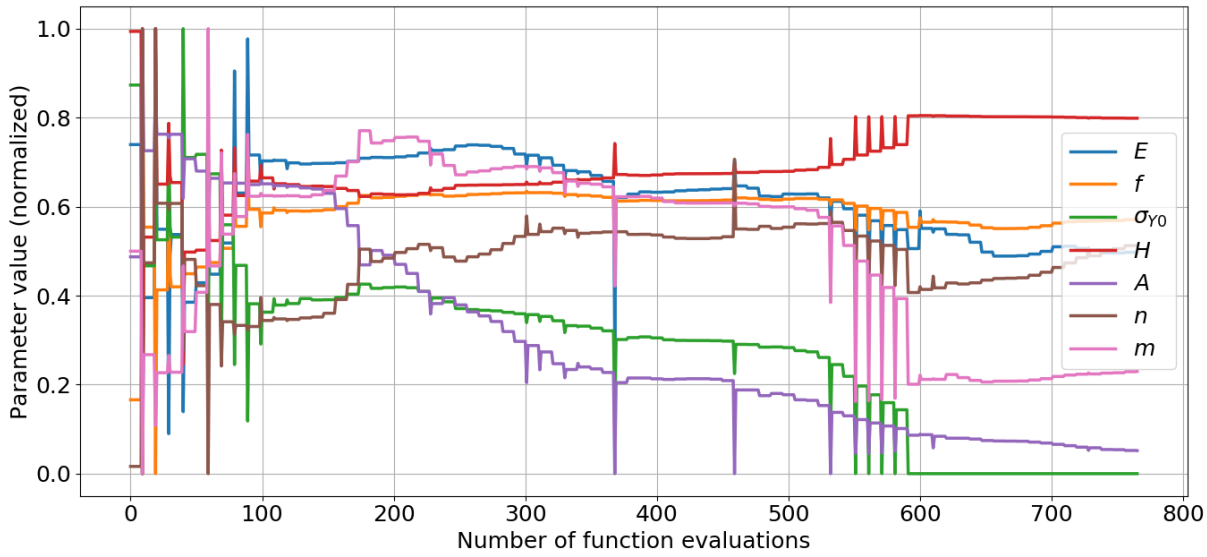

Figure 4.3. Evoultion of the material parameters

The evolution of the quality function was done on a logarithmic scale, as there are several orders of magnitude between the best and worst values.

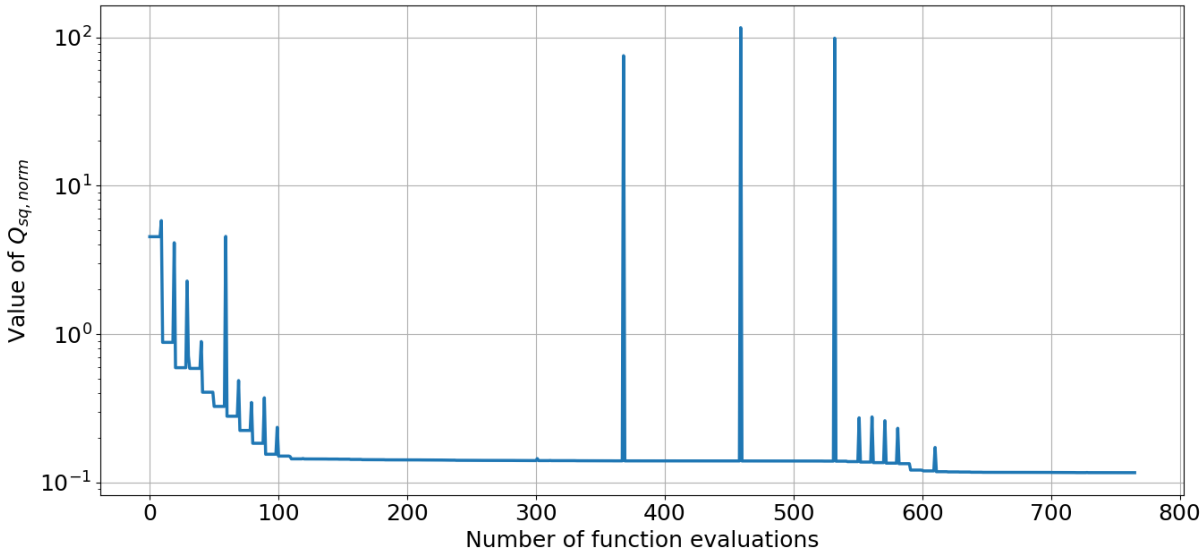

Figure 4.4. Evoultion of the quality function

It can be seen on the plot, that the parameter values change rapidly as the algorithm 'experiments' in about the first 100 function evaluations. After this initial period, there are a lot less abrupt changes. The value of the quality function improves the most in this period.

Between the approximately 100th and 550th function evaluation, the parameters change slowly compared to the initial period discounting a few 'spikes'. The improvements in the quality function are very hard to notice. New noticeable improvements are made around the 550-600th evaluation. In this period, the value of $Q_{sq,nomr}$ improves by 10-15% within 50 evaluations. We can see the value of the parameters oscillating before the improvement. After this, the value of σ_{Y0} and H does not change significantly. Other parameters (especially n) are still getting tweaked slightly.

If we zoom in on the last 100-150 evaluations in Figure 4.4, we can see some minor improvements being made consistently. This zoomed-in view can be seen in Figure 4.5. It can be observed that the quality function has several steps, each lasting for 9 function evaluations. The steps are not perfectly flat, and there are only minor differences between the values. As SLSQP is a derivative-based approach, these function evaluations are most likely related to the numerical estimation of the derivatives, evaluated at a very small distance from each other.

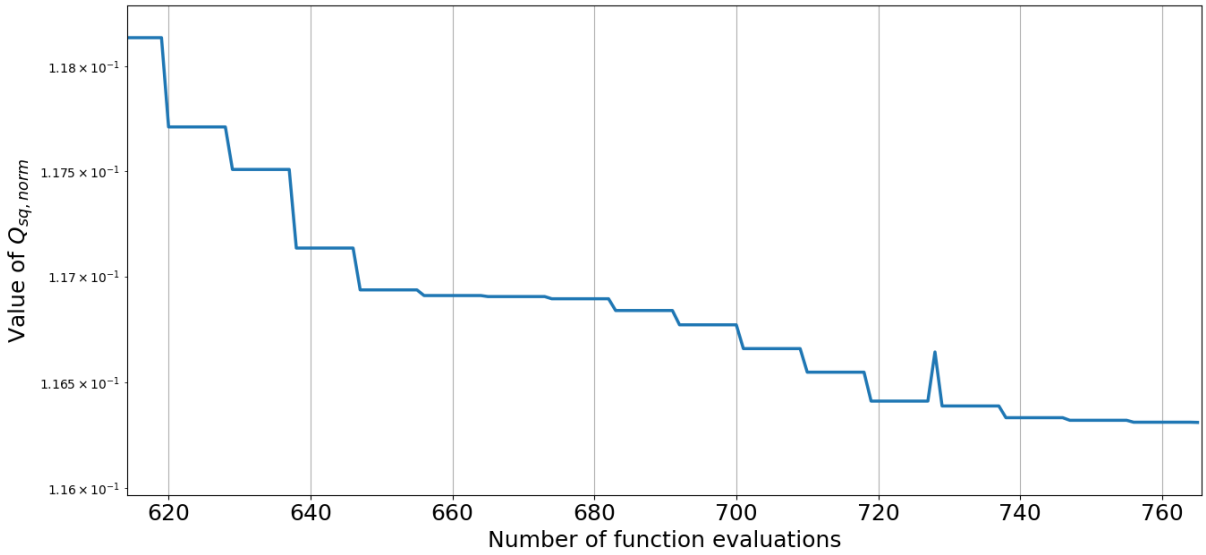

Figure 4.5. Evoultion of the quality function - zoomed-in view

Overall we can say that the optimizer makes most of the improvements in the initial phase of the optimization. However, even after a few 100 function evaluations without any significant changes, noticeable improvements can happen.

4.4 Temperature dependence

Based on the results shown in Tables 4.3 and 4.4 we can investigate the dependence of the material parameters on temperature. This is important in industrial applications. MC-PET is a thermoplastic material. For this reason manufacturing products made of it often involves heating of the material. In order to be able to predict the behavior of the material under these conditions, the material parameters are needed as a function of temperature.

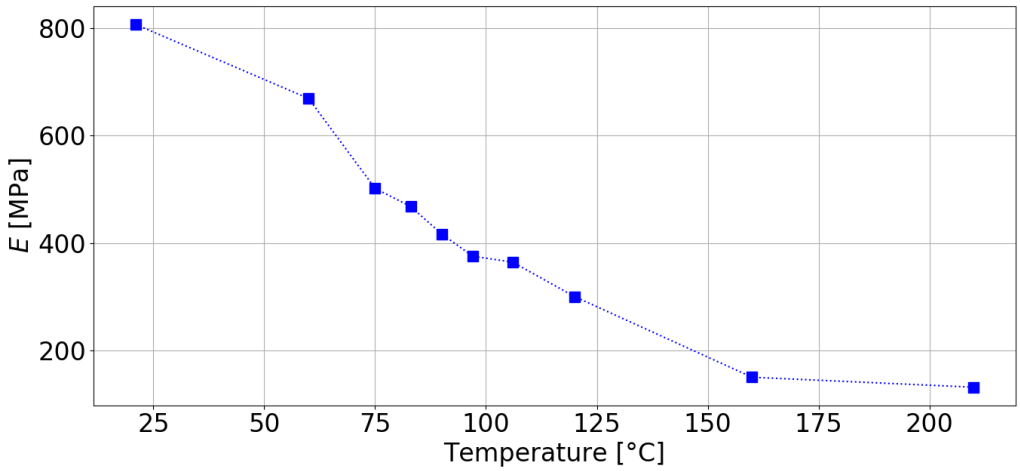

Figure 4.6. Temperature dependence - E [MPa]

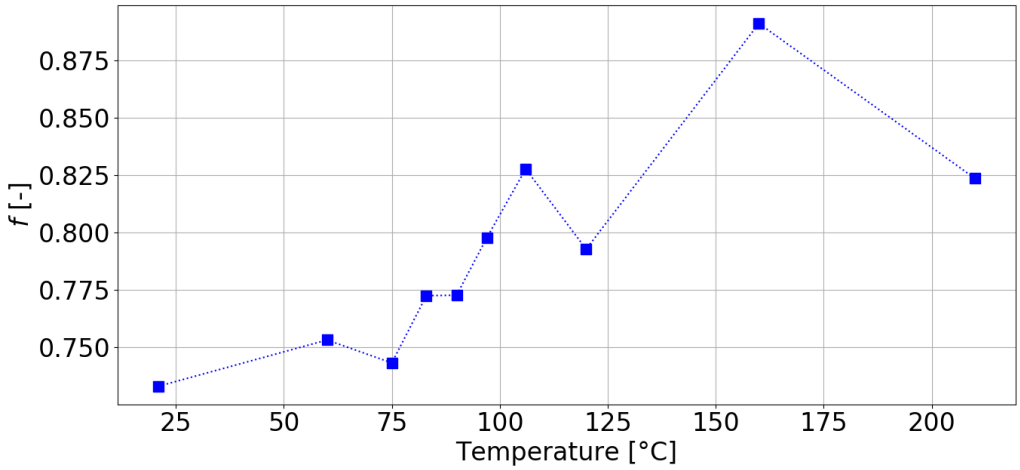

Figure 4.7. Temperature dependence - f [-]

The value of E decreases as temperature increases. This is expected of a thermoplastic material. Up to 160 [°C], the trend seems to be close to linear; however, there are not enough measurement points to draw strong conclusions. The value of f (the proportion of the viscoelastic contribution to the total elastic modulus) increases as temperature goes up, but the trend is not as 'smooth' as with E.

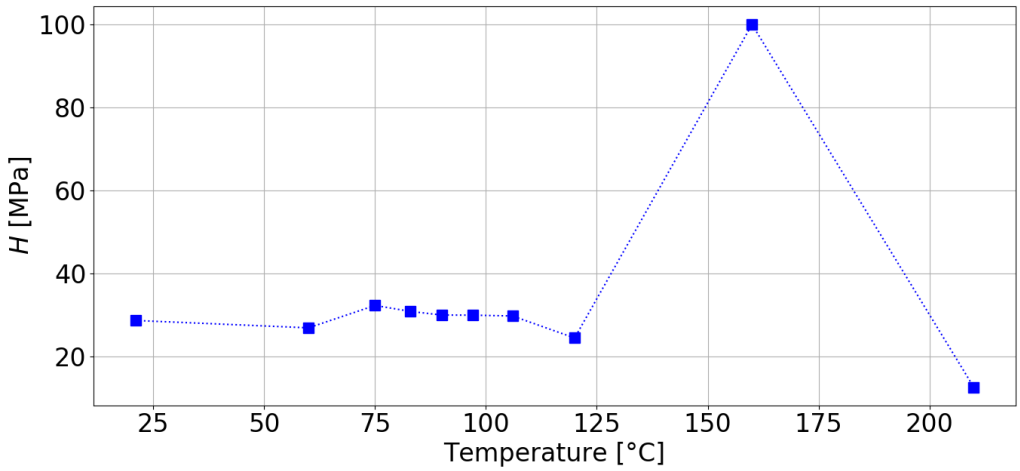

Figure 4.8. Temperature dependence - H [MPa]

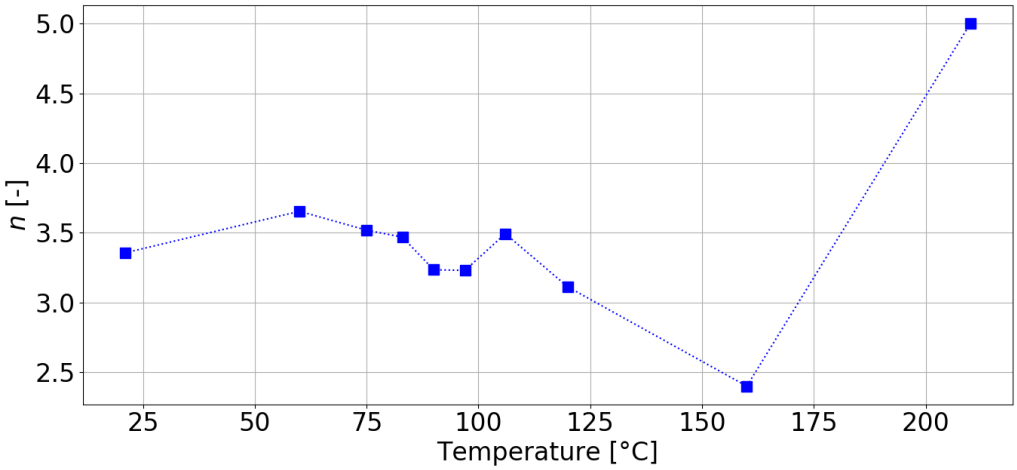

Figure 4.9. Temperature dependence - n [-]

The value of both n and H does not change significantly below 150 [°C]. Above this limit, one of the points breaks this trend quite noticeably. This can be due to measurement error, or the optimizer did not find the right parameters.

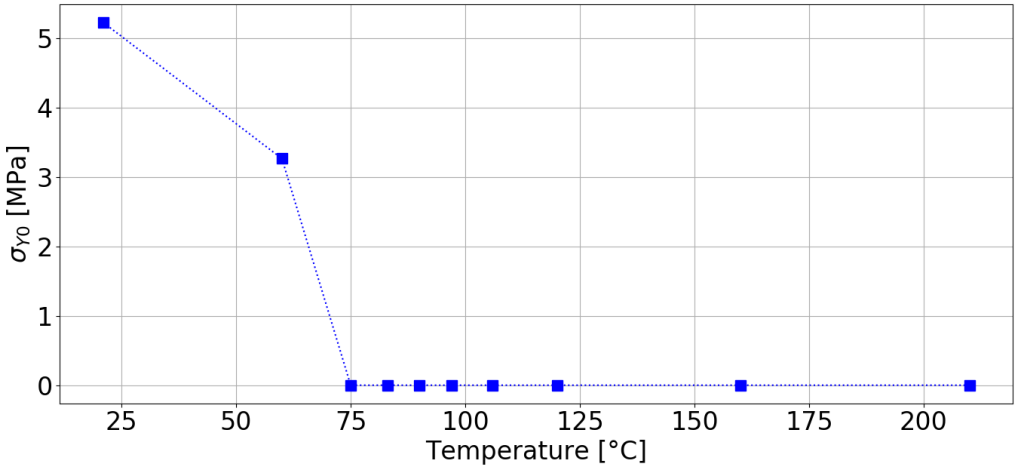

Figure 4.10. Temperature dependence - σ_{Y0} [MPa]

At 75 [°C] and above $\sigma_{Y0}[MPa]$ is zero. This means that the material enters the plastic region as soon as loaded and gets permanently deformed. The trends of A and m also change at the 75 [°C] measurement point. (A is plotted on logarithmic scale.) The value of A decreases up to 75 [°C], then increases consistently. m shows similar behavior: the trend changes at 75 [°C].

As 3 parameters have noteable behavior changes between 60 [°C] and 75 [°C] it is possible that the material structure indeed changes at this temperature.

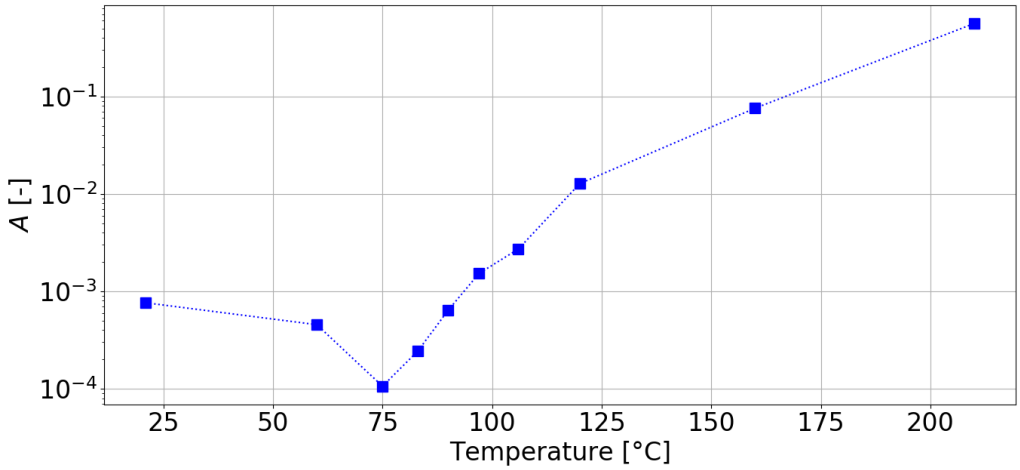

Figure 4.11. Temperature dependence - A [-]

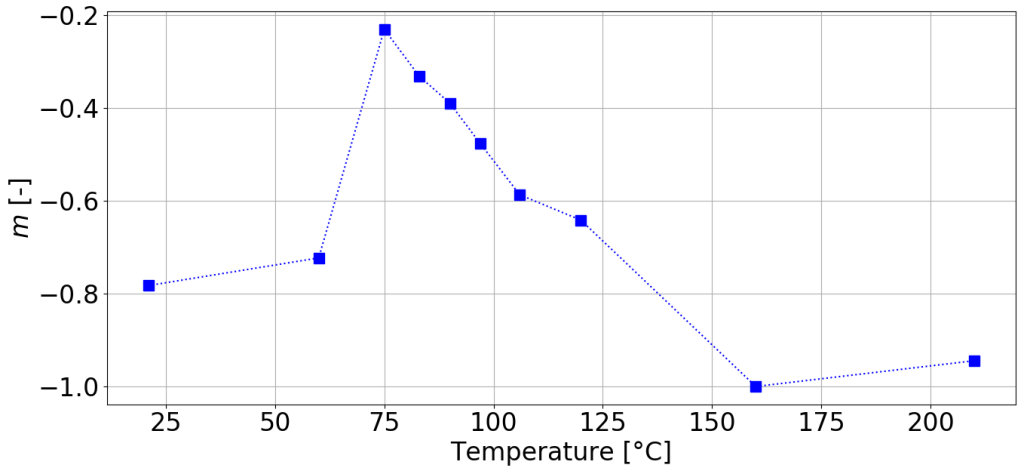

Figure 4.12. Temperature dependence - m [-]

4.5 Comparison to another method

The described general parameter fitting method can be done using other software as well. To compare my software to another solution using commercial products, I used the 21 [°C] measurement, excluding the relaxation part at the end of the load case. To have a fair comparison, both optimizers were started from the same point. The lower and upper boundaries only apply to my software.

Table 4.5. Starting point for the optimizers

Parameter	Initial guess	Lower limit	Upper limit
E [MPa]	800	400	1200
f [-]	0.5	0	1
σ_{Y0} [MPa]	5	0	25
H [MPa]	50	1	100
A [-]	0.002	0	0.1
n [-]	2	1	5
m [-]	-0.5	-1	0

4.5.1 Results with Dassault iSight

Dassault iSight is a commercial software made for 'process automation and design exploration'. (Source: [24])

It can be connected with ABAQUS and used as an external optimizer for the material fitting process. Dassault Systems owns both iSight and ABAQUS, so we can assume that the two products can communicate efficiently.

For the chosen set of measurement data, this method gave the following results:

Table 4.6. Parameters fitted with iSight

E [MPa]	f [-]	σ_{Y0} [MPa]	H [MPa]	A [-]	n [-]	m [-]
420.048	0.529	4.706	26.207	0.0018250	1.809	-0.480

This method is slowed down considerably if many measured data points are given for the load case and for the fitting. For this reason, the original set of approximately 1500 points was reduced to approximately 600. Even with this simplification, iSight needed about 6-7 hours to obtain these results on stronger hardware than my laptop.

The fitted behavior can be seen on Figures 4.13 and 4.14. The two curves are quite close to each other, indicating good fitted parameters.

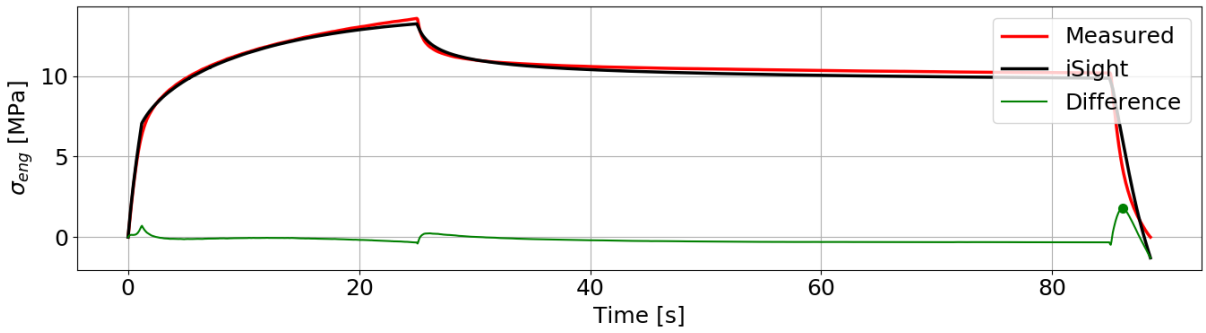

Figure 4.13. The fitted behavior by iSight - $\sigma_{eng}(t)$

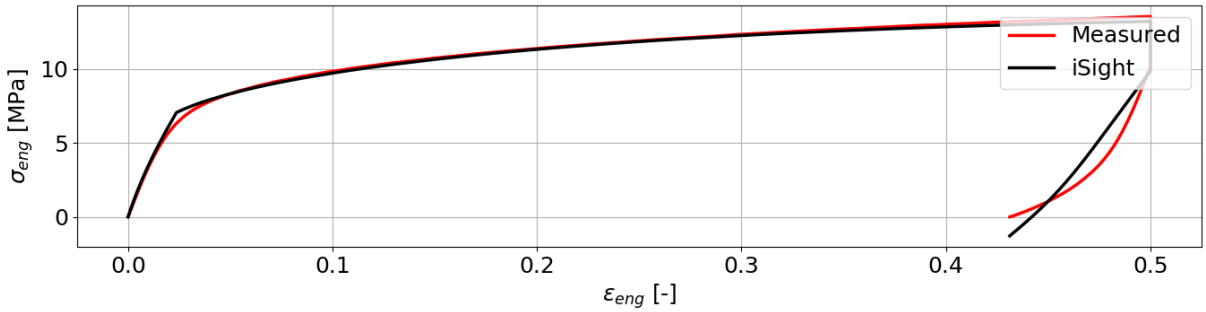

Figure 4.14. The fitted behavior by iSight - $\sigma_{eng}(\varepsilon_{eng})$

4.5.2 Results with the developed software

The starting point and parameter limits are described in Table 4.5. The SLSQP algorithm was used with 'Multipliers' parameter mode. All the available 1500 measurement points were included during the calculations, unlike iSight.

Table 4.7. Parameters fitted with my software

E [MPa]	f [-]	σ_{Y0} [MPa]	H [MPa]	A [-]	n [-]	m [-]
707.532	0.804	5.022	31.239	0.0004664	3.559	−0.913

To get these parameter values, my program only needed 68 seconds (there is some variance between runs) and 352 function evaluations. At the end of the process, the value of the quality function was $Q_{sq,norm} = 0.0772$. The material behavior is plotted on Figures 4.15 and 4.16. The two lines are very close to each other, and they overlap for most of the process.

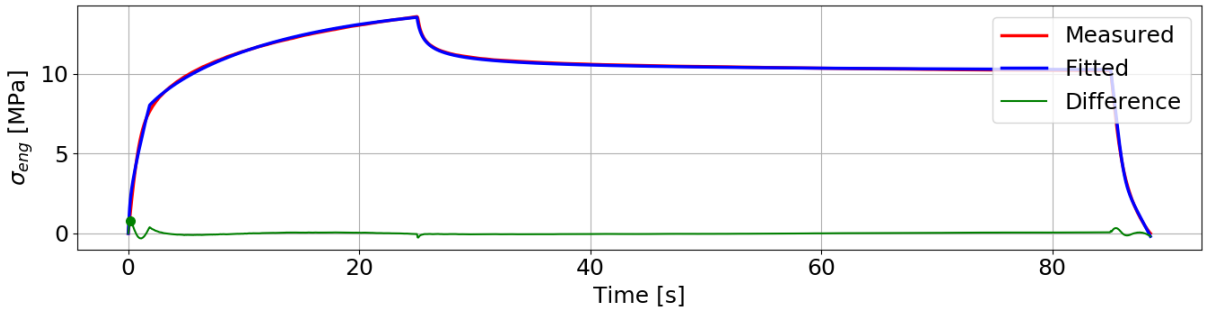

Figure 4.15. The fitted behavior by the developed program - $\sigma_{eng}(t)$

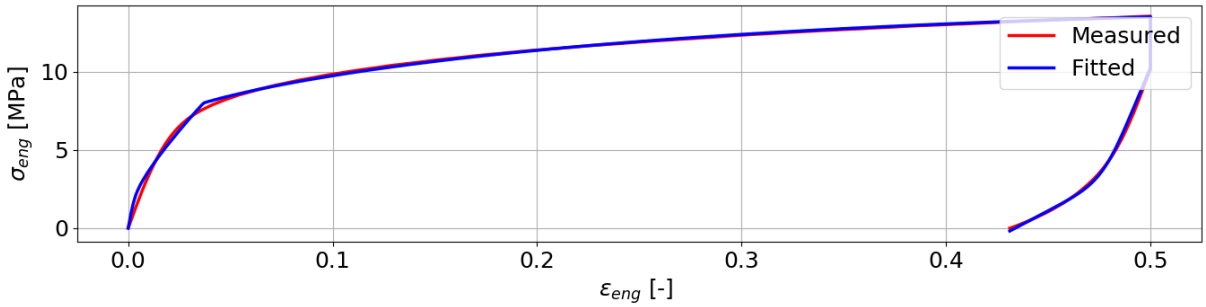

Figure 4.16. The fitted behavior by the developed program - $\sigma_{eng}(\varepsilon_{eng})$

4.5.3 Comparing the results

If we compare the fitted parameters, we can see noticeable differences. They are most likely both at local minimums of the quality function. However, based on the plots, we can say that the minimum place found my software is either the global optimum or a 'better' local optimum than the one found by iSight. Unfortunately, complicated functions (like the one between the material parameters and the value of the quality function) tend to have many local minimums. This makes the optimization task quite challenging. Choosing one parameter incorrectly can be *almost* (but not completely) compensated by tweaking the other parameters.

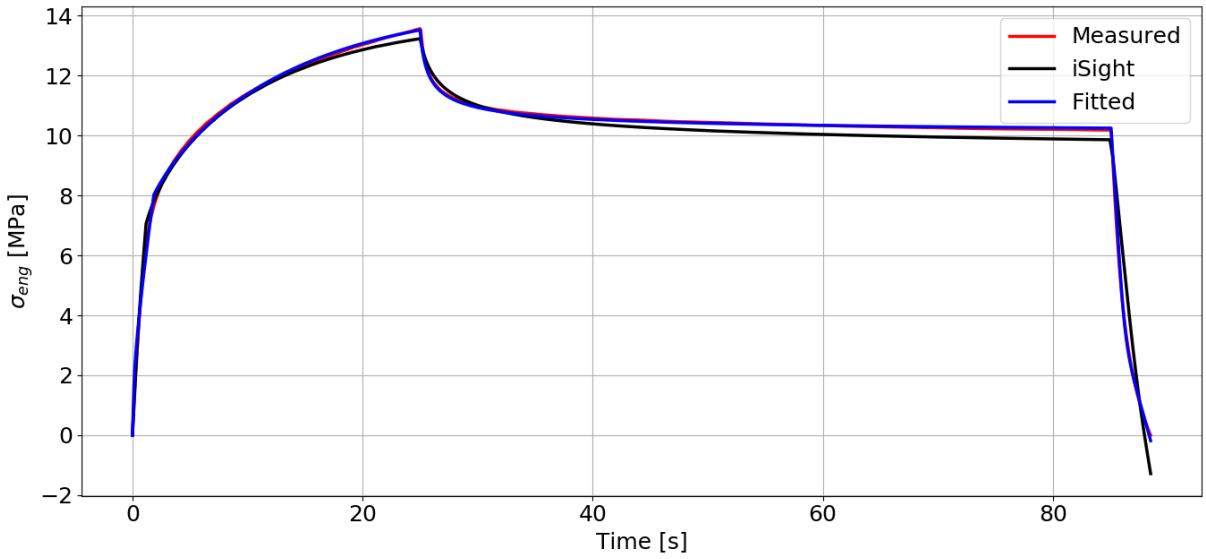

Figure 4.17. Fitted behavior comparison - $\sigma_{eng}(t)$

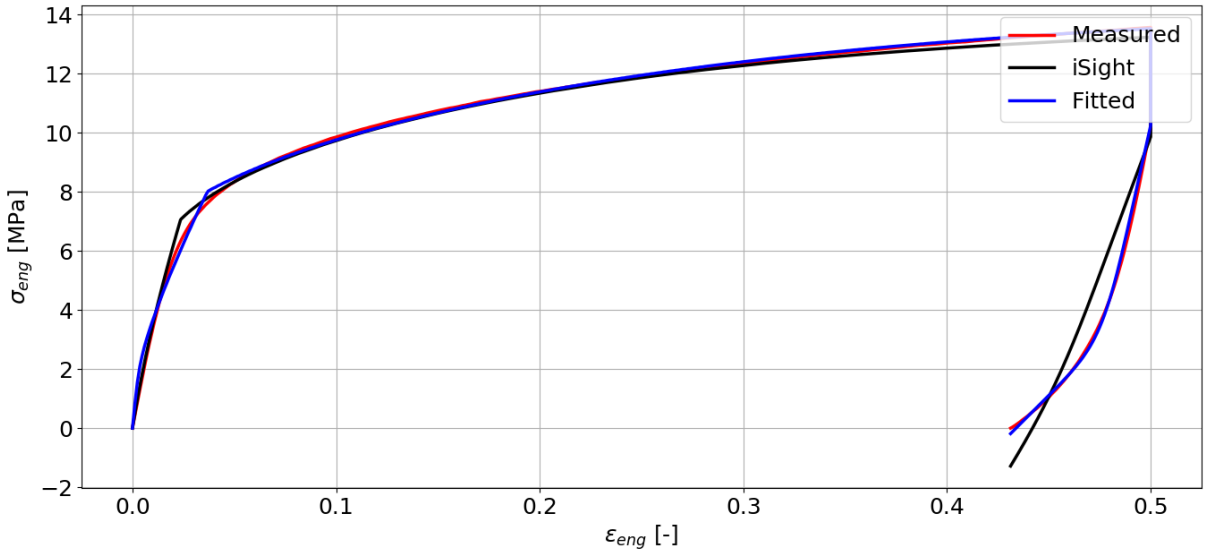

Figure 4.18. Fitted behavior comparison - $\sigma_{eng}(\varepsilon_{eng})$

In Figure 4.17 and 4.18 we can see both results as well as the measured data. It is visible that the parameters found by my code follow the measurement data more closely. This is especially visible on the $\sigma_{eng}(\varepsilon_{eng})$ curve, near the end of the load case. The parameters given by iSight fail to capture the curvature of the curve at the end.

My software gave these results about 300-400 times faster than iSight. This is a huge benefit of the developed code, especially when working with multiple measurements. This also proves that the developed code is efficient - this was the primary goal of this software.

Chapter 5

Conclusion

5.1 Evaluating the set goals

At the start of my thesis, I have defined 3 goals for the developed software. Near the end of my thesis, these goals should be investigated. Overall we can say that all the goals were achieved.

Efficiency

The developed software's performance was compared to a method using commercial products in section 4.5. Overall my software gave noticeably better results than iSight. It is capable of doing the parameter calibration process in a few minutes, compared to iSight's several hours. On the particular investigated test case, the developed software gave the parameters about 350 times faster, despite iSight getting a reduced set of data to speed up the process. My code also ran on weaker hardware than iSight.

Ease of use, graphical interface

The created graphical interface is described in detail in section 3.2. The entire parameter fitting process can be done using this interface, and the layout tries to follow the workflow as closely as possible.

Modular structure

The structure of the code is described in section 3.6. The existing interface will automatically include any new material models, numerical methods os quality functions written in the correct file in the correct format. The material models and quality functions are usable without the GUI as well.

If new models are added, the parameter fitting process can be done on them as well.

5.2 Ways to improve the method

The code is proven to be well usable in its current state. However, it can be improved further.

The simulation of the material model can be further improved, especially in the viscoelastic branch. Using another RK45 implementation from SciPy shows a potential 2-5x speedup. Currently, this method does not work well with the optimizer algorithms for unknown reasons. The code does run without error, but the obtained results are completely

wrong. Further testing is needed to identify the cause. As the evaluation of the viscoelastic branch takes much more time (50-100x) than the elastic-plastic branch, this could be a significant improvement.

In the current version of the code, there is a stability issue under some uncommon circumstances. This needs to be fixed before further improvements are made. The difference between the simulated and measured data should be plotted on the $\sigma(\varepsilon)$ curves as well.

The evolution of the parameters and the quality function cannot be seen from the GUI. The log files include the necessary data to analyze them. However, an external script is needed to do this efficiently. Such tools should be available on the interface, preferably as a popup window, which should show the state of the optimization in real-time.

As the investigated function seems to have many local minima, other optimization tools should be considered. Genetic algorithms are suited to handle such functions, but they are generally slower than other optimization tools. It should be investigated whether their use would be beneficial for this problem or not. SciPy does have tools for genetic algorithm based optimization, but a custom algorithm can be developed as well.

5.3 Summary

The software able to do the parameter calibration of the TLPV model was successfully developed. Being a specialized tool, it gives better results than the commercial software it was compared to. These results were obtained about 2 orders of magnitude faster, despite the other method having fewer data points to work with.

The program can be expanded with other material models as well. As is contains several algorithms and settings, it will most likely provide an effective solution for these new models as well.

Overall the developed software seems to be an effective tool for material parameter calibration.

Chapter 6

References

6.1 Book, thesis

[1] J. Kichenin: *Comportement Thermomécanique du Polyéthylène - application aux structures gazieres*, Ecole Polytechnique, Paris, France, 1992.

[2] Ascher, Uri M.; Petzold, Linda R: *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*, Society for Industrial and Applied Mathematics, University City Science Center Philadelphia, PA, United States 1998.

[3] K. Radhakrishnan; A. C. Hindmarsh: *Description and Use of LSODE, the Livermore Solver for Ordinary Differential Equations*, LLNL report UCRL-ID-113855, 1993., (https://computing.llnl.gov/casc/nsde/pubs/u113855.pdf)

[4] Fletcher; Roger: *Practical methods of optimization (2nd edition)*, 1987.

[5] Nocedal, J.; S. J. Wright: *Numerical Optimization*, Springer, New York, United States, 2006.

6.2 Publication

[6] Berezvai, Sz.; Kossa, A.: *Characterization of a thermoplastic foam material with the two-layer viscoplastic model*, Materials Today: Proceedings 4, 5749-5754, 2017

[7] M. J. D. Powell: *An efficient method for finding the minimum of a function of several variables without calculating derivatives*, The Computer Journal, Volume 7, Issue 2, 1964, Pages 155-162

[8] Kraft, D.: *A software package for sequential quadratic programming*, Technical Report DFVLR-FB 88-28, DLR German Aerospace Center, Institute for Flight Mechanics, Koln, Germany, 1988.

[9] Nelder, J. A.; R. Mead. *A Simplex Method for Function Minimization*, The Computer Journal 7: 308-13, 1965.

6.3 Website

[10] *Z-set homepage*,
http://www.zset-software.com/solutions/material-parameter-fitting/

[11] *HYPERFIT homepage*, http://www.hyperfit.wz.cz/

[12] *ABAQUS homepage*,
https://www.3ds.com/products-services/simulia/products/abaqus/

[13] *ABAQUS documentation: Two-layer viscoplasticity*,
https://abaqus-docs.mit.edu/2017/English/SIMACAEMATRefMap/simamat-c-viscous.htm

[14] *Wikipedia: Python (programming_language)*,
https://en.wikipedia.org/wiki/Python_(programming_language)

[15] *Python Software Foundation website*, https://www.python.org/about/

[16] *Stackoverflow blog: The Incredible Growth of Python*,
https://stackoverflow.blog/2017/09/06/incredible-growth-python/

[17] *IEEE Ranked the Top Programming Languages of 2019*,
https://learnworthy.net/ieee-ranked-the-top-programming-languages-of-2019/

[18] *Python Package Index: Auto .py to .exe*, https://pypi.org/project/auto-py-to-exe/

[19] *NumPy homepage*, https://numpy.org/

[20] *SciPy Library homepage*, https://www.scipy.org/scipylib/index.html

[21] *Matplotlib homepage*, https://matplotlib.org/

[22] *Riverbank Computing Limited: PyQt*,
https://www.riverbankcomputing.com/software/pyqt/intro

[23] *Encyclopedia of Mathematics: Runge-Kutta method*,
https://www.encyclopediaofmath.org/index.php/Runge-Kutta_method

[24] *Dassault iSight*,
https://www.3ds.com/products-services/simulia/products/isight-simulia-execution-engine/

Appendices

Appendix A

Fitted curves

This appendix contains the plots of the fitted material behavior for all measurement data.

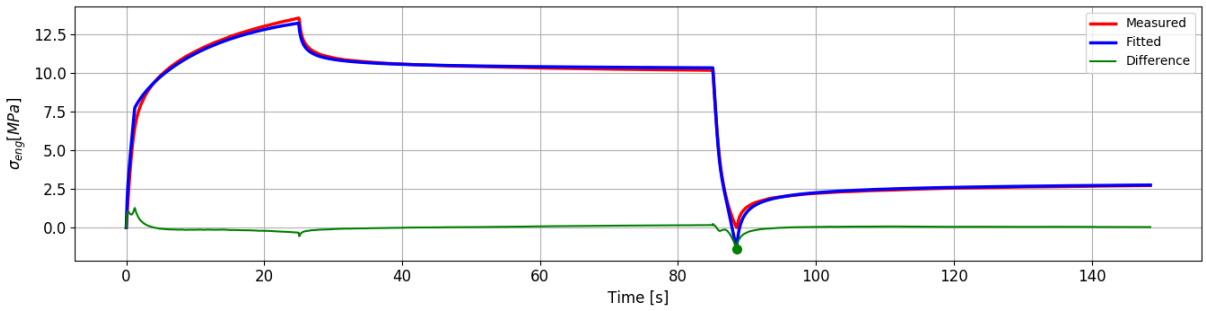

Figure A.1. The $\sigma_{eng} - t$ plot for the $T = 21$ [°C] measurement

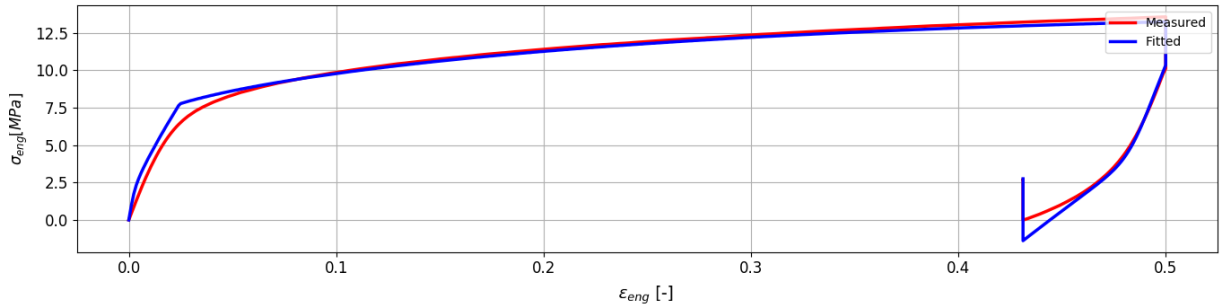

Figure A.2. The $\sigma_{eng} - \varepsilon_{eng}$ plot for the $T = 21$ [°C] measurement

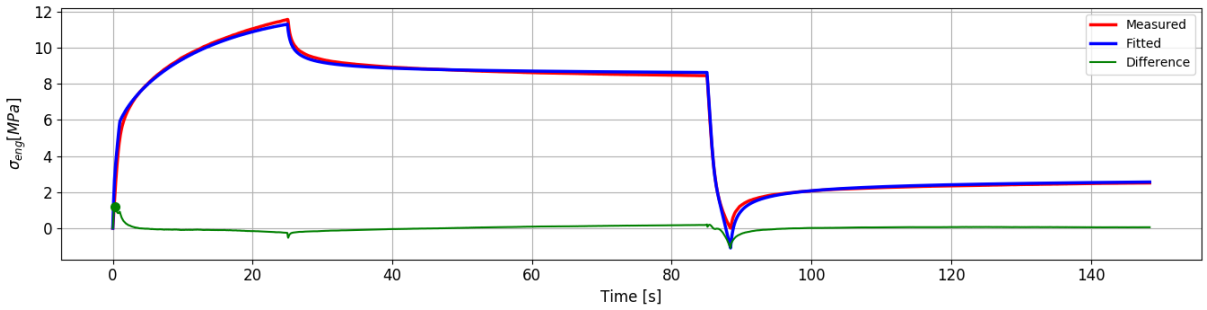

Figure A.3. The $\sigma_{eng} - t$ plot for the $T = 60$ [°C] measurement

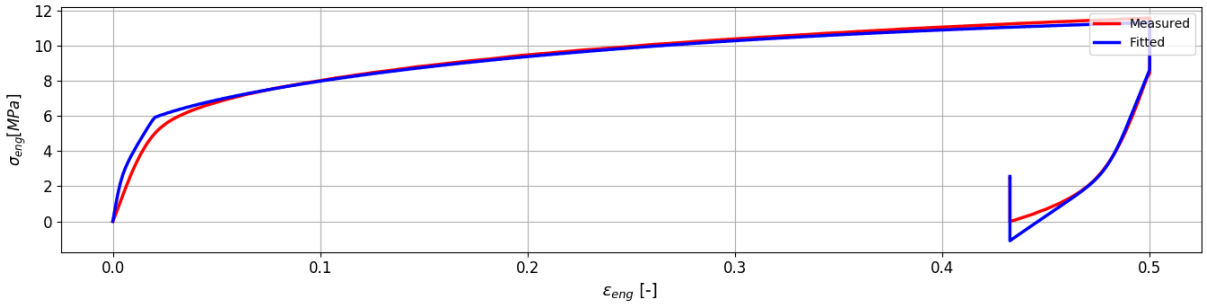

Figure A.4. The $\sigma_{eng} - \varepsilon_{eng}$ plot for the $T = 60$ [°C] measurement

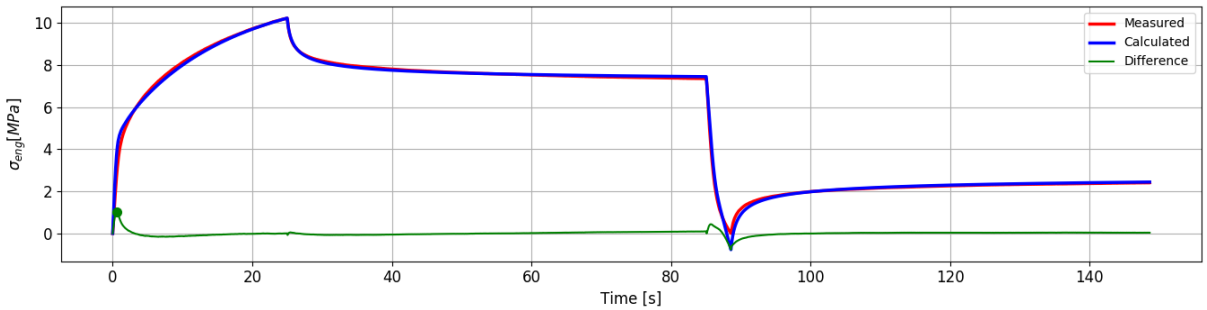

Figure A.5. The $\sigma_{eng} - t$ plot for the $T = 75$ [°C] measurement

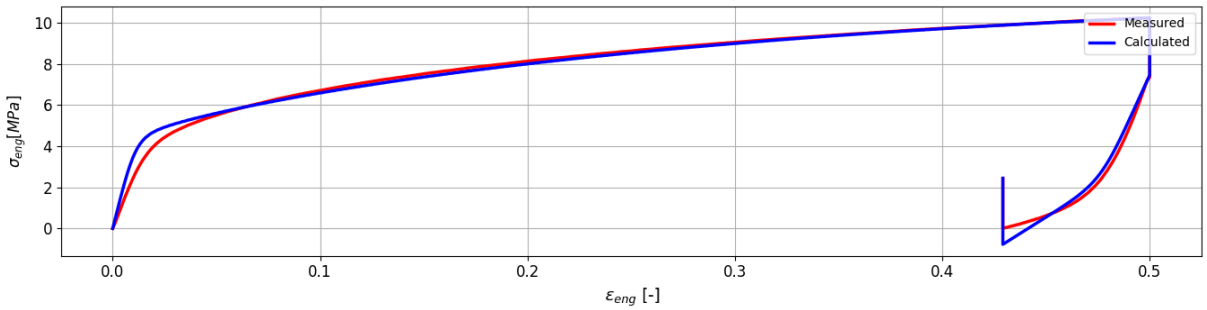

Figure A.6. The $\sigma_{eng} - \varepsilon_{eng}$ plot for the $T = 75$ [°C] measurement

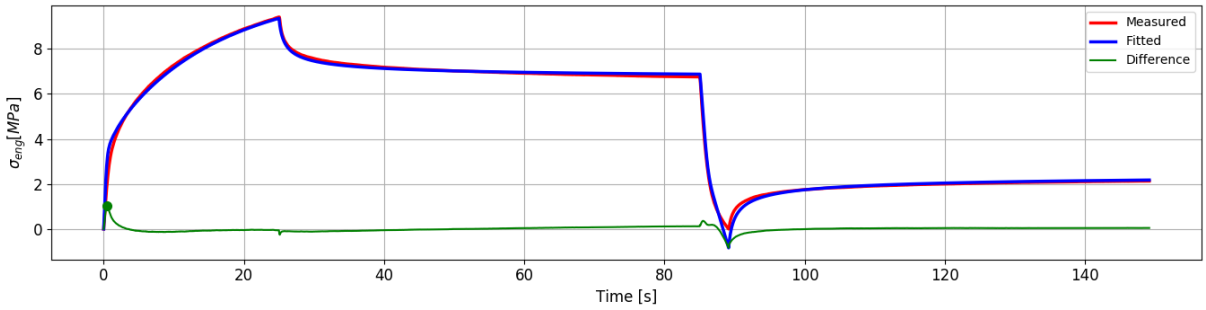

Figure A.7. The $\sigma_{eng} - t$ plot for the $T = 83$ [°C] measurement

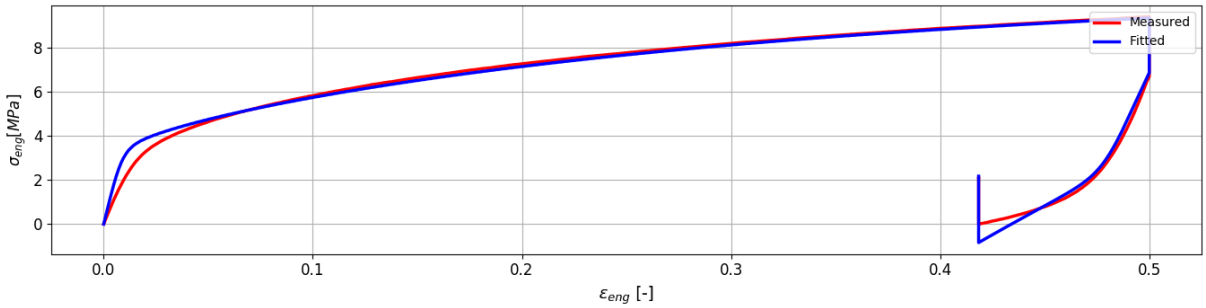

Figure A.8. The $\sigma_{eng} - \varepsilon_{eng}$ plot for the $T = 83$ [°C] measurement

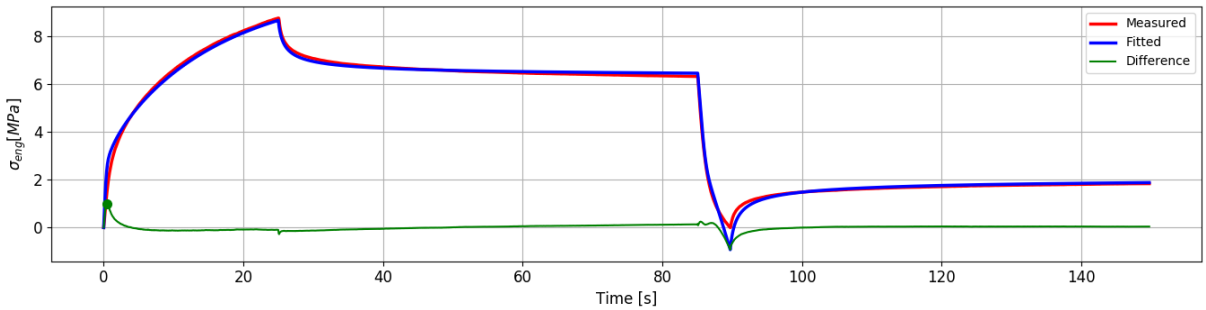

Figure A.9. The $\sigma_{eng} - t$ plot for the $T = 90$ [°C] measurement

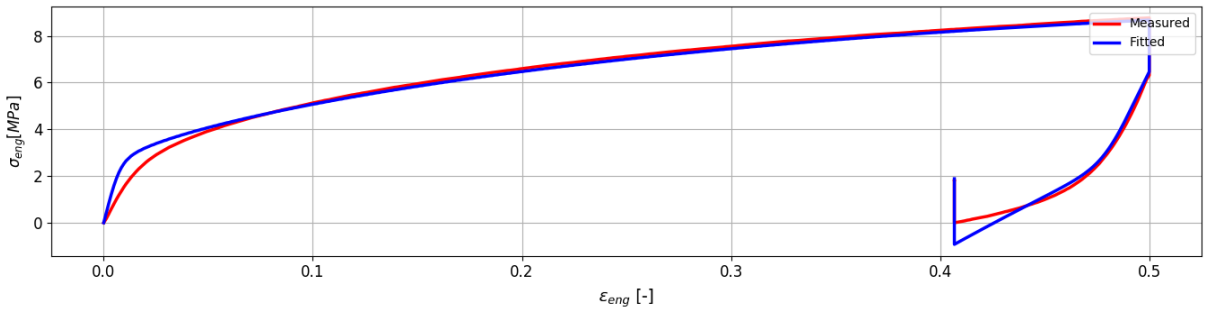

Figure A.10. The $\sigma_{eng} - \varepsilon_{eng}$ plot for the $T = 90$ [°C] measurement

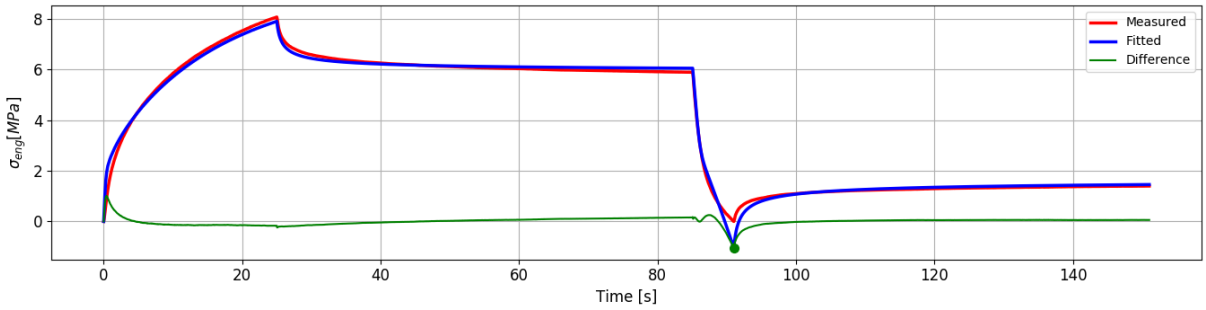

Figure A.11. The $\sigma_{eng} - t$ plot for the $T = 97$ [°C] measurement

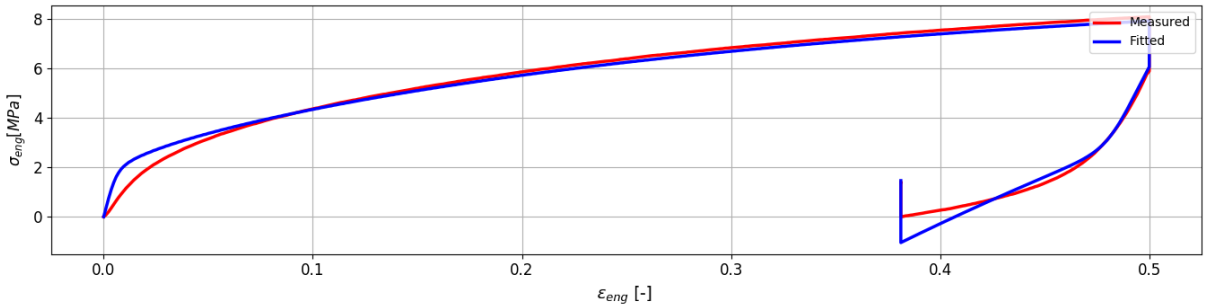

Figure A.12. The $\sigma_{eng} - \varepsilon_{eng}$ plot for the $T = 97$ [°C] measurement

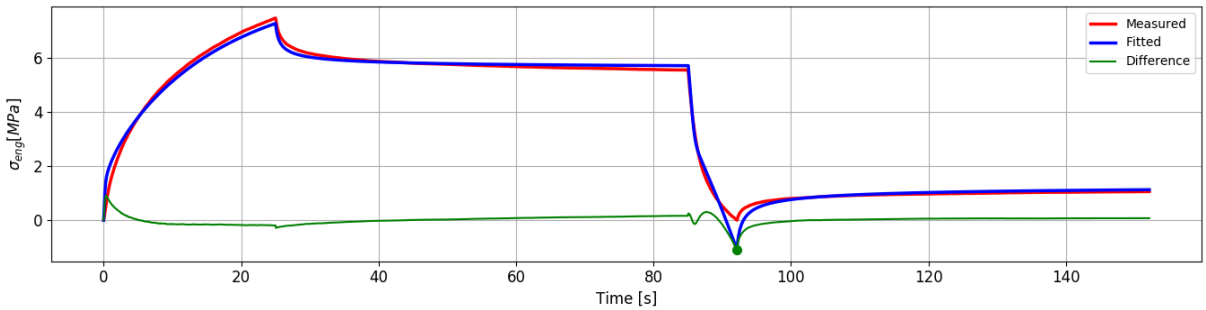

Figure A.13. The $\sigma_{eng} - t$ plot for the $T = 106$ [°C] measurement

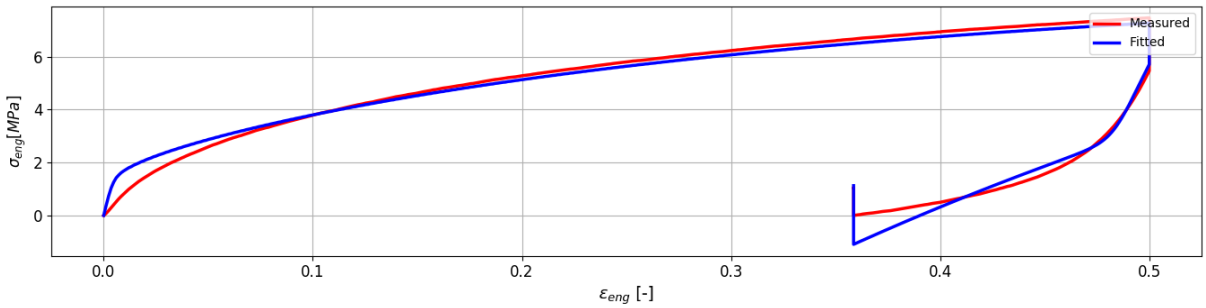

Figure A.14. The $\sigma_{eng} - \varepsilon_{eng}$ plot for the $T = 106$ [°C] measurement

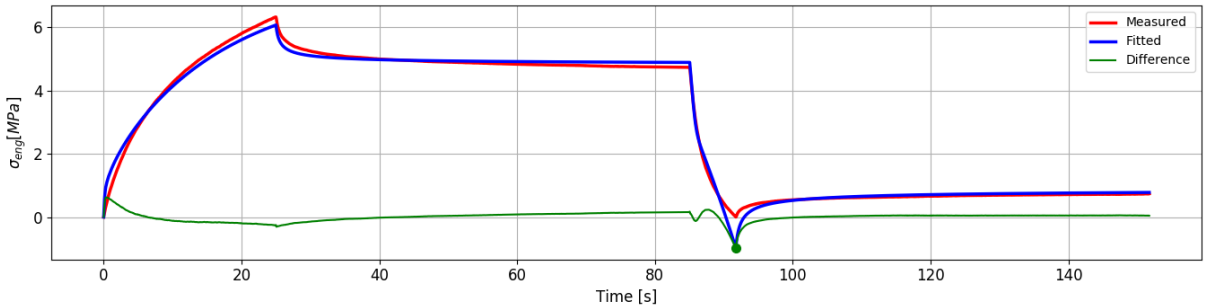

Figure A.15. The $\sigma_{eng} - t$ plot for the $T = 120$ [°C] measurement

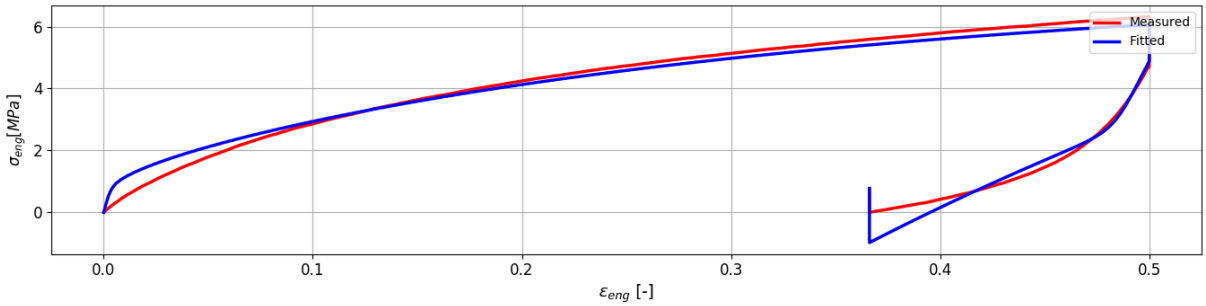

Figure A.16. The $\sigma_{eng} - \varepsilon_{eng}$ plot for the $T = 120$ [°C] measurement

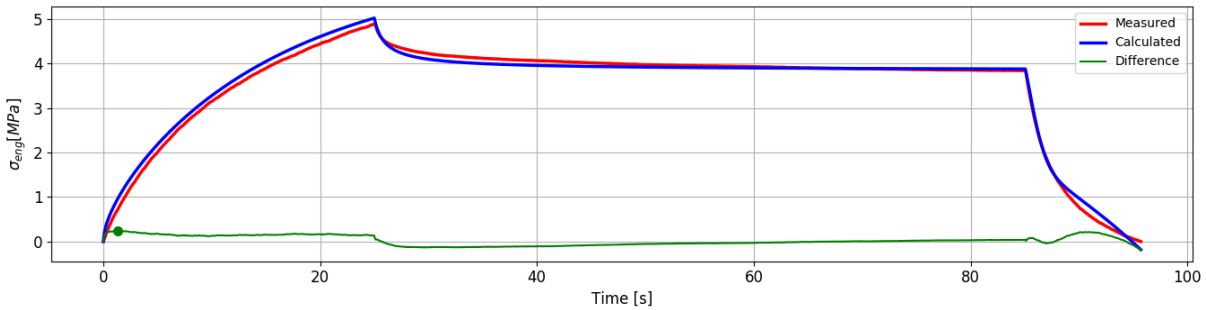

Figure A.17. The $\sigma_{eng} - t$ plot for the $T = 160$ [°C] measurement

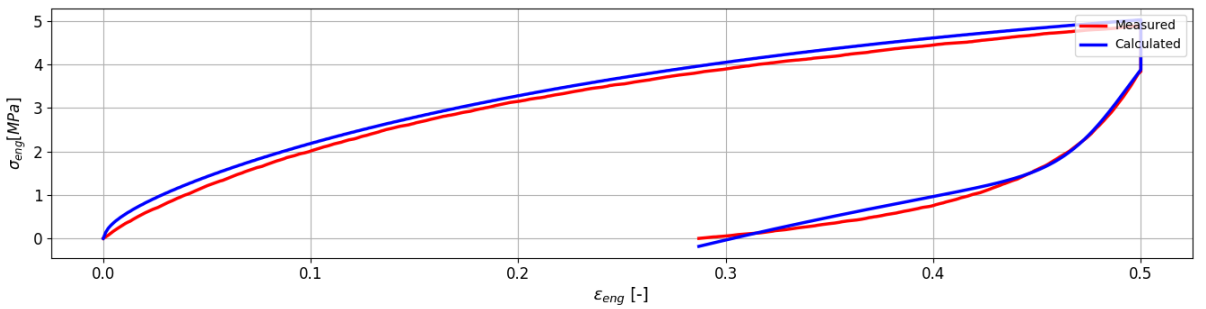

Figure A.18. The $\sigma_{eng} - \varepsilon_{eng}$ plot for the $T = 160$ [°C] measurement

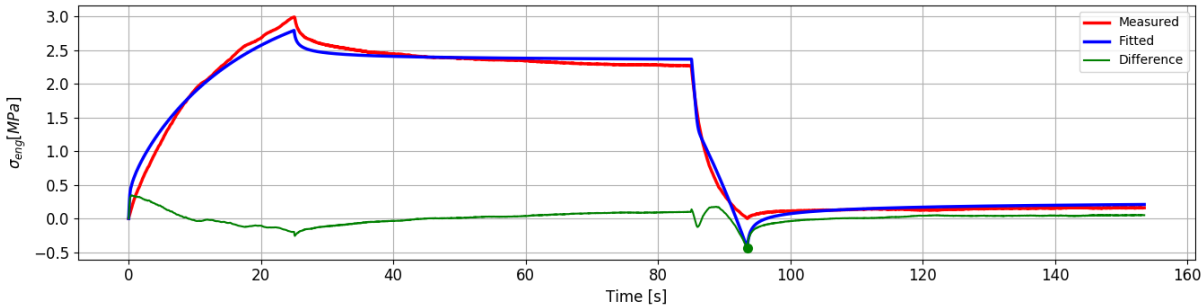

Figure A.19. The $\sigma_{eng} - t$ plot for the $T = 210$ [°C] measurement

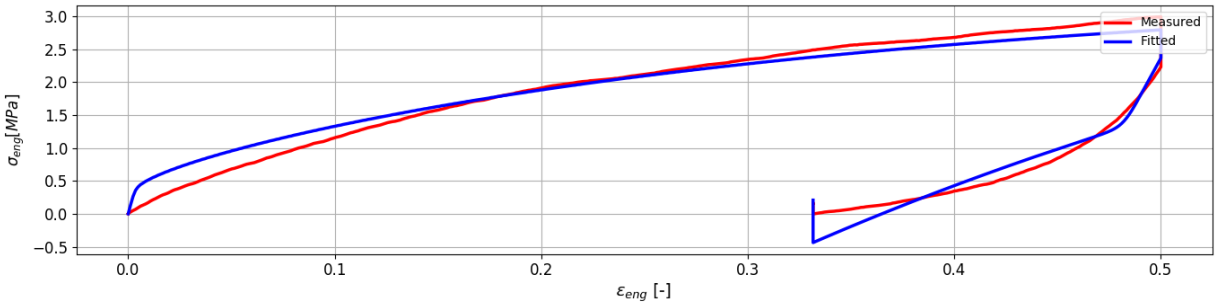

Figure A.20. The $\sigma_{eng} - \varepsilon_{eng}$ plot for the $T = 210$ [°C] measurement

Appendix B

Step-by-step user guide

B.1 Instructions for parameter fitting

1. Load the measurement file into the program. The file is valid if all conditions are met:
   (a) The file must be `.csv` format with comma (',') as separator.
   (b) The data contains one column for time, one for strain and one for stress and:
      i. time data should be measured in seconds
      ii. strain can be either engineering or true strain
      iii. stress is engineering stress measured in mega pascals.

      The order of these columns should be selected on the interface before opening the file.
   (c) The time data is strictly increasing.
   (d) The file may contain a header and other text, but all suck lines must start with a # character.
2. Select the material model to be fitted.
   (a) Change the numerical method if needed. The default methods should work good in most cases.
   (b) Check the 'Use finite strain' checkbox if needed and set the initial stress if it is not 0.
3. Set the initial guess for each material parameter in the 'Value' column.
4. Set the lower and upper boundary for each parameter in the respective 'Min' and 'Max' columns. (Optional, but strongly recommended.)
5. Select the optimizer algorithm. SLSQP is recommended, if a good initial guess is known. In other cases Powell or L-BFGS-B is recommended.
6. Select the quality function. Using 'Squared diff (normalized)' is recommended.
7. Set 'maxiter' and 'maxfev'. (Optional.)
8. Select the parameter mode. To use the 'Normalized' mode, set the parameter boundaries. Using 'Multipliers' or 'Normalized' is recommended. (Optional, but recommended.)

9. Set the other options of the optimizer.
   (a) In the tested cases the default 'eps' was good. Change it with caution, as non-default values can harm convergence.
   (b) 'Boundary cross penalty' affects only Powell from the recommended algorithms.
   (c) The use of random points not recommended if a reasonable initial guess is known.
10. Start the optimizer. Plotting mode can be changed in the 'Settings' tab.

B.2 Instructions for material behavior testing

1. Define a load case by either of the methods below.
   (a) Load a valid measurement file. The file is valid if all conditions are met:
      i. The file must be `.csv` format with comma (',') as separator.
      ii. The data contains one column for time, one for strain and one for stress and:
         A. time data should be measured in seconds
         B. strain can be either engineering or true strain
         C. stress is engineering stress measured in mega pascals.

      The order of these columns should be selected on the interface before opening the file.
      iii. The time data is strictly increasing.
      iv. The file may contain a header and other text, but all suck lines must start with a # character.

      The strain data will be used for the load case. When the simulation is done, the measured strain is plotted as well.
   (b) Manually define a load case on the 'Material behavior' tab.
      i. Define the engineering stress values in time.
      ii. Define the number of linear intervals used to connect the given points.

      If the load case manually defined, it will be used instead of the measurement file.
2. Select the material model on the 'Material behavior' tab.
   (a) Change the numerical method if needed. The default methods should work good in most cases.
   (b) Check the 'Use finite strain' checkbox if needed and set the initial stress if it is not 0.
3. Set the material parameters in the 'Values' column.
4. Simulate the behavior by hitting the 'Calculate' button.