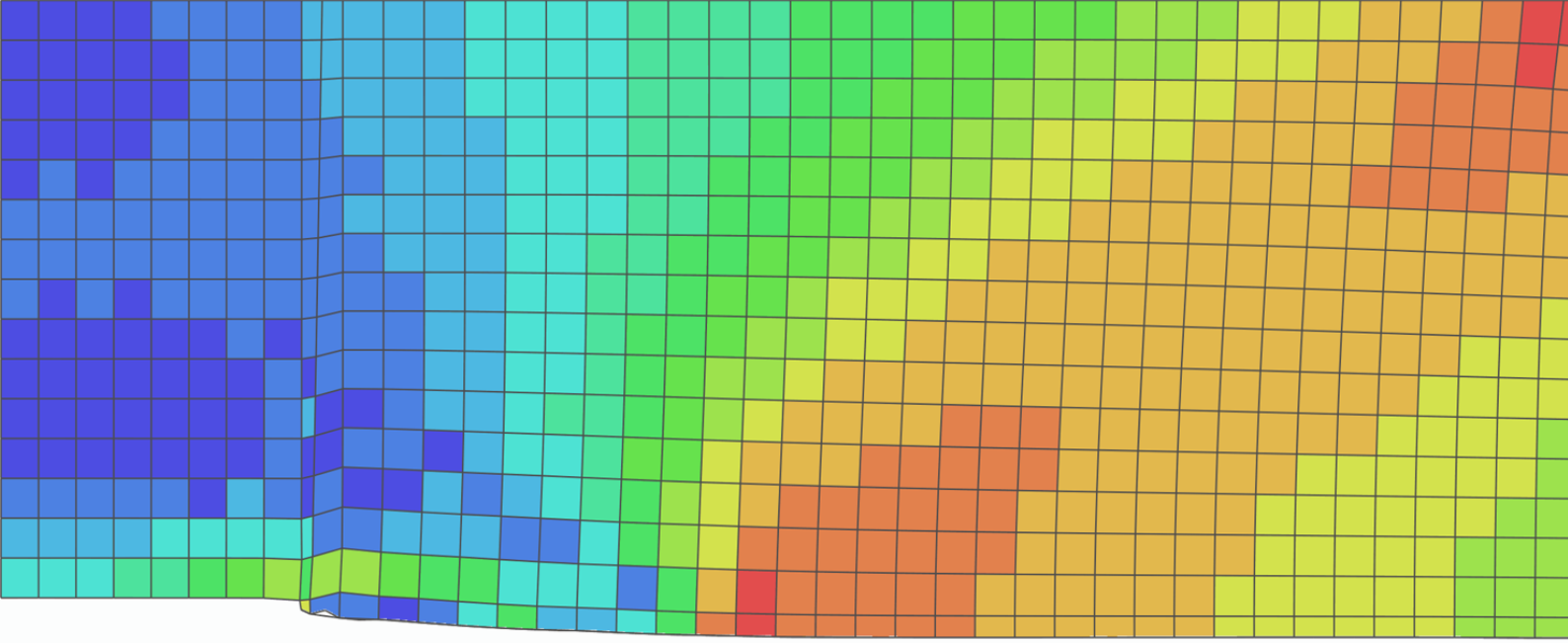
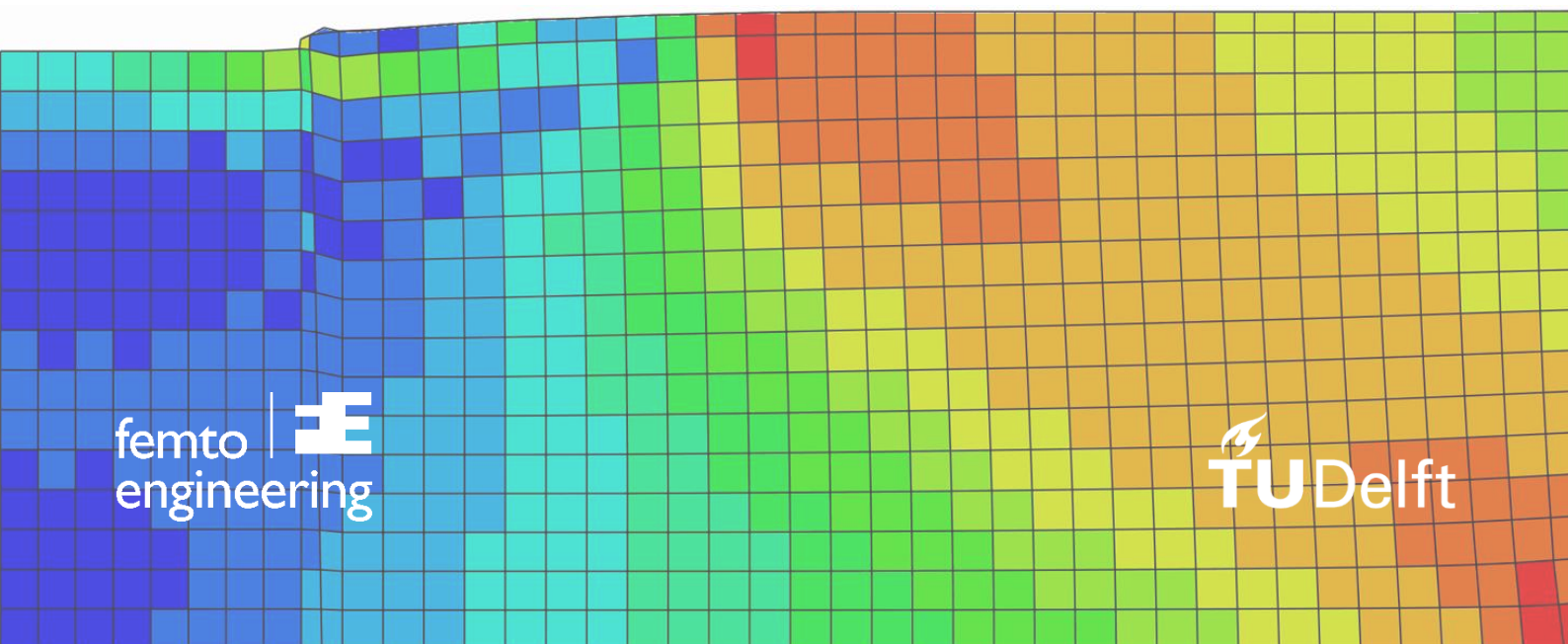


Enhancing FEA crack propagation simulations by employing stress-state-dependent TSLs

M.M. Korving



femto engineering

TU Delft

Thesis for the degree of MSc in Marine Technology in the specialization of Ship and Offshore Structures

Enhancing FEA crack propagation simulations by employing stress-state-dependent TSLs

By

M.M. Korving

Performed at

Femto Engineering

This thesis (MT.23/24.044.M) is classified as confidential in accordance with the general conditions for projects performed by the TUDelft.

To be defended publicly on Wednesday July 31, 2024

Company supervisors
Responsible supervisor: Ir. R. Vielvoije

Thesis exam committee
Chair/Responsible Professor: Dr. C.L. Walters
Staff Member: M.A.A.M. Adly
Staff Member: Prof.dr.ir. L.J. Sluys
Company Member: Ir. R. Vielvoije

Author Details
Study number: 5188369

Abstract

Multiple methods exist for simulating crack propagation in the finite element method (FEM). Among these, the extended finite element method (XFEM) shows great potential as it allows for the use of larger elements while maintaining accuracy and practicality. Within XFEM, the employment of traction separation laws (TSLs) is a promising approach to capture post-necking effects, which are typically predicted poorly with larger elements. These TSLs are often assumed to be constant along the crack length and are determined by fitting to experimental data.

However, it has been proven that necking and fracture both depend on the state of stress, expressed as stress triaxiality. It is also known that the stress triaxiality within the crack region varies as the crack progresses through a plate due to changing boundary conditions and crack blunting. Consequently, the necking behaviour ahead of the crack tip changes, necessitating multiple stress state-dependent TSLs along the crack length to accurately capture this effect.

In this thesis, a subroutine is developed to enable the use of multiple TSLs along an extending crack in Abaqus, based on the stress triaxiality of each element. Additionally, the relationship between TSL parameters and stress triaxiality was determined by minimizing the difference between simulations and experimental data of a large-scale CCT experiment. With this relationship known, it is possible to account for the effect of the changing stress state when determining the TSLs.

The study finds that stress triaxiality-dependent TSLs produce a significantly better match with experimental results compared to a stress triaxiality invariant TSL. This underscores the importance of addressing stress-state dependency in TSL determination. The study also highlights that mesh size and material dependency on TSLs must be addressed so that these TSLs can be applied universally across different conditions.

Preface

This thesis, "Enhancing FEA crack propagation simulations by employing stress-state-dependent TSLs," serves as my final project and concludes my studies at the TU Delft. Within the Master Marine Technology, I got increasingly invested in structural problems and really liked courses that focussed on ultimate strength, material behaviour and FEM. This thesis is a logical conclusion to this chapter, and I was able to develop both my FEM and programming skills, as my understanding of failure and fracture.

Numerous persons have played a major role in this project, and I am very grateful for their help. First of all, I want to thank Carey Walters, for the valuable insights and feedback. Your courses "Sterkteleer van Schepen" and "Ultimate Strength of Ships" made me enthusiastic about structures and inspired me to continue in this field. I immediately noticed that you take the time to help your students, whether this is during a course or as a graduation supervisor, and I really appreciated this.

Secondly, I am grateful to Mohammed Adly, who served as my daily supervisor at the university. I value your engagement with my project and the informal atmosphere of our meetings. You were very approachable, and I always left our meetings more secure and with a better sense of direction.

Furthermore, I want to express my gratitude to Richard Vielvoije for being my supervisor within Femto. I admire your willingness to contribute or troubleshoot, even after working hours. Whether it is by interpreting doubtful FE results or giving sharp feedback on reports and presentations, I knew I could always count on your help.

My gratitude extends to Vincent Bouwman from 4RealSim for troubleshooting the use of XFEM and subroutines, to Bert Sluys for being on my graduation committee and giving insights on my thesis, and to Tom Santegoeds for making this project possible within Femto.

Finally, I am grateful to my family, girlfriend, and friends for their support and for providing the occasional distraction.

M.M. Korving
Delft, July 2024

Contents

Nomenclature

Abbreviations

CCT	Central Crack Tension
CPS4R	4-node bilinear plane stress quadrilaterals with reduced integration and enhanced hourglass control
CRKOPEN	Crack Opening displacement
CRKPRESS	Crack Pressure
CZM	Cohesive Zone Model(ling)
FE	Finite Element
FEA	Finite Element Analysis
FEM	Finite Element Method
GTN	Gurson Tvergaard Needleman
HRR	Hutchinson-Rice-Rosengren
MAE	Mean Absolute Error
MMC	Modified Mohr-Coulomb
NS	Normal Strength
TSL	Traction Separation Law
XFEM	eXtended Finite Element Method

Symbols

$\bar{\sigma}$	Stress of undamaged model
δ	Crack opening displacement
δ_0	Failure displacement
δ_1	Displacement at damage initiation
δ_2	Displacement at moment of shear localisation

Γ_0	Cohesive energy
Γ_{II}	Energy dissipated after the onset of shear localisation
Γ_I	Energy dissipated in the necking process
σ_1	Normal stress in x-direction
σ_2	Normal stress in y-direction
σ_3	Normal stress in z-direction
σ_h	Highest defined point at stress-strain diagram
σ_m	Hydrostatic stress
σ_y	Yield stress
σ_{eq}	Von Mises equivalent stress
σ_{y0}	Cohesive stress
u	Displacement
u_{dis}	Discontinuous displacement
u_{reg}	Regular displacement
u_{sing}	Displacement resulting from stress singularity at crack tip
D	Damage parameter
K	TSL stiffness
n	Number of data points
T	Stress triaxiality
Y_i	Y-coordinate of experimental results
y_i	Y-coordinate of FE results

Introduction

The maritime industry continually seeks optimal designs to minimize material use and overall weight while still conforming to strict safety and strength criteria. Achieving lighter and more efficient structures requires advanced assessment methods to accurately predict the structural behaviour and capacity of these designs. The finite element method (FEM) is such a method, and multiple types of failure can be simulated quite accurately.

One critical condition that could affect the structural integrity of maritime structures is the formation and propagation of cracks in plates, particularly during extreme events such as collisions, ship groundings and attacks. However, despite significant advancements, accurately modelling crack propagation in a practical way remains challenging. Traditional finite element techniques often necessitate the use of very small elements (many times smaller than the plate thickness) to achieve accurate results, which is computationally impractical for large-scale applications. The extended finite element method (XFEM) presents a promising solution by allowing the use of larger elements while retaining accuracy and practicality.

One essential aspect of simulating crack propagation in XFEM is the incorporation of traction separation laws (TSLs) which can help to capture post-necking effects. TSLs represent the material behaviour after significant deformation, which is often inadequately predicted when larger elements are employed. Typically, TSLs are assumed to be constant along the crack length and are derived from fitting to experimental data. However, this approach overlooks the variations in the stress state within the crack region, particularly the stress triaxiality, which changes as the crack progresses through the material.

This thesis addresses this effect and aims to determine the TSLs based on the local state of stress that occurs within the crack region, thereby enhancing the accuracy and practicality of XFEM for simulating crack propagation in large maritime structures. By incorporating stress-state-dependent TSLs, the developed approach provides a more physically accurate assessment method for predicting the structural behaviour and capacity of these structures.

Motivation

The current research is of both engineering and social relevance. It contributes to engineering by increasing the accuracy of FE fracture simulations with coarse meshes. Employing larger elements in these simulations allows for a practical way of modelling fracture, and computational time and effort (and thus costs) are saved. This research thus improves the engineering field by reducing costs and enhancing practicality.

The social relevance of this research is that structures will be designed more efficiently and safely. If accurate FE fracture simulations can be done on extreme events such as ship collisions or grounding, lives can be saved when these events occur.

Outline

This thesis is structured as follows: first, a literature study is performed in chapter 1, which leads to a research gap and research questions, described in section 1.5.

Subsequently, chapters 2 and 3 explain the two main parts of this research: the development of a TSL-tool that incorporates stress-state dependent TSLs in Abaqus and the optimization study, which aims to find the relationship between the TSL and stress triaxiality. Finally, the results of the optimization study are presented in chapter 4, and a future outlook is given in chapter 5.

Chapter 1

Literature review

1.1 Background on ductile fracture

This chapter gives a brief overview of the most important aspects of ductile fracture and crack propagation to gain a broad understanding of the relevant theory treated within this thesis. The focus will be on ductile fracture in metals and metallic alloys. For completeness, both the microscopic and the macroscopic fracture behaviour are described.

1.1.1 Microscopic fracture behaviour

A dominant failure mechanism in ductile metals is nucleation, growth, and coalescence of microscopic voids [39][8]. When stress occurs in a metal, voids or cavities nucleate at inclusions and second-phase particles which occur in the metal matrix due to the manufacturing process. At these locations, high stress concentrations occur due to external loading.

After nucleation, the voids grow in size resulting from the increasing applied strain. Because the void itself acts as a discontinuity as well, stress concentrations will now occur around the void, causing a further increase in the amount of stress. Due to this phenomenon, the void can grow, and elongation can occur at a rate that is about two times that of the surrounding material. The growth is stimulated by the movement of dislocations in the crystal lattice of the metal. These dislocations move to allow for the expansion of the voids, resulting in local plastic deformation in the lattice [39][8].

When significant void growth occurs, the process is terminated by void linking, or coalescence. In this step, multiple voids connect and form a link that further weakens the material, resulting in reduced stress-carrying capability. The exact manner in which void coalescence occurs is highly dependent on the failure mode, microstructural factors, and loading conditions.

Eventually, void coalescence leads to the formation of microcracks, which in turn evolve into macroscopic cracks when the stress is increased further. After the initiation phase, the macroscopic crack propagates through the material. This process is marked by the tearing of atomic bonds and the dissipation of stored elastic energy.

The final stage is critical failure. The crack propagates until reaching a critical length where after the material fails and total separation occurs [39][8].

In figure 1, the stages as highlighted above are shown on a force-displacement curve of a round notched bar, obtained in an experiment executed by [9]. It can be seen when the different failure stages occur in the macroscopic degradation of a metal under tension. It is visible that after the specimen has attained maximum strength and necking is initiated, damage accumulates gradually in the form of void nucleation and growth. The picture marked by (c) shows the coalescence of the largest voids, resulting in a macroscopic crack that causes the first significant degradation of stress-carrying capacity. In the remaining descending part of the curve, the damage increases through crack propagation at the inside of the specimen and eventual failure, denoted by (f).

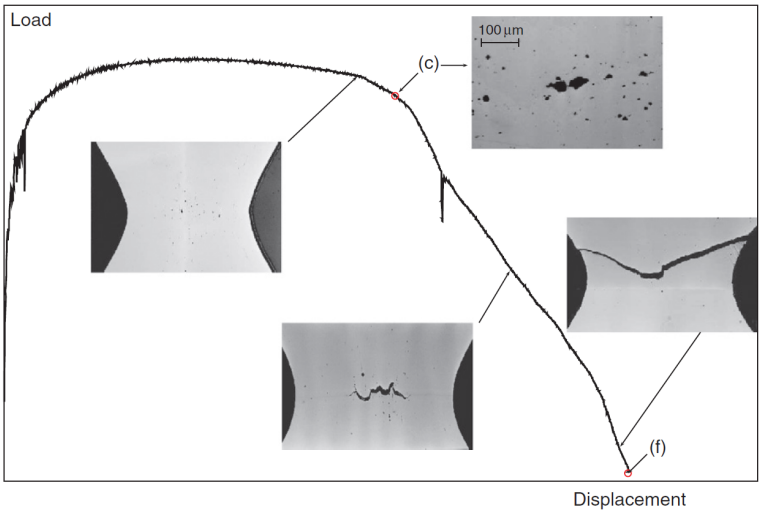


Figure 1: Force displacement curve showing the different stages of microscopic fracture behaviour: void nucleation, growth, and coalescence [9].

1.1.2 Macroscopic fracture behaviour

Figure 2 shows different steady-state conditions that are observed in an advancing mode I crack in metal plates. These stages characterise the macroscopic fracture behaviour of metals.

In the fracture process, failure occurs in the form of plastic dissipation during steady-state conditions. The first of these conditions is local neck initiation, which occurs well ahead of the crack tip (figure 2 a). The onset of necking is regarded as the moment at which damage initiates.

Subsequently, the neck will develop further (figure 2 b) until a small-scale localization of shear stress occurs in the neck, called a shear band (figure 2 c). This shear band significantly weakens the material and finally leads to failure just ahead of the crack tip (figure 2 d), and separation occurs [36].

Figure 3 depicts a force-displacement diagram that was obtained by [36]. From this figure, the steps depicted in figure 2 can be distinguished. It also becomes clear that necking begins at the maximum load. After necking is initiated, relatively high deformation is localized within the neck.

As the neck further develops, the load is gradually decreasing until shear localization occurs. After this point, the load falls abruptly with relatively small additional elongation. The small deformation in this last step is mostly limited to occur within

the shear band. With progressing shear in the shear band, damage increases, and eventually failure occurs [36].

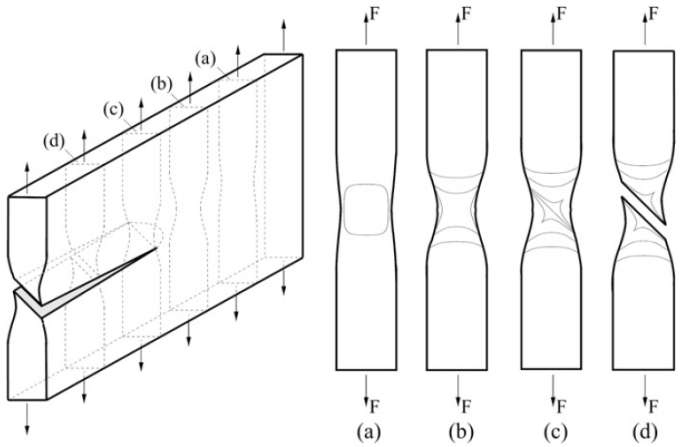


Figure 2: Fracture process of a crack subjected to mode I loading: (a) onset of necking ahead of the crack tip, (b) developed neck closer to the tip, (c) shear localization, and (d) failure at the crack tip [36].

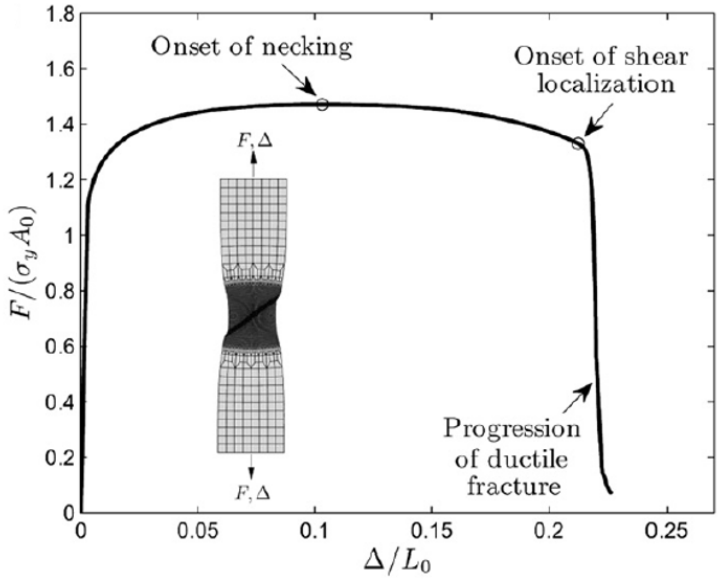


Figure 3: Normalized force-elongation curve for steady state fracture, obtained by [36]. Here, σ_y, A_0, and L are the yield stress, initial cross-sectional area, and initial length, respectively.

1.1.3 Stress state

As mentioned earlier, a dominant factor in the fracture of ductile metals is the growth and coalescence of microvoids. Various studies have proven that the growth of these voids is influenced heavily by the state of stress in a material.

A suitable parameter to define the state of stress in a material is found to be the triaxiality parameter T. T is defined as the ratio of hydrostatic stress, σ_m, to the Von Mises equivalent stress, σ_{eq} [20] and is expressed as:

$$T = \frac{-\sigma_m}{\sigma_{eq}} = \frac{\frac{1}{3}(\sigma_1 + \sigma_2 + \sigma_3)}{\sqrt{\frac{(\sigma_1-\sigma_2)^2+(\sigma_2-\sigma_3)^2+(\sigma_3-\sigma_1)^2}{2}}}, \tag{1}$$

where σ_1, σ_2 and σ_3 are the normal stresses in x-, y- and z direction, respectively.

From eq.(1), it can be concluded that low triaxiality values correspond to a lower degree of hydrostatic tensile stress and, thus, a higher degree of deviatoric stress. Deviatoric stress comes with higher levels of shear stress, and because the movement of line defects, such as dislocations in the microstructure of the metal, is mainly controlled by shear stress, plastic flow and, thereby, ductility is expected to be higher for low values of triaxiality [52]. On the other hand, a higher value of stress triaxiality, in turn, corresponds to less ductility.

This behaviour is also depicted in figure 4, where a typical fracture locus for steel is given, showing the failure strain against the stress triaxiality. Globally, a decreasing failure strain is observed for increasing levels of triaxiality. In the figure, primary stress states are indicated as well, such as uniaxial compression (T=-1/3), pure shear (T=0), uniaxial tension (T=1/3), plane strain tension (T=$1/\sqrt{3}$) and biaxial tension (T=2/3).

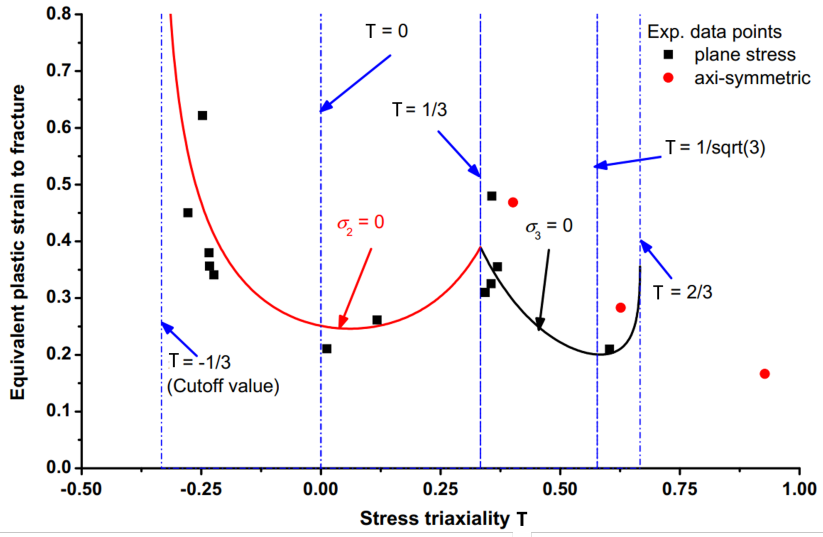


Figure 4: Typical failure locus showing failure strain for different stress states [27].

Various experiments on the effect of stress triaxiality on ductile tearing and failure have been executed. The results of these studies are generally in line with the aforementioned trend. It was found that higher values of stress triaxiality result in fracture initiation at lower plastic strains [28][20]. Also, void growth rates are increased when the material is experiencing a moderate to high level of stress triaxiality [29][43][25][10], leading to accelerated coalescence and formation of shear bands [38].

The effects described above emphasize the significant influence of stress triaxiality on the damage behaviour. Section 1.3 elaborates on the effect of stress triaxiality further.

1.2 Numerical modelling of cracks

The previous chapter explained the basic process of ductile fracture. This chapter provides information about current options for modelling this fracture behaviour and their limitations. A distinction is made between commonly used methods and methods that allow for the use of bigger elements, which is in line with this thesis's overhanging goal.

1.2.1 Motivation for numerical models

Structural components that need to be evaluated for crack-like defects or failure are commonly assessed by methods that rely on classical fracture mechanics. This field of mechanics provides theoretical principles and analytical tools to understand and predict the behaviour of cracks in materials under stress. Assessment methods based on classical fracture mechanics have been incorporated into international standards and codes (such as BS7910 and ISO 12135) and are regarded as a powerful and reliable engineering tool [16]. However, there are also limitations to these methods. The most important limitation is the fact that fracture mechanics applications describe the constraints of a cracked body or specimen based on a single parameter. This single-parameter description can only be accurate under a limited number of conditions. For example, test standards are mainly focused on plane strain conditions and disregard other stress states. For these other stress states, the description of the stress and strain conditions near a crack tip is incomplete, and accuracy is lost [16].

The problem that arises is known as the transferability problem. Because the variation of stress states in test pieces is limited, the fracture properties obtained by methods based on classical fracture mechanics are limited as well. Therefore, these methods are not always suited for the examination of structures, where a broader range of boundary conditions and constraints lead to a high variation of stress states [16].

A possible solution is the use of numerical methods, such as FE models. The evident advantage of an FE model is that it can calculate the states of stress or strain in a body more accurately. If the model can use this state of stress in determining the damage parameters, the transferability problem will be solved, and an accurate description of deformation and fracture behaviour can be given as output [49].

1.2.2 Common FE approaches

This section describes the element deletion method and damage models, which are the most commonly used approaches for modelling cracks in FE software.

1.2.2.1 Element deletion

Element deletion in Finite Element Method (FEM) is a numerical technique used to simulate the propagation of cracks in materials. In this method, elements are progressively removed as the crack propagates. Elements are typically deleted when they reach a critical threshold of stress, strain, or damage, as determined by damage

models (described in section 1.2.2.2). These damage models assess the material's condition under applied loads, predicting when and where failure occurs.

The element deletion method provides a realistic representation of how cracks develop and propagate through a material, and it is a popular approach to modelling crack propagation in the industry. However, the technique comes with challenges, such as sensitivity to mesh size and quality, which can affect the accuracy of the results. Capturing post-necking effects and obtaining an accurate description of the crack path requires a very fine mesh, as elements must have an in-plane dimension much smaller than the plate thickness. Numerical stability can also be an issue, as the sudden removal of elements may lead to convergence problems in the FEM analysis. Implementing effective damage models and deletion criteria requires careful consideration of material properties and loading conditions.

1.2.2.2 Damage models

The element deletion method selectively deactivates elements when they reach a critical threshold of stress, strain, or damage. This threshold is determined by damage models. Damage models are mathematical and computational frameworks that predict the initiation and evolution of damage in materials under various loading conditions. These models simulate how materials degrade over time, ultimately leading to failure. In this section, the distinction is made between two types of damage models: phenomenological models and damage mechanics models.

1.2.2.2.1 Phenomenological models

Phenomenological damage models are empirical in nature, meaning that they are primarily based on observed phenomena and experimental data rather than detailed knowledge of the underlying microstructural mechanisms. They are widely used because they effectively capture observed material response while simplifying the underlying physical mechanisms. Examples of damage criteria are the Modified Mohr-Coulomb (MMC) fracture criterion [6], the Hosford-Coulomb (HC) criterion [31], the Johnson-Cook (JC) criterion [24] and the shear criterion [22][1].

From these models, the MMC, HC, and JC models depend, amongst other factors, on the level of stress triaxiality. This means that the stress state is accounted for in determining damage. In section 1.1.3, it was concluded that this is indeed important, as the stress state significantly influences damage.

1.2.2.2.2 Damage mechanics models

In contrast to phenomenological models, damage mechanics models incorporate a physically based description of the damage process. In the context of ductile fracture, this means the nucleation and growth of voids in the material, with the goal of predicting the initiation and propagation of fracture.

An example of a damage mechanics model is the constitutive GTN model. After the discovery that ductile fracture was influenced so heavily by the effect of voids, the need arose for a model that could incorporate this effect in the formulation of damage. It was developed in 1977 by Gurson [19] and later modified by Needleman and Tvergaard in 1984 [33]. The model describes the softening effect resulting from the growth and coalescence of voids [40] and depends on the triaxiality of the stress state as well as the physical details of the voids, such as initial void volume fraction and critical void volume.

A drawback of the GTN model is that a significant amount of parameters are needed

to describe the process of nucleation, growth and coalescence of voids. Therefore, it is not a popular approach to use for practical applications [7].

1.2.3 FE approaches that support bigger elements

The preceding sections describe the element deletion method and how it can model crack propagation in FEM by combining it with a damage model. It was mentioned that an important aspect of this method is that very fine meshes are required to obtain reasonably accurate results.

This requirement is needed for multiple reasons. First of all, the element size and shape influence the crack path. Secondly, bigger elements tend to average out stress peaks that occur at the crack tip, which leads to an inaccurate description of the stress field near a crack. Lastly, small elements are needed to capture the effect of necking. An explanation of this last phenomenon is given by referring to figure 3. In this figure, the force-elongation curve of a plate under uniaxial tension is given. When modelling cracks in FEM, the obtained output must match this behaviour and the elements are required to capture important details of the fracture process. One of these details is the necking behaviour before the onset of significant material damage, as explained in section 1.1.2.

To capture the necking effect with element deletion, the elements must have an in-plane element size much smaller than the plate thickness. Elements with in-plane dimensions that are larger than the plate thickness can accurately describe the material behaviour prior to the onset of necking but are unfortunately unable to solve the deformation in the neck and shear band [36].

Mesh refinements, as needed by the element deletion method, are possible for very small sections of structures or test specimens but are very difficult to use when analyzing larger structures or dynamic problems. The scaling issue that arises is an important problem that needs to be overcome in order to analyse bigger structures. FE methods that can handle bigger element sizes while remaining accurate are a solution to this problem. In the following section, two of these methods, the cohesive zone model and the extended finite element method, are described.

1.2.3.1 Cohesive zone model

An often-applied approach to using larger elements is incorporating a Cohesive Zone Model (CZM). A CZM is a phenomenological concept where interface elements with no initial thicknesses are placed at crack locations in between the normal continuum elements. These interface elements, or cohesive elements, can open up in the same way as a crack, following a specific constitutive relation called a traction separation law (TSL). The surrounding continuum elements remain undamaged and capture the material behaviour outside the crack process zone [53].

In the CZM approach, the TSL must provide a phenomenological approximation of the failure process. This failure process consists of the gradual loss of load-carrying capacity beyond the onset of necking which, as was concluded earlier, could not be captured by the large elements in their standard form [36][50]. The softening behaviour is now described by a stress-displacement (or traction-separation) relationship instead of a stress-strain relationship. The idea of cohesive zone modelling is presented in figure 5. In this figure, the curve for the TSL and the continuum elements correspond to the

example of the metal sheet described in section 1.1.

The cohesive zone model is a user-friendly model that has the advantage of allowing for the use of larger elements while still being able to capture the effects of necking and shear localisation. Other advantages are the numerical robustness and the need for only a few parameters [49][16].

Figure 6 shows the TSL used in this example. The normalised traction (force per original area) is shown on the vertical axis, and the normalised separation is shown on the horizontal axis. As mentioned, the figure shows the behaviour from the onset of necking until failure. Because the TSL takes care of the behaviour that can normally only be simulated with very fine meshes, it is now possible to use larger elements. However, one can imagine that the correctness of the parameters that describe this TSL is of great importance for the accuracy of the final result. Therefore, a significant amount of research has been put into the determination of correct TSL shapes and values. A small overview of this is given in section 1.2.4.

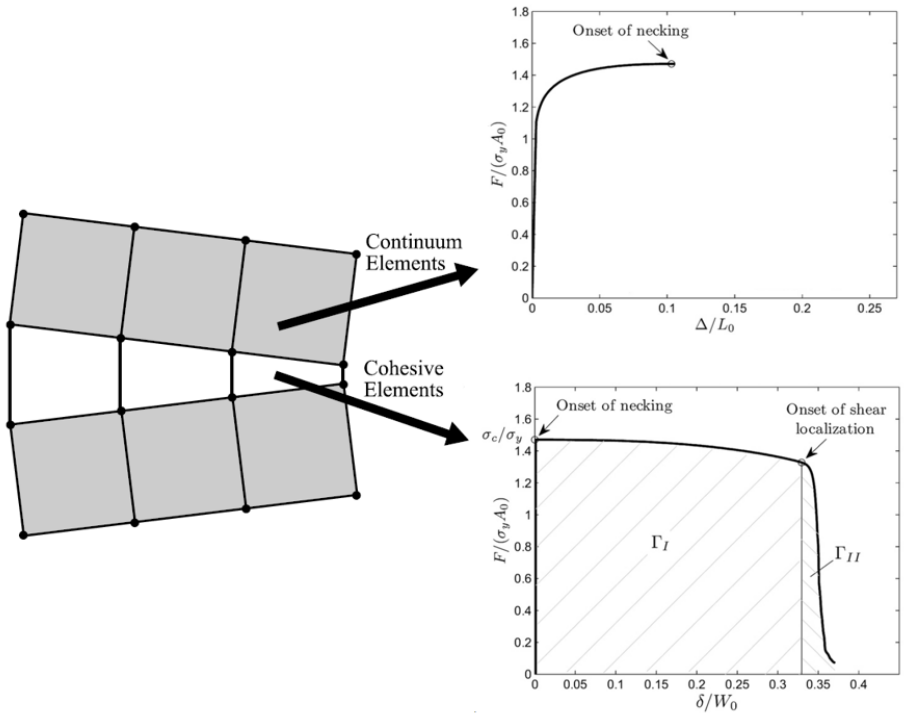


Figure 5: Application of the cohesive zone model. The crack opening displacement is controlled by a TSL, representing the damage after the onset of necking. The behaviour of the plate prior to necking is captured by the continuum elements surrounding the crack. The original figure is taken from [4].

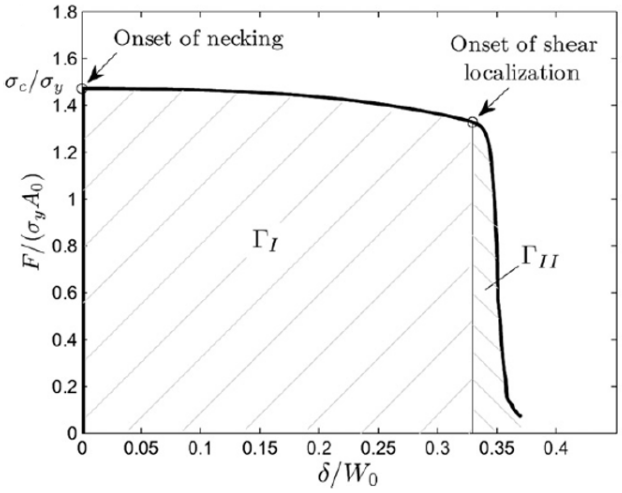


Figure 6: Traction-separation curve that describes the failure process of the metal sheet described in section 1.1. The total fracture energy is divided in fracture energy associated with necking, Γ_I, and fracture energy associated with shear localisation and fracture, Γ_{II} [36].

1.2.3.1.1 Limitations of CZM

As explained in the previous sections, the CZM approach is a suitable method to model cracking behaviour in large elements and structures. There are, however, also limitations.

A dominant limitation of the CZM method is the fact that the location of the crack path must be known in advance. If information about the crack path is not available, cohesive elements can be placed on every location in the material where it is suspected that a crack could occur [49]. The crack path is now defined as an array of 'activated' cohesive elements, whilst failure is not yet reached in the adjacent cohesive elements. Even though this method could serve as a workaround, it has multiple shortcomings. First, the amount of computational effort increases with the number of elements used. Therefore, applying multiple cohesive elements in a large region that may or may not be needed is not effective in this regard. Secondly, the crack path is still not completely arbitrary as the shape and size of the elements are of influence here. A final remark is that particular element orientations can cause heavy zigzag patterns in the crack path, which can cause numerical problems as well [49].

This shortcoming stands in the way of practically employing the cohesive zone method on real-life structures. A method that overcomes this issue is the extended finite element method.

1.2.3.2 Extended finite element method

The extended finite element method (XFEM) is an extension of the commonly used finite element method and is used for the simulation of complex geometries, discontinuities, and singularities. Just like the CZM, it can be used with a TSL which gives similar advantages concerning the use of larger elements. However, as described above, XFEM solves the problems caused by the unknown crack path that are encountered in CZM. In this section, the concept of XFEM is explained.

1.2.3.2.1 Basic concept

The working principle of XFEM is that discontinuities in an element, such as cracks, are allowed by 'enriching' the degrees of freedom of the nodes with updated displacement functions. These updated displacement functions represent the kinematics resulting from the discontinuity [58][1] and are given as

$$u(x) = u_{\text{reg}}(x) + u_{\text{dis}}(x) + u_{\text{sing}}(x). \tag{2}$$

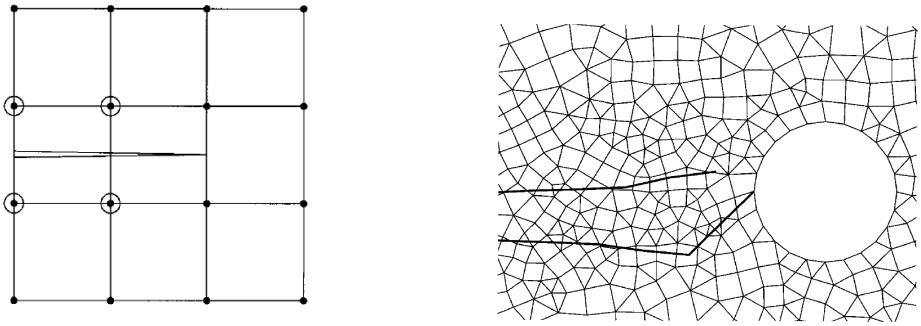
Here, u_{reg} applies to all the nodes in the model and describes the regular displacement, u_{dis} describes the discontinuous displacement at the location of the discontinuity and u_{sing} describes the displacement as a result of the stress singularity at the crack tip. The mathematical partition of unity concept [5] ensures a smooth transition between the enrichment functions, which are only locally applied, and the standard finite element basis functions that are applied over the entire domain [42].

During the process of numerical integration, both the standard functions and the enrichment functions are considered. This way, at the location of the crack, an extra accurate description of the displacement field is given, while at other locations, the normal basic functions are considered.

1.2.3.2.2 Phantom node method

The technique mentioned above leads to the advantage that the crack does not have to be located along an element, as is visible in figure 7. With the enriched nodes, the discontinuities within an element can be captured. An often-used addition that increases the applicability of XFEM even further is the phantom node method, as shown in figure 8. Phantom nodes are placed on top of real nodes and are fully constrained to this node when the element is intact. When a crack is initiated in the element and separation occurs, the phantom nodes are 'untied' of the real nodes and can move apart. The element now behaves as being cut into two parts, where each part is bounded by two real nodes and two phantom nodes. The separation between the two parts is now controlled by the TSL, in the same way as was the case for the CZM. When the maximum displacement is reached, and failure occurs, the two parts of the element can move independently from each other [1].

Due to the features mentioned here, the XFEM method is an attractive engineering approach that is very practical. It exhibits less mesh dependency, and cracks can be modelled without the use of special cohesive elements. For these reasons, the remainder of this thesis is focused on using TSLs within the context of XFEM.



(a) (b)

Figure 7: Visualisation of a crack modelled with XFEM. (a) The crack does not have to be aligned with the mesh. The encircled nodes are enriched. (b) XFEM crack resulting from a stress concentration near a hole [30].

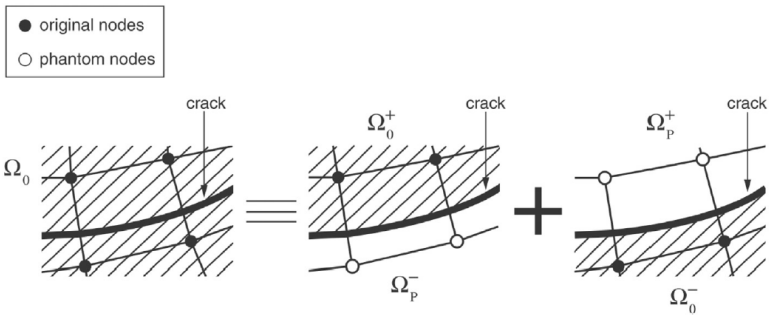


Figure 8: Working principle of the phantom node method [1]

1.2.4 Traction separation laws

As already described, traction separation laws define the constitutive response of the crack region after the onset of necking by relating the traction (T) to the separation (δ). By defining the material degradation resulting from fracture damage that the elements cannot capture, it is possible to cover a wide range of materials, geometries and fracture mechanisms. However, it should be noted that a single TSL that can be used for all applications does not exist. This section aims to explain the nature of a TSL, the different variations of shapes and how the parameters of a TSL are usually obtained.

1.2.4.1 TSL models

In figure 6, a TSL was shown that applies to the example of the metal sheet described in chapter 1.1. The area under the TSL is the total work per unit area that is required to cause total separation after the onset of necking [38]. This entity is also expressed as cohesive energy Γ_0. From figure 6, it can be observed that the total cohesive energy consists of the energy dissipated in the necking process (Γ_I) and the energy dissipated after the onset of shear localisation (Γ_{II}). It can be seen that Γ_I is much greater than Γ_{II}. This behaviour is observed in other tough ductile alloys as well [38].

Other important parameters are the failure displacement δ_0 and the cohesive stress σ_{y0}. In a TSL, δ_0 corresponds to the point at which the traction reaches zero and crack

opening occurs. The cohesive stress is the maximum stress reached in a TSL [49].

In order to implement a TSL into a numerical solver, the curve as shown in figure 6 must be simplified, and a choice has to be made of what function or shape can best be used. Figure 9 gives an overview of several shapes that were proposed in the literature. The main considerations here were the effects that a specific shape would have on the resulting fracture behaviour. However, Tvergaard and Hutchinson [55] concluded that this effect is not significant.

Figure 9c shows a TSL that describes the damage evolution typically associated with concrete. The TSL is approximated by a relatively simple linearly decreasing function that represents linear softening. This shape is representative of brittle fracture damage. Ductile materials often have a TSL shape as described by the remaining models in figure 9. The functions can be of higher order, showing a smooth response (figure 9a, 9b and 9e) or be described by a bilinear or trilinear shape (figure 9d and 9f).

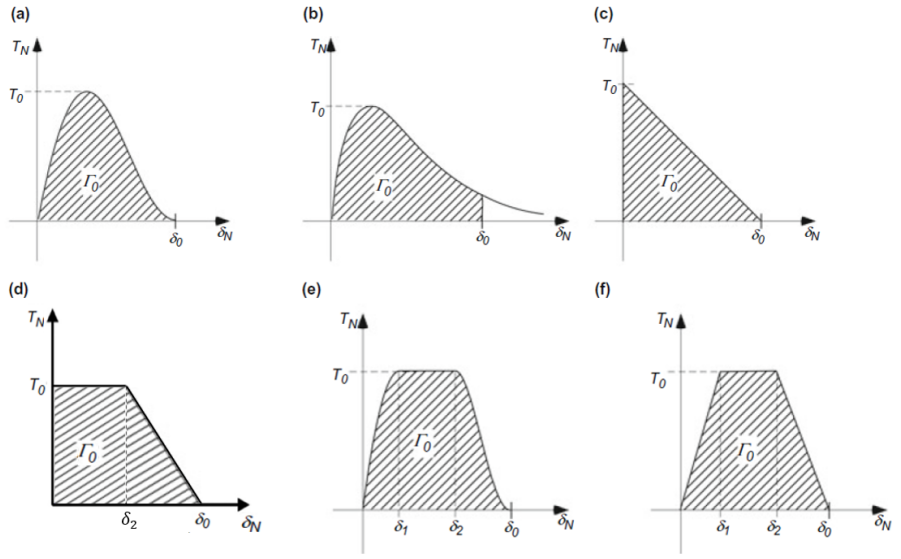


Figure 9: Different shapes of TSLs used in literature: (a) Needleman [34] (b) Needleman [32] (c) Hillerborg [21] (d) Woelke [38] (e) Scheider [45] (f) Tvergaard and Hutchinson [55]. In this figure, the cohesive stress σ_{y0} is written as T_0. Original figure taken from [14].

It also becomes clear from figure 9 that some TSLs have a finite initial slope before reaching the cohesive stress, and others do not. Having a finite initial slope means that small stresses already cause separation in the material, while for the TSLs with an 'infinite' slope, the separation starts at the moment the cohesive strength is reached (as is the case for the TSLs in figure 9c and 9d). Separation that occurs before reaching the cohesive strength reduces the overall stiffness of the material and thus affects the macroscopic response [53]. It is, therefore, advised to use a TSL with no initial slope or a very high initial stiffness. In CZM, a high initial slope can cause convergence issues [16]; however, in XFEM, no initial slope is needed, and this convergence problem is omitted.

As mentioned earlier, the shape of the TSL can be regarded as of secondary im-

portance [55]. In a study performed by Scheider and Brocks [48], it was found that different TSL shapes can produce similar output with different cohesive parameters. While it is unclear if this is generally true, it does indicate the relative insignificance of the TSL shape. It must be noted here that the cohesive parameters do still strongly depend on the chosen TSL shape [47]. It is for this reason recommended to first assume a TSL shape and then determine the cohesive parameters based on that shape. The most important factor when considering a TSL shape is that it roughly matches the softening behaviour that is expected. In the current example, this would mean that it matches the global features seen in figure 6.

When a TSL shape is known, the cohesive energy Γ_0, failure displacement δ_0, and cohesive stress σ_{y0} can be determined. These parameters are also known as cohesive parameters. From these three parameters, only two are independent and needed to fully define the TSL. In applications found in literature, often the cohesive energy and cohesive stress are used [49][44].

1.2.4.2 Determination of cohesive parameters

In order to determine the cohesive parameters, it is important to note that these parameters are dependent on material properties [38][36], but also on mesh size and stress state [50]. This means that there exists no universally applicable set of parameters for all applications, but these parameters need to be determined based on the specific application at hand.

In this section, different approaches for determining the cohesive parameters are highlighted. As stated earlier, in literature, the focus is mostly on determining the cohesive strength σ_{y0} and cohesive energy Γ_0. The focus of this section is, therefore, also on these parameters.

The cohesive parameters can generally be determined by either direct procedures or numerical fitting procedures. A final remark is that ductile fracture can occur in different failure modes, and the cohesive parameters will be different for each mode. To limit the scope of this work, this study only focuses on mode I tearing.

1.2.4.2.1 Direct procedures

One way of determining the cohesive strength σ_{y0} is with the help of a tensile test. When a notched specimen is studied in a tensile test, the ultimate stress before failure can be regarded as the cohesive strength [49].

A direct method of obtaining the cohesive energy is to set Γ_0 equal to the J-integral J_i at crack initiation, obtained in a fracture mechanics test [49]. The process of obtaining J_i is described in the international standard ISO IS 12135 [56]. No further effort is made to describe this process here.

Although direct methods are a convenient way of determining the cohesive parameters *a priori*, it is questionable whether the results are accurate enough to match experiments. For this reason, it is common practice to use one of the fitting approaches described below. Direct procedures are useful, however, to determine suitable starting values for fitting.

1.2.4.2.2 Fitting to experimental data

A second approach for obtaining the cohesive parameters is by fitting these parameters to test results in a numerical optimization procedure (so *a posteriori*) [49]. The combination of σ_{y0} and Γ_0 is chosen such that the error between the simulations and test

results is minimized. To obtain accurate results, the benchmark experiments should be performed on a test piece with similar constraints and state of stress as are expected to occur in the structure that is to be assessed [48].

Different algorithms can be employed for this optimisation, such as a trial and error approach, optimisation by error minimisation, and neural networks. More information about these processes is given in [49]. This approach of fitting to experimental data is more common than the direct procedures of obtaining σ_{y0} and Γ_0.

1.2.4.2.3 Fitting to numerical data

Lastly, the cohesive parameters could also be obtained by using a numerical method in which the fracture process itself is modelled [36]. As ductile metals fail by void growth and coalescence, the TSLs can be derived by micromechanical analysis or by using the GTN method mentioned in section 1.2.2.2 [50]. In this approach, the micromechanical model is used to capture the effect of void nucleation, growth, and coalescence to eventually obtain the macroscopic response of the specimen. This macroscopic response can then be used as a benchmark case from which the cohesive parameters can be fitted [53]. Several publications [55][46][3][2][54] describe different variations of this approach.

Much research has been done on this topic, and it seems to be a good approach to determining the cohesive parameters *a priori* without the need for experiments. However, making a micromechanical FE model is not a straightforward task because the constraint conditions of the specimen are not known in advance [50], and, as stated before, the cohesive parameters largely depend on these constraint conditions. For this reason, the most frequently applied way to determine the parameters is by fitting them to experimental results.

1.2.4.3 Applications and research on TSLs

The working principle of modelling failure with the use of a TSL can be applied to a broad range of structural integrity problems [49]. The method is already widely used for modelling crack extensions and crack paths, but also for modelling failure in interfaces such as tearing of welded joints and delamination of coatings and fibre-reinforced materials.

An advantage of the method is that damage-free material, i.e. material without a pre-existing crack, can be modelled as well. Yet, one of the most frequently used applications of TSLs is the simulation of crack propagation, where the instability of a component and the relation between applied load and cracked deformation are investigated.

Various studies have been done to enhance the capabilities of TSLs even further, allowing them to be more widely applicable or to be more physically accurate [53]. Examples are studies to the modelling of fatigue [35][11], dynamic fracture behaviour and rate dependency [59][57], temperature-dependent fracture [13] and fracture dependent on the stress state [38]. The last named subject will be described in more detail in the following chapter and the remainder of this thesis.

1.3 Stress state

In the preceding chapter, using a TSL in combination with XFEM was found to be a suitable method of modelling cracks. A common way of modelling fracture with this method is to apply a single TSL that does not depend on the state of stress in the material [49][50].

However, both numerical and experimental studies have shown that the state of stress in the crack region varies significantly along the length of the crack [39][15][26][18] [17]. In section 1.1.3, it was mentioned that the state of stress, or stress triaxiality, has a significant influence on the damage behaviour [49], and for this reason, multiple damage models already account for the triaxiality in determining damage (as described in section 1.2.2.2).

A resulting conclusion is that the TSLs must also be made stress-state-dependent, as a stress invariant TSL is expected to be insufficient to account for the material behaviour over the entire crack length [50][10]. In this chapter, the stress variations within a crack are investigated, as well as the influence that this stress state has on a TSL.

1.3.1 Influence of stress state on TSLs

Section 1.1.3 described the influence of stress triaxiality on the type of failure and failure strain. It was found that a more ductile type of failure, paired with higher failure strains, is observed for lower values of triaxiality. The reason for this is the effect that the triaxiality has on void growth and the movement of line defects in the microstructure of steels.

Because of this, the TSL is expected to change as well for different stress triaxialities. Figure 10 depicts the results of calculations done by [50]. The figure shows TSL shapes for various triaxiality values of a unit cell under uniaxial straining, obtained by using the GTN constitutive model (as described in section 1.2.4.2.3) [33]. It is visible that the GTN model predicts a clear trend, namely an increase in cohesive strength and a decrease in cohesive energy with increasing stress triaxiality.

These trends are also presented in figure 11. It is noted that in order to make use of stress-state-dependent TSLs, the exact relationships between the cohesive parameters and triaxiality must be known. However, this is complicated, as the cohesive parameters are not only dependent on the stress triaxiality but also on mesh size and material (as mentioned in section 1.2.4.2).

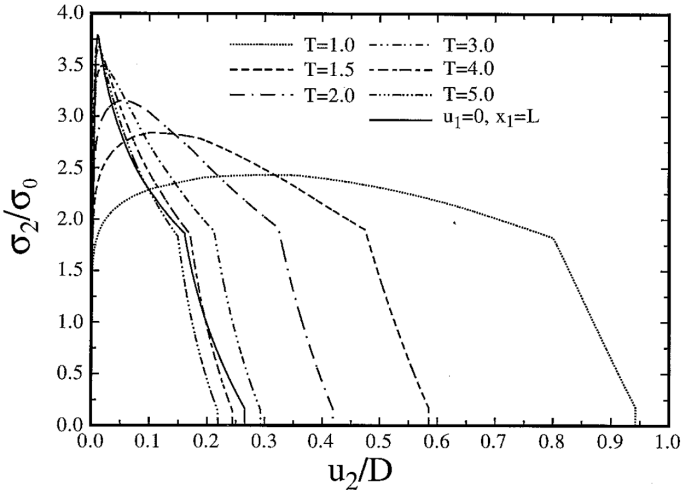
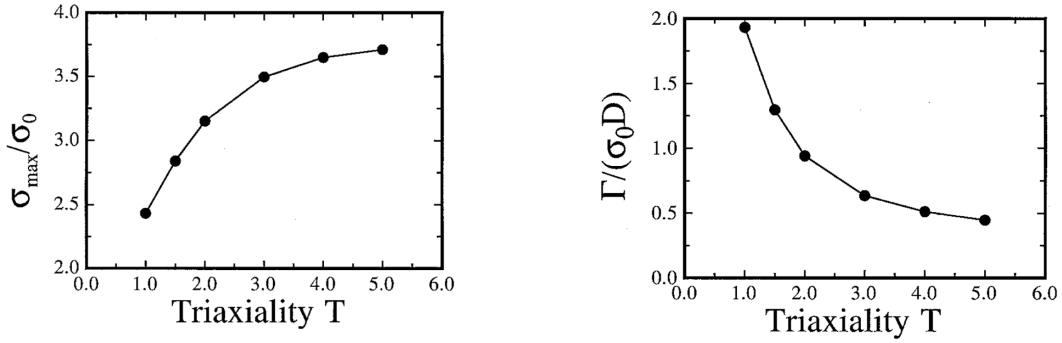


Figure 10: Traction-separation curves of a unit cell of DxD, obeying the Gurson model for varying stress triaxiality ratios [50]. Here, σ_2/σ_0 is the normalized applied stress, and u_2/D is the normalized displacement.



(a) (b)

Figure 11: Relationships between the cohesive strength (here $\sigma_{\max}$) and triaxiality (a) and cohesive energy (here Γ) and triaxiality (b) [50].

1.3.2 Stress state over crack length

In the previous section, it was concluded that the cohesive parameters are influenced heavily by the state of stress, expressed by the stress triaxiality. The current section highlights that the stress state that occurs in the specimen can vary significantly along the length of a crack. Here, it is noted that, as the current study is mainly about the stress variations over the crack length of thin metal plates, the variations in thickness directions are disregarded, and the main focus is on plane stress conditions.

Figure 12 illustrates the behaviour of the stress triaxiality as a function of the normalised distance from the crack tip, r, for a plane stress case [39]. It becomes clear that the triaxiality varies significantly over the crack length as a result of the changing boundary conditions. High values of triaxiality prevail at locations around the initial crack tip or the crack process zone. When moving away from the crack tip, the triaxiality value converts to a far-field value, also known as the HRR (Hutchinson-Rice-

Rosengren) solution. This trend is verified by results of FE simulations [15][26][18][17].

A result of the high level of stress triaxiality at the crack tip is the occurrence of less ductile failure combined with higher maximum traction in these regions, as was concluded in section 1.1.3. When moving away from the initial crack tip location, a more ductile material behaviour and higher energy are expected. This again comes with a slightly lower maximum traction [7].

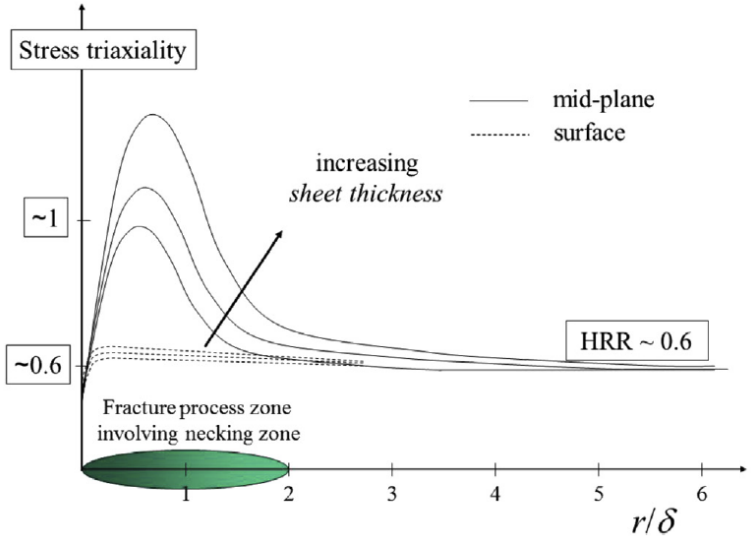


Figure 12: Variation of triaxiality over the distance from the crack tip r, normalised by opening displacement δ for a plane stress case [39].

1.3.3 Stress-state-dependent TSLs

The preceding sections explain that the stress state significantly influences the cohesive parameters and that the stress state itself varies substantially over the length of a ductile crack, showing peak values at the location of the crack tip and crack process zone. The conclusion can thus be drawn that cohesive parameters that are regarded as constants will not be able to fully capture the behaviour of a propagating crack, and a model that can incorporate stress-state-dependent TSLs is needed to increase accuracy and potentially match experiments.

A study by Woelke [38] already took a step in this direction. In this research, an advancing crack was modelled by employing multiple TSLs along the crack length rather than a constant one. This proves that multiple TSLs are needed to capture the changing conditions in a crack, and experimental results can be matched. However, the cohesive parameters of these TSLs were determined by fitting them to an experiment [51], and no effort was made to make the TSLs stress-state-dependent. Therefore, the need for stress-state-dependent TSLs remains.

1.4 Summary

In this literature review, it was found that the stress state is of great importance in fracture behaviour. In classical fracture mechanics, varying stress states are the main reason for the so-called 'transferability problem', which means that standard tests are not always able to predict the fracture behaviour of large structures. This issue can be avoided by using numerical (FE) models.

An appropriate numerical model is the Extended Finite Element Method (XFEM), which, combined with a TSL, could serve as a practical method of simulating crack propagation in ductile metals. Because the TSL, which describes the failure mechanism, can be tuned based on different conditions, it is now possible to model a larger range of structures and load cases than classical fracture mechanics can.

Combining XFEM with a TSL also allows for the use of larger elements because a TSL describes the failure behaviour that larger elements cannot capture. Other methods, such as the element deletion method, require a very fine mesh, which detracts from applicability and prevents its use on larger structures.

A weakness of the XFEM method is that even though a TSL can account for specific constraints and boundary conditions in a structure, these conditions are often taken as constant over the length of a crack. Common practice is to employ a single TSL to describe failure, and stress variations along the length of a crack are ignored.

In reality, the stress state varies significantly over the crack length, and a model is needed to account for these variations. The stress state influences a TSL, and thus, multiple stress-state-dependent TSLs need to be used over a crack path to produce physically feasible results. For this, the relationship between the cohesive parameters and stress triaxiality must be known.

Woelke [38] used multiple TSLs along a crack path and proved that it is indeed possible to match experiments by accounting for the changing conditions in a crack. However, the cohesive parameters were chosen based on a 'trial and error' based optimization using test results, and are thus not based on the stress state.

1.5 Research overview

This section presents the formulated research gap, objective, and research questions that follow from the literature review.

1.5.1 Research gap

Following from the literature review, the research gap is defined as follows:

The currently available XFEM approach, which uses TSLs to account for the failure process, is not able to account for the change of stress state along the length of a ductile crack in plate structures.

1.5.2 Objective

The research goal is to incorporate stress-state-dependent TSLs within an XFEM crack propagation simulation, thereby obtaining more accurate and physically feasible results.

1.5.3 Research questions

The main research question that follows from the research gap and objective is formulated as follows:

Can the accuracy of crack propagation simulations in a commercial FEA solver be improved by employing stress-state-dependent TSLs?

To answer the main research question, the following sub-questions are formulated:

1. How can a TSL be implemented in fracture simulations in XFEM?
2. How can a subroutine be defined in Abaqus that generates a TSL based on stress triaxiality?
3. Can relationships be defined between the cohesive parameters and triaxiality that result in a match with experimental data?

1.5.4 Scope

This research focuses on mode-I cracks occurring in a uniaxially-loaded thin plate. A plane stress shell model is used with a focus on in-plane stress variations. Variations of the stress state in thickness direction are not regarded. The simulations will be done in the commercial FE package Abaqus with the XFEM feature.

1.5.5 Methodology and support

This project is under the joint supervision of Femto Engineering and TU Delft. Both Femto Engineering and TU Delft can provide advice on how to interact with the Abaqus software. TU Delft can also advise on current research and theory on TSLs. Furthermore, the company 4RealSim can provide support on the use of subroutines.

Chapter 2

TSL implementation

The first stage of this study is developing a tool that is able to implement stress-state-dependent TSLs in Abaqus. The main stages of this process are first implementing a single, stress-state invariant TSL in Abaqus and thereafter making the TSL triaxiality-dependent with the use of a subroutine. The following chapters describe the development process of this 'TSL tool'.

2.1 Implementation of a stress-state invariant TSL

This section describes the process of implementing a TSL within Abaqus without aiming for stress state dependency. The literature review revealed that modelling in XFEM is preferred over other methods, as this allows for the use of relatively large elements. These guidelines were followed when creating the FE model. All of the simulations were done in the implicit solver in Abaqus 2023 (see also section 5.3).

2.1.1 Model description

To explore the possibilities within Abaqus, a test model of a precracked plate that will be loaded in tension is set up, as shown in figure 13. The model setup is chosen arbitrarily, with the mere goal of tuning and exploring the possibilities within Abaqus. In part 3, an actual experiment is modelled and a different FE model is used.

The dimensions of the plate are 800x600x5 mm, and the precrack is 150 mm in length with a diameter of 5 mm. Elements of the type CPS4R are used, which are 4-node bilinear plane stress quadrilateral elements of reduced integration and enhanced hourglass control. This element type was chosen because it was discovered that the XFEM functionality in Abaqus only supports the use of three-dimensional solid and two-dimensional planar elements. This restriction is also mentioned in section 5.3. The enhanced hourglass control is recommended as, in some cases, hourglassing was observed in the model.

The mesh size used is 20 mm. At the location of the crack opening, the mesh gradually transitions from 5 mm at the initial precrack to 20 mm, and this transition is 20 mm long.

The model is clamped at the bottom. At the top surface, movement in the x-direction is constrained, and a displacement is applied in the upward Y-direction.

The material used in this setup has a yield strength of 350 MPa. The Young's modulus is 210 GPa.

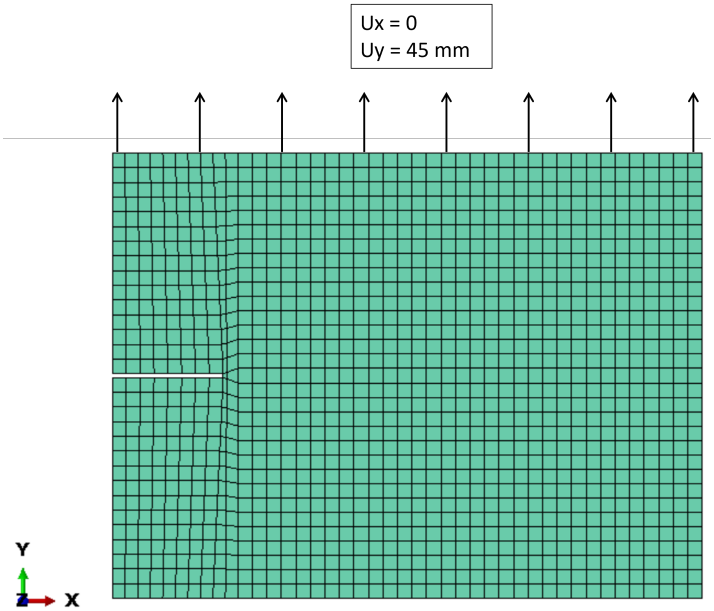


Figure 13: Model setup

2.1.2 Modelling damage

In order to accurately model the behaviour of a material when failing, multiple damage initiation criteria and damage evolution models can be defined within Abaqus. When a damage initiation criterion is met, damage begins to occur in the material. Subsequently, the damage evolution model determines how this damage evolves and when total failure will occur. It is thus the combination of the damage initiation criterion and the damage evolution model that fully determines failure.

2.1.2.1 Damage initiation

Section 1.2.2.2 already gave a short overview of different damage initiation criteria that exist. The damage initiation criteria that can be used in combination with XFEM in Abaqus are maximum principal stress or strain damage (Maxps Damage and Maxpe Damage). These initiation criteria base the point of damage initiation only on either the maximum principal stress or maximum principal strain. In the current setup, it is chosen to employ the Maxps criterion, as this corresponds to the maximum traction, or cohesive stress σ_{y0} of a TSL.

2.1.2.1.1 Local vs. nonlocal calculation of stress/strain

To accurately assess the crack initiation criterion and determine the direction of crack propagation in XFEM, it is crucial to effectively evaluate the stress and strain fields ahead of the crack tip location. Abaqus/Standard provides multiple methods for calculating these fields. The distinction is made between local and nonlocal methods.

Local methods assess the stress and strain fields within the individual element directly ahead of the crack tip. An example of this is the centroid method, which is

the default option in Abaqus. In this method, the stress and strain fields are calculated at the centroid of the element, as illustrated in figure 14. Both the crack initiation criterion and the crack propagation direction are evaluated based on the centroidal stress/strain [1].

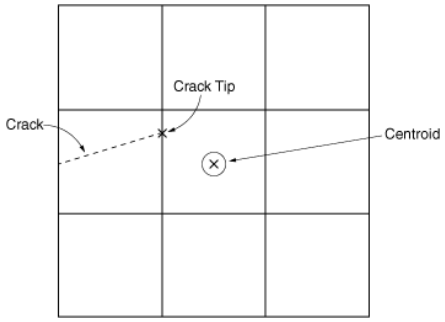


Figure 14: Centroid and crack tip locations that are used by local methods [1].

When the mesh is sufficiently refined, the centroid method proves to be both accurate and effective. However, if the mesh around the crack tip is relatively large compared to the stress/strain field gradients, the centroid method might not be sufficient. In these cases, the crack tip method can be employed. In this method, the stress/strain extrapolated to the crack tip can be used to verify whether the damage initiation criterion is met and to determine the direction of crack propagation.

A third local option would be to combine the crack tip and centroidal calculations. In this combined option, the stress/strain values extrapolated to the crack tip determine whether the damage initiation criterion is met, and the stress/strain values at the element centroid are used to calculate the crack propagation direction [1].

The three local methods for assessing the stress and strain fields mentioned earlier focus on the fields within a single element ahead of the crack tip. In some cases, however, a nonlocal approach is preferred. In the nonlocal method, the crack initiation is based on the extrapolated value of elemental stress/strain to the crack tip location, while the crack propagation direction is determined by averaging the stress of a group of elements around the crack tip. For coarse or unstructured meshes, employing nonlocal averaging of the stress and strain fields ahead of the crack tip can yield a more precise evaluation of these fields, thereby improving the accuracy of the calculated crack propagation directions.

Additionally, by default, a moving least-squared approximation by polynomials is employed to further increase the accuracy of the crack propagation direction. Instead of fitting a single polynomial to the entire data set of stress and strain data around the crack tip, the moving least-squares approximation fits polynomials to small subsets of the data points. The local polynomials are then used to approximate the values of stress and strain at any point in the vicinity of the crack tip. This creates a smooth, continuous approximation of the stress and strain fields.

Figure 15 shows the working principle of the nonlocal approach. The range of elements that are used for nonlocal averaging can be controlled by specifying the radius $\boldsymbol{r_c}$. The elements within this radius are included in the averaging [1].

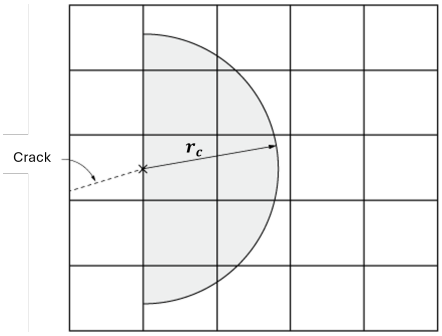


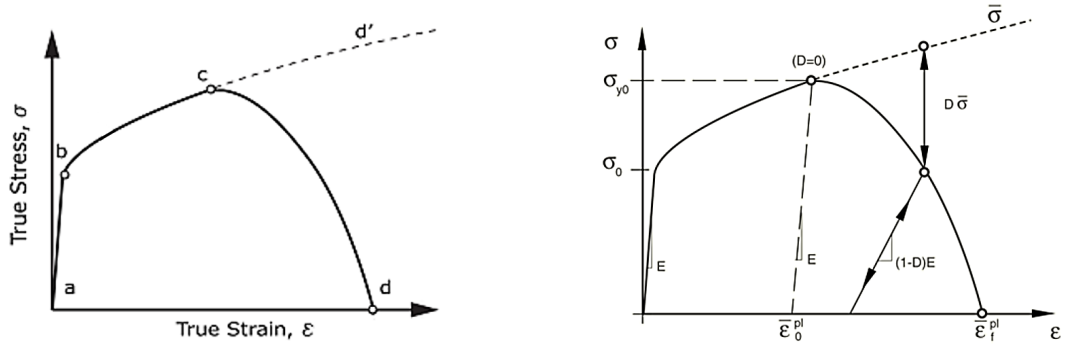
Figure 15: Nonlocal averaging region [1].

Because the nonlocal method is more suitable for larger elements, and a more accurate prediction of the crack propagation direction is made, nonlocal averaging is used within this study. $\boldsymbol{r_c}$ is set to be the default value of three times the characteristic element length.

2.1.2.2 Damage evolution

As stated earlier, the damage evolution model defines how a material degrades when damage has been initiated. It is assumed here that damage is characterized by the progressive degradation of the material stiffness, eventually leading to material failure. It is recognized that a damage evolution law serves the exact same purpose as a traction separation law and can thus be used to implement a TSL within Abaqus. For this, the working principle of a damage evolution law is explained first.

Figure 16a shows the true stress-strain response of a specimen where failure has occurred. From this figure, the initially linear elastic and subsequent plastic material behaviour can be observed in the regions a-b and b-c, respectively. Point c in this figure corresponds to the point where damage is initiated. Beyond this point there is a marked reduction of load-carrying capacity until final rupture occurs in point d. Section c-d can be viewed as the degraded response of the curve c-d', which is the curve that represents the undamaged material behaviour [1].



(a) (b)

Figure 16: True stress-strain response at the location of failure [1].

In Abaqus, the degree of damage is expressed as the damage parameter D. At the point of damage initiation, D is equal to zero, and when failure occurs, D is equal to 1.

This means that the damaged stress tensor $\boldsymbol{\sigma}$ can be obtained by the scalar damage equation:

$$\boldsymbol{\sigma} = (1 - D)\overline{\boldsymbol{\sigma}}, \tag{3}$$

where $\overline{\boldsymbol{\sigma}}$ is the undamaged stress tensor. Figure 16b shows the stress-strain curve with the damage parameter D.

2.1.3 TSL implementation

In order to incorporate a TSL in Abaqus, damage initiation and evolution are used. As mentioned previously, the initiation stress corresponds to the cohesive strength of a TSL, and the damage evolution is used to obtain the corresponding TSL shape. Damage evolution within Abaqus is expressed in terms of the damage parameter D, so in order to implement the TSL, the TSL curve that relates traction to displacement must be converted to a damage curve that relates damage to displacement. This is done by rewriting equation 3 as follows [23]:

$$D = \begin{cases} 0 & \text{for} \quad 0 \leq \delta \leq \delta_1 \\ 1 - \frac{\sigma_{y0}}{\overline{\sigma}} & \text{for} \quad \delta_1 \leq \delta \leq \delta_2 \\ 1 - \frac{\sigma_{y0}}{\overline{\sigma}} \frac{\delta_0 - \delta}{\delta_0 - \delta_2} & \text{for} \quad \delta_2 \leq \delta \leq \delta_0 \\ 1 & \text{for} \quad \delta \geq \delta_0 \end{cases} \tag{4}$$

Here, δ_1 corresponds to the displacement at damage initiation. δ_2 and δ_0 correspond to the displacement at which shear banding begins and the displacement at total failure, respectively. This is also shown in figure 17a. σ_{y0} represents the damage initiation stress. In Abaqus, the damage curve is inserted via a table with values for D as a function of $\delta - \delta_1$. This means the material behaviour before damage initiation is disregarded, as shown in figure 17b.

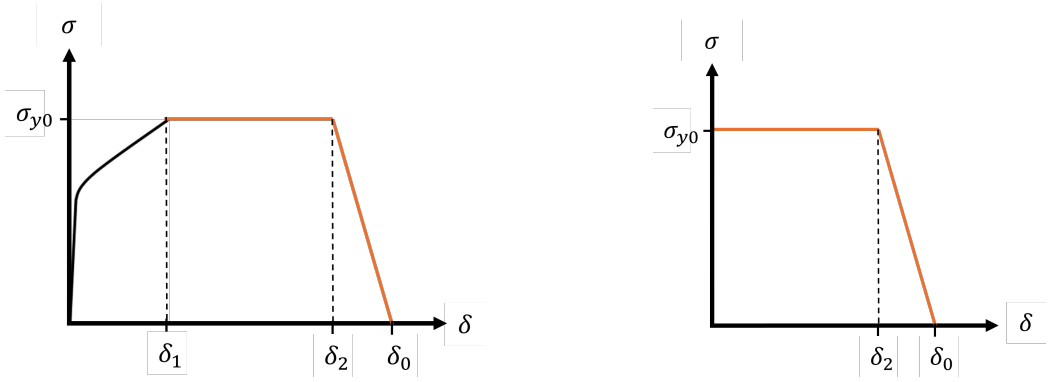


Figure 17: Typical stress-displacement plot of an element that undergoes fracture (a) and the separate TSL (b).

From equation 4, it becomes clear that in order to calculate D, the stress without damage, $\overline{\sigma}$, must be calculated. It was found that in Abaqus, this value is calculated (for 2D elements) by

$$\overline{\sigma} = K\delta, \tag{5}$$

where K, from now on indicated as the TSL stiffness, is denoted as $K = \frac{\sigma_{y0}}{\delta_1}$. It turns out that this TSL stiffness is, therefore, of significant importance for the correct calculation of the damage curve.

The TSL stiffness is implemented in Abaqus by creating 2 contact surfaces at the location of the crack, as shown in figure 20 and described in appendix A. When separation occurs, these contact surfaces are moved away from each other in the normal direction, and a contact stiffness now defines $\overline{\sigma}$. This contact stiffness is defined within the contact properties section in Abaqus and coupled to the crack surfaces. Figure 18 shows a close-up of a crack where the contact surface is visible, as the element boundary and contact surface do not fully overlap.

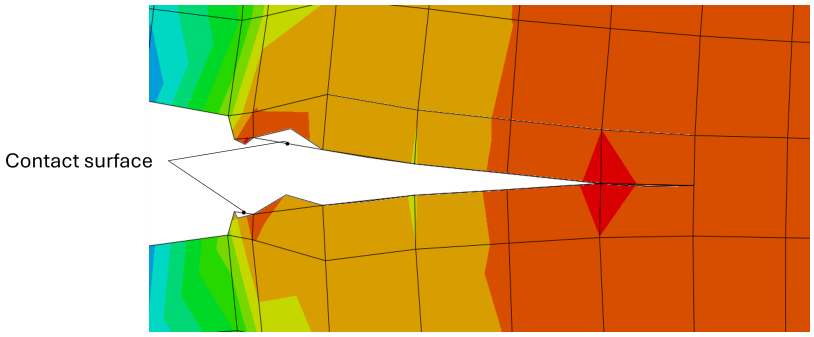


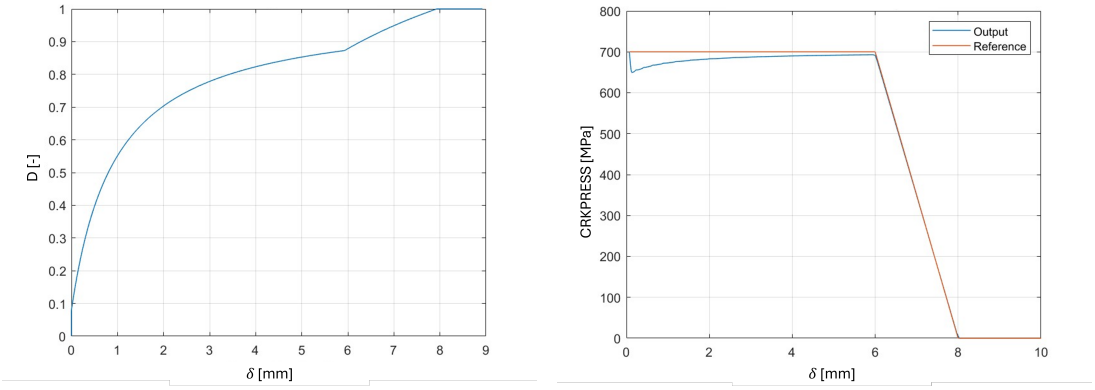
Figure 18: Contact surface within XFEM crack.

2.1.4 Resulting TSL

When the TSL parameters (σ_{y0}, δ_2 and δ_0) are known, the damage curve can be determined with the use of equation 4.

Figure 19b shows the TSL obtained in Abaqus as a result of employing the damage curve from figure 19a. The TSL curve is obtained by plotting the contact pressure (CRKPRESS), which is the pressure between two contact surfaces within a single element, to the crack opening displacement (CRKOPEN) between these contact surfaces (see figure 20 and appendix A).

The figure also shows the reference TSL. It is evident that the generated TSL matches the reference TSL quite closely. A small deviation exists in the obtained TSL, in the form of a stress drop after the moment of initiation. After this drop, the stress builds up again towards the wanted value of 700 MPa. Effort was made within this thesis to find the reason for this effect, but this was unfortunately unsuccessful.



(a) (b)

Figure 19: Damage curve (a) and resulting TSL (b).

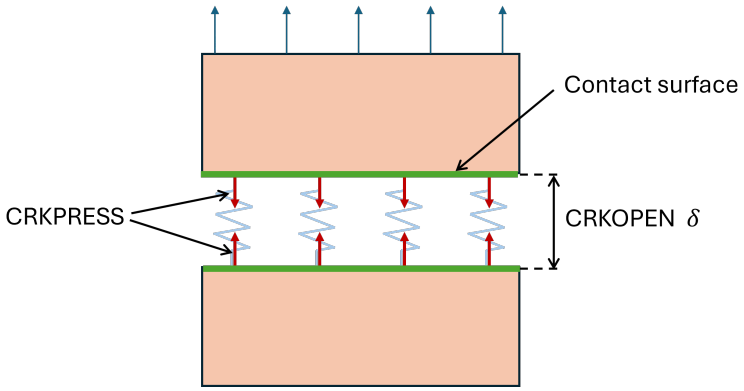


Figure 20: When a crack propagates through an element, two contact surfaces are defined, and the CRKPRESS and CRKOPEN output values follow the TSL.

In figure 21, the elemental behaviour after the implementation of the TSL is shown. In this figure, a clear distinction can be made between the linear and plastic material behaviour and the behaviour of the element after damage has been initiated. Damage initiates at a maximum principle stress of 700 MPa, as indicated by a red dot. After this, the TSL behaviour is followed.

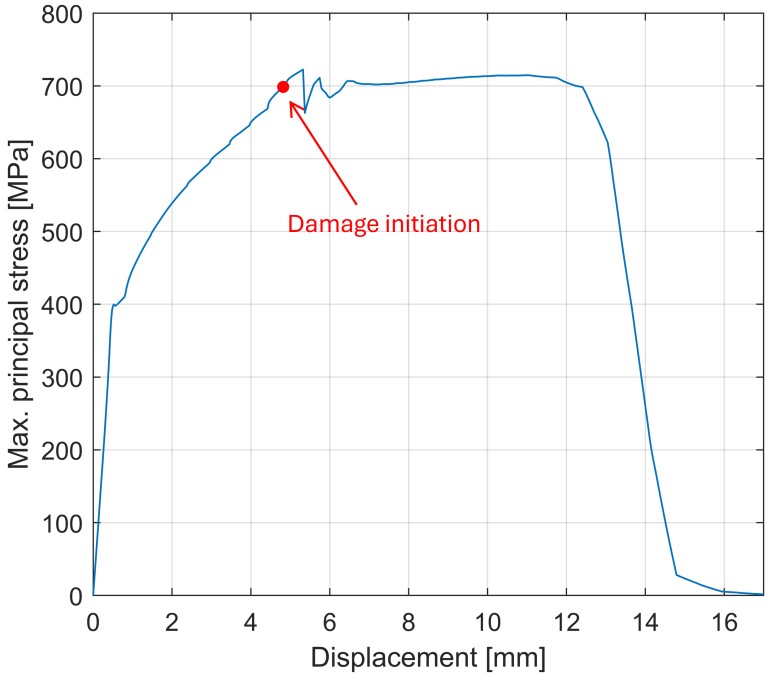


Figure 21: Stress-displacement output of an element on the crack path.

2.2 Implementation of stress-state-dependent TSLs

In the previous section, a single, stress-state invariant TSL was implemented in Abaqus. The current section describes the process of making the TSL triaxiality-dependent. First, the variance in triaxiality over the length of the crack is described, after which a subroutine is employed to generate a TSL for each degree of triaxiality.

2.2.1 Triaxiality

As described in section 1.3.2, the degree of triaxiality changes along the crack length, with higher values at the location of the initial crack tip and lower values when moving away from this location. In this section, the behaviour of the stress triaxiality in the Abaqus model is reviewed.

In figure 21, a stress-displacement curve was given for an arbitrary element that undergoes fracture. Figure 22 shows the same curve along with the values of triaxiality for that specific element, as was requested as elemental output (TRIAX) within Abaqus. From this figure, it becomes evident that the triaxiality curve has a clear peak, which occurs at the moment of damage initiation. After reaching this point, the stress gets redistributed, and the triaxiality decreases. It also becomes clear that after reaching δ_2 (in the figure corresponding to a displacement of around 12 mm), the triaxiality drops at roughly the same rate as the stress.

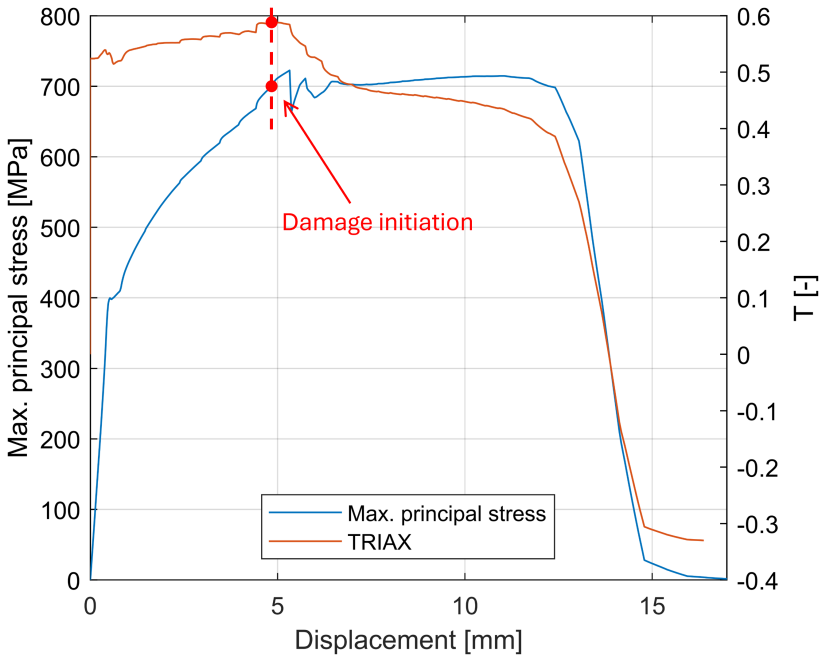


Figure 22: Triaxiality values for a single element that undergoes fracture and the max. principal stress curve for that element.

In figure 23, the triaxiality curves for 6 elements (see figure 24) along the crack length are shown. In this figure, the triaxiality peaks can clearly be distinguished and are also marked by a red dot. For these elements, the same behaviour as depicted in figure 22 is observed, i.e. the triaxiality peak of an element occurs at the moment of damage initiation of that element.

It is evident from the figure that the stress triaxiality (at the moment of damage initiation) indeed varies along the length of the crack. A peak value of 0.62 occurs at the first element, which is located close to the initial crack tip location. The last element, which is furthest away from the initial crack tip, experiences a triaxiality value of 0.36 at damage initiation. When looking back at figure 4 in section 1.1.3, it becomes clear that the triaxiality value of 0.62 falls in between the plane strain tension (T=$1/\sqrt{3}$) stress state and the biaxial tension (T=2/3) stress state. The triaxiality value of 0.36 almost coincides with the uniaxial tension (T=1/3) stress state.

It thus becomes clear that the behaviour as described in section 1.3.2, that is, a higher value of stress triaxiality at the initial crack tip location and a decreasing value when moving away from this location, is indeed observed in the FE model. The different stress states that are observed (plane strain/ biaxial tension and uniaxial tension) are due to the boundary conditions within the crack that change as the crack progresses through the plate.

It can also be seen in figure 23 that after damage initiation, the stress triaxiality drops until reaching a constant value of -0.33, corresponding to uniaxial compression. The reason for this is remaining compressive stress in the x direction (σ_1) after final separation has occurred.

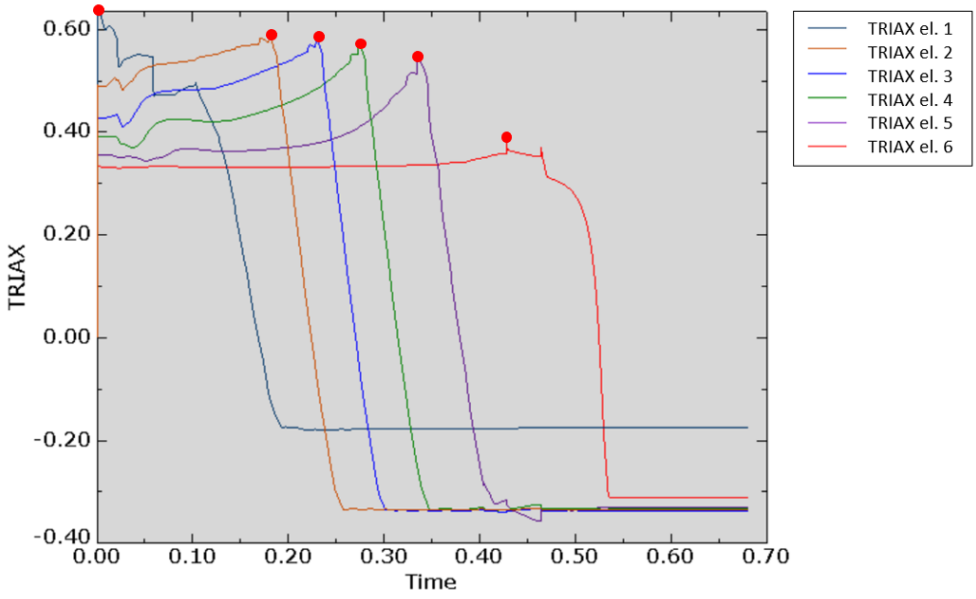


Figure 23: Triaxiality over the crack length. Each curve shows the triaxiality values of an element, see also figure 24. For each element, the moment of damage initiation is indicated by a red dot.

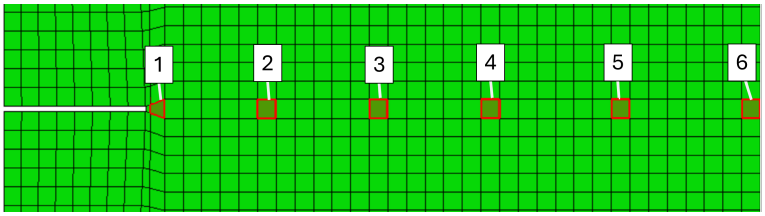


Figure 24: Selected elements along the crack path used in figure23.

2.2.2 USDFLD Subroutine

In order to vary the TSL based on the amount of triaxiality within an element, a USDFLD subroutine is employed. This section briefly describes the subroutine's working principle. The code is provided in appendix B.

The USDFLD subroutine in Abaqus allows users to make certain quantities of interest dependent on a field variable. In section 2.1, it was highlighted that a TSL is obtained within Abaqus by defining the damage initiation stress (maxps) and the damage evolution law. In the current model, the USDFLD subroutine is used to vary the damage initiation criterion and the damage evolution law based on this field variable. This is done in 2 steps, which are also shown in figure 25:

1. For every element, the triaxiality value for each iteration is retrieved. The value of interest is the triaxiality value that each element experiences at the moment of damage initiation. As described above, this corresponds to the peak value of the triaxiality. Therefore, in the subroutine, the maximum value of the triaxiality is saved for every element.

2. Based on the maximum value of triaxiality, each element is assigned a field variable.

It is chosen to define 100 field variables. The triaxiality is now related to the field variable by a factor of 100. For example, an element with a triaxiality value of 0.55 gets assigned a field variable of 55.

When every element has an assigned field variable, the damage initiation criterion and damage evolution law can be made field variable dependent within Abaqus. This is done by providing a table. In the case of the damage initiation criterion, this table consists of the max. principle stress value for every field variable. In the case of the damage evolution law, the table consists of a damage table for every field variable. These tables are generated by an additional Matlab script and implemented directly into the .inp file of the Abaqus model (see section 2.2.3).

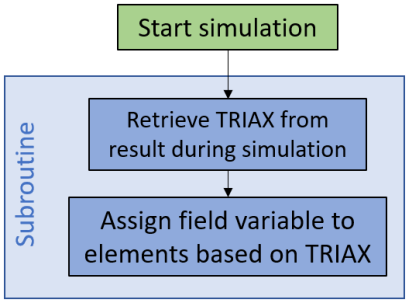


Figure 25: Working principle of the USDFLD subroutine.

2.2.3 TSL-tool

As previously mentioned, the USDFLD subroutine is employed to assign a field variable to every element. However, assigning the TSLs to these elements is done by providing damage initiation and damage evolution tables for every field variable. This

is accomplished by an additional Matlab script, provided in appendix C. The working principle of this Matlab script is shown in figure 26.

The Matlab script's input is the relationship between the TSL parameters and triaxiality. Based on this relationship, the code calculates the damage curves and creates the corresponding input tables for damage initiation and damage evolution for every field variable. These tables are formatted correctly and written to the Abaqus model's input (.inp) file so that they are incorporated into the model automatically. This way, the user needs to exert minimum effort.

The combination of the USDFLD subroutine and the additional MATLAB script forms a tool that enables the use of stress-state-dependent TSLs within Abaqus. In the remainder of this thesis, this tool is referred to as the "TSL-tool".

The results of the TSL-tool are discussed in section 2.2.5.

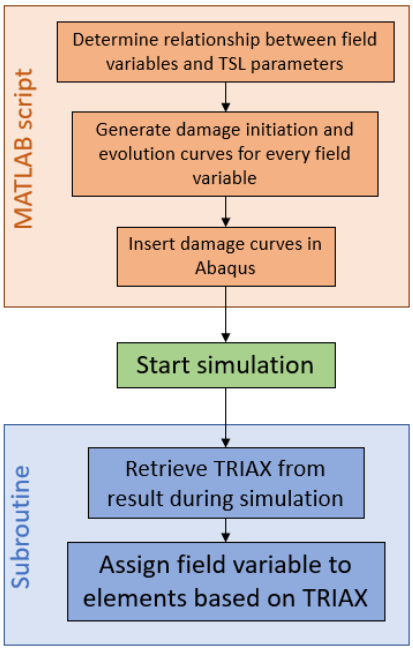


Figure 26: Working principle of the TSL-tool, consisting of a Matlab script and the USDFLD subroutine.

2.2.4 TSL stiffness

A final hurdle that needs to be overcome in order to generate stress-state-dependent TSLs is changing the TSL stiffness K. In section 2.1.3 it became clear that the TSL stiffness is of significant importance for the implementation of the TSL in Abaqus. It defines the quantity of the stress without damage, $\overline{\sigma}$, which is needed to calculate the damage curve. The TSL stiffness was implemented within Abaqus as the contact stiffness between two crack surfaces.

It is evident that when the initiation stress σ_{y0}, and thereby δ_1 of the TSL changes, the TSL stiffness must change accordingly. When this is not the case, and the TSL stiffness remains constant, changing the σ_{y0} value will yield incorrect TSL shapes.

Unfortunately, it is not possible within Abaqus to change the contact stiffness value per element with the USDFLD subroutine, as was the case for the other parameters of the damage curve. For this reason, the expression for the damage curve must be made

independent of the TSL stiffness. The goal here is to change the previously defined formulation of the damage curve so that it consists of a fixed value for the TLS stiffness (that can be defined as a constant contact property within Abaqus) and a variable TSL stiffness (that is used to properly calculate the different damage curves).

This is done by first recalling the original damage equation:

$$\sigma = (1 - D)K\delta. \tag{6}$$

Next, we can distinguish between the old damage curve (D_1), which needs a variable TSL stiffness (K_{variable}) in order to vary the TSL, as defined in section 2.1, and a new damage curve (D_2), that works with a fixed TSL stiffness (K_{fixed}):

$$\begin{aligned} \sigma_1 =& (1 - D_1)K_{\text{variable}}\delta \\ \sigma_2 =& (1 - D_2)K_{\text{fixed}}\delta. \end{aligned} \tag{7}$$

Both expressions of D must lead to the same stress, which means that we can write:

$$(1 - D_1)K_{\text{variable}}\delta = (1 - D_2)K_{\text{fixed}}\delta, \tag{8}$$

which leads to

$$D_2 = \frac{K_{\text{variable}}}{K_{\text{fixed}}}(D_1 - 1) + 1. \tag{9}$$

Here, K_{fixed} is defined as contact stiffness in Abaqus and remains fixed, D1 is calculated by equation 4, and K_{variable} is given by

$$K_{\text{variable}} = \frac{\sigma_{y0}}{\delta_1}. \tag{10}$$

The revised damage curve definition, D2, accounts for the changing σ_{y0} value and, subsequently, the changing TSL stiffness, while the contact stiffness K_{fixed} can remain constant. The calculation of D2 is incorporated in the Matlab script of the TSL-tool.

2.2.5 Results

With the TSL-tool, a TSL is generated based on triaxiality and implemented directly within Abaqus. The result is shown in figure 27. Here, the TSLs corresponding to the same 6 elements as shown in figure 24 are depicted.

It is noted that the current model assumes an arbitrary relationship between the cohesive parameters and triaxiality, as the exact relationships are not yet known. These relationships assume an increasing initiation stress and decreasing cohesive energy for increasing triaxiality, as is expected based on findings in the literature, highlighted in chapter 1.3.

From figure 27, it becomes clear that the TSLs conform to this predefined (arbitrary) relationship. Element 1, located at the initial crack tip, experiences the highest triaxiality (0.62) and is assigned a TSL with the highest initiation stress and lowest failure displacement. TSL 2 until TSL 6 correspond to elements that are further and further away from this initial crack tip location and thus experience decreasing triaxiality. For these elements, the initiation stress decreases, and the failure displacement increases, meaning a more ductile type of failure for the locations that are the furthest away from the initial crack tip.

The results presented in figure 27 show that the TSL-tool functions according to plan: it is able to assign a varying TSL over the crack length based on the state of stress of an element. However, it does become clear that an initial stress drop occurs at the moment of initiation, just as was observed in figure 19b. This stress drop occurs at the first TSL and becomes less pronounced for the subsequent TSLs. Unfortunately, the cause of this stress drop was not found, and further effort is needed to obtain a perfect trapezoidal TSL shape.

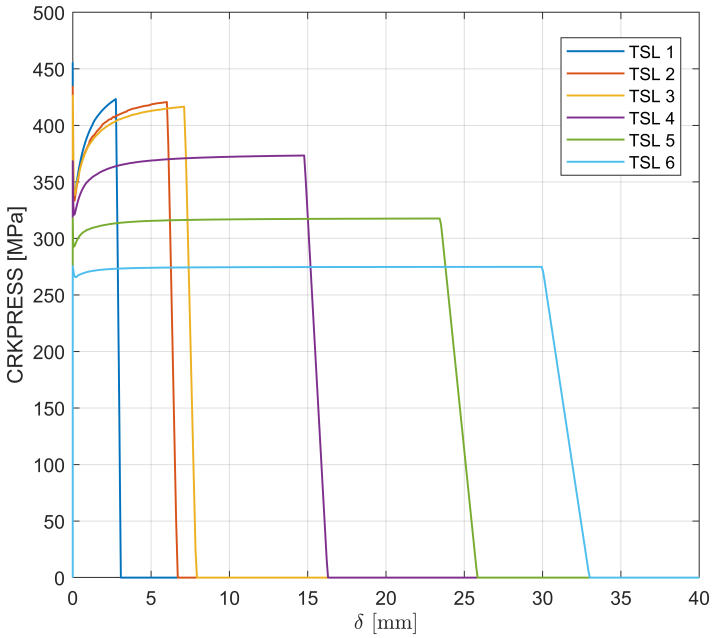


Figure 27: Generated TSLs based on the triaxiality variance in the FE model.

2.2.6 Further steps

As mentioned earlier, the most important input of the TSL-tool is the relationship between the cohesive parameters and triaxiality. This relationship determines the TSL for different stress states and is vitally important for matching experimental data. In the study described above, an arbitrary relationship is taken to test whether or not the TSL-tool works and can generate a TSL based on the stress state.

The next chapter describes the process of finding correct relationships between triaxiality and the cohesive parameters. When these relationships are known, the TSL-tool can be used to match experiments and increase the accuracy of the FE model.

Chapter 3

TSL optimization

This chapter discusses the input optimization study. Within this study, an optimization algorithm is used to find the relationship between the cohesive parameters and triaxiality, resulting in a match between the FE model and experimental data.

Within this chapter, the benchmark experiment and the optimization study are described in chapter 3.1 and 3.2. The results of the optimization study are given in chapter 4.

3.1 Benchmark experiment

3.1.1 Experimental setup

The benchmark case for the current study is an experiment executed by Simonsen and Törnqvist in 2004 [51]. This experiment consists of a Central Crack Tension (CCT) test with a large plate of normal strength (NS) steel. Figure 28 shows the details of the experimental setup.

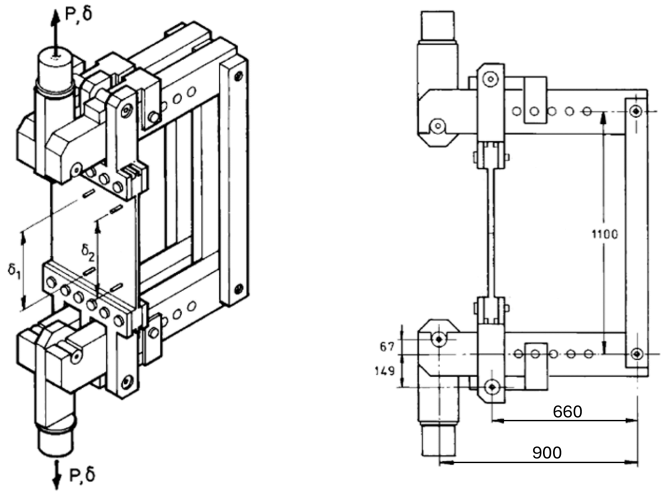


Figure 28: Experimental setup for the CCT benchmark tests [51].

In this experiment, the load, displacement and crack length elongation are measured. The dimensions of the test plate are 806x500x10 mm. The initial crack has a length of 100 mm and there was a hole at the initial crack tip with a diameter of 5 mm.

The edges of the specimens were clamped by 25-mm-thick plates, which were fixed to the specimens by bolts. These thick plates extend 50mm from the bolts into the specimen area.

The material of the plate is NS steel with a yield stress of 273 MPa and a tensile strength of 372 MPa. The stress-strain diagram is illustrated in figure 29. The failed specimen is shown in figure 30.

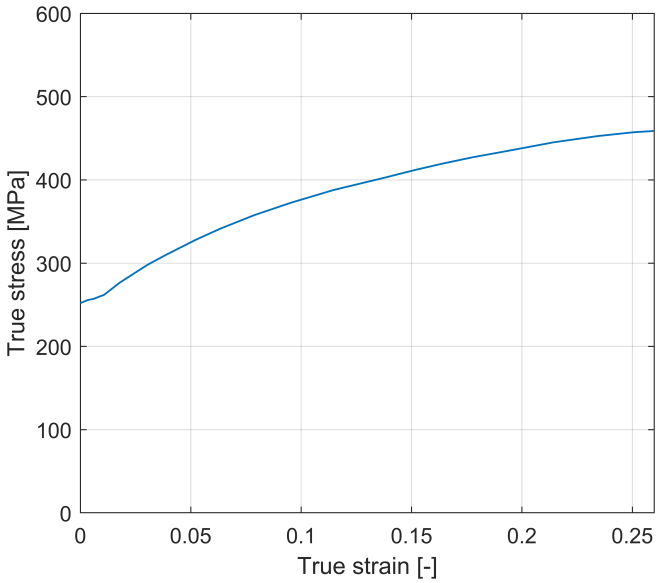


Figure 29: Stress–strain curve of the NS steel plate [51].

Figure 30: Test specimen after failure [51].

3.1.2 FE model

Figure 31 shows the FE model used to simulate the CCT experiment. The elements that are used are the same as described in section 2.1, which are CPS4R elements: 4-node bilinear plane stress quadrilaterals with reduced integration and enhanced hourglass control. The mesh size is 20 mm, and at the location of the crack tip, the mesh gradually transitions from 5 mm at the initial precrack to 20 mm. For now, this mesh size is chosen arbitrarily. The influence of the mesh size on the final results will be investigated further in section 4.3.

Because of symmetry, only half of the plate is modelled, and symmetry constraints are applied over the symmetry edge. Furthermore, the top and bottom edges are assumed to be fully clamped because of the thick bolted plates attached to them in the experiment setup. These thick clamps extend 50 mm into the specimen area. Because of simplicity, only the portion of the specimen plate that is free from these clamps is modelled. This means that the height of the plate is now 706 mm instead of 806 mm.

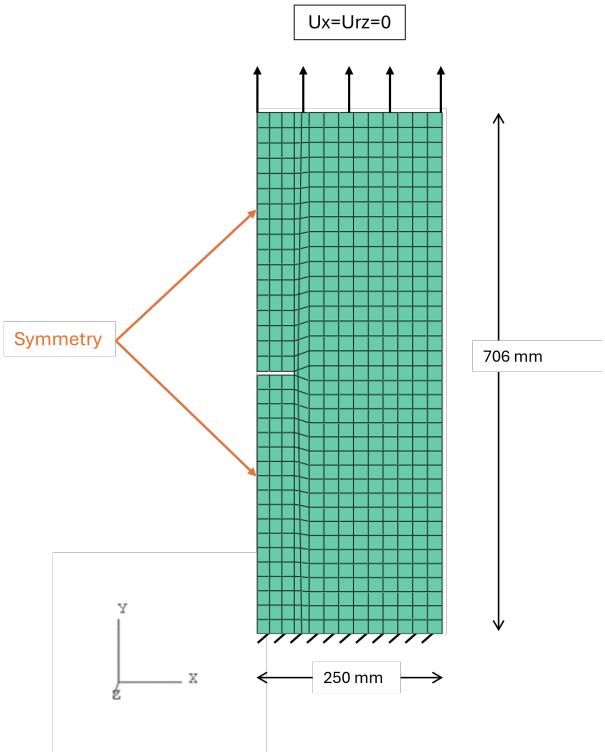


Figure 31: FE model of the CCT experiment

A final remark must be made concerning the location of the load application in the experiment. From figure 28 it becomes evident that there is a horizontal distance of 240 mm between the load application and the location of the plate. Because the vertical beam at the far left of the setup (see figure 32) prevents movement in the y direction, the plate itself only experiences a fraction of the total applied force. This fraction equals $660/900 = 0.733$.

In the FE model, the load is applied directly to the plate. Therefore, the FE results are corrected with the load correction factor of 0.733 when generating the force-displacement diagram.

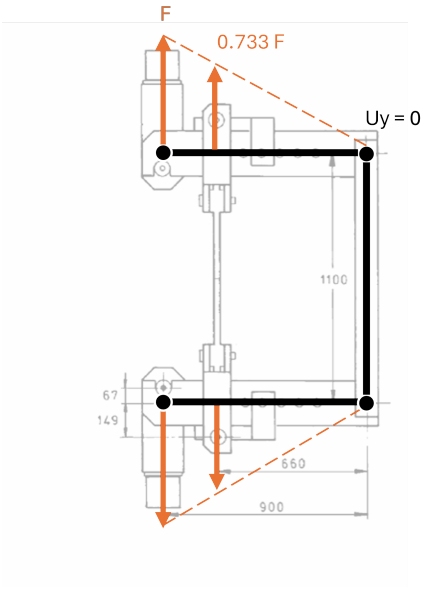


Figure 32: Due to a cantilever effect, the plate experiences only a fraction of the total applied force.

3.2 Optimization study

This chapter explains the optimization process and highlights the optimization parameters, objective function and optimization algorithm. As stated before, this optimization study aims to find relationships between the cohesive parameters and triaxiality that produce output that matches experimental data. This is done by fitting these relationships to the force-displacement and crack extension-displacement curves obtained in the above-mentioned experiment.

3.2.1 Parameters

As mentioned earlier, the exact shape of a TSL is defined by the initiation stress σ_{y0}, the displacement at which softening is initiated δ_2 and the failure displacement δ_0, as depicted in figure 17b. To reduce the number of parameters, it is assumed here that a fixed relationship exists between δ_2 and δ_0. For this relationship, a factor of 1.1 is assumed:

$$\delta_0 = 1.1 \cdot \delta_2. \tag{11}$$

This assumption is based on the fact that a realistic value for δ_0 will not be much larger than δ_2. This is because, once a shear band has formed, a rapid decrease in load carrying capacity is expected [36], resulting in the Γ_{II} value being significantly lower than Γ_I [38], as highlighted in section 1.1.2 and 1.2.4.1. On the other hand, when δ_0 is smaller than $1.1 \cdot \delta_2$, convergence issues might appear due to the steep decrease in stress.

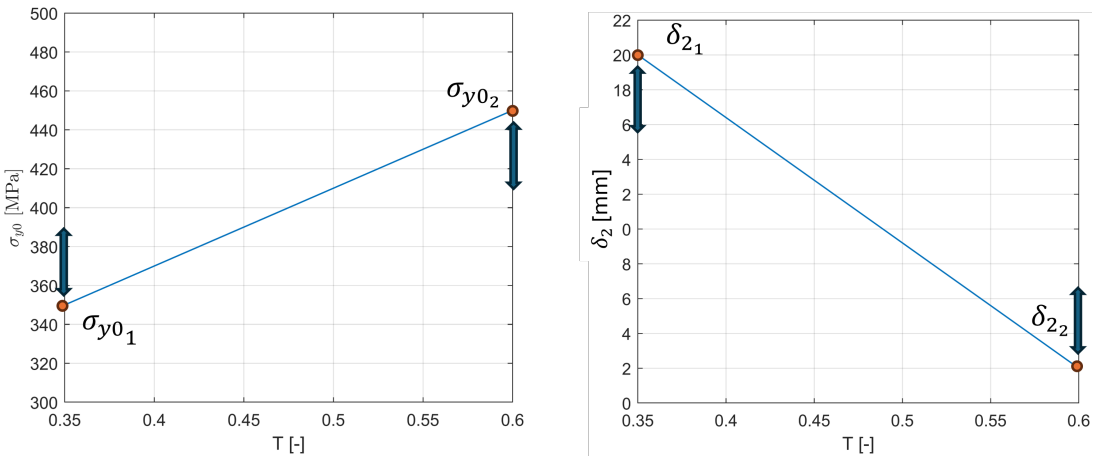
The remaining cohesive parameters, δ_2 and σ_{y0}, will be used in the optimization. Here, a linear trend is assumed between each cohesive parameter and stress triaxiality. This linear trend curve is formed between two points, which can shift in upward or downward direction. It shall be obvious that moving these points in vertical direction

changes the slope of the linear curve and, thus, the entire cohesive parameter-triaxiality relationship.

For this reason, the y-coordinates of these points are suitable optimization parameters, resulting in σ_{y0_1}, σ_{y0_2}, δ_{2_1} and δ_{2_2}. In figure 33, the parameters are shown. The allowed movement of the optimization parameters in the vertical direction is indicated with arrows.

The x-coordinates of the 4 points remain fixed during the optimization. In figure 33, it can be seen that the curves run from a triaxiality value of 0.35 to 0.6. This range is in accordance with the observed triaxiality range from the FE simulations of this load case, which is shown in figure 39.

It is noted that the assumption of the linear relationship between the TSL parameters and triaxiality may not be completely accurate. This assumption is made to further reduce the number of optimization parameters to a minimum of 4. It is highlighted that, within this thesis, no effort is made to refine these relationships by adding more points.



(a) (b)

Figure 33: For both cohesive parameters, two points define a linear relationship between the cohesive parameter and triaxiality. The y-coordinates of these 4 points are the optimization parameters.

3.2.2 Constraints

A few constraints apply to the optimization parameters. First of all, σ_{y0} must be bounded to prevent it from taking on values that are not feasible. Therefore, σ_{y0} is bounded by the yield stress of the material and the highest defined point at the stress-strain diagram σ_h:

$$\begin{aligned} \sigma_y \leq \sigma_{y0_1} \leq \sigma_h \\ \sigma_y \leq \sigma_{y0_2} \leq \sigma_h. \end{aligned} \tag{12}$$

Secondly, δ_2 is bound by a minimum value of 0.1 to prevent δ_2 to become zero. When δ_2 is zero, δ_0 will also be zero. This will not be possible, and convergence issues will occur during the simulation. It must be noted that the value of 0.1 is chosen arbitrarily, and it can be argued that this needs to be smaller. However, because the

optimization solutions for δ_2 (described in section 4.1) are all much higher than 0.1, it is assumed that the bound of 0.1 is sufficient and reducing it will not affect the results.

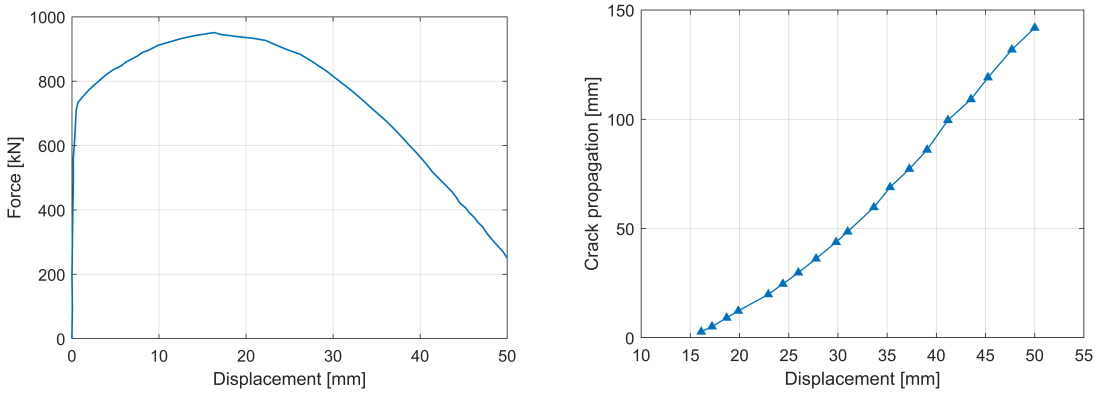
$$\begin{aligned} 0.1 &\leq \delta_{2_1} \\ 0.1 &\leq \delta_{2_2} \end{aligned} \tag{13}$$

The final constraint relates to the findings of the literature study. It was mentioned in section 1.3 that the initiation stress is expected to be an increasing function of triaxiality, and the cohesive energy (here expressed as the failure displacement) is expected to be a decreasing function of triaxiality. Two constraints are formed that ensure this behaviour:

$$\begin{aligned} \sigma_{y0_1} &\leq \sigma_{y0_2} \\ \delta_{2_2} &\leq \delta_{2_1} \end{aligned} \tag{14}$$

3.2.3 Objective function

The goal of the optimization is to find the input that results in a match with the experiment. The experimental data used for this are the force-displacement curve and the crack propagation-displacement curve, shown in figure 34.



(a) (b)

Figure 34: Objective curves for the optimization: force-displacement diagram (a) and crack propagation-displacement diagram (b).

An error can be defined when comparing the curves obtained from the experiment with those obtained by the FE simulation. This error is expressed as the mean absolute error (MAE). The MAE is expressed as the sum of absolute errors divided by the number of points along the curve, multiplied by 100 to obtain a percentage:

$$\text{MAE} = \sum_{i=1}^{n} \frac{|Y_i - y_i|}{n} \cdot 100\%. \tag{15}$$

Where Y_i is a y-coordinate of the experimental result and y_i is a y-coordinate of the FE result. n stands for the total number of data points along the curve.

The objective function is formed by minimizing the total error, i.e. the sum of the force-displacement error (MAE_1) and the crack propagation-displacement error (MAE_2). The objective function is expressed as:

$$\min_{\boldsymbol{\sigma_{y0_1}},\boldsymbol{\sigma_{y0_2}},\boldsymbol{\delta_{2_1}},\boldsymbol{\delta_{2_2}}} \text{MAE}_1 + \text{MAE}_2 \tag{16}$$

Subjected to:

$$\begin{aligned} &\sigma_y \leq \sigma_{y0_1} \leq \sigma_h \\ &\sigma_y \leq \sigma_{y0_2} \leq \sigma_h \\ &0.1 \leq \delta_{2_1} \\ &0.1 \leq \delta_{2_2} \\ &\sigma_{y0_1} \leq \sigma_{y0_2} \\ &\delta_{2_2} \leq \delta_{21} \end{aligned} \tag{17}$$

3.2.4 Algorithm

For this optimization study, the Nelder Mead algorithm is used. The Nelder–Mead method is a nonlinear, numerical optimization method for finding the minimum or maximum of an objective function in a multidimensional space. It is also a direct search method, meaning that it seeks to find the minimum or maximum by directly exploring the solution space without the need for derivative information. This is particularly useful for problems such as the current one, where obtaining the derivatives can be complex.

Additionally, the Nelder Mead method is a local method. This means that the algorithm aims to find an optimum within range of the initial starting point, which can be a local optimum instead of a global one. Local methods are regarded as more efficient and will converge to an optimum faster. However, it is noted that when information about the global minimum and the shape of the function is unknown, there is a strong dependence on the initial starting point, as different starting points can lead to different local optima. For this reason, different starting points are investigated within this study, and multiple optima are found [41].

3.2.4.1 Working principle

In the Nelder Mead method, a simplex is formed, which is a geometric figure consisting of n+1 points in an n-dimensional space. In the current problem, the simplex consists of 4+1=5 points. The working principle is to compare the values of the objective function at all the points in the simplex and gradually move in the direction that produces the best results.

The simplex can move through the solution space by reflection, expansion, contraction and shrinking. These actions are illustrated in figures 35 and 36, where an example simplex is shown in 2D space [41].

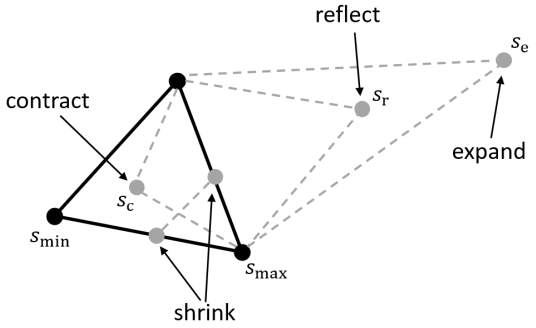


Figure 35: An example of a simplex in 2D space. The simplex consists of 2+1=3 points and can reflect, expand, contract and shrink.

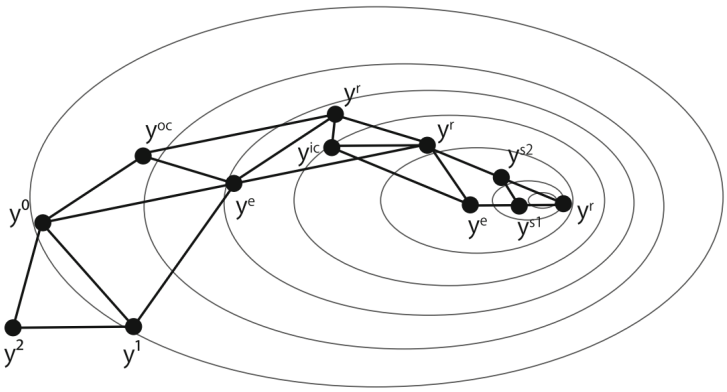


Figure 36: An example of a simplex in 2D space, reaching an optimum solution by reflection, expansion, contraction, and shrinking [37].

Chapter 4

Results

This chapter presents the results of the optimization study. Multiple optimized cohesive parameter-triaxiality relationships are proposed, and the obtained force-displacement and crack propagation curves are compared with those of the experiment. Finally, the increased accuracy gained by employing stress-state dependent TSLs instead of a stress-state invariant TSL is investigated.

4.1 Optimization results

Figure 37 shows the result of the optimization study. Multiple starting points were used, which resulted in four optimal solutions:

$$\begin{aligned} \text{Solution 1} &= [403.3,\ 410.7,\ 16.5,\ 4.7] \\ \text{Solution 2} &= [439.2,\ 455.0,\ 10.4,\ 3.4] \\ \text{Solution 3} &= [435.8,\ 437.7,\ 11.7,\ 3.7] \\ \text{Solution 4} &= [425.3,\ 427.1,\ 14.3,\ 3.9]. \end{aligned} \tag{18}$$

Each of these four solutions resulted in a minimisation of the objective function. The difference in error values that were generated with these four solutions is negligible ($< 0.6\%$ total error difference). This means that instead of one single solution, there are multiple combinations for both σ_{y0} and δ_2 that provide a good solution. The lowest error is achieved with solution 4.

When looking at figure 37, it is evident that the solutions for δ_2 are clearly decreasing, from a range of 10-15 mm to around 5 mm, which corresponds to a decrease of roughly 50-66%. However, the σ_{y0} solutions seem to behave in a more constant manner. The σ_{y0} curves of solutions 3 and 4 are almost completely constant, with a difference of around 2 MPa. The curves for solutions 1 and 2 show a slight increase, but the difference here is only 1.7% and 3.6%, respectively.

It also becomes clear that within a solution, a relatively low σ_{y0} value is paired with a relatively high δ_2 value, and vice versa. This behaviour makes sense, as these variations cancel each other out concerning their addition to the total cohesive energy.

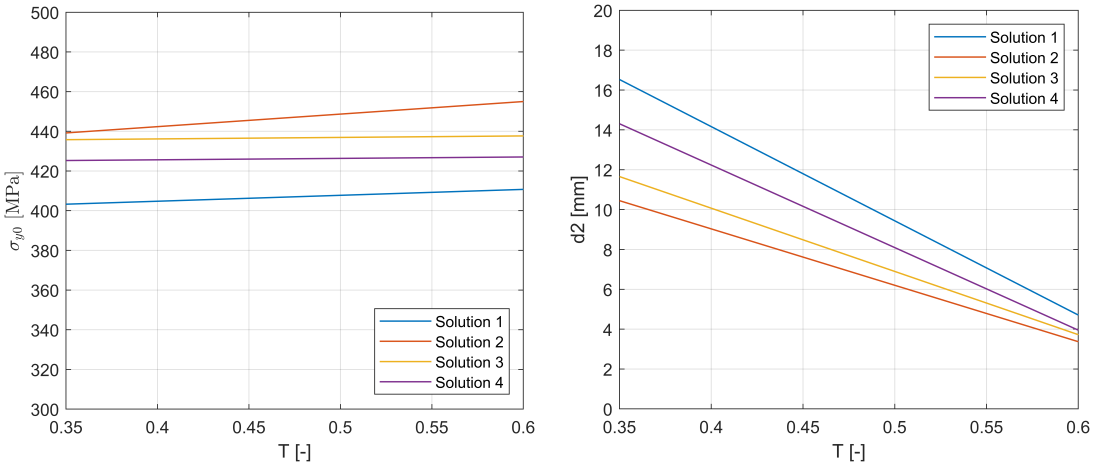
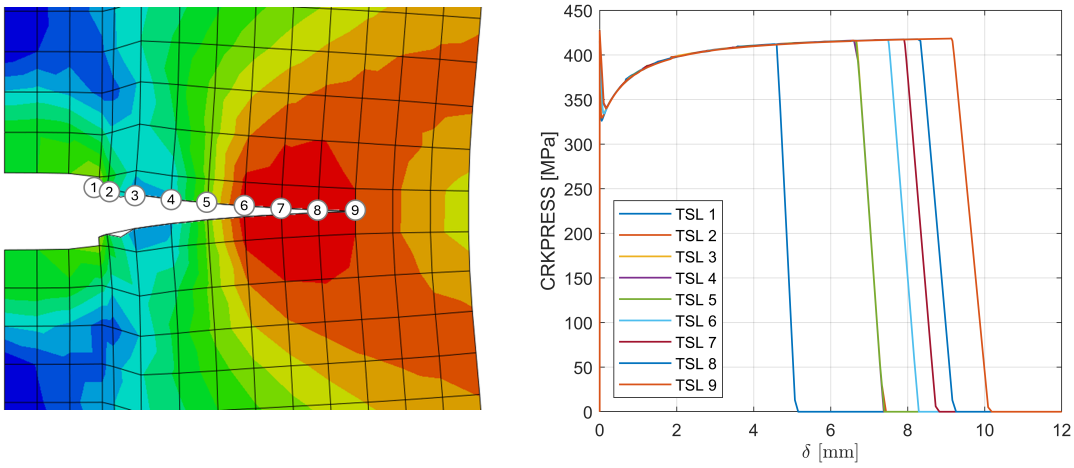


Figure 37: Found optimum curves of initiation stress σ_{y0} vs. triaxiality (a) and δ_2 vs triaxiality (b).

Figure 38 shows a close-up of the cracked specimen along with the TSLs along the crack length. The TSLs are derived by the relationships of solution 4. The triaxiality values of the elements along the crack path are 0.58, 0.53, 0.52, 0.53, 0.51, 0.5, 0.48, 0.47 and 0.46, respectively, see also figure 39. It is noted that TSLs 2-5 show overlap, as the triaxiality values for these elements are within close range of each other.



(a) (b)

Figure 38: Close-up of the cracked specimen (a) and the TSLs corresponding to the 9 highlighted nodes along the crack length (b).

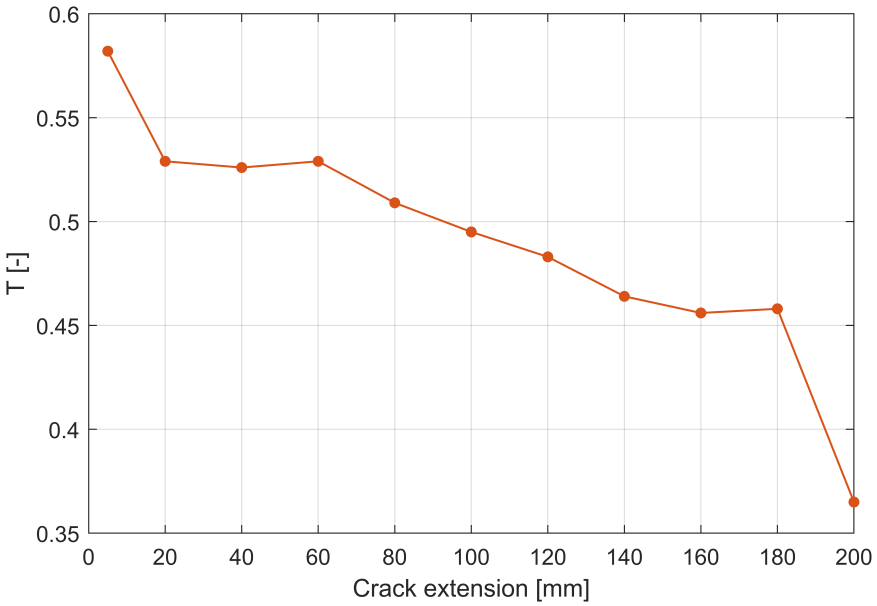


Figure 39: Triaxiality values along the crack path, measured at the moment of failure for each element.

The force-displacement curve resulting from solution 4 is depicted in figure 40. It is noticeable that a fairly good match is obtained between the experiment and the FE results, with a mean absolute error of 3.07%. It is evident that this error is mostly due to the zigzag pattern of the FE curve, resulting from the relatively large elements that break one at a time. The location of the maximum value of the FE curve is also influenced by this zigzag pattern. It is visible that a peak occurs at a displacement of 24.3 mm instead of 16.4 mm at the experimental curve. The maximum attained force of the FE curve is 964 kN, compared to 950 kN for the experimental curve. Additionally, at a final displacement of 50.0 mm, the experimental curve reaches a force value of 250 kN. This compares well with the obtained final force of the FE curve, which is 271 kN. The maximum error is 23.71% and occurs at a displacement of 48.3 mm.

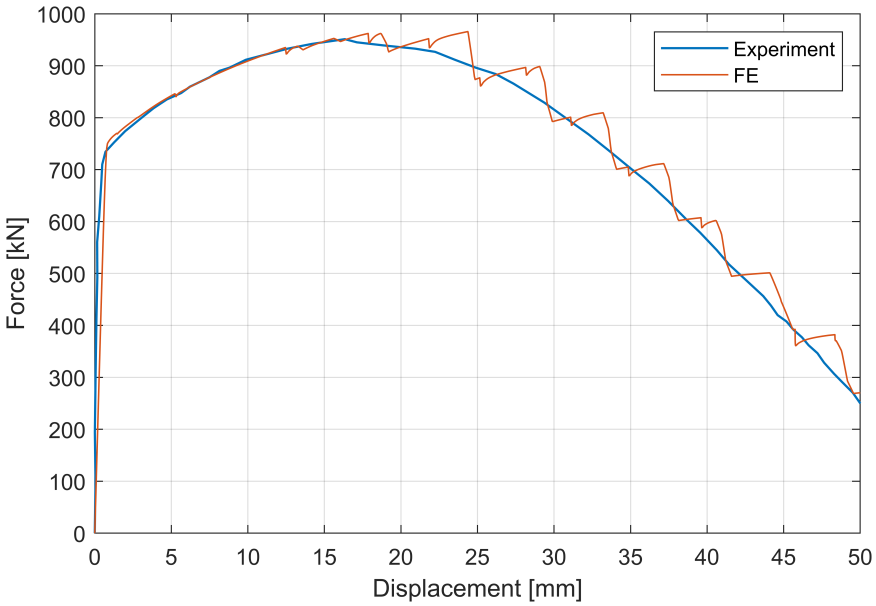


Figure 40: Force-displacement curve of experiment and FE simulation obtained by stress-state dependent TSLs, following the trends of solution 4.

Figure 41 shows the crack propagation curves. Also here, a good match is attained between the experiment and the FE model, with a mean absolute error of 1.75%. The maximum error is 7.87% and occurs at a predicted crack propagation of 20 mm.

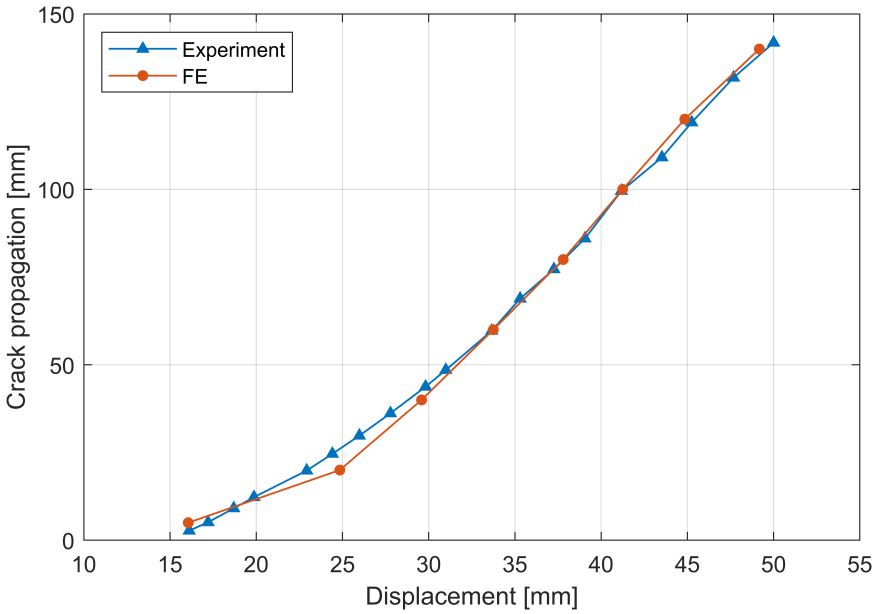


Figure 41: Crack propagation curve of experiment and FE simulation obtained by stress-state dependent TSLs, following the trends of solution 4.

4.2 Comparison to a single TSL

The results presented above show that employing a varying, stress-state dependent TSL over the crack length leads to a good match between FE and experiments. This section investigates the accuracy gained by employing a stress-state-dependent TSL rather than a fixed, stress-state invariant one.

Below are the results of an optimization study with only two optimization parameters: σ_{y0} and δ_2. These parameters form a single TSL and do not vary with respect to the triaxiality. This means that only a single TSL is employed over the entire crack length.

The found optimum is:

$$\text{Optimum } 5 = [417.1,\ 7.1] \tag{19}$$

Figure 42 shows the force-displacement diagram of this solution. It is evident that a fair match is obtained with the experimental data.

Nonetheless, some small deviations exist. The FE curve shows a slight over-prediction of the force in the 15-35 mm region compared to the experimental curve and a slight under-prediction from 35 mm onwards. This leads to a higher maximum predicted force, which is 981 kN at 21.2 mm, compared to the 950 kN at 16.4 mm of the experimental curve. Additionally, the force at the final displacement of 50 mm is 162 kN, which is significantly lower than the 250 kN observed in the experiments. Table 4 shows an overview of the maximum force and the force at a displacement of 50 mm for the experiment and the two FE cases. This leads to a maximum error of 32.21%, occurring at a displacement of 45.6 mm, and a mean absolute error of 5.12%.

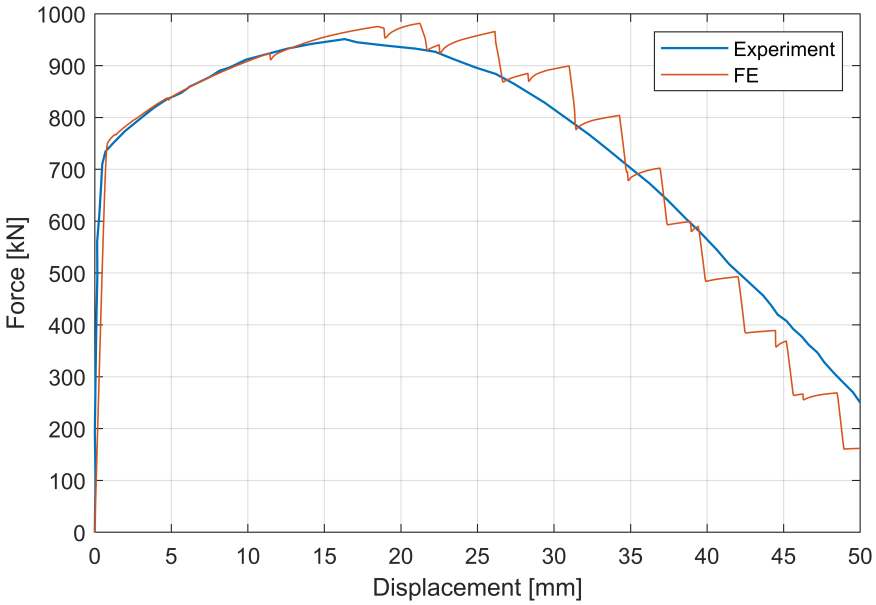


Figure 42: Force displacement curve of experiment and FE simulation obtained by employing a single TSL over the crack length.

Table 1: Comparison of the peak force and force at 50 mm for the three cases.

	Peak force	Displacement at force peak	Force at 50 mm
Experiment	950 kN	16.4 mm	250 kN
Stress-state-dependent TSLs	962 kN	24.3 mm	271 kN
Fixed TSL	981 kN	21.2 mm	162 kN

The deviations of the force-displacement diagram shown in figure 42 indicate a slight mismatch in the predicted crack propagation behaviour. This mismatch is depicted more clearly in figure 43, which shows a more significant discrepancy between the FE and the experimental results. It can be deduced from this figure that the crack is initiated later in the FE model than in the experiment. Also, the predicted crack propagation reaches its maximum value at a lower displacement, indicating a higher overall crack rate. The maximum error is 26.76% and occurs at the first data point of the FE curve at a crack propagation of 5 mm. The mean absolute error between these curves is 7.48%.

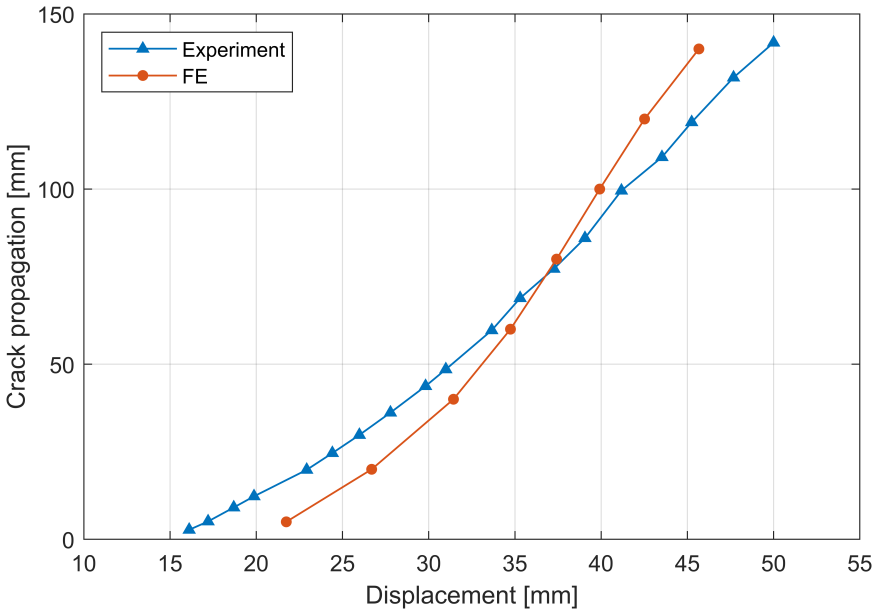


Figure 43: Crack propagation curve of experiment and FE simulation obtained by employing a single TSL over the crack length.

Figures 42 and 43 showcase the clear need for multiple stress-state-dependent TSLs along the crack length. During the first half of the total crack length, the force is over-predicted in the FE model because the crack propagation is under-predicted. The opposite is true for the second half of the crack length. It is easy to imagine that employing a TSL with an increasing δ_2 value will fix this issue, as becomes visible when looking at figures 40 and 41.

The gained accuracy of using a stress-state dependent TSL compared to a fixed TSL is shown in table 2. The maximum error difference of 18.90% stands out from this table, indicating that the crack propagation curve is highly improved when stress-state-dependent TSLs are employed.

Table 2: Comparison of the mean absolute errors of stress-state dependent TSLs vs a fixed, stress-state invariant TSL. MAE1 and max. error 1 refer to the force-displacement curve, and MAE2 and max. error 2 refer to the crack propagation curve.

	MAE1	Max. Error 1	MAE2	Max. Error 2
Stress-state dependent TSLs	3.07%	23.71%	1.75%	7.87%
Fixed TSL	5.12%	32.21%	7.48%	26.76%
Gained accuracy	2.05%	8.5%	5.73%	18.90%

4.3 Mesh sensitivity

In the previous section, it was highlighted that employing multiple, stress-state-dependent TSLs indeed increases the accuracy of the FE crack propagation simulation. This is an important finding, and the stress state can now be incorporated into the TSL definition.

However, in section 1.2.4.2, it was mentioned that besides the stress state, a TSL is also dependent on the material and mesh size of the FE model. This section shortly addresses the mesh size dependency that was observed in the current FE model.

Within the current FE model, a significant mesh dependency was found. This is illustrated in figure 44, which shows the force-displacement diagram for different mesh sizes obtained by solution 4. It is noticeable that a smaller mesh size results in an underprediction of the force, whereas a bigger mesh size results in an overprediction.

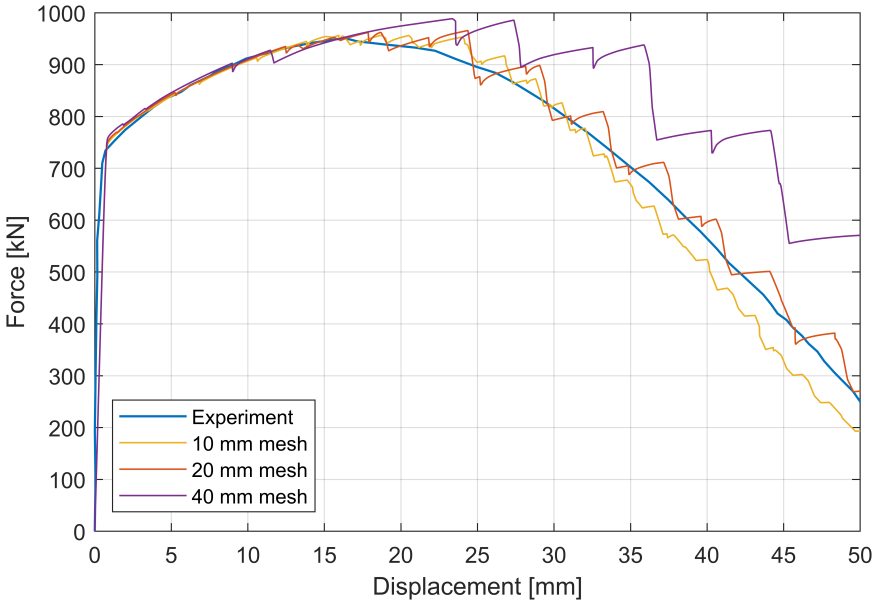


Figure 44: Force-displacement diagram of solution 4 for different mesh sizes.

This phenomenon is expected. It is widely known that smaller elements show higher stresses at the location of a stress concentration than large elements, which is caused by stress averaging within the element. In our model, the stress that is influenced the most by the element size is the stress concentration that occurs at the crack tip. Therefore, a model with a smaller mesh will reach the crack initiation stress earlier, meaning at a smaller displacement, than a model with larger elements. Table

3 illustrates this phenomenon. The table shows the stress concentration values that occur in different models at a fixed applied displacement of 3.5 mm. A stress difference of around 20 MPa can be observed between the 10 mm model and the 40 mm model. This effect is believed to be the main contributor to the large mesh dependency of the FE results.

Table 3: Max. principal stress value at an applied displacement of 3.5 mm for different models.

	Hot spot stress at 3.5 mm applied displacement
10 mm mesh	472.0 MPa
20 mm mesh	464.5 MPa
40 mm mesh	453.4 MPa

Another interesting factor is the extent to which the stress triaxiality values are influenced by the mesh size. If the stress triaxiality shows a high mesh size dependency, this would also significantly affect the results. Yet, this seems not to be the case, as shown in figure 45. This figure plots the triaxiality values against the crack extension for the different mesh sizes. It becomes clear that, apart from some small deviations, no consequential triaxiality differences exist between the different mesh sizes.

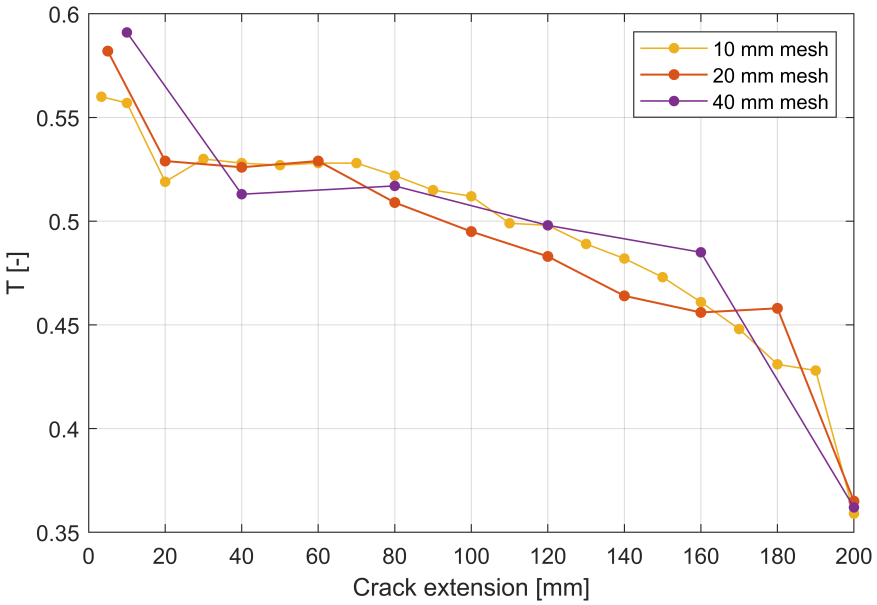


Figure 45: Variation of triaxiality over the crack length. Here, each triaxiality value is taken at the moment of damage initiation.

It is recognized that overcoming the dependency on the mesh size is an important step to take in order to obtain a more robust definition of the TSLs. Nonetheless, this thesis emphasises the stress state dependency of the TSLs, and no effort is made to overcome the mesh sensitivity. A discussion about this topic can be found in chapter 5.1.

4.4 Unconstrained optimization

As described in section 3.2.2, two particular constraints are implemented in the optimization:

$$\sigma_{y0_1} \leq \sigma_{y0_2}$$
$$\delta_{2_2} \leq \delta_{2_1}$$

These constraints enforce the TSL behaviour that was expected from the literature review, namely the increase of the initiation stress σ_{y0} and the decrease of δ_2 with increasing triaxiality. Nevertheless, the optimum solution illustrated in figure 37 shows near-constant initiation stress functions, which can indicate that the optimum tends to go towards a decreasing function for initiation stress instead of the expected increase.

For this reason, optimization runs are performed without these two constraints. This means that the cohesive parameters can increase or decrease without restrictions. The remaining constraints listed in section 3.2.2 are still enforced.

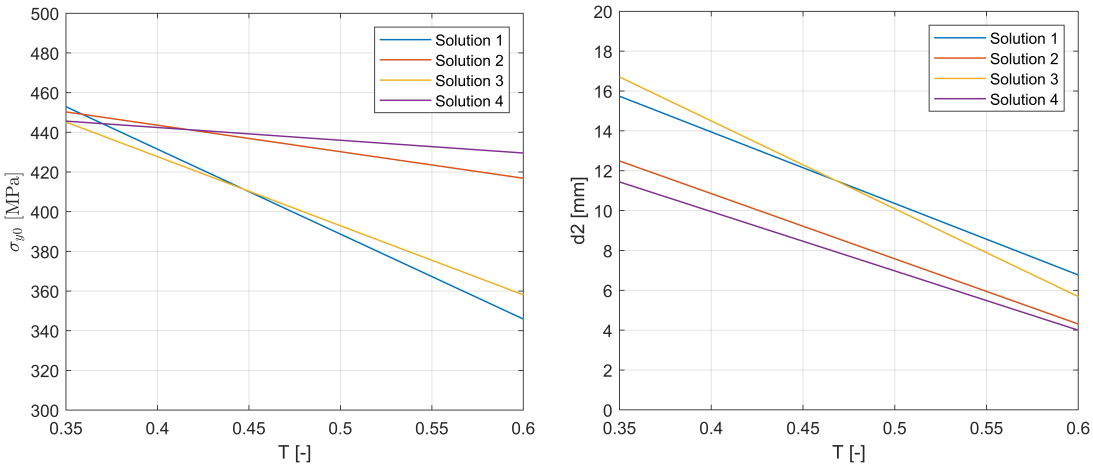
These 'unconstrained' optimization runs result in the following solutions:

$$\begin{aligned} &\text{Solution } 6 = [452.9,\ 345.9,\ 15.7,\ 6.8] \\ &\text{Solution } 7 = [450.3,\ 416.9,\ 12.5,\ 4.3] \\ &\text{Solution } 8 = [445.1,\ 358.1,\ 16.7,\ 5.7] \\ &\text{Solution } 9 = [445.6,\ 429.6,\ 11.4,\ 4.0]. \end{aligned} \tag{20}$$

The errors between of these solutions are again fairly close together ($< 1\%$ total error difference). The lowest error is achieved with solution 6. This error is shown in table 4. It is noticeable that both MAE1 and MAE2 are lower for the unconstrained optimization solution.

Figure 46 demonstrates that the four solutions can be grouped into two trend lines showing similar behaviour: solutions 6 and 8 and solutions 7 and 9. It can be observed that solutions 7 and 9 are roughly the same as solutions 1-4, which were obtained by constrained optimization. The main difference here is the slight decrease of σ_{yo} instead of the slight increase/constant trend found in solutions 1-4.

A more extensive contrast is found between solutions 1-4 and solutions 6 and 8 (which show the lowest error). Solutions 6 and 8 show a clear decrease in σ_{yo}, which does not align with the initial assumptions that were made after the literature review. This phenomenon is discussed further in section 5.1.



(a) (b)

Figure 46: Obtained optimum curves of initiation stress σ_{y0} vs. triaxiality (a) and δ_2 vs triaxiality (b) with unconstrained optimization.

Table 4: Comparison of the mean absolute errors of constrained vs. unconstrained optimization.

	MAE1	MAE2
Constrained optimization	3.07%	1.75%
Unconstrained optimization	2.67%	1.45%

Chapter 5

Discussion and future outlook

In this chapter, the work presented in this thesis is evaluated. Section 5.1 and 5.2 highlight the most important conclusions and points of interest from the results, and sections 5.3 and 5.4 critically review the limitations of the proposed research and present a future outlook.

5.1 Discussion

This study aimed to investigate whether FE simulations on crack initiation and crack propagation could be improved by employing multiple stress-state-dependent TSLs along the crack length. It is shown within this thesis that this is indeed possible. TSLs that were based on stress triaxiality had a closer match with experiments than if a triaxiality invariant TSL was used. However, this observation is subject to a few conditions. A number of important findings are discussed more thoroughly in this section.

One of these conditions regards the parameters that affect the TSL. From the literature, it was found that an important parameter of influence is the stress state, as this is well known to affect the onset of necking and ultimate fracture, which is inferred to affect the TSL. If we strive towards a TSL definition that is universally applicable, meaning that a TSL can be generated for a (near) infinite variety of conditions, addressing the stress state dependency is an important step. This stress state dependency is tackled within this thesis, and relationships between the cohesive parameters and the stress triaxiality are obtained. However, it was proven in chapter 4.3 that not only the stress state but also the mesh size influences a TSL. Additionally, it is expected that the material model is also an important consideration, as the material obviously influences the fracture behaviour and, thus, the TSLs. It is thus recognized that, in order to be able to determine the TSLs in a truly universal manner, further investigations are needed that focus on incorporating the dependencies on mesh size and material as well.

Continuing this line of thought, it would be valuable to investigate whether the dependence on mesh size is related to the specimen's geometry (such as thickness or width) or to the absolute value of the mesh size. If the latter holds true, it would be feasible to scale the TSLs, allowing large-scale problems to be addressed using data

from small-scale tests. It can be argued that *a priori* prediction of TSLs will become of less importance if laboratory tests could provide all necessary information for empirical calibration on a larger scale. Therefore, further research into the influence of geometry and the scalability of TSLs is needed.

Another interesting finding is highlighted in chapter 4.4. Here, it became clear that a σ_{y0} value that decreases with increasing stress triaxiality leads to an optimum solution of the unconstrained optimization. This is interesting, as an increasing function of σ_{y0} was initially expected. However, upon further research and within the reviewed literature, it was discovered that no strict rule seems to apply to the behaviour of σ_{y0}.

On one side, a study by Siegmund [50] suggests an increasing σ_{y0} value for increasing triaxiality, as the TSLs for higher triaxiality ranges show a higher stress peak. On the other side, a study by Walters [12] shows failure loci that do not illustrate a clear increasing or decreasing trend for σ_{y0} (it is noted that σ_{y0} physically corresponds to the necking stress, and multiple necking stress curves are provided in this reference) but rather a minimum in the triaxiality range of 0.33-0.6.

This means two things. First of all, a decreasing σ_{y0} value could be physically possible, but it is encouraged to study its behaviour more closely in literature. Secondly, if σ_{y0} shows a minimum within the occurring triaxiality range, a linear σ_{y0}-triaxiality relationship will not accurately capture this effect, and an extra point needs to be added (see section 5.4).

5.2 Conclusions

This thesis proves that the accuracy of crack propagation simulations can be improved by employing stress-state-dependent TSLs.

First, it is shown that a TSL can accurately be implemented within an XFEM context to represent crack initiation and propagation in Abaqus. Contact surfaces can be defined at the locations of the crack faces. The contact behaviour between these surfaces is expressed through a damage evolution law, which, combined with a specified damage initiation criterion, can be used to generate the desired TSL.

Secondly, it is demonstrated that a subroutine can be employed to make the TSL stress-state dependent. The written subroutine can find each element's triaxiality value and assign a specific TSL to each element based on this entity. Important input is the relationship between the cohesive parameters and the stress triaxiality.

These relationships are estimated by minimizing the differences between simulations and large-scale CCT experiments. With the fitted relationships as input, employing stress-state-dependent TSLs in a coarse-meshed model leads to a good match with experimental data. The force-displacement and crack propagation curves of an FE model with a 20 mm mesh show a minimal error compared to the experimentally obtained curves. This confirms the great potential of using stress-state-dependent TSLs in fracture simulations instead of the commonly used simulation strategies that require very fine meshes.

It is also proven that employing stress-state-dependent TSLs over the crack length results in a more accurate description of the crack behaviour than using a stress-state invariant TSL. When a stress-state invariant TSL is used, an inaccurate estimation of the crack rate and moment of crack initiation is made, resulting in an 8.50% higher maximum error of the force-displacement curve and an 18.90% higher error of the crack propagation curve. This finding emphasizes the need for stress-state-dependent TSLs.

Finally, it is noted that not only triaxiality but also mesh size and material influence the TSLs. For this reason, when aiming for a universally applicable TSL definition, these factors must also be taken into account, and it does not suffice only to consider the stress state.

5.3 Limitations

An important restriction that applies to the work presented above concerns the XFEM functionality within Abaqus. It was discovered that XFEM works only for three-dimensional solid and two-dimensional planar models, such as plane strain and plane stress. These types of elements suffice when modelling simpler cases where stresses perpendicular to the plate are negligible, such as the presented uniaxial tension load case. However, three-dimensional shell elements are more suitable for more complex geometries and loading conditions, as these elements can handle curvature and out-of-plane effects such as bending and twisting. Additionally, these elements can provide a more accurate description of the through-thickness stress distribution, which is essential for failure analysis. For this reason, the presented work is not yet suitable for applications in the industry, and a way must be found to expand the XFEM feature to three-dimensional shell elements.

It is also noted that XFEM, in its current implementation in Abaqus, only works with an implicit solver, which further limits the practicality of the proposed research. Fracture simulations are often executed within a dynamic context, such as crash or explosion simulations. These types of simulations are characterized by large strains, strain rates and deformations occurring in short time periods. An explicit solver is needed to capture these effects accurately. The restriction of using XFEM only in combination with an implicit solver means that only static or quasi-static fracture behaviour can be modelled.

Another limitation regards the boundary conditions for which this research is valid. Within this thesis, only pure Mode I fracture is considered. This means that only cracks that occur due to tensile stress normal to the crack plane can be simulated.

Further limitations follow from the assumptions made in this thesis. The first is the assumption that plane stress elements can accurately represent the stress state within the plate. Plane stress elements simplify the three-dimensional stress state to a two-dimensional problem by assuming the out-of-plane stress to be zero. This assumption holds for 'thin' plates where the thickness is much smaller than the other dimensions. In this thesis, an experiment was modelled in which this condition holds, and it is expected that the plane stress assumption doesn't influence the results significantly. However, for plates with higher relative thickness values, the plane stress assumption becomes less valid, and the use of plane strain elements must be considered.

Another assumption is made in section 3.2. Here, it was assumed that a fixed relationship exists between δ_0 and δ_2 of 1.1. This assumption was based on the expectation that once a shear band has formed, a rapid decrease in load-carrying capacity is expected, and no significant variations exist in this decrease. However, this assumption does pose a limitation on the shape of the TSLs, as the δ_0 is now unable to vary freely.

Finally, the TSL-triaxiality relationships were assumed to be linear. This simplification results in fewer optimization parameters and a more manageable number of solutions. The effect of allowing for more points along the curve, and thereby optimizing for different shapes, is assumed to be small considering the good match with the experiments already found in this thesis. However, it would be interesting to see

which TSL-triaxiality shape is found to be optimal and what the effect on the results will be.

5.4 Recommendations

Multiple recommendations can be made regarding further research on this topic. The first is to focus on expanding the XFEM feature so that it is compatible with explicit shell elements. This will significantly increase the practicality of the proposed research, as was also mentioned in section 5.3. Options for this were briefly explored, and it is expected that a user-written element subroutine (VUEL) can be used to achieve this. Details on this subroutine and further implementation remain to be determined.

Secondly, to find a universally applicable TSL definition, the mesh size and material dependency of the TSLs must be addressed, as was mentioned in chapter 5.1. An approach for this could be to repeat the optimization process described in this thesis but for multiple mesh sizes and materials and look for trends of how the results are influenced. Only when stress state, material and mesh size are considered can the TSLs be employed in a wider variety of conditions. Also, to improve the universality of the method, a study on the scalability of the TSLs and the influence on the specimen's geometry would be fruitful, as also discussed in section 5.1.

Furthermore, the mesh used in the current study consists of perfectly square elements. It is recognized that a perfect mesh is not always attainable, and elements will not always be perfectly square. If the findings of this thesis are implemented in a practical context, it would be interesting to investigate the influence of an imperfect mesh on the results.

Additionally, it is highlighted that the TSL shape that is obtained in Abaqus does not completely match the trapezoidal TSL that is aimed for. In figures 19b, 27, and 46b, it was shown that a stress drop occurs just after the moment of initiation. After this drop, the stress rises and reaches the wanted plateau value. Effort is made within this thesis to solve this issue, but it remains unclear what is happening exactly. Subsequent investigations might look into the cause of the stress drop and try to obtain a perfect trapezoidal TSL.

Another recommendation is to validate the presented results. This thesis covers a single experiment because time and experimental data were limited. However, repeating the proposed optimization study for multiple different test cases will lead to a more thorough understanding of the cohesive parameter-triaxiality relationships. It would also be interesting to see whether a different load case (a CT test rather than a CCT test, for example) will lead to similar cohesive parameter-triaxiality curves or whether the TSLs are transferable from one load case to the other. It is noted here that because of the mesh and material dependency mentioned above, the validation test cases will need to have the same material and mesh size as the experiment that was modelled in this study.

Another observation that was made within this thesis is that a decreasing σ_{y0}-triaxiality curve produces an optimal match with experiments. This behaviour was not expected initially, but this could well be physically feasible. Further research can focus on the behaviour of σ_{y0} for changing values of triaxiality to gain a more broad understanding of this topic. Also, adding an extra point to the σ_{y0}-triaxiality relationship might be needed, as discussed in section 5.1.

Finally, future work can focus on expanding the current research for mixed-mode

fracture behaviour, which will enable the analyses of a larger variety of loading conditions.

Bibliography

[1] *Abaqus User Manual*. Dassault Systemes. 2023.

[2] R. G. Andersen, C. Tekoğlu, and K. L. Nielsen. "Cohesive traction–separation relations for tearing of ductile plates with randomly distributed void nucleation sites". In: *International Journal of Fracture* 224 (2020).

[3] R.G. Andersen, C.L. Felter, and K.L. Nielsen. "Micro-mechanics based cohesive zone modeling of full scale ductile plate tearing: From initiation to steady-state". In: *International Journal of Solids and Structures* 160 (2019).

[4] M. Anvari, I. Scheider, and C. Thaulow. "Simulation of dynamic ductile crack growth using strain-rate and triaxiality-dependent cohesive elements". In: *Engineering Fracture Mechanics* 73 (2006), pp. 2210–2228.

[5] I Babuska and J.M. Melenk. "The partition of unity method". In: *Int. J. Numer.Methods Eng* 40 (1997).

[6] Y. Bai and T. Wierzbicki. "Application of Extended Mohr-Coulomb Criterion to Ductile Fracture". In: *International Journal of Fracture* 161 (2010), pp. 1–20.

[7] Anuradha Banerjee and R. Manivasagam. "Triaxiality dependent cohesive zone model". In: *Engineering Fracture Mechanics* 76 (2009).

[8] A. Amine Benzerga and Jean-Baptiste Leblond. "Ductile Fracture by Void Growth to Coalescence". In: *Advances in Applied Mechanics*. Ed. by Hassan Aref and Erik van der Giessen. Vol. 44. Elsevier, 2010, pp. 169–305. ISBN: 0065-2156. DOI: https://doi.org/10.1016/S0065-2156(10)44003-X. URL: https://www.sciencedirect.com/science/article/pii/S006521561044003X.

[9] Ahmed Amine Benzerga. "Rupture ductile des tôles anisotropes". PhD Thesis. Ecole Des Mines de Paris, 2000. DOI: 10.13140/RG.2.2.15133.74723.

[10] W. Brocks, D.-Z. Sun, and A. Hönig. "Verification of the transferability of micromechanical parameters by cell model calculations with visco-plastic materials". In: *International Journal of Plasticity* 11 (1995), pp. 971–989.

[11] Susana del Busto, Covadonga Betegón, and Emilio Martínez-Pañeda. "A cohesive zone framework for environmentally assisted fatigue". In: *Engineering Fracture Mechanics* 185 (2017), pp. 210–226. ISSN: 0013-7944. DOI: https://doi.org/10.1016/j.engfracmech.2017.05.021. URL: https://www.sciencedirect.com/science/article/pii/S001379441730098X.

[12] C.L.Walters. "Framework for adjusting for both stress triaxiality and mesh size effect for failure of metals in shell structures". In: *International Journal of Crashworthiness* 19 (2014), pp. 1–12.

[13] Sourayon Chanda and C. Q. Ru. "Cohesive zone model for temperature dependent fracture analysis of pipeline steel". In: 25th Canadian Congress of Applied Mechanics. London, Ontario, Canada, June 2015.

[14] C. R. Chen et al. "Three-dimensional modeling of ductile crack growth: Cohesive zone parameters and crack tip triaxiality". In: *Engineering Fracture Mechanics* 72.13 (2005), pp. 2072–2094. ISSN: 0013-7944. DOI: https://doi.org/10.1016/j.engfracmech.2005.01.008. URL: https://www.sciencedirect.com/science/article/pii/S0013794405000627.

[15] C.R. Chen et al. "Three-dimensional modeling of ductile crack growth: Cohesive zone parameters and crack tip triaxiality". In: *Engineering Fracture Mechanics* 72 (2005), pp. 2072–2094.

[16] Alfred Cornec, Ingo Scheider, and Karl-Heinz Schwalbe. "On the practical application of the cohesive model". In: *Engineering Fracture Mechanics* 70 (2003), pp. 1963–1987.

[17] Marcin Graba. "Characteristics of selected measures of stress triaxiality near the crack tip for 145Cr6 steel - 3D issues for stationary cracks". In: *Open Engineering* 10.1 (2020), pp. 571–585. DOI: doi:10.1515/eng-2020-0042. URL: https://doi.org/10.1515/eng-2020-0042.

[18] Marcin Graba. "On The Parameters of Geometric Constraints for Cracked Plates under Tension – Three-Dimensional Problems". In: *International Journal of Applied Mechanics and Engineering* 22 (2017). DOI: 10.1515/ijame-2017-0058.

[19] A. L. Gurson. "Continuum Theory of Ductile Rupture by Void Nucleation and Growth: Part I—Yield Criteria and Flow Rules for Porous Ductile Media". In: *Journal of Engineering Materials and Technology* 99.1 (1977), pp. 2–15. ISSN: 0094-4289. DOI: 10.1115/1.3443401. URL: https://doi.org/10.1115/1.3443401.

[20] J. W. Hancock and A. C. Mackenzie. "On the mechanisms of ductile failure in high-strength steels subjected to multi-axial stress-states". In: *Journal of the Mechanics and Physics of Solids* 24.2 (1976), pp. 147–160. ISSN: 0022-5096. DOI: https://doi.org/10.1016/0022-5096(76)90024-7. URL: https://www.sciencedirect.com/science/article/pii/0022509676900247.

[21] A. Hillerborg, M. Modéer, and P. E. Petersson. "Analysis of crack formation and crack growth in concrete by means of fracture mechanics and finite elements". In: *Cement and Concrete Research* 6.6 (1976), pp. 773–781. ISSN: 0008-8846. DOI: https://doi.org/10.1016/0008-8846(76)90007-7. URL: https://www.sciencedirect.com/science/article/pii/0008884676900077.

[22] H. Hooputra et al. "A Comprehensive Failure Model for Crashworthiness Simulation of Aluminium Extrusions". In: *International Journal of Crashworthiness* 9 (2004), pp. 449–464.

[23] M. S. Islam and K. S. Alfredsson. "Peeling of metal foil from a compliant substrate". In: *The Journal of Adhesion* (2019), pp. 1–32.

[24] G. R. Johnson and W. H. Cook. "Fracture Characteristics of Three Metals Subjected to Various Strains, Strain rates, Temperatures and Pressures". In: *Engineering Fracture Mechanics* 21 (1985), pp. 31–48.

[25] J. Koplik and A. Needleman. "Void growth and coalescence in porous plastic solids". In: *Int. J. Solids Structures* 24 (1988), pp. 835–853.

[26] H. Kordisch, E. Sommer, and W. Schmitt. "The influence of triaxiality on stable crack growth". In: *Nuclear Engineering and Design* 112 (1989), pp. 27–35. ISSN: 0029-5493. DOI: https://doi.org/10.1016/0029-5493(89)90142-8. URL: https://www.sciencedirect.com/science/article/pii/0029549389901428.

[27] Yanshan Lou et al. "New ductile fracture criterion for prediction of fracture forming limit diagrams of sheet metals". In: *International Journal of Solids and Structures* 49.25 (2012), pp. 3605–3615. ISSN: 0020-7683. DOI: https://doi.org/10.1016/j.ijsolstr.2012.02.016. URL: https://www.sciencedirect.com/science/article/pii/S002076831200056X.

[28] A. C. Mackenzie, J. W. Hancock, and D. K. Brown. "On the influence of state of stress on ductile failure initiation in high strength steels". In: *Engineering Fracture Mechanics* 9.1 (1977), pp. 167–188. ISSN: 0013-7944. DOI: https://doi.org/10.1016/0013-7944(77)90062-5. URL: https://www.sciencedirect.com/science/article/pii/0013794477900625.

[29] F. A. McClintock. "A criterion for ductile fracture by the growth of holes". In: *J. Appl. Mech* 35 (1968).

[30] Nicolas Moes, John Dolbow, and Ted Belytschko. "A finite element method for crack growth without remeshing". In: *International Journal for Numerical Methods in Engineering* 46 (1999).

[31] D. Mohr and S. J. Marcadet. "Micromechanically-Motivated Phenomenological Hosford-Coulomb Model for Predicting Ductile Fracture Initiation at Low Stress Triaxialities". In: *International Journal of Solids and Structures* 67-68 (2015), pp. 40–55.

[32] A. Needleman. "An analysis of decohesion along an imperfect interface". In: *International Journal of Fracture* 42.1 (1990), pp. 21–40. ISSN: 1573-2673. DOI: 10.1007/BF00018611. URL: https://doi.org/10.1007/BF00018611.

[33] A. Needleman and V. Tvergaard. "An analysis of ductile rupture in notched bars". In: *Journal of the Mechanics and Physics of Solids* 32.6 (1984), pp. 461–490. ISSN: 0022-5096. DOI: https://doi.org/10.1016/0022-5096(84)90031-0. URL: https://www.sciencedirect.com/science/article/pii/0022509684900310.

[34] Alan Needleman. "A Continuum Model for Void Nucleation by Inclusion Debonding". In: *Journal of Applied Mechanics-transactions of The Asme - J APPL MECH* 54 (1987). DOI: 10.1115/1.3173064.

[35] O. Nguyen et al. "A cohesive model of fatigue crack growth". In: *International Journal of Fracture* 110.4 (2001), pp. 351–369. ISSN: 1573-2673. DOI: 10.1023/A:1010839522926. URL: https://doi.org/10.1023/A:1010839522926.

[36] K.L. Nielsen and J.W. Hutchinson. "Cohesive traction-separation laws for tearing of ductile metal plates". In: *International Journal of Impact Engineering* 48 (2011).

[37] Yoshihiko Ozaki, Masaki Yano, and Masaki Onishi. "Effective hyperparameter optimization using Nelder-Mead method in deep learning". In: *IPSJ Transactions on Computer Vision and Applications* 9.1 (2017), p. 20. ISSN: 1882-6695. DOI: 10.1186/s41074-017-0030-7. URL: https://doi.org/10.1186/s41074-017-0030-7.

[38] P.B.Woelke, M.D.Shields, and J.W.Hutchinson. "Cohesive zone modeling and calibration for mode I tearing of large ductile plates". In: *Engineering Fracture Mechanics* 147 (2015).

[39] A. Pineau, A. A. Benzerga, and T. Pardoen. "Failure of metals I: Brittle and ductile fracture". In: *Acta Materialia* 107 (2016), pp. 424–483. ISSN: 1359-6454. DOI: https://doi.org/10.1016/j.actamat.2015.12.034. URL: https://www.sciencedirect.com/science/article/pii/S1359645415301403.

[40] Andre Pineau. "Modeling ductile to brittle fracture transition in steels - Micromechanical and physical challenges". In: *International Journal of Fracture* 150 (2008), pp. 129–156. DOI: 10.1007/s10704-008-9232-4.

[41] Singiresu S. Rao. *Engineering Optimization*. Vol. 5. John Wiley Sons Ltd, 220.

[42] Joris J.C. Remmers, Rene´ de Borst, and Alan Needleman. "The simulation of dynamic crack propagation using the cohesive segments method". In: *Journal of the Mechanics and Physics of Solids* 56 (2008), pp. 70–92.

[43] J. R. Rice and D. M. Tracey. "On the ductile enlargement of voids in triaxial stress fields". In: *J. Mech. Phys. Solid* 17 (1969), pp. 201–217.

[44] I. Scheider and W. Brocks. "Cohesive elements for thin-walled structures". In: *Computational Materials Science* 37 (2006), pp. 101–109.

[45] I. Scheider and W. Brocks. "Simulation of cup–cone fracture using the cohesive model". In: *Engineering Fracture Mechanics* 70 (2003), pp. 1943–1961.

[46] Ingo Scheider. "Micromechanical based derivation of traction-separation laws for cohesive model simulations". In: *Procedia Engineering* 1.1 (2009), pp. 17–21. ISSN: 1877-7058. DOI: https://doi.org/10.1016/j.proeng.2009.06.006. URL: https://www.sciencedirect.com/science/article/pii/S1877705809000071.

[47] Ingo Scheider and Wolfgang Brocks. "The Effect of the Traction Separation Law on the Results of Cohesive Zone Crack Propagation Analyses". In: *Key Engineering Materials* 251 (2003), pp. 313–318. DOI: 10.4028/www.scientific.net/KEM.251-252.313.

[48] Ingo Scheider, F. Hachez, and Wolfgang Brocks. "Effect of Cohesive Law and Triaxiality Dependence of Cohesive Parameters in Ductile Tearing". In: *Fracture of Nano and Engineering Materials and Structures - Proceedings of the 16th European Conference of Fracture* (2006). ISSN: 978-1-4020-4971-2. DOI: 10.1007/1-4020-4972-2_478.

[49] Karl-Heinz Schwalbe, Ingo Scheider, and Alfred Cornec. *Guidelines for applying cohesive models to the damage behaviour of engineering materials and structures*. Springer, 2013, p. 89. ISBN: 978-3-642-29493-8.

[50] T. Siegmund and W. Brocks. "Prediction of the work of separation and implications to modeling". In: *International Journal of Fracture* 99 (1999), pp. 97–116.

[51] Bo Cerup Simonsen and Rikard Törnqvist. "Experimental and numerical modelling of ductile crack propagation in large-scale shell structures". In: *Marine Structures* 17 (2004).

[52] Wolé Soboyejo. *Mechanical properties of engineered materials*. Marcel Dekker, Inc., 2003. ISBN: 0-8247-8900-8.

[53] I.T. Tandogan. "Ductile fracture of metallic materials through micromechanics based cohesive zone elements". MSc Thesis. Middle East Technical University, 2020.

[54] I.T. Tandogan and T. Yalcinkaya. "Development and implementation of a micromechanically motivated cohesive zone model for ductile fracture". In: *International Journal of Plasticity* 158 (2022).

[55] V. Tvergaard and J. W. Hutchinson. "The relation between crack growth resistance and fracture process parameters in elastic-plastic solids". In: *J. Mech. Phys. Solids* 40 (1992).

[56] *Unified method of test for the determination of quasistatic fracture toughness*. en. Standard. Geneva, CH: International Organization for Standardization, 2021. URL: https://www.iso.org/standard/78208.html.

[57] Nunziante Valoroso, Gilles Debruyne, and Jerome Laverne. "A cohesive zone model with rate-sensitivity for fast crack propagation". In: *Mechanics Research Communications* 58 (2014), pp. 82–87. DOI: 10.1016/j.mechrescom.2013.12.008.

[58] J. Wolf et al. "Numerical modeling of strain localization in engineering ductile materials combining cohesive models and X-FEM". In: *International Journal of Mechanics and Materials in Design* 14.2 (2018), pp. 177–193. ISSN: 1573-8841. DOI: 10.1007/s10999-017-9370-9. URL: https://doi.org/10.1007/s10999-017-9370-9.

[59] Fenghua Zhou, Jean-François Molinari, and Tadashi Shioya. "A rate-dependent cohesive model for simulating dynamic crack propagation in brittle materials". In: *Engineering Fracture Mechanics* 72.9 (2005), pp. 1383–1410. ISSN: 0013-7944. DOI: https://doi.org/10.1016/j.engfracmech.2004.10.011. URL: https://www.sciencedirect.com/science/article/pii/S0013794404002371.

Appendix A

Abaqus settings

In this appendix, the most important settings and features that were used within this thesis to obtain the results in Abaqus are presented. The goal of this appendix is to allow the obtained results to be reproducible. Within this section, all the screenshots are made within Abaqus Standard A/CAE 2023. Furthermore, references will be made to the subroutine code and Matlab script. These can be found in appendix B and C.

Material

After the definition of a part in Abaqus, the material properties must be defined. The Matlab script will fill in most of these properties, but it is nevertheless important to define the proper fields within the material properties tab in the A/CAE. Figure A.1 shows a screenshot of the edit material window. Important settings are indicated.

First of all, the material name is important. This can be chosen freely by the user, but it is important to know that this name also has to be inserted in the Matlab script. If the name in the Matlab script does not match the material name as defined here, the script will not work.

Secondly, multiple material behaviour fields must be created. These fields are indicated in the figure. Apart from the elastic and plastic material behaviour, maxps damage must be defined (via: 'Mechanical/ Damage for Traction Separation Laws/ Maxps Damage'). Also, User Defined Field ('General/User Defined Field') and Depvar ('General/Depvar') must be defined. Within the Depvar settings, set the Number of solution solution-dependent state variables on 1.

Within the Maxps Damage settings box, set the Position to Nonlocal, the Tolerance to 0.3, and the number of field variables to 1. Also, under the 'Suboptions' button, the Damage Evolution and Damage Stabilisation Cohesive can be defined. Within the Damage Evolution section, the Type is set on Displacement, the Softening on Tabular and the number of field variables on 1. Within the Damage Stabilisation Cohesive window, the Viscosity Coefficient is set to 1E-4, which is the default value.

It is again mentioned that the Matlab script implements part of the settings described above in the .inp file. This is done for the position, tolerance and the tabular values for the damage evolution/initiation for different field variables. The rest of the settings must be defined within the A/CAE.

A final word about the tolerance: this entity describes the tolerance of the damage criterion. If the tolerance is set too high the damage will not initiate at the wanted

initiation stress. If the tolerance is set too low, convergence issues might appear. It was found within this thesis that a value of 0.3 works and produces good results. However, changes can be made on this value.

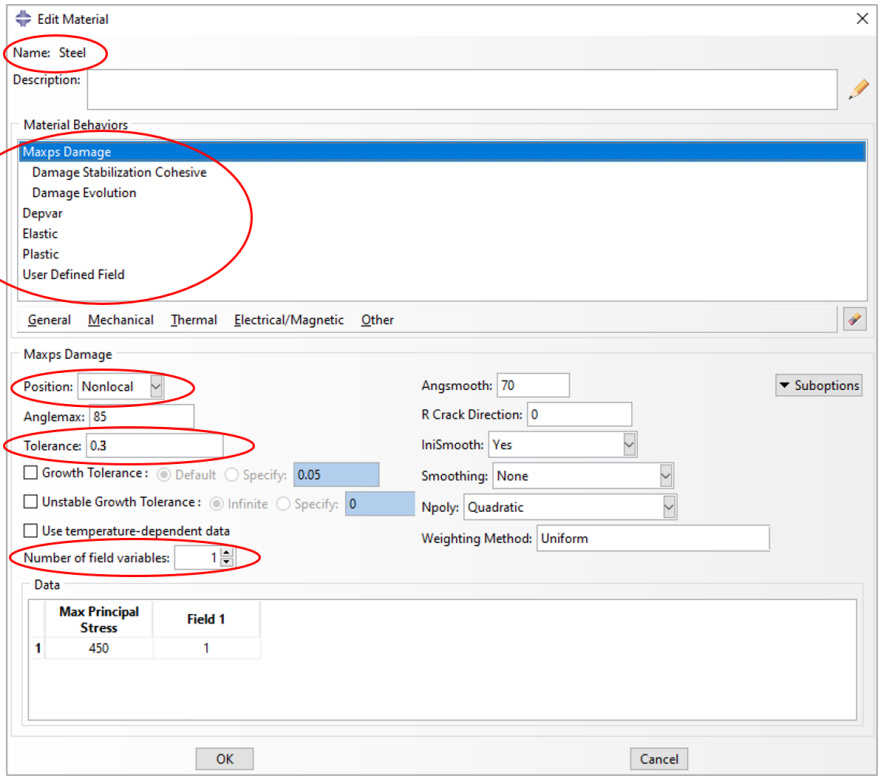


Figure A.1: Edit material window

Step

Other useful settings are found within the Step Manager (figure A.2). A step is created with the type Static, General. The Nlgeom option must be on, as this accounts for non-linear material deformations, such as necking. For the Automatic stabilisation feature, select 'Specify dissipated energy fraction', with a value of 2E-4. Also, check the box: 'Use adaptive stabilisation with max. ratio of stabilisation to strain energy' and set the corresponding value to 0.05. This automatic stabilisation option introduces artificial damping into the model to help stabilize the analysis. This resulted in a more stable simulation and smoother results. Without the automatic stabilisation, the simulation was unstable and convergence difficulties arose due to the highly nonlinear behaviour.

Within the incrementation tab, the Maximum number of increments is set to a high value of 1E8. The increment size is set to 1E-05, 1E-10, and 1E-03 for the initial, minimum, and maximum step size, respectively. The maximum step size is an important entity here. If the maximum step size is set too high, convergence issues may appear.

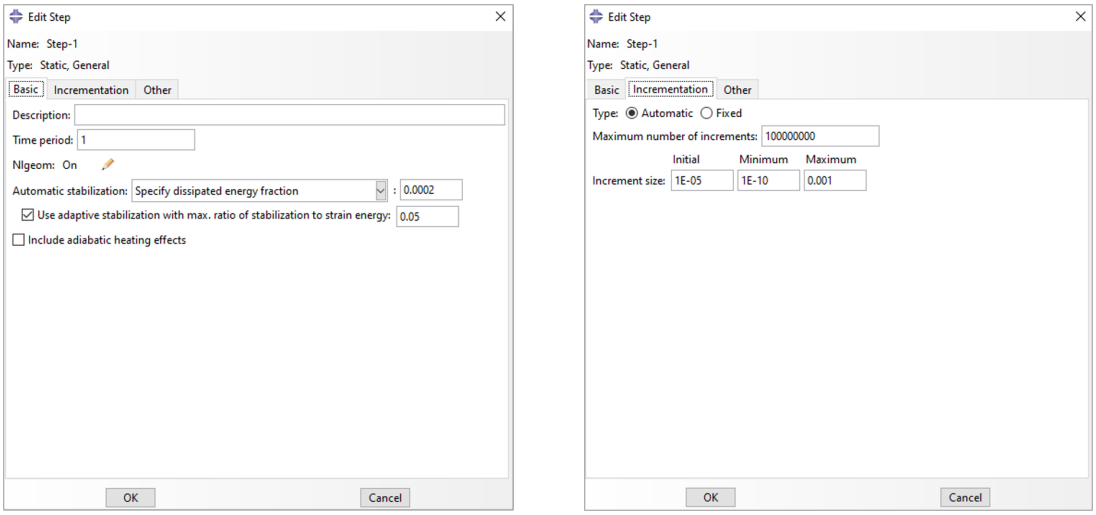


Figure A.2: Edit step windows

Next, adjustments must be made to the convergence controls within the step module. It was found that convergence could not always be reached within the executed simulations. This is resolved by increasing the convergence control on the displacement correction C_n^α and time incrementation I_A.

In Abaqus, the convergence control on the displacement correction C_n^α refers to the criteria used to determine whether the solution for a nonlinear analysis step has converged. During each iteration, the FE solver adjusts the displacement field to reduce the residual forces, which are the differences between the internal forces in the model and the externally applied loads. These adjustments to the displacement field are known as the displacement corrections. The convergence control on displacement correction determines when the solution is converged. If the C_n^α is increased, the tolerances are tightened, meaning that to reach convergence, the displacement corrections need to be smaller relative to the overall displacement in the model. This enforces a more stable and accurate solution.

Secondly, the time incrementation parameter I_A manages the size of time increments during the simulation based on the convergence behaviour. When I_A is increased, Abaqus adjust the time increments more aggressively, allowing for larger increments when the solution converges well, reducing computational time. When convergence issues are encountered, the time step is reduced less conservatively.

Within the A/CAE, C_n^α and I_A are altered via the Step module/Other/General Solution Controls/Edit (figure A.3). Within the current study, increasing C_n^α from the default value of 0.01 to 1 and increasing I_A from the default value of 1.0 to 10 lead to good convergence behaviour throughout the simulations.

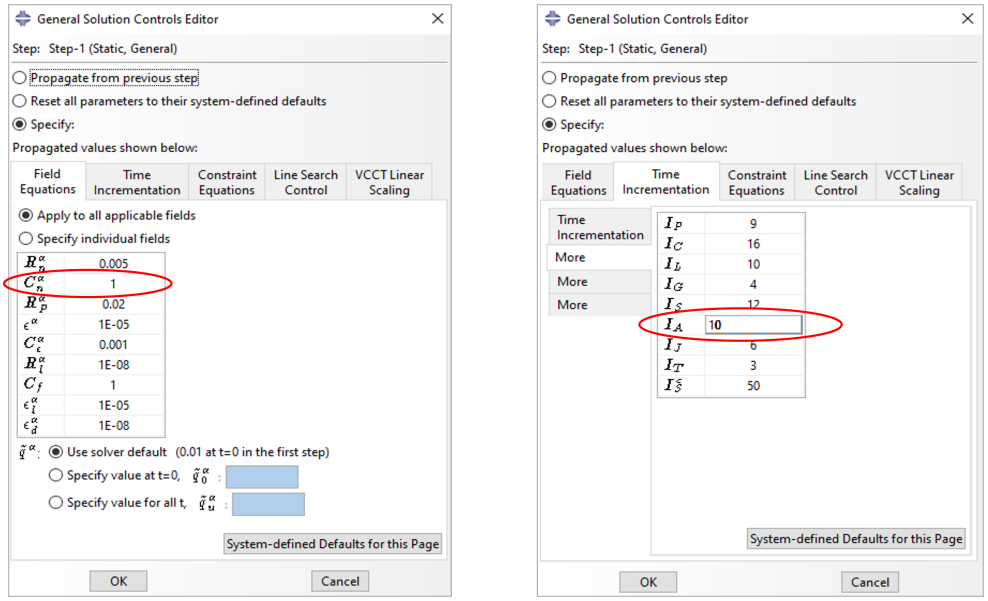


Figure A.3: Changing the convergence controls

Interaction

In order to create the XFEM crack, some steps have to be taken within the interaction module.

First, the TSL stiffness must be defined as a contact property. For this, go to the Interaction Property Manager (figure A.4). As Contact Property Options, choose 'Mechanical/ Normal Behaviour'. As a Constraint enforcement method, choose 'Penalty (Standard)'. Finally, specify the stiffness value. Please note that this 'Penalty (Standard)' value corresponds to the TSL stiffness value K_{fixed}, as discussed in section 2.2.4 and can be any value. Here, this value is set to 800.

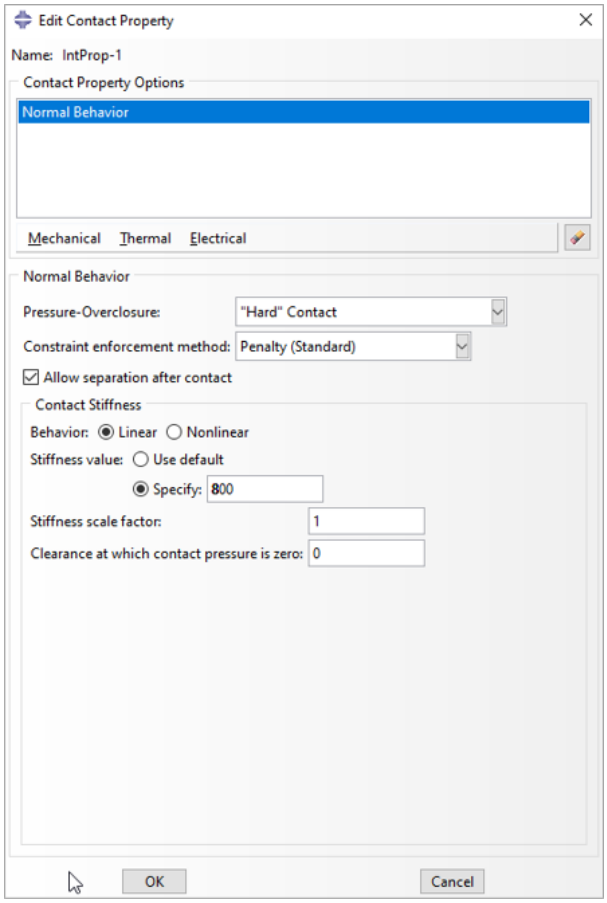


Figure A.4: Edit Contact Property window

After this, the crack must be defined within the interaction module. To do this, go to the 'Special' tab/ Crack/Create (figure A.5). Choose XFEM as the crack type and choose the part as the crack domain.

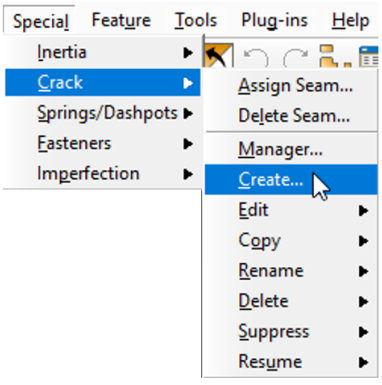


Figure A.5: Create XFEM crack

Within the Edit Crack window, it is important to link the specified contact property (where the TSL stiffness was defined) to the XFEM crack. This is done at the 'Specify contact property' box, as shown in figure A.6.

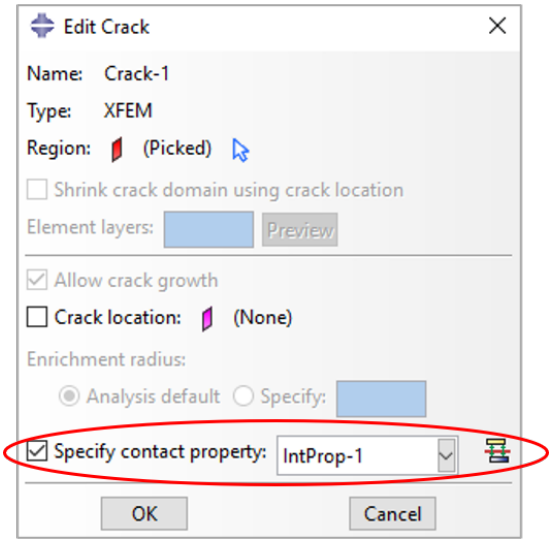


Figure A.6: Edit Crack window

Next, an interaction must be defined. This is done within the interaction manager. Create an interaction of the type 'XFEM crack growth'. Next, define the XFEM crack as specified in the previous step and check the 'Allow crack growth in this step' checkbox.

Mesh

CPS4R elements are used within this thesis. It is important to specify Enhanced Hourglass Control within the Element Type window, as depicted in figure A.7. This is because, in some cases, convergence could not be attained with the default hourglass control option, as some elements are subjected to heavy shearing. Also, the Geometric Order is Linear.

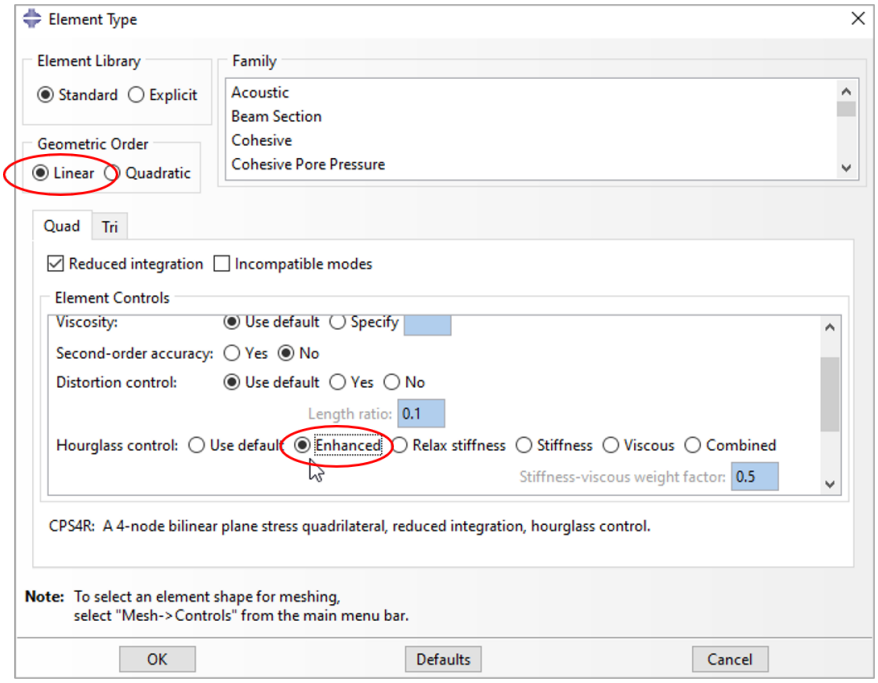


Figure A.7: Element type window

Jobs

Finally, the job is created within the Edit Job window (figure A.8). Make sure to insert the path to the subroutine.

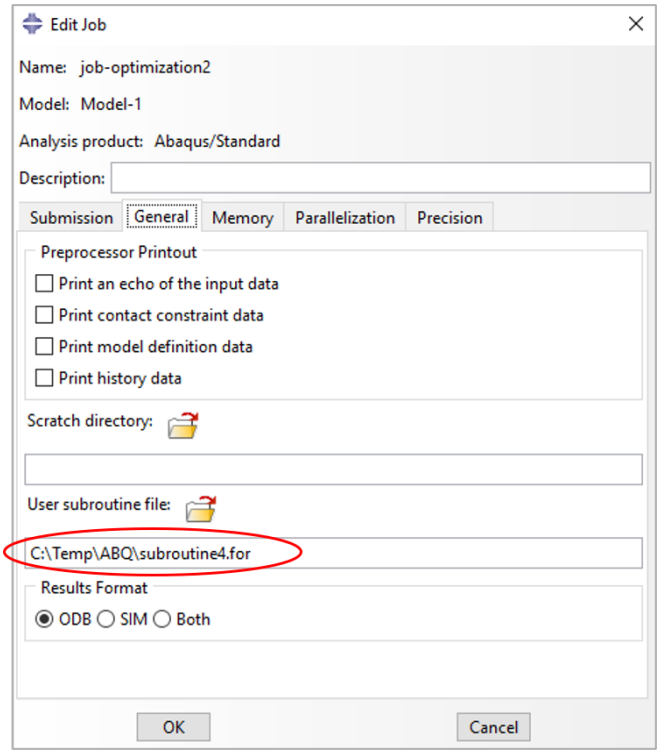


Figure A.8: Edit Job window

Matlab script

Some final settings are implemented directly to the .inp file of the Abaqus model by the Matlab script. First, a command is added in the Assembly section of the .inp file that generates the contact surfaces at the crack location. This command is shown in figure A.9. Note that 'Crack-1' is the name of the XFEM crack as defined earlier.

```
** Constraint: Constraint-1
*Coupling, constraint name=Constraint-1, ref node=m_Set-56, surface=s_Set-56_CNS_
*Kinematic
*Enrichment, name=Crack-1, type=PROPAGATION CRACK, elset=_PickedSet104, interaction=IntProp-1
*SURFACE, name=CrackSurface, type=XFEM
CRACK-1
*End Assembly
**
```

Figure A.9: The highlighted command lines generate contact surfaces at the location of the XFEM crack.

Next, the damage initiation and evolution tables are generated in the Matlab script (figure A.10). Please note that some material settings are listed at the top of these tables (such as Position: Nonlocal and tolerance). When these settings need to be changed, it is necessary to change them both in the Abaqus A/CAE as in the Matlab script.

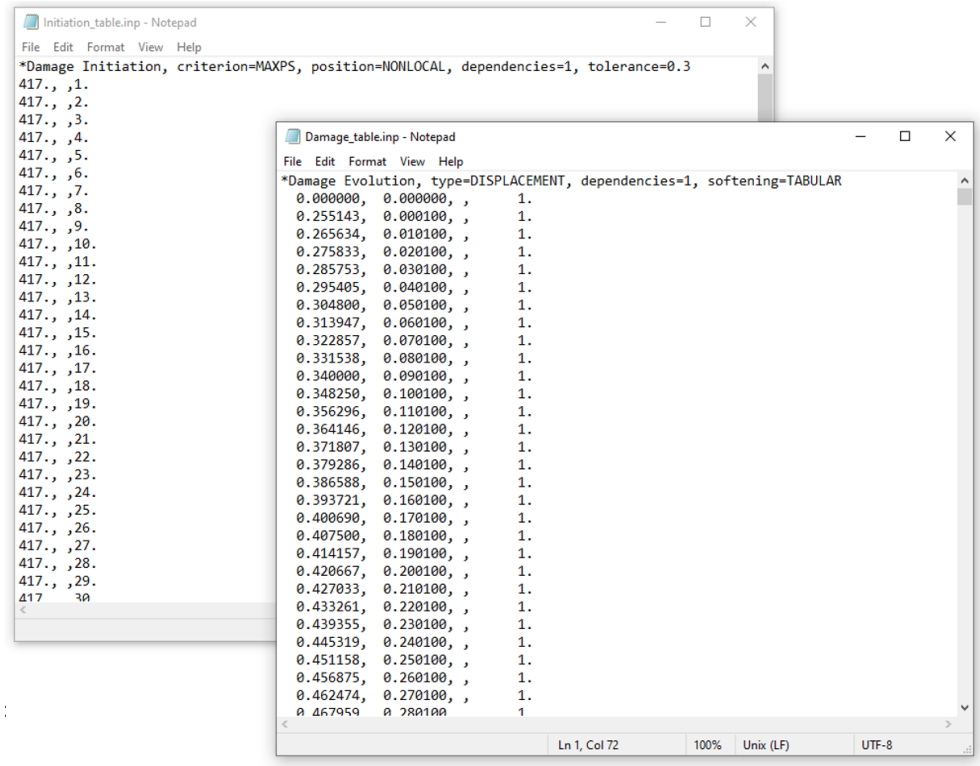


Figure A.10: Damage initiation and evolution tables, generated by the Matlab script.

These damage tables are saved in external .txt files. The 'Include' command in the .inp file includes these external files to the .inp file, as is shown in figure A.11.

```
** MATERIALS
**
*Material, name=Steel
*INCLUDE, INPUT=Initiation_table.inp
*INCLUDE, INPUT=Damage_table.inp
*Damage Stabilization
0.0001
*Depvar
      1,
```

Figure A.11: The highlighted lines include the damage initiation and evolution tables.

A final addition that is made to the .inp file is requesting the output of the contact surfaces. Two of these contact outputs are CRKPRESS and CRKOPEN, which are used to plot the TSL behaviour in the crack, as described in chapter 2.1.4.

```
** OUTPUT REQUESTS
**
*Restart, write, frequency=0
**
** FIELD OUTPUT: F-Output-1
**
*Output, field
*Node Output
CF, PHILSM, PSILSM, RF, U, UT
*Element Output, directions=YES
FV, LE, MISES, MISESONLY, PE, PEEQ, PEMAG, PRESSONLY, S, SDEG, SDV, STATUSXFEM, TRIAX
*Contact Output
CDISP, CSDMG, CSTRESS, CRKDISP, CSDMG, CRKSTRESS
**
```

Figure A.12: Request the contact output

Appendix B

Subroutine code

Below, the Fortran code of the subroutine is provided:

```
1
2
3
4        SUBROUTINE USDFLD(FIELD,STATEV,PNEWDT,DIRECT,T,CELENT,TIME,DTIME,
5       &CMNAME,ORNAME,NFIELD,NSTATV,NOEL,NPT,LAYER,KSPT,KSTEP,KINC,
6       &NDI,NSHR,COORD,JMAC,JMATYP,MATLAYO,LACCFLA)
7  C
8        INCLUDE 'ABA_PARAM.INC'
9  C
10       CHARACTER*80 CMNAME,ORNAME
11       CHARACTER*3  FLGRAY(15)
12       DIMENSION FIELD(NFIELD),STATEV(NSTATV),DIRECT(3,3),
13      &T(3,3),TIME(2)
14       DIMENSION ARRAY(15),JARRAY(15),JMAC(*),JMATYP(*),COORD(*)
15
16
17
18 C Catagorize the input
19       REAL :: numberofTSLS, maxobservedTRIAX, minobservedTRIAX, delta
20       INTEGER :: i
21
22
23 C STEP 1. Calculate TRIAX and SDEG
24       CALL GETVRM('SINV',ARRAY,JARRAY,FLGRAY,JRCD,JMAC,JMATYP,MATLAYO,
25      &LACCFLA)
26
27
28       TRIAX=-ARRAY(3)/ARRAY(1)
29
30
31       CALL GETVRM('SDEG',ARRAY,JARRAY,FLGRAY,JRCD,JMAC,JMATYP,MATLAYO,
32      &LACCFLA)
33
34       SDEG=ARRAY(1)
35
36
37 C STEP 2. Determine the maximum value of TRIAX up to this point in time
38       CALL GETVRM('SDV',ARRAY,JARRAY,FLGRAY,JRCD,JMAC,JMATYP,MATLAYO,
39      &LACCFLA)
40
41
42       TRIAXCURRENT = ARRAY(1)
```

```
43
44
45
46  C Save the maximum value
47        IF (SDEG==0) THEN
48        TRIAXMAX = MAX( TRIAX , TRIAXCURRENT )
49        ELSE
50        TRIAXMAX=TRIAXCURRENT
51        END IF
52
53
54
55  C STEP 3. Set the number of triaxiality levels
56        maxobservedTRIAX = 1
57        minobservedTRIAX= 0
58        numberofTSLS  = 100
59
60
61        delta=(maxobservedTRIAX-minobservedTRIAX)/numberofTSLS
62
63
64  C STEP 4. Assign a field variable to the element
65        do i = 1,numberofTSLS
66            if (TRIAXMAX <= minobservedTRIAX+i*delta) THEN
67                FIELD(1) = i
68                exit
69            end if
70        end do
71
72
73  C STEP 5. Store the maximum triax as a solution dependent state
74  Cvariable
75
76        STATEV(1) = TRIAXMAX
77
78
79
80  C Return from subroutine
81        RETURN
82        END
```

Appendix C

Matlab code

Below, the Matlab code of the TSL-Tool is provided:

```
1
2
3  clc
4  clear
5
6
7  %%  Input
8
9  E = 210000;                          % Youngs modulus
10 Y = 252;                             % Yield strength
11 Penalty_stiff = 800;                 % TSL stiffness defined in abaqus
12                                      % contact properties settings
13
14
15
16 %------Load data from optimization file
17 % Step 1: Load data from the source MATLAB file
18 load('data.mat');
19 % Step 2: Access the desired data
20
21 % Parameters in the format: [sigma_y01; sigma_y02; delta_21; delta_22]
22 params;
23
24  % X-parameters in the format: [min. triax; max.triax].
25  % This should be [0 1], and should not be changed.
26 params_x;
27 %-------------
28
29
30
31 % Base the TSL properties on TRIAX:
32
33 numberofTSLs= 100;           % Number of different TSLs that are generated.
34              % If changed it should also be changed in the subroutine code.
35
36  for i=1:numberofTSLs
37      triax= i/numberofTSLs;
38      %stress_ini(i) =interp1(params_x, params(1:2), triax);
39      stress_ini(1:34)=params(1);
40      stress_ini(35:60)= linspace(params(1),params(2),26);
41      stress_ini(61:100)=params(2);
42
```

```
43      %d2(i)= interp1(params_x, params(3:4), triax);
44      d2(1:34)=params(3);
45      d2(35:60)= linspace(params(3),params(4),26);
46      d2(61:100)=params(4);
47      d3(i)= d2(i)*1.1;
48  end
49
50
51
52  % Write parameters to .txt file, so that you can monitor them during the
53  % optimization study
54  fileID = fopen('parameters.txt', 'a');
55
56
57  % Write the value to the file
58  fprintf(fileID, '----\n');
59  fprintf(fileID, '%f ', params(1));
60  fprintf(fileID, '%f ', params(2));
61  fprintf(fileID, '%f ', params(3));
62  fprintf(fileID, '%f\n', params(4));
63  % Close the file
64  fclose(fileID);
65
66
67
68  % Manual option of setting the TSL-TRIAX relationship:
69  % numberofTSLs= 100;
70  %  for i=1:numberofTSLs
71  %      triax= i/numberofTSLs;
72  %      %stress_ini(i) =Y*(-0.116*triax^2+triax+1.6);
73  %      stress_ini(i) =triax*1000+200;
74  %      d2(i)= 0.6/triax*5;
75  %      d3(i)= 0.6/triax*5+2;
76  %  end
77
78
79
80  %% settings
81
82  % The maximum displacement for which you want to define your damage.
83  % Make sure this is large enough!
84  maxdisp = 300;
85
86  % Stepsize of the damage table
87  stepsize = 0.01;
88
89  % How much points do you want to add to the material model
90  % (adjust range for x axis of plot)
91  add2mat= 100;
92
93
94
95  %% Values that are automatically calculated
96
97  % Plasticity model and initiation point
98  PM =readtable('PM.txt');                                % Plasticity model
99
100 % Find initiation displacement from plasticity model
101 disp_ini = interp1(PM{:, 1}, PM{:, 2}, stress_ini, 'linear') ;
102
```

```
% TSL strength and stiffness
TSL_s = stress_ini;                                          % TSL strenght
TSL_stiff = TSL_s./disp_ini ;                                % TSL stiffness



%% DAMAGE MODEL
% Here the values of Damage and Displacement are determined.

% Displacement column
D_disp = 0:stepsize:maxdisp;


% Generate the Damage column
for j = 1:numel(stress_ini)
    for i = 1:numel(D_disp)
        if D_disp(i) <= disp_ini(j)
            D(i,j) = 0;
        elseif D_disp(i) <= d2(j)
            D(i,j) = 1-TSL_s(j)/(TSL_s(j)/disp_ini(j)*(D_disp(i)));
        elseif D_disp(i) >= d3(j)
            D(i,j) = 1;
        else
            D(i,j)=1-TSL_s(j)/(TSL_s(j)/disp_ini(j)*(D_disp(i)))*...
                ((d3(j)-(D_disp(i)))/(d3(j)-d2(j)));
        end

    end
end


% Now calculate the Damage corrected by the penalty stiffness

% Generate Damage column: D_c
for j = 1:numel(stress_ini)
    for i = 1:numel(D_disp)
        if (TSL_s(j)/ disp_ini(j)*(D(i,j)-1))/Penalty_stiff+1 >=0
           D_c(i,j) = (TSL_s(j)/ disp_ini(j)*(D(i,j)-1))/Penalty_stiff+1;
            else
            D_c(i,j)=0;
        end
        if (TSL_s(j)/ disp_ini(j)*(D(i,j)-1))/Penalty_stiff+1 >=1
           D_c(i,j) = 1;
        end
    end
end


D_disp_c = zeros(size(D_c)); % Initialize D_disp_c
nonzero_indices = zeros(1, numel(stress_ini));


for j = 1:numel(stress_ini)
    nonzero_found = false; % Flag to check if nonzero value has been found
    for i = 2:numel(D_disp)
        D_disp_c(1,j) = 0;
        if D_c(i,j) > 0
            if ~nonzero_found
                nonzero_indices(j) = i; %Store index of first nonzero value
                nonzero_found = true; % Set flag to true
```

```
                D_disp(nonzero_indices(j));
            end
            if D_c(i-1,j) == 0
                D_disp_c(i,j) = 0.0001;%+D_disp(nonzero_indices(j));
            else
                D_disp_c(i,j) = D_disp_c(i-1,j) + stepsize;
            end
        else
            D_disp_c(i,j) = 0;
        end
    end
end



%% Account for zero's in beginning of damage curve

% In the code above, the disp value of the first nonzero D_c indice was
% found. Below, we correct for this value and overwrite the D and D_c
% values.

% Generate the Damage column
for j = 1:numel(stress_ini)
    for i = 1:numel(D_disp)
        if D_disp(i) <= disp_ini(j)
            D(i,j) = 0;
        elseif D_disp(i) <= d2(j)
            if nonzero_indices(j) > 0
                 D(i,j) = 1-TSL_s(j)/(TSL_s(j)/disp_ini(j)*(D_disp(i)...
                     +D_disp(nonzero_indices(j))));
            else
                 D(i,j) = 1-TSL_s(j)/(TSL_s(j)/disp_ini(j)*(D_disp(i)));
            end
        elseif D_disp(i) >= d3(j)
            D(i,j) = 1;
        else
            if nonzero_indices(j) > 0
                D(i,j)=1-TSL_s(j)/(TSL_s(j)/disp_ini(j)*(D_disp(i)...
                    +D_disp(nonzero_indices(j))))*((d3(j)...
                    -(D_disp(i)))/(d3(j)-d2(j)));
            else
                D(i,j)=1-TSL_s(j)/(TSL_s(j)/disp_ini(j)*(D_disp(i)))...
                    *((d3(j)-(D_disp(i)))/(d3(j)-d2(j)));
            end
        end

    end
end


% Now calculate the Damage corrected by the penalty stiffness

% Generate Damage column: D_c
for j = 1:numel(stress_ini)
    for i = 1:numel(D_disp)
        if (TSL_s(j)/ disp_ini(j)*(D(i,j)-1))/Penalty_stiff+1 >=0
           D_c(i,j) = (TSL_s(j)/ disp_ini(j)*(D(i,j)-1))/Penalty_stiff+1;
            else
            D_c(i,j)=0;
        end
```

```
223         if (TSL_s(j)/ disp_ini(j)*(D(i,j)-1))/Penalty_stiff+1 >=1
224            D_c(i,j) = 1;
225         end
226     end
227 end
228
229
230 D_disp_c = zeros(size(D_c)); % Initialize D_disp_c
231
232 % Initialize array to store indices of first nonzero values
233 nonzero_indices = zeros(1, numel(stress_ini));
234
235 for j = 1:numel(stress_ini)
236     nonzero_found = false; % Flag to check if nonzero value has been found
237     for i = 2:numel(D_disp)
238         D_disp_c(1,j) = 0;
239         if D_c(i,j) > 0
240             if ~nonzero_found
241                 nonzero_indices(j) = i; %Store index of first nonzero value
242                 nonzero_found = true; % Set flag to true
243             end
244             if D_c(i-1,j) == 0
245                 D_disp_c(i,j) = 0.0001;%+D_disp(nonzero_indices(j));
246             else
247                 D_disp_c(i,j) = D_disp_c(i-1,j) + stepsize;
248             end
249         else
250             D_disp_c(i,j) = 0;
251         end
252     end
253 end
254
255
256
257 %% We now have all the data. Let's store it in a way so that it can be
258 % copied directly into ABAQUS .inp file
259
260 % Damage table
261
262 % In this forloop we loop though all the TSL's and create the damage table
263 % for each one. After this we put all the data in one big array that can
264 % be copied directly into ABAQUS
265
266 for i= 1:numel(disp_ini)
267     % Create an array with D_c values and corresponding D_disp_c values.
268     D_c2 = [D_c(:,i),D_disp_c(:,i)];
269
270     % Add a zero to each first row of the array
271     D_c3{i} = [0, 0; D_c2];
272
273
274     % Add additional columns to each array in D_c3.
275     % These columns are the field variables
276     fieldvar =  repmat(i, size(D_c3{i}, 1), 1);  % Create column with
277                                                  % values equal to i
278     D_c3{i} = [D_c3{i}, fieldvar];
279
280
281
282     % Check how much zero's the array consists of at the beginning
```

```
    zero_rows{i} = find(D_c3{i}(:, 1) == 0);

    % Make sure that there is only one row of zero's
    D_c4{i} = D_c3{i}(zero_rows{i}(end):end,:);


    % Add a small displacement that corresponds to the first nonzero
    % damage value (this is normally zero but ABAQUS requires the array
    % to begin with (0,0) instead of (value,0). )
    D_c4{i}(2,2)=0.0001;



    % Nex step is to cut off the table when the damage has reached 1.

    % Find the indices of all occurrences of 1
    one_indices = find(D_c4{i}(:,1) == 1);

    % Keep only the first occurrence of 1
    one_indices(2:end) = [];

    stored_one_indices{i} = one_indices;

    % Update the array, removing all occurrences of 1 except the first one
    D_c4{i}(stored_one_indices{i}+1:end,:) = [];

end



% End result: an array and a table with the Damage tabular data for
% multiple TSL's. The array is written to a .txt file and can be copied to
% the ABAQUS .inp file. The table can directly be copied into ABAQUS CAE.

%Make the array of Damage vs. field var.
D_c_array=vertcat(D_c4{:});

% Open a file for writing
fileID = fopen('Damage_table.inp', 'w');

% Write the header text
headerText = ['*Damage Evolution, type=DISPLACEMENT, ' ...
    'dependencies=1, softening=TABULAR\n'];
fprintf(fileID, headerText);

% Define the format string for each row
formatSpec = '  %f,  %f, ,      %.0f.\n';

% Write the array to file/command window with the specified format
fprintf(fileID, formatSpec, D_c_array');

% Close the file
fclose(fileID);

% Make a table that can be used in the ABAQUS GUI:
D_c_tab= array2table(D_c_array, 'VariableNames', ...
    {'Damage', 'Displacement','Field 1'});
```

```
% Same for the table with damage initiation data:

% Add additional column to the array of stress_ini. This column is the
% field variable
stress_iniT=stress_ini';

% Make the array of initiation stress vs. field var.
for i = 1: numel(stress_ini)
    stress_ini_array(i,:) = [stress_iniT(i), i];
end

% Open a file for writing
fileID = fopen('Initiation_table.inp', 'w');

% Write the header text
headerText = ['*Damage Initiation, criterion=MAXPS, ' ...
    'position=NONLOCAL, dependencies=1, tolerance=0.3\n'];
fprintf(fileID, headerText);

% Define the format string for each row
formatSpec = '%.0f., ,%.0f.\n';

% Write the array to file/command window with the specified format
fprintf(fileID, formatSpec, stress_ini_array');

% Close the file
fclose(fileID);


% Make a table that can be used in the ABAQUS CAE:
stress_ini_tab= array2table(stress_ini_array, 'VariableNames', ...
    {'Max Principal Stress', 'Field 1'});


%% Adjust the Abaqus model .inp file so that above and some settings are
% incorporated:

% Open the .inp file for reading
fileID = fopen('job-optimization2.inp', 'r');
if fileID == -1
    error('Unable to open input file.');
end

% Read the content of the file
fileContent = fread(fileID, '*char')'; % Read entire file as char

% Close the file
fclose(fileID);

% Define the specific text you want to locate
specificText1 = '*Material, name=Steel';
specificText2 = '*Damage Stabilization';
specificText3 = '*Static,';
specificText4 = '*Contact Output';


%---Write the INPUT commands to the inp file

% Find the position of the specific text 1 in the file content
startIndex1 = strfind(fileContent, specificText1);
```

```
403
404 if ~isempty(startIndex1)
405     % Move the startIndex1 to the next line
406     startIndex1 = startIndex1 + regexp(fileContent(startIndex1:end), ...
407         '\n', 'once');
408
409     % Define the end index for specific text 1
410     endIndex1 = strfind(fileContent(startIndex1:end), specificText2);
411     if isempty(endIndex1)
412         %nothing
413     else
414         % Adjust endIndex1 based on startIndex1
415         endIndex1 = endIndex1(1) + startIndex1 - 2; % Adjust for indexing
416         endIndex1 = endIndex1 + regexp(fileContent(endIndex1:end), '\n',
                ...
417             'once');
418     end
419
420     % Delete the block (including specific text 1 until specific text 2)
421     modifiedContent = [fileContent(1:startIndex1-1), ...
422         fileContent(endIndex1:end)];
423
424     % Define the new content to replace the deleted block for specific
425     % text 1
426     newContent1 = sprintf(['*INCLUDE, ' ...
427         'INPUT=Initiation_table.inp\n*INCLUDE, INPUT=Damage_table.inp\n']);
428
429     % Insert the new content at the position of the deleted block for
430     % specific text 1
431     modifiedContent = [modifiedContent(1:startIndex1-1), newContent1, ...
432         modifiedContent(startIndex1:end)];
433
434
435
436
437
438     %---Define a contact surface
439
440     % Define the new lines to add for the second location
441     newLines = {', interaction=IntProp-1', ['*SURFACE, ' ...
442         'name=CrackSurface, type=XFEM'], 'CRACK-1'};
443
444     % Define the text to be replaced for the second location
445     textToReplace = ', interaction=IntProp-1';
446
447     % Check if only the 2nd and 3rd cells of the newLines array
448     % are present in the file content
449     if any(contains(fileContent, newLines{2})) || ...
450             any(contains(fileContent, newLines{3}))
451         % Do nothing
452     else
453         % Replace the text with the new lines for the second location
454         modifiedContent = strrep(modifiedContent, textToReplace, ...
455             sprintf('%s\n%s\n%s', newLines{:}));
456     end
457
458
459
460
461
```

```
462     %---  Replace '*Static,' with '*Static**' and add the Controls,
463     %parameter,field command.
464
465     % Find the position of the specific text in the file content
466     startIndex2 = strfind(modifiedContent, specificText3);
467
468     if ~isempty(startIndex2)
469
470         % Find the index of the newline character immediately after
471         % startIndex1
472         newlineIndex = regexp(modifiedContent(startIndex2:end), '\n', ...
473             'once');
474         if isempty(newlineIndex)
475
476             return; % Exit if newline character not found
477         end
478
479         % Define the end index for specific text 1
480         endIndex2 = startIndex2 + newlineIndex;
481
482         % Find the index of the second newline character after
483         % startIndex2
484         secondNewlineIndex = regexp(modifiedContent(endIndex2:end), ...
485             '\n', 'once');
486         if isempty(secondNewlineIndex)
487
488             return; % Exit if second newline character not found
489         end
490
491         % Update the end index to skip two lines
492         endIndex2 = endIndex2 + secondNewlineIndex;
493
494         % Delete the block
495         modifiedContent = [modifiedContent(1:startIndex2-1), ...
496             modifiedContent(endIndex2:end)];
497
498         % Define the new content to replace the deleted block for
499         % specific text 1
500         newContent2 = sprintf(['*Static\n1e-05, 1., 1e-10, ' ...
501             '0.001\n**\n*Controls, parameters=field\n,1.0\n ']);
502
503         % Insert the new content at the position of the deleted block
504         % for specific text 1
505         modifiedContent = [modifiedContent(1:startIndex2-1), ...
506             newContent2, modifiedContent(startIndex2:end)];
507
508     else
509         %nothing
510     end
511
512
513     %---- Deactivate the controls,reset command
514     % Check if **Controls, reset exists in the file content
515     if ~contains(modifiedContent, '**Controls, reset')
516         % Replace *Controls, reset with **Controls, reset
517         modifiedContent = strrep(modifiedContent, '*Controls, reset', ...
518             '**Controls, reset');
519     end
520
521
```

```
522
523
524     %---Add XFEM contact surface output to requested output
525
526     % Replace '*Contact Output' and the two lines below it
527     startIndex4 = strfind(modifiedContent, specificText4);
528     if ~isempty(startIndex4)
529         % Find the index of the newline character immediately after
530         % startIndex4
531         newlineIndex4 = regexp(modifiedContent(startIndex4:end), '\n', ...
532             'once');
533         if isempty(newlineIndex4)
534
535             return; % Exit if newline character not found
536         end
537         % Define the end index for specific text 4
538         endIndex4 = startIndex4 + newlineIndex4;
539
540         % Skip two lines
541         for i = 1:1
542             endIndex4 = endIndex4 + ...
543             regexp(modifiedContent(endIndex4:end), '\n', 'once');
544         end
545
546         % Delete the lines
547         modifiedContent = [modifiedContent(1:startIndex4-1), ...
548             modifiedContent(endIndex4+1:end)];
549
550         % Insert the new content
551         newContent4 = sprintf(['*Contact Output\nCDISP, CSDMG, ' ...
552             'CSTRESS, CRKDISP, CSDMG, CRKSTRESS\n*']);
553         modifiedContent = [modifiedContent(1:startIndex4-1), ...
554             newContent4, modifiedContent(startIndex4:end)];
555
556         disp('inp. file successfully modified');
557     end
558 end
559
560
561
562 %---
563
564 % Overwrite the original file with the modified content
565 fileID = fopen('job-optimization2.inp', 'w');
566 if fileID == -1
567     error('Unable to open output file.');
568 end
569
570 % Write the modified content back to the file
571 fwrite(fileID, modifiedContent, 'char');
572
573 % Close the file
574 fclose(fileID);
575
576
577
578
579 %% Make pictures of Damage curves and TSL's
580
581 % Go from plasticity model to material model (include the elastic part)
```

```
PM_array= table2array(PM);
MatModel= [0,0 ; PM_array];
MatModel(2,2)=Y/E;


% Add points to the disp column
valuesToAdd = MatModel(end) + (1:add2mat) * 0.1;
disp_mat = [MatModel(:,2); valuesToAdd'];


for j = 1:numel(stress_ini)
    for i = 1:length(disp_mat)
        if disp_mat(i) <= disp_ini(j)
            Dm(i,j) = 0;
        elseif disp_mat(i) <= d2(j)
            Dm(i,j) = 1-TSL_s(j)/(TSL_stiff(j)*(disp_mat(i)));
        elseif disp_mat(i) >= d3(j)
            Dm(i,j) = 1;
        else
            Dm(i,j)=1-TSL_s(j)/(TSL_stiff(j)*(disp_mat(i)))*((d3(j)- ...
                (disp_mat(i)))/(d3(j)-d2(j)));
        end

        if Dm(i,j) == 0
            stress(i,j) = MatModel(i,1);
        else
            stress(i,j) = (1-Dm(i,j))*TSL_stiff(j)*disp_mat(i);
        end
    end
end



%% Run abaqus
try
    % Define the paths to the Abaqus executable and input file
    abaqusPath = 'C:\SIMULIA\Commands\abq2023.bat';
    lck_file = 'D:\Abq_temp\job-optimization2.lck';
    inputFile = 'job-optimization2';
    userSubroutine = 'subroutine4.for';


    % Construct the command to run Abaqus
    command = sprintf('%s job=%s user=%s', abaqusPath, inputFile, ...
        userSubroutine);

    % Execute the command
    status = system(command);

    disp('Short pause to check if licences are available...');
    pause(10)


    if status ~= 0
        error('Abaqus execution failed.');
        pause(10)
    else
        disp('Abaqus execution successful.');
    end
catch ME
```

```
642     disp('Error occurred:');
643     disp(ME.message);
644 end
645 
646 
647 
648 %% Once the simulation is finished, create a flag file
649 % Define ODB file name
650 
651 odb_file = 'D:\Abq_temp\job-optimization2.odb'; % for Femto
652 
653 
654 
655 % If there is a licence issue and the job is queued, this loop
656 % waits until there are licences again
657 while exist(lck_file, 'file') == 2 && exist(odb_file, 'file') ~= 2
658     disp('Waiting for licenses...');
659     pause(40); % Wait before checking again
660 end
661 
662 
663 
664 % Get initial size of the ODB file
665 initial_size = 0;
666 while initial_size == 0
667     % Get the size of the ODB file
668     initial_dir_info = dir(odb_file);
669     initial_size = initial_dir_info.bytes;
670     disp(['Initial size: ', num2str(initial_size)]);
671     pause(15); % Wait for 10 seconds before checking again
672 end
673 
674 % Wait until the size of the ODB file stops increasing
675 stable_time = 0;
676 while stable_time < 3*60*60 % Wait for a maximum of 3 hours
677     % Wait for 15 seconds before checking again
678     pause(15);
679 
680     % Get the current size of the ODB file
681     current_dir_info = dir(odb_file);
682     current_size = current_dir_info.bytes;
683     disp(['Current size: ', num2str(current_size)]);
684 
685     % Check if the size remains the same for a certain duration
686     if current_size == initial_size
687         break;
688     else
689         stable_time = 0; % Reset the stable time if the size changes
690         initial_size = current_size; % Update initial size
691     end
692 
693     % Check if stable time reached 3 hours
694     if stable_time >= 3*60*60
695         break; % Exit the loop if stable time reached
696     end
697 end
698 
699 
700 % Once the simulation is finished, create the flag file
701 fid = fopen('simulation_complete.flag', 'w');
```

```
fclose(fid);
disp('Simulation complete. Flag file created.');


%% Plot FE and experimental curves and write the error to a file

system('abaqus cae noGUI="python2.py"');
pause(60)




%_____________________________________
% Calculate the crack propagation vs. displacement diagram:
% In order to plot this diagram, I used a trick. I know the position of
% the nodes along the crack length. I use this when calculating the crack
% propagation. For example, when a node that is 200 mm from the initial
% crack tip reaches a damage value of 1, the crack has propagated 200 mm.
% So, in the following code I plot for each node along the crack length the
% applied displacement to the damage value of that node. Then I look up
% at what displacement value each node reaches D=1 and save these values.

%---
node1 = readtable('XYdata_node_558.txt'); % Read the FE output table
% for this node
% Convert the second column of the table to a numeric array
y_values_1 = node1{:, 2};
% Find the moment where D=1
index_1 = find(y_values_1 == 1, 1, 'first');
% Get the corresponding displacement-value
corresponding_x_1 = node1{index_1, 1};

%---
node2 = readtable('XYdata_node_562.txt'); % Read the FE output table
% for this node
% Convert the second column of the table to a numeric array
y_values_2 = node2{:, 2};
% Find the moment where D=1
index_2 = find(y_values_2 == 1, 1, 'first');
% Get the corresponding displacement-value
corresponding_x_2 = node2{index_2, 1};

%---
node3 = readtable('XYdata_node_566.txt'); % Read the FE output table
% for this node
% Convert the second column of the table to a numeric array
y_values_3 = node3{:, 2};
% Find the moment where D=1
index_3 = find(y_values_3 == 1, 1, 'first');
% Get the corresponding displacement-value
corresponding_x_3 = node3{index_3, 1};

%---
node4 = readtable('XYdata_node_570.txt'); % Read the FE output table
% for this node
% Convert the second column of the table to a numeric array
y_values_4 = node4{:, 2};
% Find the moment where D=1
index_4 = find(y_values_4 == 1, 1, 'first');
% Get the corresponding displacement-value
```

```
762 corresponding_x_4 = node4{index_4, 1};
763
764 %---
765 node5 = readtable('XYdata_node_574.txt'); % Read the FE output table
766 % for this node
767 % Convert the second column of the table to a numeric array
768 y_values_5 = node5{:, 2};
769 % Find the moment where D=1
770 index_5 = find(y_values_5 == 1, 1, 'first');
771 % Get the corresponding displacement-value
772 corresponding_x_5 = node5{index_5, 1};
773
774 %---
775 node6 = readtable('XYdata_node_578.txt'); % Read the FE output table
776 % for this node
777 % Convert the second column of the table to a numeric array
778 y_values_6 = node6{:, 2};
779 % Find the moment where D=1
780 index_6 = find(y_values_6 == 1, 1, 'first');
781 % Get the corresponding displacement-value
782 corresponding_x_6 = node6{index_6, 1};
783
784 %---
785 node7 = readtable('XYdata_node_582.txt'); % Read the FE output table
786 % for this node
787 % Convert the second column of the table to a numeric array
788 y_values_7 = node7{:, 2};
789 % Find the moment where D=1
790 index_7 = find(y_values_7 == 1, 1, 'first');
791 % Get the corresponding displacement-value
792 corresponding_x_7 = node7{index_7, 1};
793
794 %---
795 node8 = readtable('XYdata_node_586.txt'); % Read the FE output table
796 % for this node
797 % Convert the second column of the table to a numeric array
798 y_values_8 = node8{:, 2};
799 % Find the moment where D=1
800 index_8 = find(y_values_8 == 1, 1, 'first');
801 % Get the corresponding displacement-value
802 corresponding_x_8 = node8{index_8, 1};
803
804 %---
805 node9 = readtable('XYdata_node_590.txt'); % Read the FE output table
806 % for this node
807 % Convert the second column of the table to a numeric array
808 y_values_9 = node9{:, 2};
809 % Find the moment where D=1
810 index_9 = find(y_values_9 == 1, 1, 'first');
811 % Get the corresponding displacement-value
812 corresponding_x_9 = node9{index_9, 1};
813
814 %---
815 node10 = readtable('XYdata_node_594.txt'); % Read the FE output table
816 % for this node
817 % Convert the second column of the table to a numeric array
818 y_values_10 = node10{:, 2};
819 % Find the moment where D=1
820 index_10 = find(y_values_10 == 1, 1, 'first');
821 % Get the corresponding displacement-value
```

```
corresponding_x_10 = node10{index_10, 1};

%---
node11 = readtable('XYdata_node_595.txt'); % Read the FE output table
% for this node
% Convert the second column of the table to a numeric array
y_values_11 = node11{:, 2};
% Find the moment where D=1
index_11 = find(y_values_11 == 1, 1, 'first');
% Get the corresponding displacement-value
corresponding_x_11 = node11{index_11, 1};




%---RESULT
% Calculated crack propagation- displacement curve by abaqus:
crackprop_y_predicted=[5 20 40 60 80 100 120 140 160 180];
crackprop_x_predicted=[corresponding_x_2 corresponding_x_3 ...
    corresponding_x_4 corresponding_x_5 corresponding_x_6 ...
    corresponding_x_7 corresponding_x_8 corresponding_x_9 ...
    corresponding_x_10 corresponding_x_11 ];




% Load target curve data
targetData2 = table2array(readtable('objectivecurve2.txt'));

% Extract x and y data from target curve
target_crackprop_x = targetData2(:, 1);
target_crackprop_y = targetData2(:, 2);
target_crackprop_x = interp1(target_crackprop_y, target_crackprop_x, ...
    crackprop_y_predicted);
target_crackprop_x = target_crackprop_x(1:8);




 if length(crackprop_x_predicted)> 8
     crackprop_x_predicted = crackprop_x_predicted...
         (1:length(target_crackprop_x));
 end


 % If the simulation is ended prematurely, an penalty will be added to the
 % error.
 penalty = 0;
 if length(crackprop_x_predicted)< length(target_crackprop_x)
     target_crackprop_x = target_crackprop_x...
         (1:length(crackprop_x_predicted));
     penalty = 20;
 end


 % You can also add a penalty to the error if you want a increasing or
 % decreasing behaviour:
 % if params(1)>params(2)
 %     penalty = 100;
```

```
 % end
 %
 % if params(3)<params(4)
 %     penalty = 100;
 % end


 crackprop_y_predicted=crackprop_y_predicted...
     (1:length(crackprop_x_predicted));




 %_____________________________________
% Calculated Force displacement curve by Abaqus:

XYdata = importdata('XYdata.rpt');

x_predicted = XYdata.data(:,1);
y_predicted = XYdata.data(:,2)*2*0.733;

% Load target curve data
targetData = table2array(readtable('objectivecurve.txt'));

% Extract x and y data from target curve
target_x = targetData(:, 1);
target_y = targetData(:, 2)*1000;
target_y = interp1(target_x, target_y, x_predicted);

%__________




% Calculate error
error1 = (sum(abs(target_y - y_predicted)) / sum(target_y) * 100);
error2 = (sum(abs(target_crackprop_x - crackprop_x_predicted)) / ...
    sum(target_crackprop_x) * 100);

error = error1 + error2 + penalty;




% Next, I want to write some useful data to .txt files, to monitor the
% behaviour of the error and parameters during the optimization process:


% Get minError value from .txt file
error_array = importdata('minError.txt');
error_xvalues = 1:numel(error_array);

%-----

% Open the text file in append mode (so existing contents are
% not overwritten)
fileID = fopen('error.txt', 'a');

% Write the value to the file
fprintf(fileID, '%.2f\n', error);
```

```
942
943 % Close the file
944 fclose(fileID);
945
946
947 error_array2 = importdata('error.txt');
948 error_xvalues2 = 1:numel(error_array2);
949
950
951 %-----
952
953 % Open the text file in append mode (so existing contents are
954 % not overwritten)
955 fileID = fopen('error_breakdown.txt', 'a');
956
957 % Check if the file was opened successfully
958 if fileID == -1
959 error('Unable to open file for writing.');
960 end
961
962 % Write the values to the file in the desired format
963 fprintf(fileID, ['error1= %.2f  error2 (*3)= %.2f  ' ...
964     'total error= %.2f\n'], error1, error2, error);
965
966 % Close the file
967 fclose(fileID);
968
969
970 %-----
971
972
973 % Write parameters to file
974 fileID = fopen('parameters.txt', 'a');
975
976 % Check if the file was opened successfully
977 if fileID == -1
978 error('Unable to open file for writing.');
979 end
980
981 % Write the value to the file
982 fprintf(fileID, '----\n');
983 fprintf(fileID, 'produced error with the above input:\n');
984 fprintf(fileID, '%.2f\n', error_array2(end));
985
986 % Close the file
987 fclose(fileID);
988
989 %% Plot the data
990
991
992 subplot(3,3,1);
993 plot(disp_mat, stress);
994 xlabel('Separation [mm]');
995 ylabel('Traction [MPa]');
996 title('TSLs');
997
998
999 subplot(3,3,2);
1000 plot(disp_mat, Dm);
1001 xlabel('Displacement [mm]');
```

```
ylabel('Damage [-]');
title('Damage');

subplot(3,3,3);
plot(x_predicted,y_predicted, x_predicted, target_y);
xlabel('Displacement [mm]');
ylabel('Force [N]');
title('F-d curve');
xlim([0, 50]);
legend ('FE', 'Experiment','Location','best');

subplot(3,3,4);
plot(crackprop_x_predicted,crackprop_y_predicted,target_crackprop_x, ...
    crackprop_y_predicted);
xlabel('Displacement [mm]');
ylabel('Crack Propagation [mm]');
title('Crack propagation curve');
xlim([0, 60]);
legend ('FE', 'Experiment','Location','best');


%---- and the input data as well:
params_x=[0.35 0.6];
subplot(3,3,5);
plot(params_x, params(1:2));
xlabel('TRIAX');
ylabel('Initiation stress [MPa]');
title('Initiation stress vs. TRIAX');


subplot(3,3,6);
plot(params_x, params(3:4));
xlabel('TRIAX');
ylabel('d2 [mm]');
title('d2 vs. TRIAX');


% subplot(3,3,7);
% plot(params_x, params(5:6));
% xlabel('TRIAX');
% ylabel('d3 [mm]');
% title('d3 vs. TRIAX');


subplot(3,3,8);
plot(error_xvalues,error_array);
xlabel('Number of iterations');
ylabel('Error [%]');
title('Global min.error');

subplot(3,3,9);
plot(error_xvalues2,error_array2);
xlabel('Number of iterations');
ylabel('Error [%]');
title('Error per run');

% Set the background color of the whole figure to white
set(gcf, 'Color', 'w');
```

Appendix D

Optimization code

The optimization code is provided below. Two separate scripts were used: Python code that extracts output data from Abaqus for each iteration and a Matlab script with the Nelder Mead algorithm.

The Python code:

```
# Python script (python2.py) executed by Abaqus
try:
    # Initialize Abaqus
    from abaqus import *
    from abaqusConstants import *
    from caeModules import *
    from driverUtils import executeOnCaeStartup
    executeOnCaeStartup()

    # Open CAE file
    openMdb(pathName='Optimization2_20.cae')

    # Open output database
    o = session.openOdb(name='job-optimization2.odb')

    # Create displacement data
    odb = session.odbs['job-optimization2.odb']
    xy_displacement = xyPlot.XYDataFromHistory(odb=odb, outputVariableName=
        'Spatial displacement: U2 PI: rootAssembly Node 1 in NSET SET-42',
        steps=('Step-1', ), suppressQuery=True, __linkedVpName__='Viewport:
        1')

    # Create reaction force data
    xy_reaction_force = xyPlot.XYDataFromHistory(odb=odb,
        outputVariableName='Reaction force: RF2 PI: rootAssembly Node 1 in
        NSET SET-42', steps=('Step-1', ), suppressQuery=True,
        __linkedVpName__='Viewport: 1')

    # Combine the displacement and reaction force data
    combined_data = combine(xy_displacement, xy_reaction_force)

    # Save the combined data to a report
    session.xyReportOptions.setValues(layout=SEPARATE_TABLES)
    session.writeXYReport(fileName='XYdata.rpt', appendMode=OFF, xyData=(
        combined_data,))

    # List of node labels to process
```

```
    node_labels = [558, 562, 566, 570, 574, 578, 582, 586, 590, 594, 595]
        # Add your node labels here

    # Loop through each node label to extract and save data to separate .
        txt files
    for node_label in node_labels:
        field_output_name = 'CSDMG    ASSEMBLY_CRACKSURFACE/
            ASSEMBLY_CRACKSURFACE'
        csdmgData = []

        # Verify the field output exists
        step = odb.steps['Step-1']
        if field_output_name not in step.frames[0].fieldOutputs:
            raise ValueError("Field output '{}' not found in the output
                database".format(field_output_name))

        for frame in step.frames:
            field_output = frame.fieldOutputs[field_output_name]
            node_found = False
            for value in field_output.values:
                if value.nodeLabel == node_label:
                    csdmgData.append((frame.frameValue, value.data))
                    node_found = True
                    break
            if not node_found:
                raise ValueError("Node label {} not found in frame {}".
                    format(node_label, frame.frameId))

        # Convert csdmgData to XYData format
        xyCsdmg = xyPlot.XYData(data=csdmgData, sourceDescription='Field
            output:CSDMG    ASSEMBLY_CRACKSURFACE/ASSEMBLY_CRACKSURFACE')

        # Combine the displacement and reaction force data
        combined_data_node = combine(xy_displacement, xyCsdmg)

        # Write the combined data to a .txt file
        with open('XYdata_node_{}.txt'.format(node_label), 'w') as file:
            for xy in combined_data_node:
                file.write('{} {}\n'.format(xy[0], xy[1]))

        del session.xyDataObjects[xyCsdmg.name]

except Exception as e:
    # If an error occurs during simulation, create an error flag file
    open('abaqus_error.flag', 'w').close()
    # Print the error message for debugging
    import traceback
    traceback.print_exc()
    raise e
```

The Matlab code:

```
% clc
% clear

% Load target curve data
targetData = table2array(readtable('objectivecurve.txt'));

```

```
% Extract x and y data from target curve
target_x = targetData(:, 1);
target_y = targetData(:, 2);

% Define initial guess and tolerance
num_points = 4; % Number of points to interpolate
x_predicted = linspace(min(target_x), max(target_x), num_points);
initialGuess = [415.267684 437.535569 15.267330 3.196011];

tol = 0.3; % Tolerance for convergence
alpha = 1; % Reflection coefficient: standard value = 1
gamma = 2; % Expansion coefficient: standard value = 2
rho = 0.5; % Contraction coefficient: standard value = 1/2
sigma = 0.5; % Shrink coefficient: standard value = 1/2
maxIterations = 1500000000;

% Define lower and upper bounds for each parameter
lb = [275 275 0.1 0.1]; % value for min stress must be higher than yield
ub = [455 455 40 40]; % value for max stress must be lower than UTS

% Run Nelder-Mead optimization
[optimalParams, minError] = nelderMead(@objective, initialGuess, tol, ...
    alpha, gamma, rho, sigma, maxIterations, lb, ub);

% Display the optimal parameters and minimum error
disp('Optimal parameters:');
disp(optimalParams);
disp('Minimum error:');
disp(minError);

% Calculated Force displacement curve by abaqus:
XYdata = importdata('XYdata.rpt');

x_predicted = XYdata.data(:,1);
y_predicted = XYdata.data(:,2) * 2 * 0.733;

target_x = targetData(:, 1);
target_y = targetData(:, 2) * 1000;
target_y = interp1(target_x, target_y, x_predicted);

% Plot the target curve and the fitted curve
figure;
plot(x_predicted, target_y, 'bo', x_predicted, y_predicted, 'r-');
legend('Target Data', 'Optimized Curve', 'Location', 'best');
xlabel('x');
ylabel('y');
title('Curve Fitting using Nelder-Mead Optimization');

% Define the objective function
function error = objective(params)
    try
        % Define lower and upper bounds for each parameter
        lb = [275 275 0.1  0.1]; % value for min stress must be higher
        % than yield
        ub = [455 455 40 40]; % value for max stress must be lower than
        % max. defined point at stress strain diagram

        % Enforce constraints before saving parameters
        params = enforceConstraints(params, lb, ub);

```

```
        % Save input parameters to a MAT-file
        num_points = 2;
        params_x = [0 1];
        save('data.mat', 'params', 'params_x');

        run('TSLtool.m');

        % Wait for the completion of the Abaqus simulation
        while ~exist('simulation_complete.flag', 'file')
            pause(1); % Wait for 1 second before checking again
        end

        % Calculated Force displacement curve by abaqus:
        system('abaqus cae noGUI="python2.py"');
        pause(60)

        % After running Abaqus
        if exist('abaqus_success.flag', 'file')
            % Abaqus ran successfully, proceed with reading the output
            XYdata = importdata('XYdata.rpt');


            % Calculated Force displacement curve by abaqus:
            XYdata = importdata('XYdata.rpt');
            x_predicted = XYdata.data(:,1);
            y_predicted = XYdata.data(:,2) * 2 * 0.733;

            % Load target curve data
            targetData = table2array(readtable('objectivecurve.txt'));
            target_x = targetData(:, 1);
            target_y = targetData(:, 2) * 1000;
            target_y = interp1(target_x, target_y, x_predicted);



            %_____________________________________
            % Calculate the crack propagation vs. displacement diagram:
            %---
            node1 = readtable('XYdata_node_558.txt'); % Read the table
            % Convert the second column of the table to a numeric array
            y_values_1 = node1{:, 2};
            index_1 = find(y_values_1 == 1, 1, 'first');
            % Get the corresponding x-value
            corresponding_x_1 = node1{index_1, 1};

            %---

            node2 = readtable('XYdata_node_562.txt'); % Read the table
            % Convert the second column of the table to a numeric array
            y_values_2 = node2{:, 2};
            index_2 = find(y_values_2 == 1, 1, 'first');
            % Get the corresponding x-value
            corresponding_x_2 = node2{index_2, 1};

            %---
            node3 = readtable('XYdata_node_566.txt'); % Read the table
            % Convert the second column of the table to a numeric array
            y_values_3 = node3{:, 2};
            index_3 = find(y_values_3 == 1, 1, 'first');
            % Get the corresponding x-value
```

```
corresponding_x_3 = node3{index_3, 1};

%---
node4 = readtable('XYdata_node_570.txt'); % Read the table
% Convert the second column of the table to a numeric array
y_values_4 = node4{:, 2};
index_4 = find(y_values_4 == 1, 1, 'first');
% Get the corresponding x-value
corresponding_x_4 = node4{index_4, 1};

%---
node5 = readtable('XYdata_node_574.txt'); % Read the table
% Convert the second column of the table to a numeric array
y_values_5 = node5{:, 2};
index_5 = find(y_values_5 == 1, 1, 'first');
% Get the corresponding x-value
corresponding_x_5 = node5{index_5, 1};

%---
node6 = readtable('XYdata_node_578.txt'); % Read the table
% Convert the second column of the table to a numeric array
y_values_6 = node6{:, 2};
index_6 = find(y_values_6 == 1, 1, 'first');
% Get the corresponding x-value
corresponding_x_6 = node6{index_6, 1};

%---
node7 = readtable('XYdata_node_582.txt'); % Read the table
% Convert the second column of the table to a numeric array
y_values_7 = node7{:, 2};
index_7 = find(y_values_7 == 1, 1, 'first');
% Get the corresponding x-value
corresponding_x_7 = node7{index_7, 1};

%---
node8 = readtable('XYdata_node_586.txt'); % Read the table
% Convert the second column of the table to a numeric array
y_values_8 = node8{:, 2};
index_8 = find(y_values_8 == 1, 1, 'first');
% Get the corresponding x-value
corresponding_x_8 = node8{index_8, 1};

%---
node9 = readtable('XYdata_node_590.txt'); % Read the table
% Convert the second column of the table to a numeric array
y_values_9 = node9{:, 2};
index_9 = find(y_values_9 == 1, 1, 'first');
% Get the corresponding x-value
corresponding_x_9 = node9{index_9, 1};

%---
node10 = readtable('XYdata_node_594.txt'); % Read the table
% Convert the second column of the table to a numeric array
y_values_10 = node10{:, 2};
index_10 = find(y_values_10 == 1, 1, 'first');
% Get the corresponding x-value
corresponding_x_10 = node10{index_10, 1};

%---
node11 = readtable('XYdata_node_595.txt'); % Read the table
```

```
% Convert the second column of the table to a numeric array
y_values_11 = node11{:, 2};
index_11 = find(y_values_11 == 1, 1, 'first');
% Get the corresponding x-value
corresponding_x_11 = node11{index_11, 1};




%---
% Calculated crack propagation- displacement curve by abaqus:
crackprop_y_predicted=[5 20 40 60 80 100 120 140 160 180];
crackprop_x_predicted=[corresponding_x_2 corresponding_x_3 ...
    corresponding_x_4 corresponding_x_5 corresponding_x_6 ...
    corresponding_x_7 corresponding_x_8 corresponding_x_9 ...
    corresponding_x_10 corresponding_x_11 ];




% Load target curve data
targetData2 = table2array(readtable('objectivecurve2.txt'));

% Extract x and y data from target curve
target_crackprop_x = targetData2(:, 1);
target_crackprop_y = targetData2(:, 2);
target_crackprop_x = interp1(target_crackprop_y, ...
    target_crackprop_x, crackprop_y_predicted);
target_crackprop_x = target_crackprop_x(1:8);


if length(crackprop_x_predicted)> 8
    crackprop_x_predicted = crackprop_x_predicted...
    (1:length(target_crackprop_x));
end

penalty = 0;
if length(crackprop_x_predicted)< length(target_crackprop_x)
    target_crackprop_x = target_crackprop_x...
        (1:length(crackprop_x_predicted));
    penalty = 20;
end

% if params(1)>params(2)
%     penalty = 100;
% end
%
% if params(3)<params(4)
%     penalty = 100;
% end


crackprop_y_predicted=crackprop_y_predicted...
    (1:length(crackprop_x_predicted));


% Calculate error
error1 = (sum(abs(target_y-y_predicted))/sum(target_y)*100);
error2 = (sum(abs(target_crackprop_x-crackprop_x_predicted))...
    / sum(target_crackprop_x) * 100);
```

```
247
248             error = error1 + error2 + penalty;
249
250         else
251             % Abaqus encountered an error, set error value to a high value
252             error = 100;
253         end
254
255     catch
256         % If an error occurs in Matlab, set error to a high value
257         error = 100;
258     end
259 end
260
261
262 %%
263 function [optimalParams, minError] = nelderMead(objective, initialGuess,
        ...
264     tol, alpha, gamma, rho, sigma, maxIterations, lb, ub)
265     % Define initial simplex
266     n = length(initialGuess);
267     simplex = zeros(n+1, n);
268
269     % Adjust initial guess to stay within bounds
270     initialGuess = max(min(initialGuess, ub), lb);
271     simplex(1,:) = initialGuess;
272
273     for i = 1:n
274         % Perturb each dimension of the initial guess by 0.25 times
275         % its value
276         perturbation = 0.25 * initialGuess(i);
277         simplex(i+1,:) = initialGuess;
278         simplex(i+1,i) = simplex(i+1,i) + perturbation;
279         % Ensure points stay within bounds and satisfy constraints
280         simplex(i+1,:) = enforceConstraints(simplex(i+1,:), lb, ub);
281     end
282
283     % Evaluate objective function for initial simplex
284     f = zeros(n+1, 1);
285     for i = 1:n+1
286         f(i) = objective(simplex(i, :));
287     end
288
289     % Start iterations
290     for iter = 1:maxIterations
291         % Sort the simplex based on objective function values
292         [f, order] = sort(f);
293         simplex = simplex(order,:);
294
295         % Compute centroid of best vertices (excluding worst)
296         x_0 = mean(simplex(1:end-1,:), 1);
297
298         % Reflection
299         x_r = x_0 + alpha*(x_0 - simplex(end,:));
300         x_r = enforceConstraints(x_r, lb, ub); % Ensure reflection stays
301         % within bounds and constraints
302         f_r = objective(x_r);
303
304         if f_r < f(1)
305             % Expansion
```

```
306             x_e = x_0 + gamma*(x_r - x_0);
307             x_e = enforceConstraints(x_e, lb, ub); % Ensure expansion
308             % stays within bounds and constraints
309             f_e = objective(x_e);
310 
311             if f_e < f_r
312                 simplex(end,:) = x_e;
313                 f(end) = f_e;
314             else
315                 simplex(end,:) = x_r;
316                 f(end) = f_r;
317             end
318         elseif f_r >= f(1) && f_r < f(end-1)
319             simplex(end,:) = x_r;
320             f(end) = f_r;
321         elseif f_r >= f(end-1) && f_r < f(end)
322             % Outside contraction
323             x_c = x_0 + rho*(x_r - x_0);
324             x_c = enforceConstraints(x_c, lb, ub); % Ensure contraction
325             % stays within bounds and constraints
326             f_c = objective(x_c);
327 
328             if f_c <= f_r
329                 simplex(end,:) = x_c;
330                 f(end) = f_c;
331             else
332                 % Shrink
333                 best_vertex = simplex(1,:);
334                 for i = 2:n+1
335                     simplex(i,:) = best_vertex + sigma*(simplex(i,:) - ...
336                         best_vertex);
337                     % Ensure shrink stays within bounds and constraints
338                     simplex(i,:) = enforceConstraints(simplex(i,:), lb,ub);
339                     f(i) = objective(simplex(i,:));
340                 end
341             end
342         else
343             % Inside contraction
344             x_c = x_0 - 0.5*(x_r - x_0);
345             % Ensure contraction stays within bounds and constraints
346             x_c = enforceConstraints(x_c, lb, ub);
347             f_c = objective(x_c);
348 
349             if f_c < f(end)
350                 simplex(end,:) = x_c;
351                 f(end) = f_c;
352             else
353                 % Shrink
354                 best_vertex = simplex(1,:);
355                 for i = 2:n+1
356                     simplex(i,:) = best_vertex + sigma*(simplex(i,:)...
357                         - best_vertex);
358                     % Ensure shrink stays within bounds and constraints
359                     simplex(i,:) = enforceConstraints(simplex(i,:), ...
360                         lb, ub);
361                     f(i) = objective(simplex(i,:));
362                 end
363             end
364         end
365 
```

```
366
367         %write error
368         minError = f(1);
369         % Open the text file in append mode (so existing contents are
370         % not overwritten)
371         fileID = fopen('minError.txt', 'a');
372
373         % Check if the file was opened successfully
374         if fileID == -1
375         error('Unable to open file for writing.');
376         end
377
378         % Write the value to the file
379         fprintf(fileID, '%.2f\n', minError);
380
381         % Close the file
382         fclose(fileID);
383
384         %error_array = importdata('minError.txt');
385         %error_xvalues = 1:numel(error_array);
386
387
388         % Check convergence based on change in minimum error
389         if minError < 3
390         break;  % Terminate optimization if change in minimum error is
391         % below tolerance
392         end
393
394     end
395
396     % Return optimal parameters and error
397     optimalParams = simplex(1,:);
398     minError = f(1);
399 end
400
401 function params = enforceConstraints(params, lb, ub)
402     % Ensure parameters stay within bounds
403     params = max(min(params, ub), lb);
404     % Ensure params(1) <= params(2)
405     % params(1) = min(params(1), params(2));
406     %
407     % % Ensure params(4) <= params(3)
408     % params(4) = min(params(3), params(4));
409 end
```