Chair of Mechanics

Doctoral Thesis

Process Models for the Manufacturing of Railway Components

Dipl.-Ing. Jakob Bialowas, BSc

May 2023

MONTANUNIVERSITÄT LEOBEN
www.unileoben.ac.at

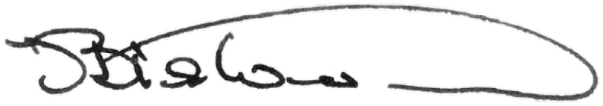
EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt, und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Ich erkläre, dass ich die Richtlinien des Senats der Montanuniversität Leoben zu "Gute wissenschaftliche Praxis" gelesen, verstanden und befolgt habe.

Weiters erkläre ich, dass die elektronische und gedruckte Version der eingereichten wissenschaftlichen Abschlussarbeit formal und inhaltlich identisch sind.

Datum 26.05.2023



Unterschrift Verfasser/in
Jakob Bialowas

Acknowledgements

Das sind hier die letzten Zeilen, die ich im Zuge des Verfassens dieser Arbeit schreibe. Ich blicke auf vier lehrreiche Jahre zurück und möchte mich bei allen, die mich in dieser Zeit begleitet haben ganz herzlich bedanken. Zuallererst gilt mein Dank den Menschen, die mir am nächsten sind, meiner großen Liebe Anja und meiner Familie. Liebe Anja, vielen Dank für alles. Die glücklichen Momente, die wir gemeinsam abseits der Arbeit verbracht haben, die unzähligen schönen Erinnerungen an gemeinsame Reisen und dass du mich auch durch diese Reise des Doktoratsstudiums begleitet hast. Danke!

Chciałbym również bardzo podziękować mojej rodzinie. Moi rodzice Elżbieta i Krzysztof położyli fundamenty pod moją edukację i zawsze zachęcali mnie do dążenia do gwiazd. Myślę też, że moje zainteresowanie transportem kolejowym narodziło się w dzieciństwie podczas niezliczonych podróży pociągiem. Chciałbym również podziękować dwóm nadzwyczajnym kobietom w moim życiu, mojej młodszej siostrze Natalii, która zawsze była dla mnie wielkim wzorem, oraz mojej babci Krystynie, która obchodzi właśnie dziewięćdziesiąte urodziny i odgrywała kluczową rolę w moim wychowaniu. Dziękuję!

Without a place to work, doctoral studies and related research would not be possible. For me, that place was MCL. Here I was able to develop personally and professionally and learn a lot. Much of the credit for this goes to Jürgen and Hans-Peter, who managed a very diverse and extensive project and still found the time for detailed expert advice. What would work be without colleagues? I don't know how it was possible, but I simply had the best ones. I want to thank Sebastian and Dominik,who were a great help in creating complex finite element models, and Daniel for his contribution to my dissertation. Only through the preliminary work in material modeling by Manuel, Konstantin, Silvia and Christian I was able to get this far in this field myself. Peter, Anatol and Tristan taught me a lot about python programming and the science of coffee brewing. Thank you very much! To my office colleagues, with whom I was lucky to share an office, I would like to say the biggest thanks regarding the motivation and the more than friendly working atmosphere. Georg, thank you for the funniest moments of our countless working days together. You have always been a role model for me in the way you work. Timna, without you I would have been lost the last weeks and months. Thank you for your attention to detail, you have contributed greatly to the quality of my written work. Thank you so much!

Es geht im Leben auch nicht immer nur um Arbeit. Was wirklich zählt ist das Vergnügen. An dieser Stelle möchte ich mich bei all meinen Freunden und Studienkollegen bedanken, mit denen ich unzählige Berggipfel erklommen habe und tausende Kilometer im besten Zug der Welt, dem Weißbier-express gefahren bin. Danke Thomas, Alex, Claus, Clemens, Hari, Marlene... Ihr bereichert das Leben ungemein.

A research project needs partnerships. In the DRaCo project, I was able to enjoy a very good partnership with industry and academia. From my colleagues at Siemens, Lucchini RS, HEGENSCHEIDT-MFD and ÖBB I was able to learn an incredible amount about the manufacturing and use of railway components. I would also like to thank the academic partners Polimi, ESI and the Montanuniversität Leoben. Thank you Reinhard for always questioning everything critically. You contributed a lot to the development of the models in this work. Thanks to Martin for mentoring me during my PhD. Regarding the scientific work, my greatest thanks go to Thomas. With your unbiased nature, you laid the foundation for my scientific and professional career and were the best supervisor imaginable. Many, many thanks!

The author gratefully acknowledges the financial support under the scope of the COMET program within the K2 Center "Integrated Computational Material, Process and Product Engineering (IC-MPPE)" (Project No 886385). This program is supported by the Austrian Federal Ministries for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK) and for Labour and Economy (BMAW), represented by the Austrian Research Promotion Agency (FFG), and the federal states of Styria, Upper Austria and Tyrol.

This thesis is dedicated to
my parents
the first two engineers I met in my life

Kurzfassung

Da die Eisenbahnindustrie eine zentrale Rolle im weltweiten Transportwesen spielt, ist die Entwicklung effizienter und zuverlässiger Eisenbahnkomponenten erforderlich. In dieser Arbeit liegt der Fokus auf der Optimierung der Herstellungsprozesse dieser Komponenten mit Hilfe von neu entwickelten Prozessmodellen. Diese Modelle ermöglichen es, die Zuverlässigkeit der Komponenten zu optimieren, die Sicherheit zu erhöhen und die Nachhaltigkeit in der Eisenbahnindustrie zu fördern.

Die entwickelten Prozessmodelle helfen die im Betrieb auftretenden Bedingungen besser zu beurteilen. Weiters können die Herstellungsprozesse von Eisenbahnkomponenten durch die Anwendung komplexer Materialmodelle besser verstanden und optimiert werden. Durch die genaue Vorhersage von Eigenspannungen sowie der Spannungsumlagerung und der Bewertung beeinflussender Prozessparameter tragen diese Prozessmodelle dazu bei, Eisenbahnkomponenten laufend weiterzuentwickeln.

In dieser Arbeit wird ein neues, innovatives Prozessmodell präsentiert, welches das Festwalzen von Radsatzwellen mittels der Finite Elemente Methode beschreibt. Dieses fortschrittliche Modell berücksichtigt dabei sowohl das zyklische Materialverhalten, als auch verschiedene Prozess- und Geometrieparameter, um die Eigenspannungsverteilung über den gesamten Querschnitt der Radsatzwelle präzise vorherzusagen. Durch die Einbeziehung und Kombination periodischer Randbedingungen wird eine erhebliche Reduzierung der Modellgröße und der Berechnungszeit erreicht, wobei die Integrität der Ergebnisse erhalten bleibt. Dabei liefert das Prozessmodell präzisere Vorhersagen des Eigenspannungszustands im Vergleich zu früheren Ansätzen.

Darüber hinaus befasst sich diese Arbeit mit dem Wärmebehandlungsprozess von Eisenbahnrädern, wobei der Schwerpunkt auf dem Einfluss heterogen verteilter, fester Phasen von Stahl liegt. Zudem wird ein Materialmodell präsentiert, welches lokale Effekte während der Wärmebehandlung, einschließlich der Phasenumwandlungen und umwandlungsinduzierter Plastizität, berücksichtigt. Um das komplexe Materialverhalten in einem Finite Elemente Prozessmodell adäquat abzubilden, ist eine präzise Abstimmung zwischen den Experimenten, den mathematischen und physikalischen Modellen und dem Finite Elemente Modell des Prozesses selbst unerlässlich. Ein solches Materialmodell bietet umfassende Einblicke in das Verhalten des Eisenbahnstahls ER7, während das entwickelte Prozessmodell es ermöglicht, die Beanspruchungen des Eisenbahnrades unter verschiedenen und sich schnell ändernden Bedingungen zu verstehen.

Zusammenfassend kann gesagt werden, dass diese Arbeit einen wesentlichen Beitrag zur Weiterentwicklung der Herstellung von Eisenbahnkomponenten leistet. Die entwickelten Prozessmodelle ermöglichen tiefe Einblicke in das Eigenspannungsverhalten und nachfolgender Spannungsumlagerung in Eisenbahnkomponenten. Außerdem ebnen diese Modelle den Weg hinsichtlich der Optimierung der Herstellungsprozesse.

Abstract

The railway industry plays a pivotal role in global transportation, necessitating the development of efficient and reliable railway components. This thesis focuses on advancing the manufacturing process of these components through the development of process models. These models aim to optimize component performance, enhance safety, and promote sustainability within the railway industry.

Process models including advanced material models allow for understanding and optimizing not only the manufacturing processes but also the in-service conditions that arise. Within the context of railway components, these models provide valuable insights into the material behavior and guide the optimization of these processes. By accurately predicting residual stresses, the stress redistribution, and the influencing process parameters, these process models contribute to improve and better understand the behavior of railway components.

This thesis presents a novel process model specifically designed for the cold rolling of wheelset axles with the finite element method. This advanced model incorporates cyclic material behavior, various process parameters, and geometrical features to precisely predict the residual stress distribution across the entire cross section of the wheelset axle. By incorporating and combining smart periodic boundary conditions, a notable reduction in model size and computation time is achieved, while maintaining the integrity of the results. Notably, the process model developed for the cold rolling of wheelset axles enhances the accuracy of the prediction of the residual stress state compared to previous approaches.

Additionally, this thesis addresses the manufacturing process of railway wheels, focusing on the influence of heterogeneously distributed solid phases of steel. An advanced material model is developed to accurately capture localized effects during the heat treatment, including phase transformations and transformation induced plasticity. To adequately represent the intricate material behavior within a finite element process model, precise coordination among experiments, mathematical and physical models for changing material properties, and the finite element model of the process itself is imperative. Such a material model provides comprehensive insights into the behavior of the railway steel ER7, while the developed process model allows to understand the stress evolution of a railway wheel in various and rapidly changing conditions.

In summary, this thesis contributes to the advancement of the manufacturing of railway components. The established process models provide deep insights into the development of residual stresses and stress redistribution of railway components enabling a further process optimization.

Contents

List of Symbols

Symbol	Unit	Description
A_1	$\mathrm{mJ}\sqrt{\mathrm{K}}$	material constant for calculation of critical nucleation radius
A_3	$\mathrm{J}^3\sqrt{\mathrm{K}}$	material constant for calculation of Gibbs free energy at its peak
A_5	-	material constant for calculation of the growth rate of the product phase fraction
B	K	material constant for calculation of the growth rate of a nucleus during transformation
b	-	Parameter of isotropic hardening rule, that defines the rate at which the size of the yield surface changes as plastic straining develops.
B_1	-	material constant for calculation of the growth rate of the product phase fraction
C_k	MPa	Parameter of kinematic hardening rule, that defines the initial kinematic hardening modulus. The index k represents the number of the particular backstress.
c	J/kgK	specific heat capacity
$\boldsymbol{C}$	$-$	matrix of the regularization, that determines the zero, first and second derivatives
D_{axle}	m	diameter of the wheelset axle
D_{WR}	m	diameter of the work roller
$f'(\xi_i)$	-	Derived saturation function
$f(\boldsymbol{\sigma}, H)$	-	yield criterion
$\boldsymbol{G}$	MPa	compliance matrix
G_{ij}	MPa	coefficient of compliace matrix
G^*	J	peak value of Gibbs free energy
H	MPa	tensorial hardening variable
$\boldsymbol{H}$	$-$	matrix of the regularization, that weights the contributions of individual evaluation points
$\boldsymbol{I}$	$-$	identity matrix
K_i	MPa^{-1}	Greenwood-Johnson transformation coefficient for the i-th phase transformation.
L	N	load
f	N	feed: distance in axial direction of the work roller covered in one turn

Symbol	Unit	Description
$\boldsymbol{n}$	-	normal vector
n	-	material constant for calculation of the growth rate of the product phase fraction
p_H	MPa	herzian pressure
Q	J	thermal energy
Q_a	J	activation energy
$\dot{\boldsymbol{q}}$	$\mathrm{W/m^2}$	heat flux
Q_∞	MPa	Parameter of isotropic hardening rule, that defines the maximum change in the size of the yield surface.
R	J/(mol K)	universal gas constant
r	mm	first coordinate of a cylindrical coordinate system
r^*	m	critical nucleation radius
r_n	m	nucleation radius
R_{WR}	m	edge radius of the work roller
$\boldsymbol{S}$	MPa	deviatoric part of the stress tensor
$\boldsymbol{S}$	–	diagonal matrix of the regularization, that indicates the measurement error
U	J	internal energy
u	J	specific internal energy
$\mathbf{u}$	m	displacement vector
W	J	mechanical energy
z	mm	third coordinate of a cylindrical coordinate system
α	-	constant in the Koistinen Marburger equation describing the velocity of the phase transformation
α_a	–	aperture angle of finite element model of the cold rolling process
$\boldsymbol{\alpha}$	MPa	backstress tensor
$\boldsymbol{\alpha}_d$	MPa	deviatoric part of the backstress tensor
α_{th}	K^{-1}	coefficient of thermal expansion
α_k	$\mathrm{W/m^2K}$	convective heat coefficient
α_n	s^{-1}	material constant for calculation of the growth rate of a nucleus during transformation
β	–	scalar factor of the regularization, that control the amount of regularization
$\Delta\theta$	K	undercooling
ε	–	emissivity
$\boldsymbol{\varepsilon}_{CC}$	m^{-1}	strain vector for cut compliance method
$\boldsymbol{\varepsilon}$	-	total strain tensor
$\dot{\boldsymbol{\varepsilon}}_i^{tp}$	-	Transformation induced strain rate by Leblond
$\boldsymbol{\varepsilon}^{el}$	-	elastic strain tensor
ε_{eq}^{pl}	-	eqivalent plastic strain

Symbol	Unit	Description
$\dot{\varepsilon}_{eq}^{pl}$	-	eqivalent plastic strain rate
$\varepsilon'_{\mathrm{ij}}$	m^{-1}	local strain component of a single layer within the cut compliance method
$\boldsymbol{\varepsilon}^{pl}$	-	plastic strain tensor
$\dot{\boldsymbol{\varepsilon}}^{pl}$	-	rate of plastic flow
$\boldsymbol{\varepsilon}^{\theta}$	-	thermal expansion tensor
$\boldsymbol{\varepsilon}^{tp}$	-	TRIP strain tensor
$\boldsymbol{\varepsilon}^{tv}$	-	transformation extension tensor
γ	-	material constant for calculation of the growth rate of the product phase fraction
γ_k	-	Parameter of kinematic hardening rule, that defines the rate at which the kinematic hardening moduli decrease with increasing plastic deformation. The index k represents the number of the particular backstress.
λ	hs	cooling parameter
$\dot{\lambda}$	-	plastic multiplier
λ	W/mK	thermal conductivity
φ	rad	second coordinate of a cylindrical coordinate system
ϕ_{axle}	rad	rotational DOF of the axle
ϕ_{WR}	rad	rotational DOF of the work roller
$r(\varepsilon_{eq})$	-	isotropic hardening function
ρ	kg/m^3	mass density
$\sigma\vert_0$	MPa	Parameter of isotropic hardening rule, that defines the yield stress at zero plastic strain.
$\boldsymbol{\sigma}_{CC}$	MPa	stress vector for cut compliance method
σ_{dil}	MPa	applied stress during deformation dilatometry experiments
σ_e	MPa	von Mises equivalent stress
σ'_{ij}	MPa	local stress component of a single layer within the cut compliance method
σ_{max}	MPa	maximum permissible load for deformation dilatometry tests
$\sigma_{\varphi\varphi}$	MPa	normal stress component in φ-direction
σ_{rr}	MPa	normal stress component in r-direction
$\boldsymbol{\sigma}$	MPa	stress tensor
σ_{y0}	MPa	initial yield stress
σ_y	MPa	yield stress
σ_{zz}	MPa	normal stress component in z-direction
θ	K	current temperature
θ_0	K	equilibrium temperature, at which the parent phase has the same free energy as the product phase
θ_{LO}	K	load onset temperature
θ_{MS}	K	martensite start temperature

Symbol	Unit	Description
$\dot{\theta}$	$^{K}/_{s}$	rate of temperature change
θ^*	K	temperature corresponding to the shortes incubation time
ξ	-	volume fraction of the product phase in Leblond's law
$\dot{\xi}$	-	time derivative of the volume fraction of the product phase in Leblond's law
ξ_M	-	volume fraction of martensite according to the Koistinen Marburger equation

List of Abbreviations

ÖBB	Austrian Federal Railways
2D	two dimensional
3D	three dimensional
bcc	body centered cubic
bct	body centered tetragonal
CC	cut compliance
CCT	continuous cooling transformation
CFD	computational fluid dynamics
DB	Deutsche Bahn
DOF	degree of freedom
EA4T	standardized quenched and tempered steel for wheelset axles
ER7	standardized quenched and tempered steel for railway wheels
fcc	face centered cubic
FE	finite element
FEA	finite element analysis
FEM	finite element method
GUI	graphical user interface
GWJ	Greenwood-Johnson
hex	hexagonal
ICE	Inter City Express
LCF	low cycle fatigue
MPC	multi point constraint
PBC	periodic boundary conditions
py	python
ROI	region of interest
RP	reference point
RVE	representative volume element
TRIP	transformation induced plasticity
TTT	time-temperature transformation
UHCF	ultra high cycle fatigue
XRD	X-ray-diffraction

1 Introduction

Mobility plays a crucial role in modern life, facilitating commuting, leisure travel, and the transportation of goods. However, as society moves towards a CO_2-neutral future, the importance of sustainable mobility concepts has grown significantly. In Austria, transport is responsible for 28.2% of greenhouse gas emissions, with road traffic accounting for nearly 99% of these emissions. Despite efforts in other sectors, transport emissions continue to rise each year. Consequently, reducing emissions has become a top priority, necessitating effective strategies and solutions. [1–3]

Motivation

A key approach to reducing greenhouse gas emissions from transport is to shift road traffic to rail. However, for this transition to succeed, rail transport must become even more appealing to commuters, and capacity must be expanded. In 2019, Austrian Federal Railways (ÖBB) transported 476 million passengers using up to 600 trains simultaneously. Addressing the challenge of accommodating even higher traffic volumes in the future is critical. One possible strategy would be to create longer inspection intervals, although this should not jeopardize safety. To accomplish this, engineers need to develop methods to better predict component behavior, especially of critical components like the wheelset. Therefore, exploring methods to enhance the attractiveness and efficiency of rail transport is crucial. [3]

Goals

The objective of this thesis is to develop advanced methods for analyzing critical components, focusing specifically on the wheelset, which comprises the wheelset axle and wheels. The wheelset is subject to stringent safety standards, and its lifetime assessment and dimensioning are typically conducted using conventional safety engineering approaches. However, these methods often employ conservative assumptions, considering average values and disregarding local property variations. To address these limitations, this thesis aims to utilize computational modeling and simulations to gain a deeper understanding of the wheelset's behavior both due to manufacturing and in service. Based on these validated software tools and simulation models it will be possible to incorporate local property changes of the material to the existing dimensioning framework and thereby increase the inspection intervals for these railway components.

By employing more advanced analysis methods, the thesis seeks to enhance the safety, reliability, and efficiency of rail transport. This proactive approach can reduce downtime, increase train availability, and ultimately contribute to the attractiveness of rail transport.

Moreover, by improving the sustainability of rail transportation, this work aligns with the broader goal of reducing greenhouse gas emissions in the transport sector.

Overall, this thesis endeavors to explore sustainable mobility concepts with a focus on enhancing the wheelset's behavior through advanced analysis methods. By achieving these goals, we can contribute to a more sustainable and efficient transport system, thereby advancing the vision of a CO_2-neutral future. [1–3]

State of the Art

Presently, the existing guidelines and standards governing the design and dimensioning of railway components rely on traditional fatigue calculations, based on Smith or Haigh diagrams, as well as fracture mechanics analyses. However, these conventional approaches are susceptible to inherent uncertainties and rely on a worst-case assumption, typically considering a visible crack scenario. Moreover, such calculations employ average material properties, disregarding local variations in these properties. As a result, there is a need for more comprehensive methodologies that can account for the complex nature of material behavior and address the limitations associated with current practices. [4]

The current approach to addressing the corrosion and stone chip protection requirements of railway components relies predominantly on the application of paints and coatings. However, this method incurs substantial costs, both in terms of materials and ongoing maintenance. Notably, coatings need to be removed periodically for component inspection and subsequently reapplied to meet in-service standards. Nevertheless, research findings have unveiled the potential of mechanical surface treatments, particularly the cold rolling process. By inducing compressive stresses in the vicinity of the component surface, cold rolling leads to crack-closure effects capable of mitigating both typical crack growth and corrosion-enhanced crack growth. The utilization of such mechanically treated components not only streamlines the manufacturing process but also yields favorable implications for maintenance procedures, expediting the overall maintenance timeline and augmenting the safety and quality in-service of these components.

Detailed elaborations of the current state of the art for the manufacturing and modeling of the components wheelset axle and railway wheel can be found in sections 3.2 through 3.3 and 4.1 through 4.3 respectively.

Open Questions

The challenges surrounding the manufacturing and maintenance of railway components give rise to several unanswered questions. The current models available for characterizing residual stresses generated during the manufacturing process provide only approximate estimations of the stress state. To achieve more precise predictions of the stress distribution in these components throughout and after their manufacturing and in service, advanced models are required. These models must account for the local property changes that have global implications on the component behavior. In the case of the stress profile within the depth of the wheelset axle, it is crucial to identify the process parameters that influence it and understand its distribution at greater depths, where conventional measurement techniques may have limitations. Further-

more, local changes in material properties during the heat treatment of railway wheels must be considered. This entails investigating the formation of solid phases at localized regions during typical heat treatments and their subsequent impact on the overall component behavior. A key focus lies in the development of integrated material and process models that extend beyond academic research and can be effectively utilized for the design and advancement of these critical railway components.

Structure

The thesis is organized into the following sections. The introduction provides an overview of the subject of process models in railway component manufacturing and highlights the unresolved questions within this field. Following the introduction, the fundamentals chapter delves into the essential elements required for the development of material models. This includes a discussion on the fundamental mechanisms of phase transformation, heat transfer, and the influential variables pertaining to residual stresses and their measurement. This is followed by two substantive chapters that provide comprehensive information critical to the formulation of process models in railway component manufacturing, namely the cold rolling of wheelset axles and the heat treatment of railway wheels. These respective chapters form thematically self-contained sections within the large subject area of railway component manufacturing, and each consists of a state of the art section, a methods section, a results section, and a conclusion. The thesis concludes with a summary chapter, which briefly recaps the results and highlights the scientific contribution. Throughout the work, readability and language has been enhanced through the utilization of AI software tools such as DeepL and ChatGPT. After using these tools, the author reviewed and edited the content and takes full responsibility for the content of the publication.

2 Theoretical Background

This chapter provides an overview of the fundamental concepts and theoretical background essential to this thesis. Initially, the constitutive equations governing the mechanical behavior of materials are briefly summarized. Subsequently, residual stresses resulting from the plastic material behavior are discussed from an engineering perspective. This section elaborates on commonly employed techniques to measure these stresses, as well as a novel technique capable of measuring residual stresses at considerable depths below the surface of a component, namely the cut compliance method. Finally, the chapter concludes with a discussion of heat transfer, particularly with regards to the implementation of heat transfer calculations through the finite element method.

2.1 Constitutive Equations

An essential part of any process model is the material description. Besides geometry and contact, it is another source of nonlinearity of the system to be solved. To represent or describe the real material behavior, material models are created that are valid for specific purposes or domains. Basically, a material model is the mathematical relationship between the stress and the strain state, or from another perspective between an applied force and the resulting displacement. This relationship can take various forms, such as elastic, plastic, or viscoplastic, that also include a monotonic or cyclic behavior. More generally, a material model aims to describe how and to what extent the material reacts to external influences.

In this thesis, two key processes are studied: cold rolling and heat treatment. The material properties play a vital role in both processes and their simulations. During both cold rolling and heat treatment, the material plasticizes, i.e. it experiences irreversible deformation. The stress-strain relationship for a uniaxial tensile test, which is commonly used to evaluate material properties, is shown in Figure 2.1. Understanding material behavior under different loading conditions is critical to accurately model manufacturing processes. Therefore, it is essential to understand how material models are developed and the key ingredients they use.

The transition from the elastic to the plastic regime is marked by the yield point σ_y, which represents the stress level at which plastic deformation initiates, irrespective of its subsequent behavior. Once the material enters the plastic regime and experiences a certain strain, that strain becomes permanent even after unloading or load reversal, with only the elastic portion of the strain being recoverable. This distinction allows for the additive decomposition of the total strain tensor ε into an elastic component ε^{el} and a plastic component ε^{pl}, as shown

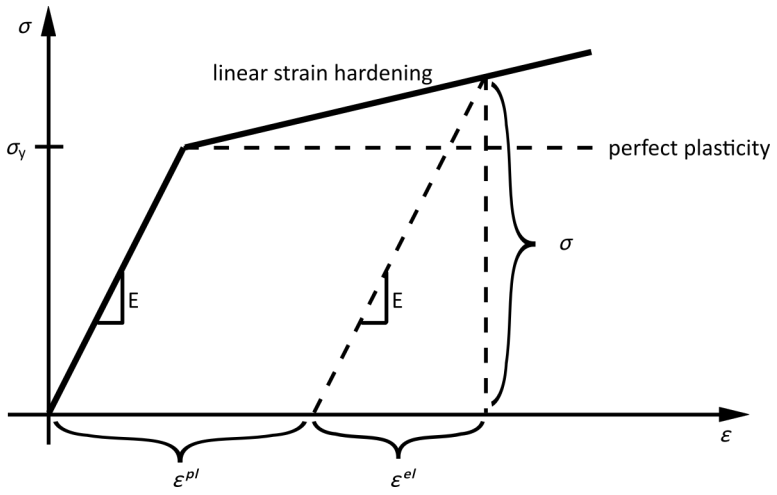


Figure 2.1: Decomposition of the strain into an elastic and plastic part. Figure and caption adapted from [5].

in Equation 2.1. The stress-strain relationship during loading and unloading follows Hooke's[1] law, which states that stress is proportional to strain.

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^{el} + \boldsymbol{\varepsilon}^{pl} \tag{2.1}$$

Yield criterion

In the context of the uniaxial case, identifying the onset of plastic deformation is relatively straightforward by using the yield point. However, in more complex scenarios involving a multiaxial stress states, where the stress tensor comprises components in multiple spatial directions, determining the transition to plastic behavior becomes more challenging. To address this, a yield criterion $f(\boldsymbol{\sigma}, H)$ needs to be defined. In the general case, this criterion is a function of the current stress state $\boldsymbol{\sigma}$ and internal hardening variables, denoted by H. Equation 2.2 establishes that the material exhibits purely elastic behavior when this function is less than zero, while plastification occurs when the yield criterion is satisfied at $f(\boldsymbol{\sigma}, H) = 0$ and $\dot{f}(\boldsymbol{\sigma}, H) = 0$, with the latter being the consistency condition. The consistency condition enforces that the current point remains at the border of the elastic domain during the plastic flow. It should be noted that because of the consistency condition $f(\boldsymbol{\sigma}, H) > 0$ is inadmissible. This relationship is also shown in Figure 2.3 for the plane stress state, where the ellipse represents the yield surface with $f(\boldsymbol{\sigma}, H) = 0$. [5, 6]

$$f\left(\sigma, H\right) < 0 : \text{elastic} \tag{2.2}$$

$$f\left(\sigma, H\right) = 0 \quad \text{and} \quad \dot{f}\left(\sigma, H\right) < 0 : \text{elastic unloading} \tag{2.3}$$

$$f\left(\sigma, H\right) = 0 \quad \text{and} \quad \dot{f}\left(\sigma, H\right) = 0 : \text{plastic} \tag{2.4}$$

$$f\left(\sigma, H\right) > 0 : \text{inadmissible} \tag{2.5}$$

[1] Robert Hooke (18 July 1635 – 3 March 1703): English mathematician.

The choice of the yield criterion function depends on the specific characteristics of the material under consideration. In this study, the von Mises[1] yield criterion is exclusively employed due to its suitability for ductile materials like steel, which is prevalent in the manufacturing of railway components. Equation 2.6 presents the tensorial representation of the von Mises yield function (with σ_e as the (von Mises) equivalent stress), wherein $\boldsymbol{S}$ represents the stress deviator of the stress tensor $\boldsymbol{\sigma}$. [5, 6]

$$f(\sigma) = \sigma_e - \sigma_y = \sqrt{\frac{3}{2}\,\boldsymbol{S} \,:\, \boldsymbol{S}} - \sigma_y \tag{2.6}$$

Furthermore, Equation 2.6 unifies three essential criteria of this law: (1) the yield condition is invariant to the hydrostatic stress, meaning it solely relies on the deviatoric component, (2) the material response is isotropic, ensuring consistent behavior regardless of the direction of loading, and (3) the yield stress in tension equals to the yield stress in compression, ensuring symmetry in the material's response to different loading conditions. This material behavior is sometimes referred to as J_2 plasticity because the expression $J_2 = \sqrt{\frac{1}{2}\boldsymbol{S} : \boldsymbol{S}}$ describes the second invariant of the deviatoric stress tensor. A graphical representation of the von Mises yield surface is shown in Figure 2.2, where the yield condition is shown in the principal stress space as a green cylinder. The red part around this surface is part of the hardening rules, that are described later. [5, 6]

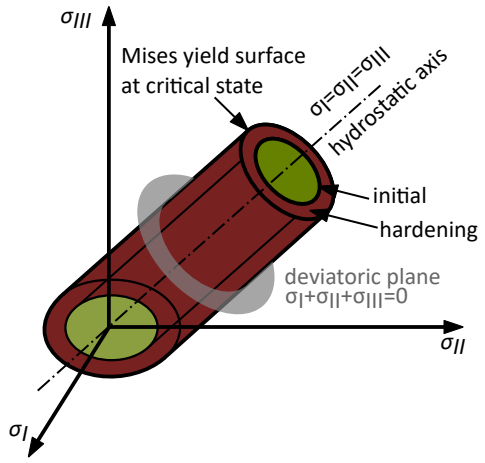


Figure 2.2: Illustration of the Mises yield surface in the principle stress space. Figure adapted from [6].

Flow rule

The yield stress serves as a criterion to determine the onset of plastic deformation, marking the limit of the elastic region. In contrast, the flow rule characterizes the material's behavior beyond the yield point and provides information on the direction of flow. This direction is determined by the plastic potential. When the yield surface and the plastic potential surface

[1] Richard von Mises (19 April 1883 – 14 July 1953): Austrian mathematician.

projected in the stress space are congruent, the flow rule is referred to as associated flow. However, in the case of non-associated flow, the plastic deformation direction, governed by the plastic potential, does not align with the normal vector of the yield surface, representing a more general case. Figure 2.3 shows this relationship graphically for the case of plane stress, which can also be expressed mathematically by Equation 2.7. In this equation, $\frac{\partial f}{\partial \boldsymbol{\sigma}}$ represents the direction of the plastic strain increment, and $\dot{\lambda}$, the plastic multiplier which corresponds to its magnitude.

$$\dot{\boldsymbol{\varepsilon}}^{pl} = \dot{\lambda} \frac{\partial f}{\partial \boldsymbol{\sigma}} \tag{2.7}$$

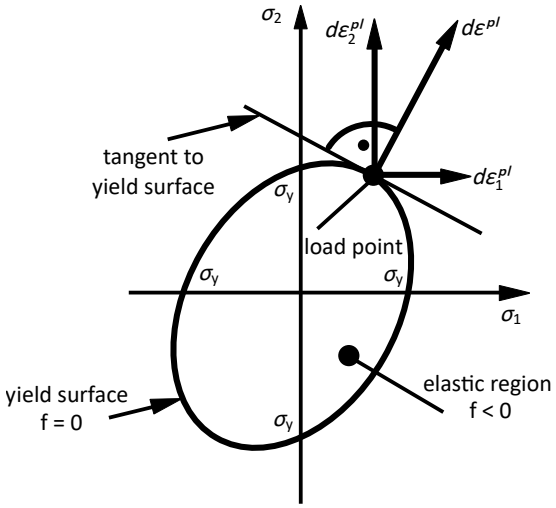


Figure 2.3: The von Mises yield surface for the plane stress condition illustrating the direction of the incremental plastic strain normal to the tangent of the surface. Figure and caption adapted from [5].

During plastic deformation satisfying the yield condition at any time is crucial, especially for time-independent plasticity. This condition, known as the consistency condition, ensures that the total differential of the yield condition is zero throughout the plastic deformation process. Mathematically, this is represented as

$$df = 0 \quad , \tag{2.8}$$

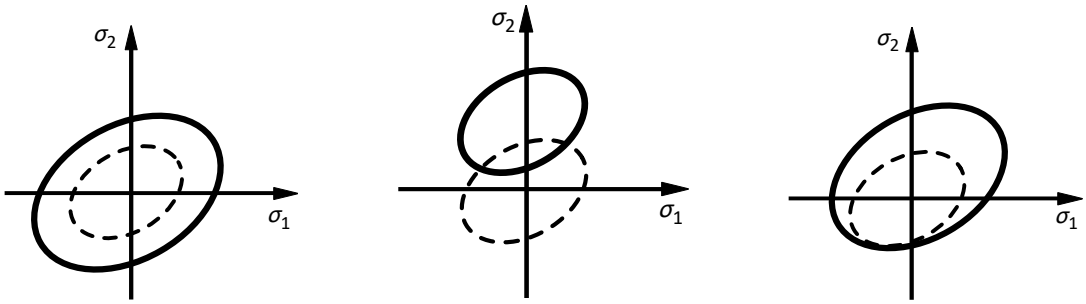
indicating that the yield condition remains unchanged during the entire plastic deformation.

Hardening Rule

To capture the complex stress states encountered during the manufacturing of railway components, it is necessary to consider the evolution of the yield surface. The area marked in red in Figure 2.2 illustrates the expansion of the yield surface, which is one possible modification of the yield surface. This expansion signifies the modification of the yield surface to accommodate a broader range of stress states. This change of the yield surface can also be

described as increasing stress for further plastification and is called hardening. The hardening rule affects both the yield criterion and the flow rule. The fact that the yield surface changes is a consequence of the aforementioned consistency condition, since there is no valid material state beyond the limit of the yield surface.

Isotropic and kinematic strain hardening are widely recognized methods for characterizing the intricate stress evolution experienced by materials during plastic deformation. While each model effectively captures specific aspects of this phenomenon in isolation, their combined utilization is often favored to mitigate the limitations inherent to the other model. Figure 2.4 shows the modification of the yield surface under plane stress conditions when subjected to the hardening laws respectively. Isotropic strain hardening induces a uniform expansion of the yield surface in all directions, whereas kinematic hardening shifts the position of the yield surface. Notably, the third part of Figure 2.4 portrays the yield surface's behavior when the combined variant of isotropic and kinematic hardening is employed.



(a) isotropic **(b)** kinematic **(c)** combined

Figure 2.4: Change of the yield surface of (a) kinematic, (b) isotropic and (c) combined hardening. Figure adapted from [7].

The expansion of the yield surface in isotropic hardening is a function of the accumulated plastic strain. Considering the consistency condition, the following relationship is obtained for the yield function:

$$f(\boldsymbol{\sigma}, \varepsilon_{eq}^{pl}) = \sigma_e - \sigma_y(\varepsilon_{eq}^{pl}) = 0 \quad , \tag{2.9}$$

where ε_{eq}^{pl} is the equivalent plastic strain, which is similar to the equivalent von Mises stress and calculated via integrating the equivalent plastic strain rate $\dot{\varepsilon}_{eq}^{pl}$, which again is dependent on the rate of the plastic strain tensor $\dot{\boldsymbol{\varepsilon}}^{pl}$:

$$\dot{\varepsilon}_{eq}^{pl} = \sqrt{\frac{2}{3}\dot{\boldsymbol{\varepsilon}}^{pl} : \dot{\boldsymbol{\varepsilon}}^{pl}} \quad , \tag{2.10}$$

$$\varepsilon_{eq}^{pl} = \int \dot{\varepsilon}_{eq}^{pl} dt \quad . \tag{2.11}$$

$\sigma_y\left(\varepsilon_{eq}\right)$ can be decomposed into an initial yield stress σ_{y0} and the isotropic hardening function $r(\varepsilon_{eq})$:

$$\sigma_y\left(\varepsilon_{eq}\right) = \sigma_{y0} - r\left(\varepsilon_{eq}\right) \quad , \tag{2.12}$$

where $r(\varepsilon_{eq})$ in its most common form is:

$$r\left(\varepsilon_{eq}\right) = Q_\infty \left(1 - e^{-b\varepsilon_{eq}}\right) \quad . \tag{2.13}$$

In this function Q_∞ gives the saturation of a nonlinear stress strain relationship and b denotes the rate at which the saturation is achieved, leading to an exponential evolution of the corresponding stress strain relationship. By combining Equation 2.12 and 2.13 the change of the yield stress defining the yield surface is given as:

$$\sigma_y\left(\varepsilon_{eq}\right) = \sigma_{y0} - Q_\infty \left(1 - e^{-b\varepsilon_{eq}}\right) \quad . \tag{2.14}$$

Isotropic hardening models are suitable for accurately describing the stress-strain relationship under monotonically increasing loads or loads without load reversal. However, when load reversal occurs, the ability to accurately reproduce the relationship between stress and strain diminishes. This is where kinematic hardening proves advantageous. By incorporating kinematic hardening, the elastic region is effectively reduced through the displacement of the yield surface, as depicted in Figure 2.4. When the kinematic hardening rule is included in the yield criterion using the von Mises yield function, the following relationship is obtained:

$$f(\boldsymbol{\sigma} - \boldsymbol{\alpha}) = \sqrt{\frac{3}{2}(\boldsymbol{S} - \boldsymbol{\alpha}_d) : (\boldsymbol{S} - \boldsymbol{\alpha}_d)} \quad , \tag{2.15}$$

Here, $\boldsymbol{\alpha}_d$ represents the deviatoric component of the backstress tensor $\boldsymbol{\alpha}$, which in turn is the key variable for describing kinematic hardening.

To accurately capture the material behavior under alternating loading in the plastic regime, Chaboche developed a comprehensive nonlinear model incorporating both isotropic and kinematic hardening. This combined model has gained widespread acceptance for its effectiveness in commercially available finite element (FE) programs and is employed in this work to describe the material behavior in the manufacturing of railway components. The kinematic hardening component proves particularly valuable in characterizing the response during a single load cycle, while the isotropic hardening component allows predicting the maximum stresses after a certain number of cycles. In the Chaboche model, the evolution of the back stress is described by the sum of multiple backstresses:

$$\dot{\alpha} = \sum_{k=1}^{N} C_k \dot{\varepsilon}_{eq} \frac{1}{\sigma_y\left(\varepsilon_{eq}\right)} \left(\boldsymbol{\sigma} - \boldsymbol{\alpha}\right) - \gamma_k \alpha_k \dot{\varepsilon}_{eq} \quad , \tag{2.16}$$

where the term $\sigma_y\left(\varepsilon_{eq}\right)$ represents the inclusion of isotropic hardening in this evolution. By superimposing multiple backstresses and subsequently adapting the values of the material parameters C_k and γ_k it is possible to calibrate the evolution of the backstress accurately to the experimental data. For the application in this work it has proven useful to model the kinematic hardening by calibrating 3 backstresses. [5, 6, 8, 9]

For a better understanding of the manufacturing of railway components, it is essential to be able to describe their material behavior. For this description, material models are used which have as key ingredients the yield criterion, the flow rule, the consistency condition and the hardening. The combination of these rules constitutes a material model.

2.2 Phase Transformation

An additional significant aspect considered in this work is the influence of heat treatment on the material behavior. Temperature variations initiate metallurgical processes that impact the material properties of railway components, not only during the heat treatment process but also throughout their operational lifetime. To gain initial insights into these metallurgical processes, models describing the crystal structure of metals are elucidated. These models serve to demonstrate the underlying reasons for alterations in the stress field during phase transformations. A comprehensive explanation of these phenomena and their modeling will be provided in Chapter 4 dedicated to the heat treatment of railway wheels.

Crystal Structures

The fundamental structure of materials, in particular metals can be understood as a periodic arrangement of atoms within a lattice structure. The specific way in which these atoms arrange themselves within the lattice plays a significant role in determining the overall behavior of the material. The microstructure, including the arrangement, size, distribution, shape, and composition of these lattice cells, further influences the material's properties. The lower parts of Figures 2.5 and 2.6, respectively, illustrate this concept of the grouped atoms in periodic unit cells. Metals, in particular, tend to crystallize in three primary lattice structures, namely: body centered cubic (bcc), face centered cubic (fcc) and hexagonal (hex). The mechanical properties of materials are closely related to these underlying crystal structures. Through heat treatments, the arrangement of atoms within the lattice can be altered, consequently leading to changes in the material's properties. [10–14]

During the heat treatment of steel, different phases are formed, each characterized by their distinct lattice structure. These phases also exhibit variations in their material properties, playing a crucial role in determining the ductility, hardness, strength, impact toughness, and creep strength of the material. The ability to intentionally manipulate and harness these properties makes steel an incredibly versatile material. Through careful control of the heat treatment process, steel can be tailored to meet specific performance requirements in a wide range of applications. [10–12, 14]

Different temperature-time profiles during the heat treatment process are instrumental in the formation of specific solid phases. This enables the manufacturing of components from a single material that simultaneously meet diverse requirements. As illustrated in this thesis using the example of a railway wheel, specific heat treatment conditions can be applied to meet the specific performance criteria. Detailed requirements for the railway wheel can be found in Subsection 4.1.1. [12, 15]

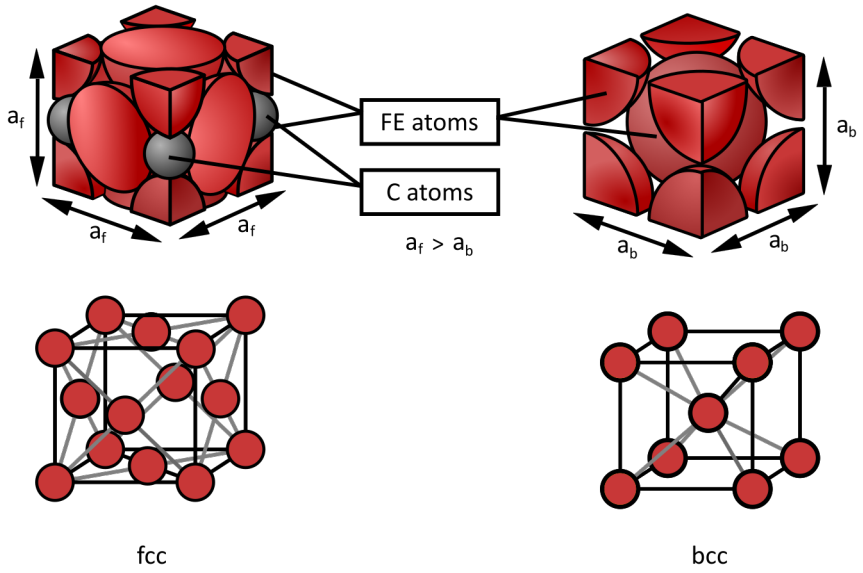


Figure 2.5: Change in the crystal structure of steel during the transformation of austenite into pearlite. Figure adapted from [16].

Transformation Mechanisms

The residual strains and stresses observed in a railway wheel are a result of the phase transformations that take place within its austenitic microstructure. During these transformations, the crystal structure changes from fcc to bcc as the austenite transforms into ferrite, pearlite, and bainite. Figure 2.5 shows the distinct atomic arrangements in the fcc and bcc lattices. This transformation process, known as reconstructive or diffusive transformation, involves the diffusion of atoms, which rearrange themselves into a new structure. It is important to note that such atomic movements require time for diffusion to occur. However, increasing cooling rates restrict the available time for diffusion, impeding the movement of atoms. When the cooling rate reaches a critical speed, the atoms do not have enough time to diffuse, resulting in an abrupt transformation. In the case of steel, the austenite transforms into martensite[1] through a displacive or martensitic transformation. Unlike diffusive transformations, this type of transformation is solely dependent on temperature and not time. The limited diffusion time leads to a presence of embedded C atoms that cause a distortion of the original cuboid geometry, transforming it into a tetragonal lattice. The structural transition in Figure 2.6 shows the transformation from the fcc lattice, which represents the most densely packed arrangement for spheres, to the distorted body centered tetragonal (bct) lattice of martensite. The resulting volume change and lattice distortion play a significant role in generating stresses during heat treatment processes involving phase transformations. These stresses do not necessarily have a negative influence on the component behavior. The distorted lattice not only enhances strength and hardness but also impacts the component's service life positively, particularly when the residual stresses are compressive. [10–14, 17, 18]

[1] Adolf Karl Gottfried Martens (6 March 1850 – 24 July 1914): German materials scientist and metallurgist.

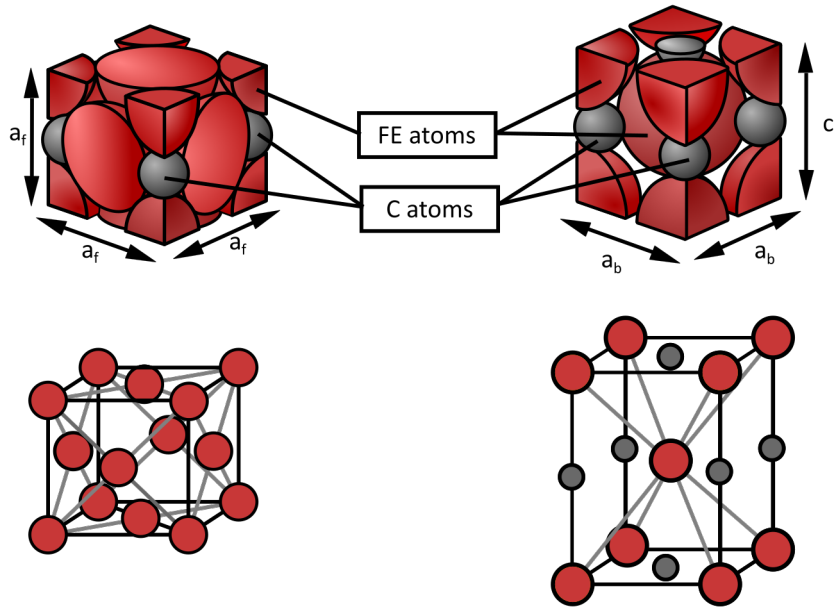


Figure 2.6: Change in the crystal structure during the transformation of austenite into martensite. Figure adapted from [16].

Strain decomposition

The heat treatment of steels involves numerous variables that give rise to stresses, primarily attributed to changes in both strain and localized strain alterations. The total strain encompasses various factors contributing to its magnitude. In the framework of small strain theory, these individual contributions to the total strain can be decomposed in an additive manner. By understanding and accounting for each specific component, a comprehensive analysis of the overall strain and its associated stresses can be achieved. This decomposition approach allows for a more detailed examination of the factors influencing the resulting stresses in the context of heat treatment processes. [19–22]

The total strain tensor ε is composed of the components ε^{el}, ε^{pl}, ε^{θ}, ε^{tp} and ε^{tv} as presented in Equation 2.17. Here ε^{el} represents the elastic strain tensor, ε^{pl} the plastic strain tensor, ε^{θ} the tensor of thermal expansion, ε^{tp} the transformation induced plasticity (TRIP) strain tensor, and ε^{tv} the transformation expansion tensor.

$$\varepsilon = \varepsilon^{el} + \varepsilon^{pl} + \varepsilon^{\theta} + \varepsilon^{tp} + \varepsilon^{tv} \tag{2.17}$$

The decomposition of the strain tensor is a key aspect of the analysis conducted in this work. To provide a clearer depiction of these contributions, refer to Figure 2.7, which depicts a scenario where a wire is subjected to both applied weight and simultaneous heat removal. As the temperature decreases, the wire contracts, representing the strain component of thermal expansion ε^{θ}. The applied weight stretches the wire, constituting the elastic component of the strain tensor ε^{el}, assuming reversible behavior upon unloading. If the temperature drop triggers a phase transformation, it can be accompanied by a volume jump, resulting in a transformation

expansion strain ε^{tv}. When all these effects occur simultaneously, an additional phenomenon known as TRIP can arise. The TRIP strain ε^{tp} entails plastic deformation occurring during phase transformations under the concurrent influence of an external load. It is important to note that the load itself remains within the purely elastic regime and does not cause plasticity of the product phase. Plastic deformation manifests in the softer region in the vicinity of the transformation, particularly in the case of steel where it occurs during the martensitic transformation in the austenitic regions. Interestingly, in practical scenarios, the additional load does not necessarily need to be externally applied. Pre-existing stresses within the microstructure can be interpreted as an additional load in regions that will undergo transformation later on. In the case of the heat treatment of the railway wheel, this is the case when the outer layers contract due to cooling with water and press on the inner regions, where phase transformations occur later in the process. [19–22]

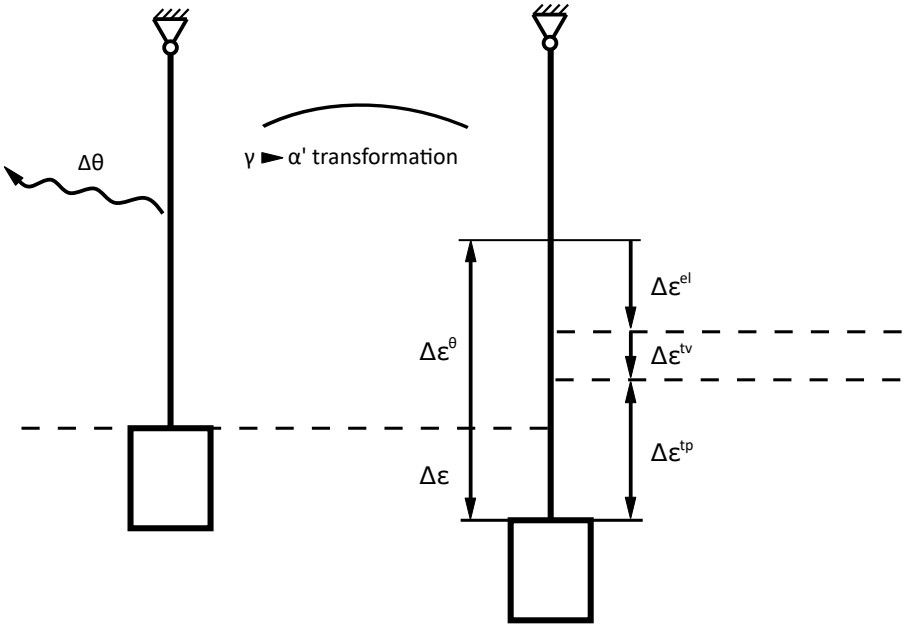


Figure 2.7: Various contributions of strain under load, temperature change and phase transformation. Figure adapted from [23].

In section 4.3, one can find detailed information on the current state of the art in modeling such processes, such as phase transformation and the associated property changes, including the modeling of the individual components of the strain tensor ε.

2.3 Residual Stresses

Residual stress refers to the internal stresses that persist within a component or material even after the removal of the external forces or loads that caused them. These stresses can exist in different forms, such as tensile, compressive, or shear stresses, and are a result of various manufacturing processes like welding, heat treatment, machining, or forming operations. These

processes induce localized changes in geometry like for example during the cold rolling process of wheelset axles and temperature gradients, and microstructure changes, leading to non-uniform and inelastic strains. [12, 24, 25]

2.3.1 Definition of Residual Stresses

In the realm of continuum mechanics, the equilibrium of residual stresses is essential. It mandates that the forces and moments arising from these stresses must sum up to zero. In the light of this principle, residual stresses are classified into three categories: (1) first-order residual stresses, which are balanced across larger regions and account for the inhomogeneity of multiple grains, (2) second-order residual stresses, which exhibit homogeneity over microscopic areas and compensate for the inhomogeneity within individual grain regions, and (3) third-order residual stresses, which reflect inhomogeneities at the submicroscopic scale. Another consideration of this classification is the impact of residual stresses on the macroscopic level. Alterations in first-order residual stresses always lead to macroscopic changes, while substantial modifications are necessary to yield discernible macroscopic effects in second-order residual stresses. Third-order residual stresses, however, are imperceptible at the macroscopic level. [12, 24]

The focus of this work is primarily on first-order residual stresses, although the involvement of second-order residual stresses may be considered in specific cases. Residual stresses have been a subject of extensive investigation within the scientific community for many years, encompassing numerous aspects. However, for the scope of this thesis, it is sufficient to acknowledge the existence of residual stresses and understand how they can be influenced at the macroscopic level. The simulation of manufacturing processes has emerged as a valuable approach to assess the occurrence of residual stresses. Given that this thesis involves comparing simulation results with experimental measurements of residual stresses, it is pertinent to briefly discuss the key aspects of residual stress measurement.

2.3.2 Measurement of Residual Stresses

This section provides a concise overview of various experimental methods commonly employed for the measurement of residual stresses. Subsequently, it delves into a specific method utilized in this work, namely the cut compliance (CC) method. The CC method is a destructive testing method that derives residual stresses by analyzing the strains during the cutting process of a component.

Commonly Used Measuring Techniques

In the field of residual stress measurement, numerous well-established methods have been utilized for several decades. However, these methods face two recurring challenges when applied to real components. Firstly, the accuracy of measurements is not always satisfactory, particularly in cases involving complex geometries or positions where significant changes in geometry occur. Secondly, there is a limitation in the depth to which measurements can penetrate. Non-destructive techniques such as X-ray-diffraction (XRD), ultrasonic velocity

measurements, and the utilization of the Barkhausen effect yield reliable results within the near-surface region, typically limited to the first 2 mm below the surface. [26–28]

XRD is a versatile method. In materials science, it is used to determine the crystal structure, but also to determine the stress state. The diffraction of electromagnetic waves using Bragg's law is applied to the surface to be examined. Thin layers can even be transmitted by radiation. Based on constructive and destructive interference, the stress state can be obtained. [26–28]

The ultrasonic stress measurement is based on the acoustoelastic effect. It takes advantage of the fact that the speed of propagation of elastic waves in the continuum is related to the present stress. [26–28]

Barkhausen Noise Analysis offers an alternative non-destructive method for measuring residual stresses. This technique is specifically applicable to ferromagnetic materials, as it requires the component to be magnetized. By subjecting the material to an external magnetic field, the magnetic domains within the workpiece align themselves with the field lines. Once the saturation state is reached, where all domains are aligned parallel to the external field, the field is removed. The magnetic domains then undergo a process where they sequentially return to their original positions, producing detectable fluctuations in the magnetic field known as Barkhausen noise. These noise characteristics are influenced by the presence of stresses in the material. To accurately determine the stress state, the measurement system must be properly calibrated to the specific material. Once calibrated, the method provides a convenient and rapid means of assessing the orientation and magnitude of residual stresses. However, a limitation of this method is that it can only provide an average measurement value within a depth of approximately one millimeter, starting just below the surface of the material. [29, 30]

In contrast to non-destructive methods, destructive or mechanical methods for measuring residual stresses rely on the principle of strain measurement during material removal, which leads to the relief of residual stresses. [31, 32]

The hole drilling technique is the most popular and widely used method for measuring residual stresses, constituting approximately 30% of all measurements. This technique involves drilling a hole into the surface of a component, where strain gauges are used to measure the strains in the vicinity of the hole resulting from the relaxation of the residual stresses. By analyzing these strains, the residual stresses in the material can be determined. One of the main advantages of this technique is its capability to measure strains in different directions, providing comprehensive information about the stress composition leading to the residual stress state. [31, 32]

Destructive testing methods also include the so-called contour method. It can be used to determine the stress for an entire exposed surface of a component. In this method, a cut is made through the component using wire erosion to avoid introducing additional stresses. The residual stresses in the component cause the cut surface to deform or warp. The deviation of the cut surface from a flat plane is then measured using a coordinate measuring system. By incorporating the warped geometry into FE calculations, the residual stress state can be derived. Although this method provides meaningful stress distributions over a large exposed area, it is a particularly complex procedure that requires expertise in both measuring and simulation. [31]

Cut Compliance Method

The CC method, employed in this work for stress measurement, was initially proposed by Schindler in [33, 34] for determining stress intensity factor and residual stresses in CT specimens. It was later extended and applied to cylindrical geometries such as wheelset axles by Gänser in [35]. Similar to the contour method, the CC method is a destructive measurement technique that involves making a cut on the component, which can be performed using saws as well as wire erosion. This enables the determination of stresses perpendicular to the cutting plane. Strain gauges are strategically positioned to measure the strain resulting from relaxation during the cutting process. A common arrangement involves placing two gauges on either side of the cut, with a third gauge positioned opposite to the cut, where the highest strain due to relaxation is expected. Unlike the contour method, the CC method measures strain throughout the cut and provides information about each layer removed. However, the measured strains cannot be directly converted into stresses. Instead, FE simulations are conducted iteratively to simulate the relaxation process and determine an appropriate stress distribution for the given geometry. [31, 33–36]

The measurement process begins by identifying a suitable position on the component, ensuring a minimum distance of three times the diameter to the component edges in the case of wheelset axles. This distance requirement aims to ensure that the measured stresses are representative and uninfluenced. In the case of wheelset axles subjected to alternating bending, the middle region is of particular interest since it experiences the highest load stresses. Once the strain gauges are securely attached, the cutting process can commence, taking care to minimize the introduction of additional stresses. When using saws, it is recommended to make the cut as narrow as possible, and the feed rate should be kept as low as possible. Typically, a feed rate of 0.1 mm per cut has been found to yield satisfactory results. This incremental material removal is necessary for subsequent FE calculations. The cut is then made to the desired measurement depth, which is typically slightly deeper than the depth at which the stresses are to be evaluated. This additional depth allows for averaging of the stress distribution over a certain range. For example, if a 2 mm range is desired to determine the stress distribution up to a depth of 10 mm, the total depth of the cut should be 12 mm. While it is theoretically possible to perform a complete slicing of the component to obtain a comprehensive stress profile, experiments have shown that excessively large cutting depths can yield unrealistic results. [35]

Upon completing the measurement, the subsequent stage of stress determination using the CC method commences. Initially, the measured stresses are plotted as a function of the depth of the cut, and these data points are approximated using polynomials or other suitable mathematical representations to facilitate subsequent calculations. The stress vector $\boldsymbol{\sigma}_{CC}$ obtained from the CC method is derived by establishing the relationship between the strain vector $\boldsymbol{\varepsilon}_{CC}$ of the CC method and the compliance matrix $\boldsymbol{G}$. It is important to note that the compliance matrix is the reciprocal of the well-known elasticity matrix:

$$\boldsymbol{\sigma}_{CC} = \boldsymbol{G}^{-1} \cdot \boldsymbol{\varepsilon}_{CC} \quad . \tag{2.18}$$

While the compliance matrix establishes a direct relationship between the measured strains and the desired stresses, determining the coefficients of the matrix for each strain gauge necessitates the use of FE simulations. Unfortunately this is not a trivial task. An important prerequisite underlying the determination of these coefficients is the axial symmetry of the stress distribution within the component. Figure 2.8 illustrates the sequential exposure of axially symmetric stress layers as the sectioning progresses, providing an overview of the general framework of the CC method.

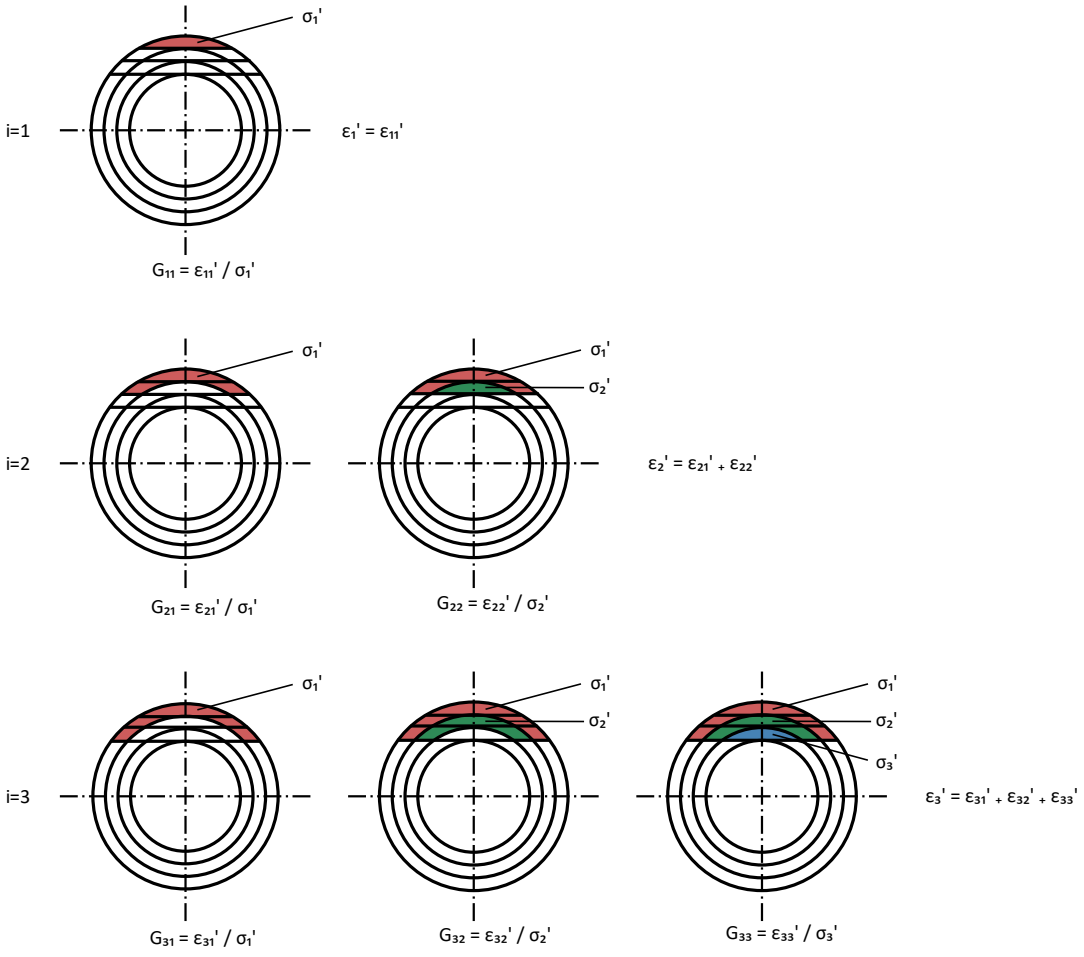


Figure 2.8: Determination of the coefficients of the compliance matrix. Figure taken from [35].

It is important to note that a single section cuts through multiple regions with varying stress levels. To determine the coefficients of the compliance matrix $\boldsymbol{G}$, an arbitrary stress value is assumed, resulting in a corresponding strain. By combining the known values of σ'_{ij} and $\varepsilon'_{\mathrm{ij}}$, the individual coefficient G_{ij} of the compliance matrix is obtained, as follows:

$$G_{ij} = \sigma'_{ij}/\varepsilon'_{ij} \quad . \tag{2.19}$$

The influencing factors G_{ij} of the compliance matrix are determined with the help of the finite element method (FEM). A separate FEM calculation is performed for each individual section that is cut. To reduce computational complexity, a stress value of $\sigma'_{ij} = 1$ MPa is

assumed, resulting in $G_{ij} = \varepsilon'_{\mathrm{ij}}$ for the influencing factors. As a result, the measured strain at the strain gauge is equal to G_{ij}.

In practical applications, it has been found that using polynomials alone for smoothing is insufficient. To address this limitation, regularization techniques are applied. Specifically, Tikhonov regularization is employed [37], leading to the following system of equations:

$$\left(\boldsymbol{G}^T\boldsymbol{G} + \beta\boldsymbol{C}^T\boldsymbol{S}^T\boldsymbol{H}\boldsymbol{S}\boldsymbol{C}\right)\boldsymbol{\sigma}_{CC} = \boldsymbol{G}^T \cdot \boldsymbol{\varepsilon}_{CC} \quad . \tag{2.20}$$

Here $\boldsymbol{C}$ is a matrix that determines the zero, first and second derivatives of the stress. It determines whether there is (0) stress uniformity, (1) stress gradient reduction, or (2) stress jump reduction. The $\boldsymbol{H}$-matrix weights the contributions of the individual evaluation points. If the spacing is constant, $\boldsymbol{H} = \boldsymbol{I}$. The matrix $\boldsymbol{S}$ is a diagonal matrix indicating the measurement error and β controls the amount of regularization.

2.4 Heat Transfer

To accurately model the heat treatment process of a railway wheel, it is crucial to determine the temperature distribution at every point of the component and at each point in time. Both material parameters that influence heat transfer, as well as those that leave the heat transport unaffected, are temperature-dependent. Furthermore, phenomena like phase transformations are as well influenced by both the temperature and the temperature rate. Therefore, it is important to explore the fundamental mathematical relationship of heat conduction in an isotropic solid. In addition, the feasibility of sequentially coupled calculations, eliminating the need for full coupling in both directions, to determine the stress field resulting from the heat treatment of railway wheels will be discussed. [38, 39]

The solution to the heat conduction problem involves calculating the temperature distribution as a function of time and space. This is achieved by applying the first law of thermodynamics in the form of its time derivative to a closed system, which can be expressed as the power balance equation

$$\frac{dU}{dt} = \frac{dQ}{dt} + \frac{dW}{dt} \quad . \tag{2.21}$$

The change in internal energy U over time is influenced by two factors: the thermal energy Q and the mechanical energy W that enter or leave the defined region through its boundaries. The change in internal energy can also be described by the volume integral of the specific internal energy u and the density of the material ρ by

$$\frac{dU}{dt} = \int\limits_{(V)} \rho \frac{du}{dt} dV \quad . \tag{2.22}$$

The specific internal energy u again results from the relation of a (temperature dependent) specific heat capacity c and the temperature θ to

$$du = c\left(\theta\right) d\theta \quad . \tag{2.23}$$

Taking into account the mass $m = \int \rho dV$, the time derivative of the internal energy thus results in

$$\frac{dU}{dt} = \int\limits_{(V)} \rho c\left(\theta\right) \frac{d\theta}{dt} dV \quad . \tag{2.24}$$

In order to calculate the heat flow $d\dot{Q}$ which moves over the surface of the area dA in direction of the normal vector $\boldsymbol{n}$, the following relationship results

$$d\dot{Q} = -\dot{\boldsymbol{q}}\boldsymbol{n} dA \quad , \tag{2.25}$$

where $\dot{\boldsymbol{q}}$ is the heat flux vector, a power density necessary to describe the heat flow.

Using the divergence theorem that states that the flux through a surface is equal to the volume integral of the divergence over the region inside the surface the heat flow can be rewritten to

$$\oint\limits_{(A)} \dot{\boldsymbol{q}}\boldsymbol{n} dA = \int\limits_{(V)} \nabla \dot{\boldsymbol{q}} dV \quad . \tag{2.26}$$

The law of heat conduction, also known as Fourier's[1] law, states that the rate of heat transfer through a material is proportional to the negative gradient in the temperature and the the area normal to this gradient and thus gives the equation for the heat flux with its proportionality factor λ, which is called thermal conductivity

$$\dot{\boldsymbol{q}} = -\lambda \nabla \theta \quad . \tag{2.27}$$

Finally the change of the thermal energy can be rewritten as

$$\frac{dQ}{dt} = \int\limits_{(V)} \nabla \left(\lambda \nabla \theta\right) dV \quad . \tag{2.28}$$

The power, the time derivative of the mechanical energy W, supplied to the region usually consists of two parts, a power that causes a change in volume and the power that is dissipated within the region. In the case of solids, such as railway wheels, the first part can be neglected and only the part associated with the dissipated energy remains leading to

$$\frac{dW}{dt} = \int\limits_{(V)} \dot{q}^{diss} dV \quad . \tag{2.29}$$

By combining the individual contributions of the energy and differentiating with respect to the volume the partial differential equation, which governs the temperature field in a solid yields as

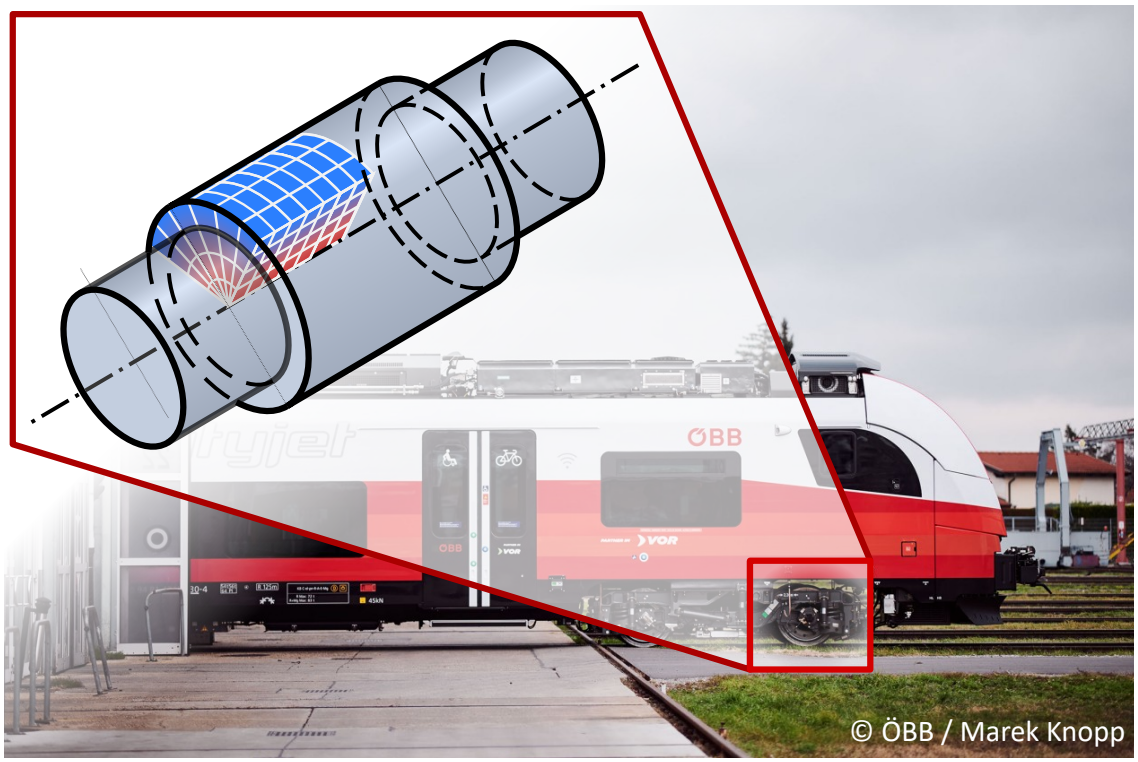
[1] Joseph Fourier (21 March 1768 - 16 May 1830): French mathematician.

$$\rho c\left(\theta\right)\frac{d\theta}{dt}=\nabla\left(\lambda\nabla\theta\right)+\dot{q}^{diss} \quad . \tag{2.30}$$

In various thermal calculations, the dissipated energy plays a crucial role in describing different effects, characterized by irreversible energy conversion. For example, in electrical calculations, it represents the heat generated by a resistor, while in mass transfer calculations, it accounts for heat sources resulting from chemical reactions. In mechanical calculations involving significant deformations, the dissipated energy arises from internal friction within the material. The magnitude of this fraction is influenced by various variables, including the strain rate, making it necessary to perform fully coupled calculations. Fully coupled means that the temperature field is dependent on the stress field and, conversely, the stress field is dependent on the temperature field. However, in the case of the heat treatment of the railway wheel examined in this work, large deformations are not involved, allowing the neglect of internal friction effects. Consequently, the dissipated energy term becomes independent of the strain rate and mechanical calculations, although it can still be utilized to account for other dissipative components, such as the latent heat of phase transformations. Thus, a sequential approach is adopted, wherein the thermal calculation is initially performed independently, followed by the mechanical calculation, which relies on the results obtained from the thermal analysis.

This concludes the Chapter Theoretical Background. With the knowledge gained on material models and residual stresses, the upcoming chapter delves into the specifics of cold rolled wheelset axles.

3 Cold Rolling of Wheelset Axles



This chapter focuses on investigating the effects of the cold rolling process on the wheelset axle component. It begins with a brief introduction to the component itself, including its requirements and manufacturing process. A closer examination is then given to the specific process of cold rolling. As the main objective of this work is process modeling, a section is dedicated to reviewing the current state of the art in modeling this process. This section provides insights into different modeling strategies, their respective advantages and disadvantages, as well as the selection of appropriate materials and material models. In order to develop a meaningful process model, the comparison with reality is crucial. Therefore, parts of various sections are dedicated to experimental approaches and the necessary measurements. The heart of this chapter is the Section 3.4 *Finite Element Model of Cold Rolled Wheelset Axles*. It details all the relevant models that have been developed in this work. Finally, this chapter concludes with the presentation and discussion of the results and findings obtained through these models.

3.1 The Component Wheelset Axle

Wheelset axles are key components for the railroad sector as they play a crucial role regarding safety. The wheelset comprises an axle with two wheels fitted tightly onto it. These axles act as a connection between the two wheels and facilitate the transfer of power and torque. A schematic representation of a typical wheelset configuration is depicted in Figure 3.1.

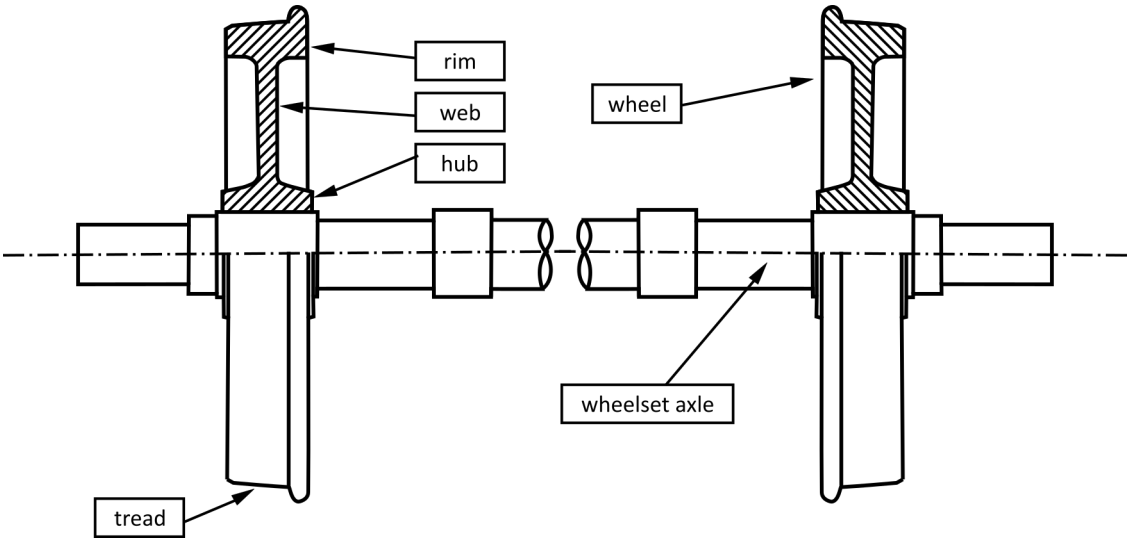


Figure 3.1: Wheelset comprised of an axle and two wheels including the destinations of the key components of a railway wheel.

In the field of railroad engineering, wheelset axles are categorized based on their functionality and geometry. There are two main types: powered and non-powered wheelset axles. Powered axles are responsible for transmitting torque and power to drive the wheels, while non-powered axles are primarily responsible for providing support and stability to the wheelset. Additionally, wheelset axles can have either hollow or solid geometries. Hollow axles have a central void or cavity, which helps reduce weight while maintaining the structural integrity. It is important to note that in railroad engineering terminology, the term "axle" is commonly used to refer to components that transmit torque, taking on the role of a shaft in conventional mechanical engineering. [15]

Wheelset axles are designed to withstand long service lives and a high number of load cycles, typically up to 10^9 cycles. According to Wöhler's concept of fatigue strength, components are designed to be fatigue-resistant up to approximately 10^8 load cycles. Beyond this point, the regime of ultra-high-cycle fatigue (UHCF) begins, which imposes further limitations on the permissible load.

To illustrate the relationship between the number of load cycles and the allowable stress amplitude, an S-N curve is commonly used. The S-N curve, typically plotted on a logarithmic scale, provides a graphical representation of the fatigue behavior of a material. In Figure 3.2, a schematic S-N curve is depicted, showing multiple stages. The ultra high cycle fatigue (UHCF) regime is marked as stage III. Despite being subjected to an extremely high number of load cycles, wheelset axles are considered to be particularly safe.

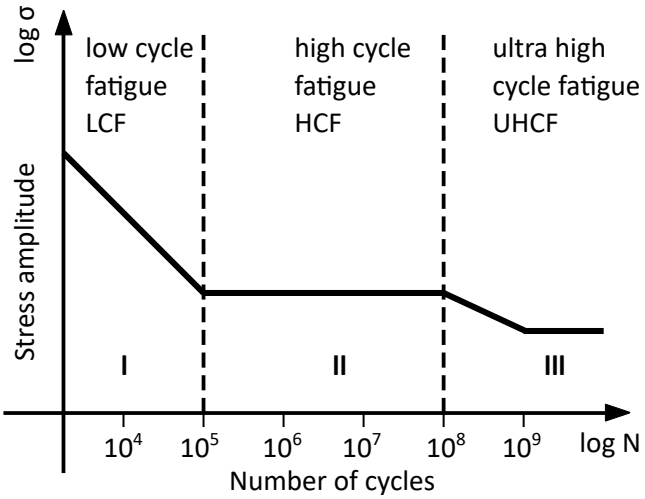


Figure 3.2: Schematic illustration of a multi-stage S-N curve.

Nevertheless, the consequences of an axle failure must not be completely disregarded and work must continue on increasing the safety of these components. Uncertainties present a particular difficulty in predicting component behavior. These include: changing load amplitudes, especially when a significant number of load cycles exceed the fatigue strength; changing or unknown track conditions; uneven distribution of loads in the cars; load peaks due to damage on the track or on other components of the train; and operator errors. This "safe-life" design approach is complemented by additional regular inspections to counteract the uncertainties mentioned above. [40–44]

3.1.1 Requirements for Wheelset Axles

The European standards EN13103 [45] and EN13104 [46] describe the design method for powered and non powered axles and the standard EN13261 [47] regulates the product requirements with focus on the material behavior. The axle design proceeds according to the following steps. Firstly, the forces acting on the axle must be precisely defined, and the resulting moments in all sections of the axle are calculated. Subsequently, the diameters of the axle body and journals are determined. Using these parameters, the stresses in each section of the axle are computed and compared against the maximum allowable stress limits. These limits vary depending on factors such as the steel type, whether the axle is hollow or solid, and whether it is powered or non-powered. For the definition of the forces the following influences are taken into account: moving masses, effects due to braking, effects due to curving, the wheel geometry, as well as effects due to traction. Since the wheelset rotates during the travel of a train, all loads are to be considered as alternating loads. These requirements result in a clearly specified manufacturing process. [45–47]

3.1.2 Manufacturing of Wheelset Axles

The casting process plays a crucial role in the manufacturing of wheelset axles, as it provides the initial cylindrical ingot from which the axle will be further processed. However, it is important to note that this stage of manufacturing can introduce potential sources of defects that may impact the performance of the axle in the long term. One of the primary causes of failure in the UHCF regime is the presence of non-metallic inclusions. [41]

In addition to employing continuous casting, steel plants also employ the method of individual ingot casting. This approach offers the advantage of enhanced control over the formation of non-metallic inclusions and other defects specifically for the production of wheelset axles. The process control is meticulously designed to ensure that potential defects are localized in regions that are not integral to the final component. Consequently, any defective areas can be precisely identified and removed through targeted cutting prior to subsequent processing stages.

The actual manufacturing process of a wheelset axle begins with the defect-free ingot. Figure 3.3 depicts the necessary steps of the manufacturing of a wheelset axle from ingot to shipment. First, the ingots are cut to the necessary pre-size to be subsequently preheated for the forging process. A press with semicircular dies forges the wheelset axle using the open-die forging process.

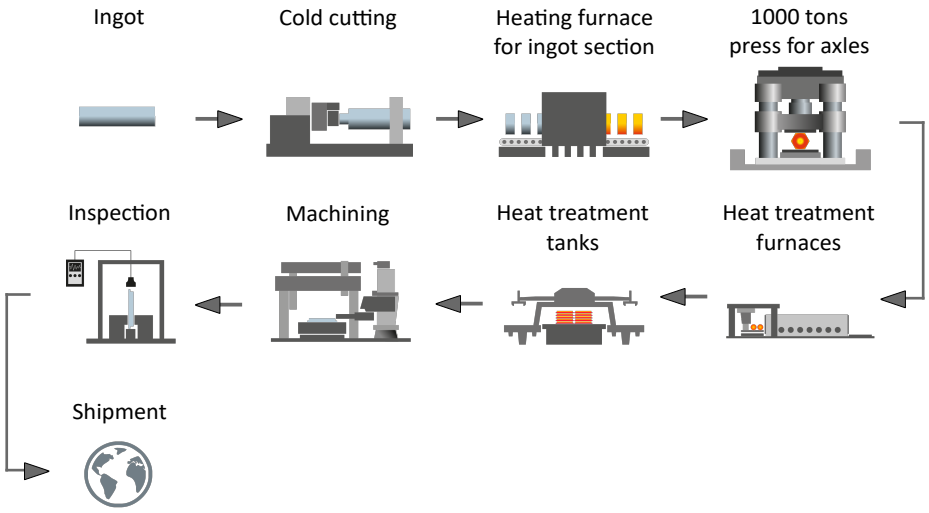


Figure 3.3: Manufacturing steps of a wheelset axle starting with a defect-free ingot with the process stages: cold cutting, heating of ingots, hammer forging, heat treatment in a furnace, heat treatment in a tank, machining, inspection and shipment. Figure adapted from [48].

The actual manufacturing process of a wheelset axle commences with a defect-free ingot. 3.3 depicts the necessary steps of the manufacturing of a wheelset axle from ingot to shipment. Initially, the ingots undergo cutting operations to achieve the required pre-size, followed by preheating in preparation for the subsequent forging process. One possibility to forge a wheelset axle is by employing the open die process with semicircular dies. Another possibility is the radial forging process. This forging processes involve significant plastic deformation, resulting in a

concomitant decrease of the component in radial direction and an expansion in the longitudinal direction. In cases where the forging cools below the recrystallization temperature, reheating is necessary to minimize forming forces and tool wear. The forging process serves two main purposes: geometric modification and alteration of the microstructure, thereby enhancing the mechanical properties of the wheelset axle and contributing to its improved service life and safety compared to non-forged counterparts. Following forging, the heat treatment stage ensues, encompassing microstructure normalization and subsequent quenching in a tank. Prior to being shipped to rail vehicle manufacturers, the finished components undergo meticulous quality inspections utilizing a variety of testing methods. Figure 3.4 presents an illustration of a completed wheelset axle. [15, 48, 49]

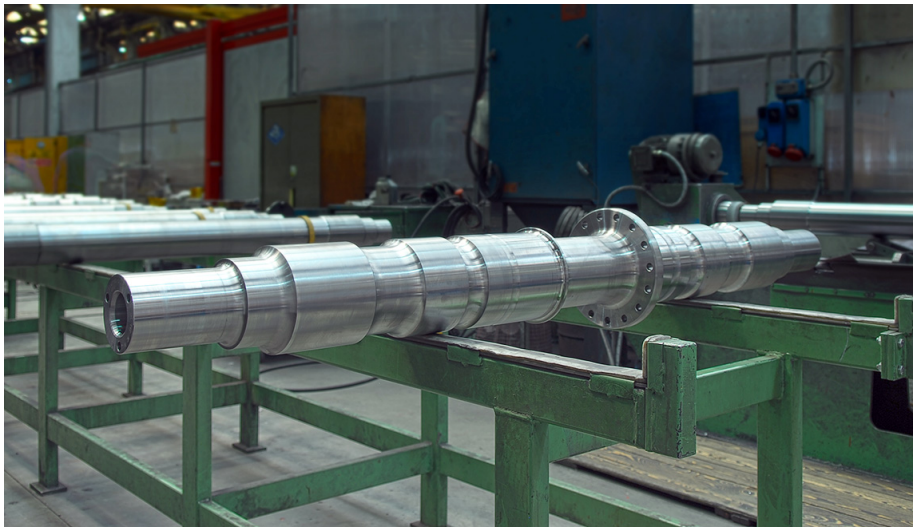


Figure 3.4: Finished wheelset axle. Figure taken from [49].

3.2 Cold Rolling Process

Cold rolling is a versatile and straightforward manufacturing process that can be adapted based on the type of tool and process control employed. Regardless of the specific variant, the fundamental principle remains consistent. During the process, the tool rolls over the component's surface while simultaneously applying pressure. At the contact point, the load is increased to a level that surpasses the material's yield point, causing plastic deformation. This plastic deformation results in residual stresses within the component's rolled-over region. With this work-hardening process local compressive stresses can be introduced to the components in a very targeted manner. [50–56]

The process of cold rolling has been extensively researched and has found widespread use in both manufacturing and maintenance applications. It is particularly suitable for enhancing the durability of components subjected to cyclic loading. Notably, Scholtes has made significant contributions to the understanding of mechanical surface treatment processes, with a specific emphasis on the resulting residual stresses [28]. Subsequent research efforts have focused on achieving precise descriptions of the cold rolling process itself, as well as investigating the

behavior of different materials. An essential aspect of these investigations is the accurate prediction of the residual stress distribution following cold rolling, as well as the subsequent redistribution of stresses during operational conditions. [50, 56, 57]

In addition to its positive impact on service life, cold rolling also imparts surface hardening to the component. Unlike alternative surface treatments like shot peening, cold rolling generates a smoother surface, facilitating more accurate prediction of component properties after manufacturing. Furthermore, the process offers the advantage of precise control over the depth at which residual stresses are introduced. According to [56], 85% of the damage to components originates from the surface zone. By introducing residual stresses that counteract applied loads, the service life of the component can be improved, as stress peaks in this region remain below the permissible tensile stress threshold. [50, 56, 57]

Figure 3.5 shows a schematic of the cold rolling process for cylindrical geometries, such as wheelset axles. In this particular example, a machine resembling a lathe is employed, where a tool known as the work roller is pressed against the surface of the wheelset axle with a specified force denoted as L in Figure 3.5. Both the wheelset axle and the work roller rotate around their respective axes (c.f. ϕ_{WR} and ϕ_{axle}), while the work roller also undergoes an axial[1] movement. This relative motion generates a helical path on the component's surface. The distance covered thereby in z-direction during one complete revolution is referred to as the feed, which is measured in millimeters per revolution.

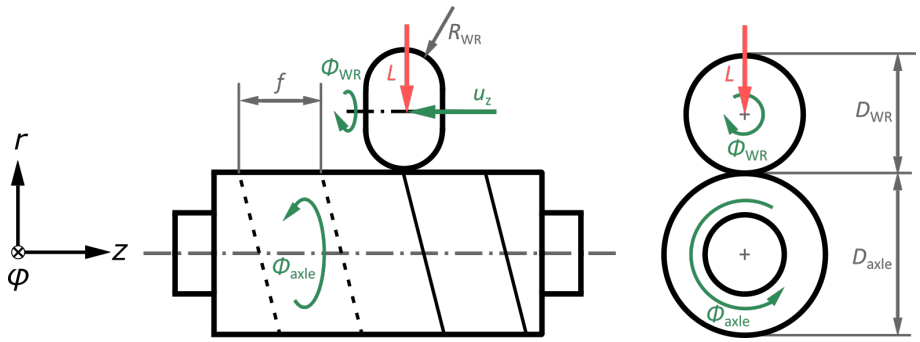


Figure 3.5: Simplified representation of the cold rolling process with the process-determining parameters: load (L), feed (f), edge radius of the work roller (R_{WR}), work roller diameter (D_{WR}) and axle diameter (D_{axle}).

Using the described process, it is feasible to roll larger areas. The rolling procedure commences on one side with a running-in phase, wherein the force is gradually augmented to attain the desired surface pressure. Rolling continues at this defined force until a transition point is reached. At the transition point, a run-out phase follows during which the force is gradually diminished. In contrast, transitions or ring notches are rolled in the single pass process without feed.

[1] Note that unless otherwise specified in this thesis, axisymmetric geometries use a cylindrical coordinate system. The designation of the axes is r, φ and z in radial, tangential and axial directions respectively.

3.3 Modeling of Cold Rolling

Despite its relative simplicity as a mechanical surface treatment, predicting component properties after cold rolling is a challenging task. Numerous calculation methods have been researched to predict various properties of components subjected to cold rolling. Numerical approaches are the most commonly used, but analytical methods have also been developed for specific component properties. The choice of the calculation method depends on factors such as component geometry or its simplified representation. Analytical methods may not be suitable for complex geometries but offer greater efficiency for simple ones. The modeling of the cold rolling process takes into account different geometries, the material properties, the desired component properties, the purpose of cold rolling, and the intended application, as well as the required level of accuracy in the results. The interest in modeling arises from the limitations of experimental methods in determining certain properties, such as measuring stresses at significant depths within the component.

Given that cold rolling involves contact between two bodies, an initial approximation is made using the Hertzian[1] theory, which describes the behavior of two bodies under a contact load. However, this theory is limited to purely elastic material behavior and does not accurately capture plastic behavior, leading to an underestimation of the contact area and an overestimation of the mean surface pressure. To address this, Maierhofer et al. [58] developed an efficient process model by combining the Hertzian theory with Prandtl's[2] theory and incorporating plasticity. This model provides a reliable initial estimation of the depth of residual stresses in axisymmetric components resulting from cold rolling. By using such a tool, it becomes possible to assess the potential suitability of the cold rolling process for enhancing the damage tolerance in early stages of the design. However, in advanced design stages, an accurate prediction of the stress distribution throughout the entire component depth is crucial. Numerical methods, such as the FEM, are well-suited for this purpose. [58]

In the field of numerical modeling of the cold rolling process, there have been numerous studies. To gain a comprehensive understanding of the current state of the art, previous works in this area are categorized based on the geometry and its simplification, material and models, and the specific quantities of interest of the investigation, respectively.

3.3.1 Geometries for Cold Rolling Simulations

A first approximation, initially independent of any particular geometry, is certainly the use of a two dimensional (2D) model. It is suitable for modeling processes without feed. Choi et al. developed in [59] a model for the cold rolling of fillets on crankshafts. Another example of a 2D simulation realized by Yen et al. is [60]. Here, the effect of smoothing an uneven or rough component surface is studied. A more recent example is the work of Mombeini et al. [61], which includes a 2D model of a section of a cylindrical fatigue test specimen and is used for analyzing the effect of cold rolling on the fatigue behavior. [59–61]

[1] Heinrich Rudolf Hertz (22 February 1857 - 1 January 1894): German physicist.
[2] Ludwig Prandtl (4 February 1875 - 15 August 1953): German engineer.

The majority of simulations is in three dimensional (3D). It is necessary to simplify the geometry of the component as much as possible, either to planar [54, 62–68] or cylindrical geometries [69–73]. Even complex components, such as turbine blades, can be approximated with planar cuboids by selecting sufficiently small sections [74, 75]. The three most commonly studied components by the FEM are crankshafts [59, 76], turbine blades [74, 75] and railway axles [68, 77].

When considering the geometry, or even just a section of it, the choice of boundary conditions has a major impact on the results. Especially when reducing the region of interest (ROI) to a section, it must be ensured that the boundary conditions are adapted to the component, the material and the process. 3.6 illustrates a simplified representation of the wheelset axle geometry as a cylinder sector. As a result of this simplification, appropriate boundary conditions need to be applied to the exposed surfaces of the cutout section. One possible approach is to define the boundary conditions sufficiently far from the investigated zone, so that different boundary conditions lead to the same result in the investigated zone. In the following, already implemented combinations of boundary conditions are presented. [70, 71, 73, 78] constrain the inside face of a cylindrical cutout and [72] additionally constrain the faces in tangential direction. When using cubic geometries, there is often a lack of information on boundary conditions, but it can be assumed that the surface facing away from the cold rolling process is fixed, as for example in [63–65, 65–68, 74, 75]. In [69] there are special boundary conditions. Here, the ROI is followed by semi infinite elements. At one end, these elements are connected to the component and at the other "infinite" end they are connected in such a way that the far field displacements approach zero [9]. Another method, which many works make use of, is the use of several parts fit and tied to each other. This method allows to create finely meshed areas in the ROI and insert them into more coarsely meshed peripheral areas, reducing the total number of elements of a model.

In summary, it is not possible to represent the entire component in a simulation of the cold rolling process. When choosing the section and the boundary conditions, it is necessary to adapt them to the exact application to be investigated.

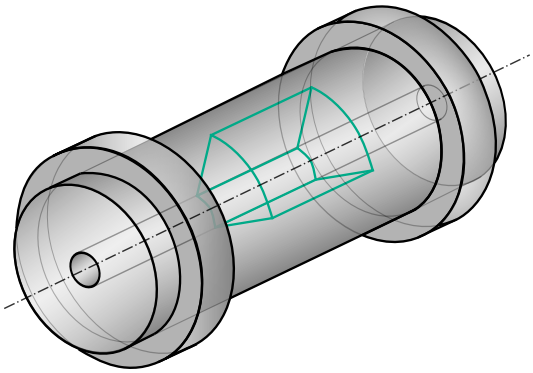


Figure 3.6: Simplified model geometry used in the FE model.

3.3.2 Materials and Material Models for Cold Rolling Simulations

The material selection is a crucial factor in process simulation, as many properties and outcomes are dependent on the chosen material. The material not only influences the process itself but also interacts with the geometry, thereby affecting the overall model. In the context of the cold rolling process, several materials are of significant interest, including steel, nickel alloys, aluminum alloys, and titanium alloys. These materials share a common characteristic, which is the enhancement of fatigue strength or improved surface quality as a result of cold rolling.

In many instances, a specific spot on a component undergoes multiple passes of cold rolling by the roller. As the material is subjected to successive passes it hardens. To accurately simulate this behavior, the material model employed should be capable of reproducing cyclic plasticity.

In [59] a new material model was developed that can reproduce anisotropic hardening behavior. Although this material model is particularly well suited for cyclic compressive loads, material models with a combined isotropic and kinematic material behavior prevail. This may be attributed to the wider availability and greater familiarity of these models within the field.

EA4T

The material used for the wheelset axles in this work is called EA4T. It is a standardized quenched and tempered steel for wheelset axles (EA4T). The chemical composition, mechanical properties and the necessary test methods are regulated in EN13261 [47].

C	Si	Mn	P	S	Cr	Cu	Mo	Ni	V
0.40	0.50	1.20	0.02	0.02	0.30	0.30	0.08	0.30	0.06

Table 3.1: Maximum percentage content of the various specified elements of EA4T according to [47].

EA4T is a low-alloy steel that has a long-standing history of usage in the railroad industry. In the late 2000s, the entire Deutsche Bahn (DB) Inter City Express (ICE) fleet transitioned from a higher-strength and more modern steel, which was chosen for lightweight construction purposes, back to the reliable EA4T steel. Although components made of EA4T are not as lightweight as those made of high-strength steels like 34CrNiMo6, they offer certain advantages. However, it should be noted that EA4T steel results in higher energy consumption during service and occupies more space in the passenger compartment. One significant advantage of EA4T steel is its well-established track record. It has undergone extensive testing and has proven to meet the stringent safety requirements set by railway operators and the general public. However, it is important to note that factors such as curve radii, track conditions, weather influences, and other variables can only be partially considered in the design and performance of components made from EA4T steel. Nonetheless, its extensive service on Central European railway lines has demonstrated its ability to meet the demanding safety standards of the industry. [79]

3.4 Finite Element Model of Cold Rolled Wheelset Axles

This section describes the FE model used for all calculations related to the cold rolling of wheelset axles. First, the requirements for such a model are discussed. This is followed by a subsection describing the selected geometry and focusing on the necessary model simplifications. Among other things, these simplifications can only be implemented if the boundary conditions are selected correctly. The subsection 3.4.3 *Coupling and Boundary Conditions* is devoted to this problem and its implementation.

3.4.1 Requirements for a FE Model of Cold Rolled Wheelset Axles

This FE model is mainly used to predict the production-related residual stresses of wheelset axles caused by cold rolling. Furthermore, it aims to identify the factors influencing the distribution of these residual stresses. The influences in this work refer primarily to parameters that can be adjusted in the course of the manufacturing process. In particular, the focus is on the local property changes caused by this manufacturing process.

The outcomes of the FE calculations serve as the foundation for a specialized software that accurately predicts the stress state of cold rolled wheelset axles. To achieve this, a considerable number of calculations are necessary, which impose a maximum computing time for each parameter set. Ideally, the developed process model enables automation of all calculations, facilitating integration into a higher-level optimization process.

In addition to accuracy and efficiency, the model should provide insights into the impact of subsequent maintenance measures on the residual stress state of cold rolled components. It should offer the capability to qualitatively predict the effects of maintenance actions and easily adapt to future calculations. The following subsections provide detailed explanations of how these complex requirements were incorporated into different aspects of the FE model.

3.4.2 Geometry for the Cold Rolling Process Model

In the context of cold rolling wheelset axles, the machining process involves treating significant sections of the axle surface. Consequently, it is generally desirable to employ a model that encompasses as large a portion of the geometry as possible. The wheelset axle's geometry is primarily characterized by rotational symmetry, suggesting the potential for an axially symmetrical approach in modeling. However, due to the multidirectional movements involved in the cold rolling process (radial, tangential, and axial), it is not feasible to reduce the model solely to its cross-sectional area.

Even if this simplification were achievable, it would still result in a model too large to thoroughly investigate local property changes induced by cold rolling. The challenge lies not in the disparity between the component size and the analyzed zone, but rather in the need for a fine resolution of results throughout the entire component, which cannot be consistently maintained. Consequently, it becomes necessary to focus on a small section of the wheelset axle and generate high-resolution and representative results in that region, with the aim of capturing the overall stress state of the entire component.

However, the roller used in the cold rolling process can come into contact with the edges or at least close to the edges of the simplified geometry, leading to heterogeneously distributed results in this area. The challenge of these edge effects can only be addressed by implementing carefully chosen boundary conditions, which play a crucial role in mitigating these effects and ensuring accurate and meaningful simulations.

3.4.3 Coupling and Boundary Conditions

The primary purpose of implementing boundary conditions in the model is to accurately represent the behavior of the entire component within the free-cut portion of the geometry. Thus, at least the faces with the colors yellow and salmon in Figure 3.9 need boundary conditions. Another consequence of the presence of the free-cut region is the disruption of the continuous nature of the cold rolling process in the simulation. This discontinuity can impact the fidelity of the results and the ability to capture the true behavior of the system. To address this issue, periodic boundary conditions (PBC) are employed. These conditions restore the continuity of the cold rolling process in the simulation, enabling a more faithful representation of the phenomenon under investigation.

Free cutting results in free surfaces in the tangential and axial directions. These surfaces require boundary conditions that meet the requirements already discussed. In order to maintain the overview of the boundary conditions at this point, they are subdivided as follows: (1) Coupling of nodes: the nodes of the free-cut rz planes are coupled together to create a periodic cell in the direction of rotation. (2) Shadow elements: the shadow elements, which are additional elements representing part of the surface of the missing geometry in the free-cut region, are coupled with surface elements of the actual model. (3) Boundary conditions at free-cut $r\varphi$ planes: these are meant to compensate for the missing geometry in the axial direction. (4) Tilting of the model geometry: the model geometry is tilted by a defined angle to represent the feed of the cold rolling process.

Before delving into the detailed description and implementation of boundary conditions, it is important to understand what periodic boundary conditions (PBCs) are and why they are necessary for this particular application. PBCs were first applied in the field of molecular dynamics. In principle, a large but finite simulation domain is chosen for such calculations and this domain is evaluated only in a small region far from the edges. The use of PBCs in all spatial directions results in an infinite grid of repeating regions of interest (ROIs). This works as follows: if a molecule leaves the ROI on one side, then the same molecule re-enters the model on the opposite side. This idea then made its way into continuum mechanics and was applied, for example, by Mayer et al. [80] in rolling contact simulations of railway wheels on rails. What makes the model of Mayer et al. special is the fact that the railway wheel can roll on the free-cut edges and even beyond. In analogy to molecular dynamics, the wheel rolls up onto the "beginning" of the rail when it rolls out over the "end". In the model of the cold rolling process presented in this work, this idea is applied, adapted and extended for the special motion procedure in cold rolling compared with rolling straight on a rail.

(1) The top part of Figure 3.7 shows one of the relevant couplings of this model. The coupled faces in the rz-plane are marked with the color magenta. The coupling itself is achieved applying equations to individual pairs of nodes of the FE mesh. The equations for the couplings must be defined for each node pair in the displacement degrees of freedom.

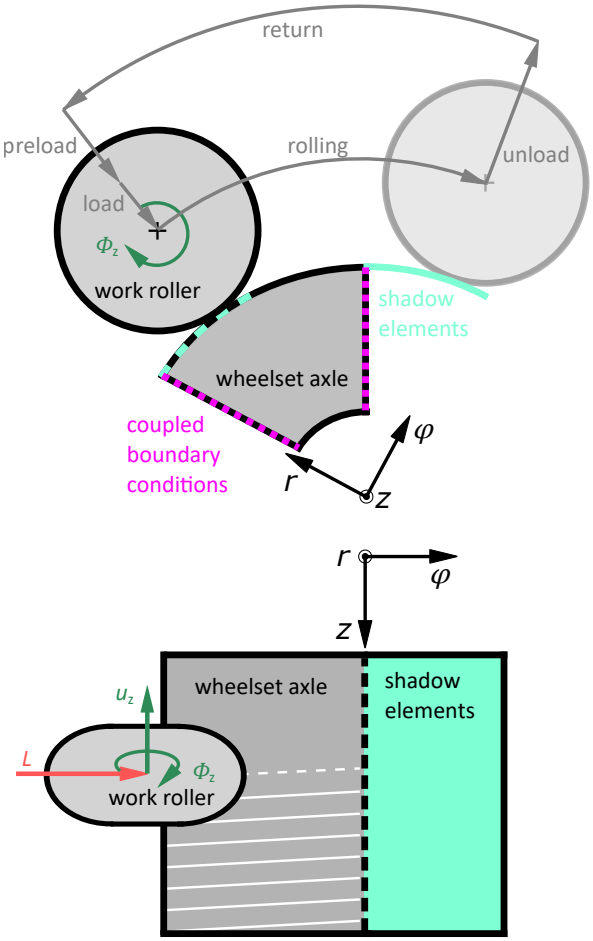


Figure 3.7: Coupling of nodes in the rz-plane and arrangement of faces of the simplified FE model.

To realize the implementation of these equations independent of the opening angle of the model, a separate local coordinate system is defined for each node. This allows to simply equate the displacement of the coupled nodes without having to consider the tangential part due to the opening angle. A set of equations for the coupling of nodes n^b_{11} and $\hat{n}^b_{11}$ in Figure 3.8 is defined as follows where $\mathbf{u}$ represents the displacement vector:

$$u_r^{n^b_{11}} - u_r^{\hat{n}^b_{11}} = 0 \tag{3.1}$$

$$u_\varphi^{n^b_{11}} - u_\varphi^{\hat{n}^b_{11}} = 0 \tag{3.2}$$

$$u_z^{n^b_{11}} - u_z^{\hat{n}^b_{11}} = 0 \tag{3.3}$$

This set of equations states that the distance between two coupled nodes must be constant. It follows that if one of these nodes moves, its coupled partner will also move. This creates a cell which is periodic in the tangential direction, and that repeats $360/\alpha_a$ times.

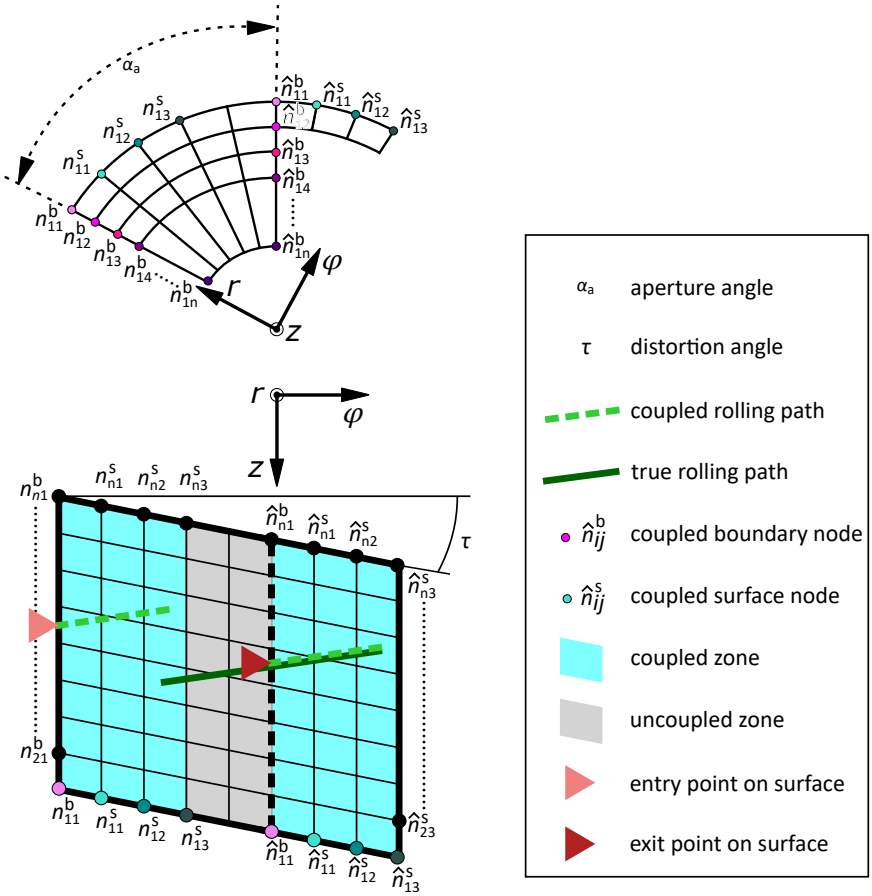


Figure 3.8: Coupling of nodes in the φz-plane and arrangement of faces of the simplified FE model.

(2) The previously described coupling establishes a periodic cell in the tangential direction but does not enable complete rolling over the edge. To address this, additional elements known as shadow elements are introduced. In Figure 3.7, these elements are depicted in cyan and are strategically positioned to extend the surface of the cylinder sector. It's important to note that the coupling occurs between the nodes on the surface of the shadow elements and the surface of the actual model, rather than between the elements themselves. The use of elements in this model simplifies the implementation of these nodes. The coupling starts with the first node after the edge, because the edge nodes were already considered in the first coupling. This coupling is shown in Figure 3.8, where the coupled nodes are depicted in different shades of cyan.

The surface nodes of the shadow elements are coupled to the surface of the actual model using equations that equate the displacement degrees of freedom. This allows the roller to roll over the edge and simultaneously reenter the model from the opposite side. In other words, during a rolling cycle, the roller follows the dark green path in Figure 3.8. For the cylinder

sector, it "feels" as if the roller is moving along the light green path. It is the combination of both, the two boundary conditions and couplings that allow the roller to smoothly traverse the edge, thereby transforming the discontinuous process of cold rolling a cylinder sector into a continuous process. This approach ensures that the simulation accurately captures the helical path of cold rolling a cylinder and preserves the overall continuity of the process.

(3) The primary objective of this model is to accurately determine the stress state, with a particular focus on the stresses in the axial direction. During the cold rolling process, a complex multi-axial stress state arises, which is predominantly influenced by the boundary conditions in the axial direction. Hence, it is crucial to carefully select the appropriate boundary conditions on the faces of the $r\varphi$ plane.

Throughout the development of this model, several variables that affect the stresses in the axial direction have been identified. These variables include the length of the model in the axial direction, the number of constrained degrees of freedom on the side surfaces, the specific location on the model where the boundary conditions are applied, and the utilization of special features in conjunction with the boundary conditions.

Extensive investigations have revealed that using a model length that is either too large or too small has undesirable effects on the calculation time and the accuracy of results. Excessively long model lengths do not significantly improve the accuracy but increase computation time, while excessively short model lengths lead to artificially high stresses due to a stiffening effect. Furthermore, convergence issues arise when the roller is positioned too close to the edges of the model.

An alternative approach considered was to simulate the fixed-loose bearing arrangement of the wheelset axle during the cold rolling process. In this case, the face on the fixed bearing side was fixed in all spatial directions, while the face on the loose bearing side was allowed to deform in the axial direction. Although this boundary condition variation yielded satisfactory results in the near-surface region, it produced unrealistic outcomes as the depth increased.

Throughout these investigations, various positions for applying the boundary conditions were tested, including individual edges, entire faces, and different combinations thereof. However, the results exhibited considerable variation, indicating that none of these arrangements accurately represented the real-world scenario.

Based on this empirical knowledge, a special feature, namely springs, was introduced to the faces in the axial direction. The concept behind this approach is to strike a balance between the stiffening effect of fully constrained edges and the underestimation of stresses caused by completely free edges (even in only one spatial direction). By implementing the springs, a compromise solution is achieved that better approximates the real stress distribution in the wheelset axle during the cold rolling process.

By incorporating spring elements, a solution is achieved that lies between the extremes of fully constrained and completely free edges, providing better control over the boundary conditions. The model is reconstructed using a different coupling approach compared to previous ones. Instead of coupling individual nodes, all nodes on a surface are coupled to a reference node. This reference node serves multiple purposes: it enables better control over

the boundary conditions and facilitates the application of loads through the aforementioned spring elements.

As depicted in Figure 3.9, a spring element is positioned between a fixed node at the boundary and the reference node to which all nodes on the free-cut surface are coupled. This configuration allows for displacements in the axial direction, which, in turn, induce stresses in the model. The stiffness of the springs can be easily adjusted to meet specific requirements, offering flexibility in controlling the behavior of the boundary conditions.

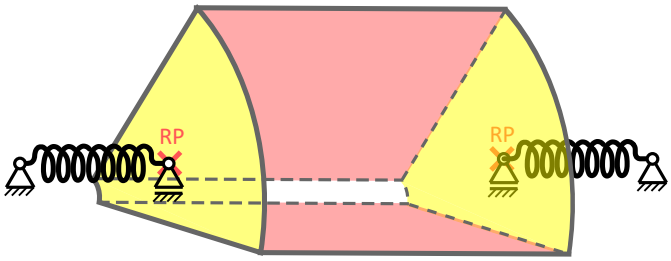


Figure 3.9: Spring elements used in FE model.

The determination of the spring stiffness for the inserted spring element is based on considering the removed or cut-off portion of the wheelset axle as a bar spring. In the scenario where the cold rolling process is simulated at the center of the wheelset axle, the length of the bar spring is chosen as 1 meter, and the modulus of elasticity is assigned as that of steel.

(4) The non-perpendicular sides of the cut-out cylinder sector depicted in Figure 3.8 necessitate the introduction of a distortion angle, denoted as τ, in order to model the roll's feed using the couplings described earlier. The feed causes the roller to move in a helical pattern across the surface as it rolls over. To compensate for the missing geometry of the cylinder, the roller needs to advance the distance it would have traveled on the remaining geometry when it rolls back onto the cylinder sector. However, this presents two challenges: the need to adjust the feed rate between simulations and the potential difficulty in reproducing the exact feed rate using finite element subdivisions.

By distorting the geometry by the τ angle, both challenges can be addressed simultaneously. A uniform finite element mesh is employed, and the angle only needs to be adjusted according to the desired feed rate. This distortion allows the roller to move in a helical path over the surface, and the coupled nodes at the edges are positioned relative to each other in a way that compensates for the roller's virtual completion of the rollover offset due to the feed rate. The entire geometry can be adjusted based on the feed parameter using a Python script for parametrization.

In summary, the combination of the various boundary conditions used in the finite element model of the cold rolling process ensures both a realistic representation of the process and computational efficiency.

3.4.4 Material Model for Cold Rolling Simulation

As discussed in Subsection 3.3.2, a material model is required that can represent cyclic-plastic behavior. The underlying constitutive equations are given in Chapter 2.

The calibration of the material model used in the analysis involves conducting low cycle fatigue (LCF) experiments. These experiments are performed under controlled strain conditions at various load levels. Typically, the experiment maintains a constant load level until a stabilized state is achieved before applying the next higher strain level. Additionally, the stress ratio is varied to account for further influences on the material model, with a stress ratio of $R = -1$ being of particular significance.

The experimental data obtained from the LCF tests is used to fit the parameters of the Lemaitre-Chaboche model. The fitting process aims to find the parameter values that best replicate the behavior observed in the experiments. This calibrated material model is then tested for its suitability in numerical simulations.

The parameter set for the simulations in this work is provided in Table 3.2. It is important to note that the literature does not always specify the number of cycles for which a material model has been fitted. However, this information is crucial in process simulations, as the model's ability to accurately capture the cyclic behavior over a specific number of cycles is essential for reliable predictions.

$\sigma\|_0$	Q_∞	b	Index k	C_k	γ_k
(MPa)	(MPa)	(-)	(-)	(MPa)	(-)
			1	432369	3592.6
345.7	-150	100	2	78409	470.2
			3	7023	47.4

Table 3.2: Parameters of combined isotropic-kinematic hardening behavior of EA4T used for FE simulations.

In the context of simulating the cold rolling manufacturing process, it is necessary to conduct experiments with a specific range of cycle numbers, typically between 10 and 100 cycles. This range has been found to yield accurate results when utilizing the material model for simulation purposes. This material model has been fitted to 100 cycles, which suits the purpose of this work. However, it should be noted that this particular parameterization is not suitable for simulating the operational behavior of the wheelset axle, which experiences significantly higher cycle numbers ranging from 10^5 to 10^9 cycles. To effectively capture the long-term cyclic behavior in such operational scenarios, alternative experiments and parameter values must be employed to calibrate the material model appropriately. These considerations ensure that the simulation accurately reflects the observed behavior under the anticipated cycle range in real-world operational conditions.

3.4.5 Kinematics of Cold Rolling Simulation

The simplification of the geometry in the cold rolling process transforms the originally continuous process into a discontinuous one. Consequently, the kinematics of the simulation, including the motion steps, are also affected by this change.

In order to accommodate the force-controlled nature of the cold rolling process, the simulation makes use of special features offered by finite element (FE) programs, known as multi point constraint (MPC). MPCs are utilized to fix or connect degree of freedom (DOF) of individual nodes or components within the simulation. Similar to boundary conditions, MPCs enable the design of relative movements between different parts of the system. Most FE packages provide predefined MPCs, such as connector elements in Abaqus, which are named based on their intended purpose and specify the fixed DOF in each case.

This model incorporates two connector elements to facilitate the force-controlled movements required in the cold rolling process. The first connector element utilized is the Translator, which enables movement of the roller along the r-axis. In this case, all but this translational DOF are fixed, allowing free movement in the first direction of the locally defined coordinate system. The second connector element employed is the Hinge, which permits free rotation of the roller around its own axis.

The configuration of these features is depicted in Figure 3.10, where they are connected through individual reference nodes. The combination of the RP Center and RP Hinge forms the MPC Translator, with RP Center being stationary and RP Hinge capable of movement along the connecting axis. The Hinge connector is positioned between RP Hinge and RP WR, with RP Hinge being fixed while RP WR is allowed to rotate freely relative to it.

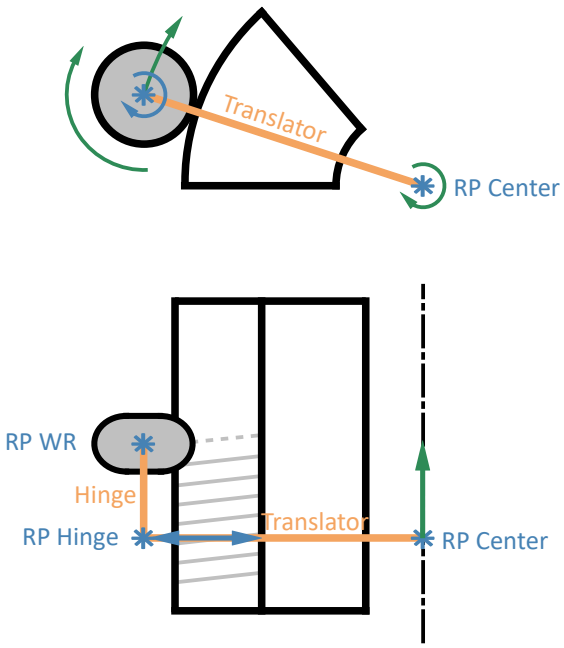


Figure 3.10: Multi point constrains of the FE model. Gold: connector elements, blue: reference points, green: DOFs of components.

The force-controlled loading of the cold rolling process is initiated by the RP Hinge and can be adjusted according to specific requirements. The rotation of the roller is also precisely defined, with the default setting allowing the roller to roll over the wheelset axle surface without experiencing friction. The Python (Py) script calculates the necessary rotation based on the selected geometry. For processes involving slippage or similar phenomena, the rotation can be adjusted accordingly.

A successful rollover of the roller requires a specific sequence of steps, which are: (1) preload, (2) load, (3) rolling, (4) unload, and (5) return. These steps are depicted schematically in Figure 3.7 at the top. The division into these individual steps serves to effectively control the different motions involved and efficiently handle the numerical nonlinearities.

(1) The preload step serves the purpose of establishing contact between the roller and the surface of the wheelset axle. In the initial cycle, this contact is already present since the axle surface is perfectly cylindrical and the roller is initially positioned precisely on the surface. However, even the first rolling causes permanent deformation of the surface, which cannot be determined a priori. As the number of cycles increases, the surface of the wheelset axle continues to evolve due to material hardening, leading to changes in plastic strains and consequently altering the surface position.

In an earlier version, the preload step was controlled based on displacement, with minimal adjustments made to ensure contact between the two parts. However, it was observed that during certain phases of the simulation, the displacement-controlled preload triggered reaction forces higher than the maximum forces occurring in the load step. To address this issue, the preload step is now stress controlled, employing a negligible load on the order of 1% of the total load. However, stress control introduces other numerical challenges. If the roller has no initial contact with the wheelset axle during the preload step, applying a force would result in infinitely large acceleration and lead to numerical instabilities.

To overcome this problem, a workaround is implemented by adjusting the contact settings to virtually bring the two bodies into contact, even though there is a small gap between them. However, it is crucial to use such settings judiciously and ensure they do not impact the simulation results. Therefore, this setting is enabled only during the preload step and disabled in all other steps. This approach allows the roller to establish contact with the wheelset axle in a stable manner without altering the simulated process itself.

(2) The second step of a cycle is called Load and is used to apply the full force to the roller. This step also reestablishes the correct contact settings and begins the actual cold rolling process. When using the default load application settings, the force is applied linearly over the step time and reaches the desired magnitude at the end of the step.

(3) During the rolling step, the actual process of cold rolling takes place. In the previous steps, all degrees of freedom of each component were fixed, except for the translator that allowed movement along the r-direction while preventing any other movements. However, in the rolling step, the entire movement of the roller, including rotation about the center axis of the wheelset axle and translation along this axis, can be controlled.

This control is achieved through the use of the reference point (RP) Center, which is connected to other reference points via connectors, enabling the roller to move over the surface

of the component. The translational part of this motion represents the feed rate of the cold rolling process. In Figure 3.10, these motions are depicted in green color, indicating the controlled movement of the roller during the rolling step.

(4) and (5) In the unload and return steps of the process, the roller is maneuvered back to its initial position for the next rollover. This involves overcoming the remaining rotation and ensuring that the roller precisely returns to the same position it occupied before, virtually through the coupling mechanism.

To visualize the actual and virtual positions of the roller, a comparison can be made between the true rolling path and the coupled rolling path. This comparison provides a better understanding of how the roller moves during the process and how its position is controlled through the coupling. Figure 3.7 illustrates this comparison, highlighting both the true rolling path and the path achieved through the coupling mechanism.

3.4.6 Contact Settings

When simulating the cold rolling process, two components come into contact. Modeling the contact introduces another nonlinearity in addition to geometry and material. This model uses the extended Lagrange[1] contact formulation. It follows three steps: (1) the solver finds a converged solution using the penalty method, (2) if a slave node penetrates the master and exceeds the maximum penetration depth, the contact pressure is "increased" and iterations start over until convergence is regained, (3) this contact pressure is changed until the penetration depth falls below an allowable maximum value.

The use of this contact formulation leads to a higher number of iterations and thus to higher computation times, but is characterized by high stability and avoids problems of overconstraints. [9]

3.4.7 Python Script for Model Generation

The model of the cold rolling process described in Section 3.4 is controlled by a python (py)-script. The geometry is built and adapted to the process parameters as discussed in the tilted mesh example in subsection 3.4.3. All described couplings are defined node by node. This requires precise search algorithms that find the nodes in the model and couple them correctly. Properties such as the part geometry, the underlying material model, and all process parameters of the cold rolling process can be changed quickly. This ease of use allows to perform parameter studies and to investigate the influence of individual process parameters on the residual stress state. The combination of individual couplings and the resulting reduction of the geometry improve the calculation time by a factor of 25 compared to previous models of the cold rolling process. The boundary conditions with the additional spring elements homogenize the results and increase the accuracy, especially in deeper regions below the surface. See Appendix A.2 for the code of the py script for Abaqus used for the fully automatic generation of the finite element models.

[1] Joseph-Louis Lagrange (25 January 1736 - 10 April 1813): Italian mathematician, physicist and astronomer.

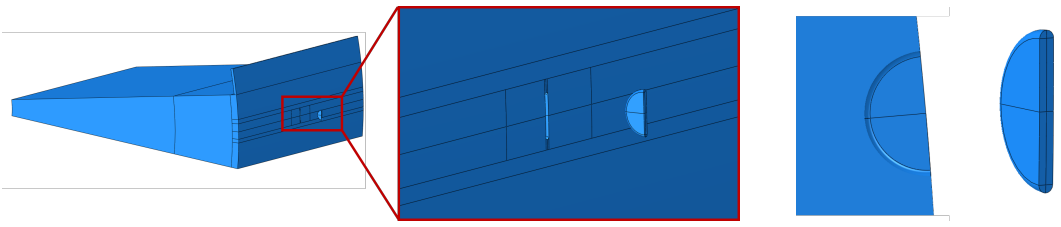
3.4.8 Model Extension

The FE model of the cold rolling process described so far represents an idealized scenario where the surfaces are initially perfectly smooth and the cylinder sector represents the central area of the wheelset axle. However, in real-world operation, these components are subjected to additional hazards beyond high loads. One common issue is ballast impact, where individual stones from the ballast bed are thrown up by the train and strike the undercarriage components. This impact can lead to the formation of notches, which serve as potential initiation sites for crack growth. Furthermore, these impacts can cause paint damage, exposing the surface to corrosion. To demonstrate the capabilities of the model for further calculations, this subsection presents an example of a possible extension to incorporate these additional factors into the existing model. [81]

The objective of extending the existing model is not to simulate the impact of a rock on the surface of the wheelset axle, but rather to gain insights from experimental studies. Pourheidar et al. [82] conducted experiments on full-scale wheelset axles, aiming to investigate the behavior of crack propagation under rotating bending and corrosion conditions in cold-rolled wheelset axles. During the study, notches were intentionally introduced on the surface to examine the formation and propagation of cracks. However, one aspect that remained unresolved in the experiments was the stress state in the vicinity of the notch. To address this aspect, the existing FE model was expanded.

To incorporate the notch into the surface of the FE model, a suitable location needs to be chosen where no existing couplings are present. In Figure 3.8, the gray area is designated for this purpose as it is free from couplings. This region serves as a neutral zone between the coupled surface and the shadow elements, allowing for the insertion of additional features such as notches or cracks. The objective in this instance is to replicate a semicircular notch, resembling the one employed in Pourheidar's [82] experiments.

The notch is semicircular in shape with a depth (or radius) of 3 mm. Its width measures 0.3 mm, resulting in an edge radius of 0.15 mm at the notch. Figure 3.11 provides a schematic representation of the notch on the axle surface. To prevent the introduction of additional stresses from the machining process, the notch is created using electrical discharge machining.



(a) Partitioning of model with a notch as a partition.

(b) Detail view of notch geometry.

Figure 3.11: Model geometry of cold rolled axle with notch.

The model needs to fulfill the following requirements: (1) ensuring the wheelset axle and notch dimensions match the tests, (2) maintaining consistency with the cold rolling parameters used in the tests, (3) introducing the notch with minimal additional stress, and (4) evaluating the stress state before and after introducing the notch.

To accommodate the 6 mm diameter of the notch in the uncoupled zone, the model must be enlarged. However, this poses a challenge of maintaining fine geometries while managing the dramatic increase in the number of elements. To ensure reasonable computational time, the model needs to be partitioned strategically, reducing resolution near the notch while still obtaining meaningful results. Another complexity arises from the requirement of having identical element sizes at the transition between coupled and uncoupled zones to facilitate accurate coupling assignments. This necessitates partitioning the wheelset axle and specifying the mesh in a way that preserves the original coupled areas at the edges. Figure 3.12 provides a front view of the wheelset axle surface and a detailed view of the transition region between the coupled and uncoupled zones.

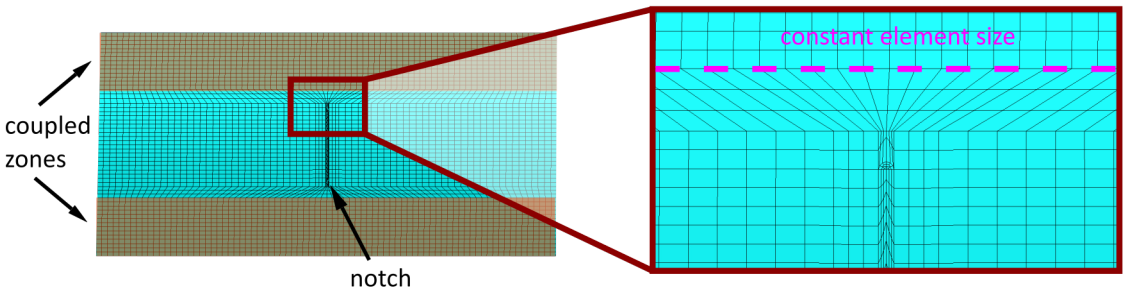


Figure 3.12: Front view on the mesh of the wheelset axle surface and a detail view on the transition between the coupled and uncoupled zones.

After the partitioning of the wheelset axle, the notch can be removed via element deletion. This is achieved by running the simulation of the cold rolling process as usual. At the end of the simulation, an additional step is included to remove the notch and analyze the resulting stress redistribution. It's important to note that this removal of the notch is an idealized geometry change and does not introduce any additional stresses.

3.5 Results and Discussion

The results section of the study primarily examines the distribution of residual stresses in the cold rolled wheelset axle. It begins with a validation process to ensure the appropriate size of the model. Subsequently, a comprehensive analysis is conducted to investigate the distribution of residual stresses throughout the depth of the axle. This analysis provides a detailed understanding of the residual stress profile. Furthermore, a parameter study is conducted, which leads to the development of a software tool capable of predicting the residual

stress profile of cold rolled wheelset axles. This tool enables the estimation of residual stress distribution based on various process parameters. Lastly, the section discusses the redistribution of residual stresses resulting from maintenance procedures or the presence of a notch, as described earlier. This examination sheds light on how these factors impact the distribution of residual stresses within the wheelset axle.

3.5.1 Validation of Model Size

This subsection is partly taken verbatim from [83].

One of the questions that arose with the first results of the developed cold rolling model was to which extent it would be possible to reduce the size of the model through homogenization. Due to the fact that the computational time of previous models was very large, no relevant advantage over other methods was achieved. However, by creating the periodic cell, the boundary effects caused by ordinary boundary conditions are eliminated.

Figure 3.13 (a) shows a stress contour plot of a first attempt to model the cold rolling process with the possibility of rolling over the edges of a cylinder sector with a total opening angle of 45°. This model consists of two parts, a plastic region of interest in the center surrounded by an elastic part. The rollover starts at the elastic part and continues over the entire plastic region before lifting off from the elastic part on the other side. This model is used as a reference for the updated model described in Section 3.4.

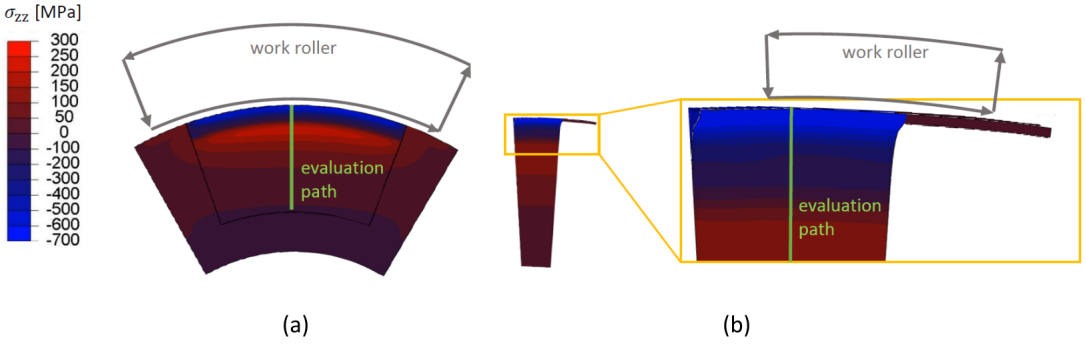


Figure 3.13: Contour plots of the residual σ_{zz} stresses, the position of the evaluation path and the modeled movement of the work roller:
(a) Reference model with 45° aperture angle of the cylinder sector and (b) the corresponding updated model with 6° aperture angle with an additional detail view.

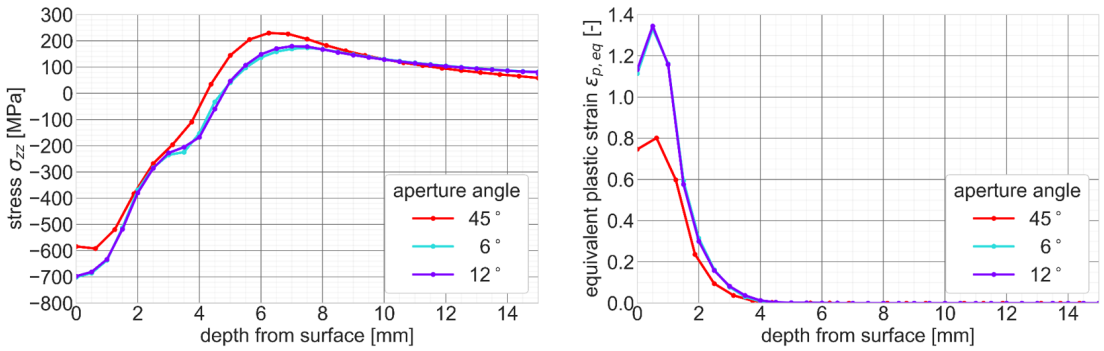
It is important to note that all three of the 6°, 12°, and 45° opening angle models are supported axially by a fixed/free bearing arrangement. The model size refers only to the φ-direction. The outlined results already include the update with the springs for bearing in the z-direction. The difference in the results compared to the updated model can be seen from about 5 mm below the surface. However, they are sufficient to illustrate the effects of size in the φ-direction.

Figure 3.13 (b) shows the stress distribution of the model described in section 3.4 with an aperture angle of 6°. The distribution of residual stresses in the tangential direction is more

homogeneous compared to the reference model with an aperture angle of 45° in Figure 3.13 (a). In addition, the number of elements is significantly smaller (200 000 for the reference model and 80 000 for the updated model), although the mesh size in the tangential and radial directions could be decreased by about 50 % per side. Thus, the mesh of the updated model is finer and can better reproduce the high stress and strain gradients. The biggest difference between the two models is the computation time. It is reduced from approximately 24 hours per rollover for the 45° aperture angle reference model to less than one hour for the 6° aperture angle updated model using an Intel Xeon CPU with 64 GB of memory for both computations.

In order to achieve a steady state after cold rolling, a certain number of rollovers must be achieved in the simulation. The evaluation path in Figure 3.13 is located at half the length of the cylinder sector, which is also half the rolled-over length in the axial direction (i.e., the number of rollovers times the feed rate). The stress state in this path is already influenced before the work roller comes into direct contact with this path and continues to change while the work roller rolls over the zone behind the path. Considering the material model and the possible values of the process parameters for wheelset axles, at least 30 rollovers are required to reach a steady state in the region of interest, resulting in a total computation time of less than 30 hours.

Figure 3.14 shows the axial stress σ_{zz} and equivalent plastic strain ε_{eq}^{pl} distributions evaluated along the evaluation path of Figure 3.13. The results labeled 45° correspond to the reference model in Figure 3.13 (a) and those labeled 6° and 12° correspond to the updated model in Figure 3.13 (b). The good agreement between the results of the 6° and 12° model shows that the reduction to a 6° aperture angle is acceptable since they were calculated with the same element size. The different results between 6° and 45° in Figure 3.14 (a) can be explained by the choice of a finer mesh in the tangential and axial directions, since smaller elements can better resolve the high stress gradients towards the surface. The different results in Figure 3.14 (b) can be explained by the same argument, although the difference for the equivalent plastic strain ε_{eq}^{pl} is even more pronounced. This also indicates that the element size of the reference model does not lead to an adequate resolution of the plastic deformations in the cold rolled zone.



(a) (b)

Figure 3.14: (a) Residual σ_{zz} stresses and (b) equivalent plastic strain ε_{eq}^{pl} distributions of the updated model with aperture angles 6° and 12° and the reference model with an aperture angle of 45°. The two curves of the updated model with 6° and 12° nearly coincide.

The simulation methodology with periodic boundary conditions is equivalent to multiple work rollers cold rolling the axle simultaneously. This means that in the model with an opening angle of 6°, there are 60 rollers distributed around the circumference in a row along the helical rolling path, each cold rolling an angle of 6°. By further reducing the opening angle, the stress fields of the individual work roller could influence each other and thus affect the results. It is also important to ensure that the size of the contact area is a fraction of the area of the cylinder sector in order to prevent the contact from being overlapped by the shadow elements.

In summary, the following statements can be made about the model size: (1) the periodic boundary conditions allow a significant reduction in the opening angle, (2) the combination of component geometry and roller size determines the size of the contact patch. The contact patch must not exceed a fraction of the size of the uncoupled zone to ensure that no feedback caused by the coupling occurs in the model. (3) The ratio of cylinder sector size to shadow elements must be adapted to the process. A ratio of 2:1 of cylinder sector:shadow elements has been found to work well. This ensures that the uncoupled zone is the same size as the protruding shadow elements, and adjusting the size from point (2) ensures that the shadow elements are also of a size that will not cause problems.

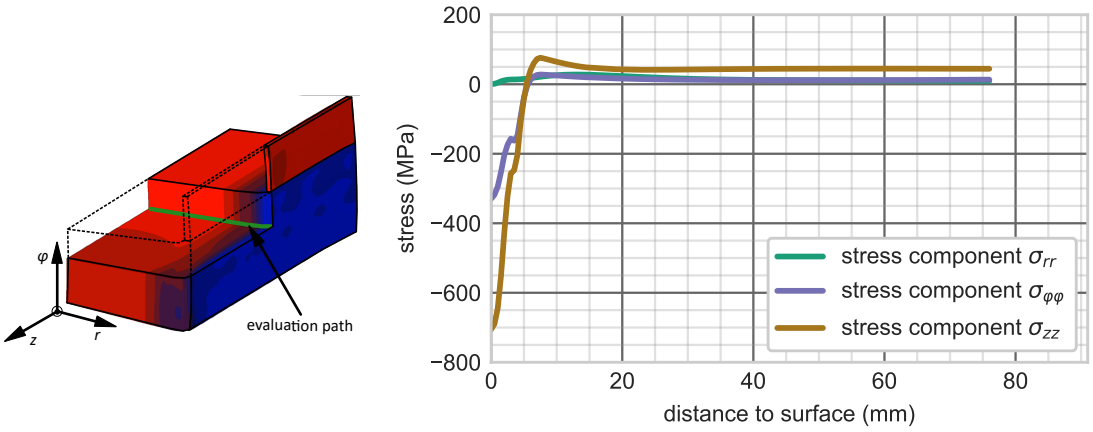
3.5.2 Stress Distribution in Cold Rolled Wheelset Axles

The previous subsection demonstrates the impressive capability of simple models with coarse resolution to accurately represent the stress state in the near-surface region. However, as the depth increases, deviations arise between the models and the expected values in terms of mechanical stress equilibrium. The following considerations are made regarding this equilibrium: The stresses or forces must be in balance. It is assumed that the stresses at the surface are already accurately represented, that the stresses in the axial direction remain unchanged over long distances, and that the plastic strains occur only in the region close to the surface. If these assumptions hold true, the stresses at greater depths should approach a constant value and, most importantly, be in equilibrium with the surface stresses.

A typical stress profile, represented by σ_{zz}, exhibits compressive stresses at the surface of the wheelset axle. As depth increases, these compressive stresses gradually decrease until they eventually transition to tensile stresses. The depth at which the compressive stresses penetrate is a significant characteristic of the cold rolling process. At a certain depth along the stress curve, the tensile stresses reach a maximum peak before gradually decreasing to a constant value that persists towards the center of the wheelset axle. Figure 3.15 illustrates such a curve spanning the entire cross section of the wheelset axle, which is designed as a hollow shaft. For a closer examination of the near-surface area, Figure 3.16 provides a detailed view of the same stress curve. It is noteworthy that cold rolling typically results in an almost planar stress state near the surface, with primarily σ_{zz} and $\sigma_{\varphi\varphi}$ stresses being present. The z-direction is particularly relevant for design considerations involving rotating bending, hence the subsequent results predominantly focus on stresses in the z-direction.

For a further assessment of the tensile stress state, an equilibrium is calculated between the compressive stresses from the simulation and the tensile stresses that would form if they

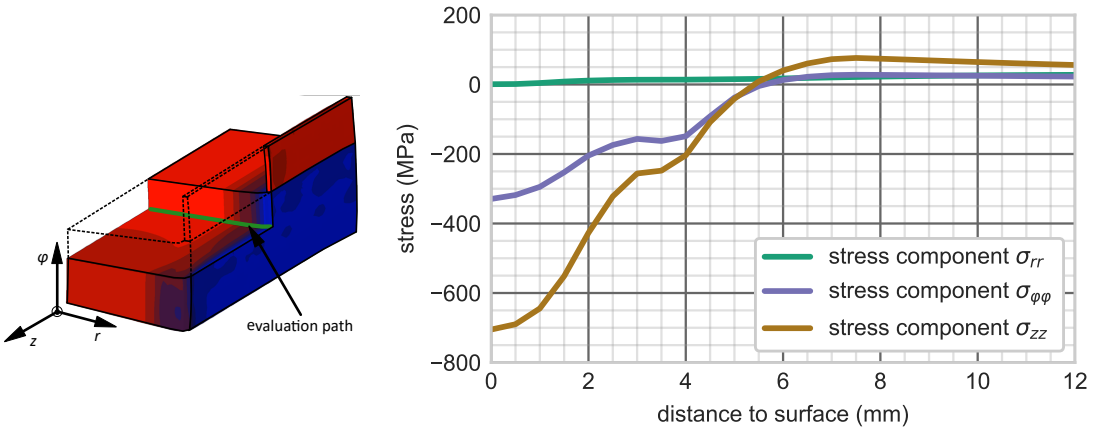
were constant all the way to the center. As Figure 3.15 presents a plot along the r-axis, it represents an axially symmetric view. The equilibrium is therefore calculated using the volume under pressure or tension. The following subsection applies the knowledge gained here in the form of parameter studies and shows which geometry and process parameters have an influence on the residual stress state of cold-rolled railway components.



(a) Isometric view of cold rolling model.

(b) Distribution of the normal stress components σ_{rr}, $\sigma_{\varphi\varphi}$ and σ_{zz} in radial direction.

Figure 3.15: Distribution of the normal stress components over entire axle geometry of the cold rolled FE model with the process parameters feed $f = 0.5$ mm/rev and load $L = 30$ kN.



(a) Isometric view of cold rolling model.

(b) Distribution of the normal stress components σ_{rr}, $\sigma_{\varphi\varphi}$ and σ_{zz} in radial direction.

Figure 3.16: Distribution of the normal stress components in the near surface region of the cold rolled FE model with the process parameters feed $f = 0.5$ mm/rev and load $L = 30$ kN.

3.5.3 Parameter Study for the Cold Rolling of Wheelset Axles

With the significantly improved computational efficiency compared to previous models, parameter studies can now be conducted to investigate the influencing factors of the cold rolling process. The first study aims to provide an overview of the geometry parameters that affect the stress distribution. Subsequently, these parameters will be further examined to determine the precise extent of their influence.

The initial investigation focuses on varying the geometry parameters, namely the wheelset axle diameter (D_{axle}), work roller diameter (D_{WR}), and work roller edge radius (R_{WR}). For a visual reference, please refer to Figure 3.5 for the corresponding designations. Table 3.3 provides an overview of the varied values, constant parameters, and resulting values for the remaining parameters.

varied values			constant values	calculated values	
D_{axle}	D_{WR}	R_{WR}	p_H	L	f
(mm)	(mm)	(mm)	(MPa)	(N)	(mm/rev)
160	100	7.5	5000	min 9 700	min 0,37
180	150	15		max 74 700	max 1,33
200	200	30			
220					

Table 3.3: Values of the first parameter study for the influencing factors of the cold rolling process.

In previous studies, a Hertzian pressure of 5000 MPa has proven to be suitable for wheelset axles. Therefore, this value is maintained as a constant in the initial parameter study, while the other values are adjusted accordingly. To accomplish this, the process model developed by Maierhofer et al. is utilized [58]. This model is primarily used to estimate the depth of residual stress penetration resulting from cold rolling. However, it also provides the capability to estimate the required cold rolling force and feed rate for a given combination of geometry parameters and Hertzian pressure.

In the parameter study, the required force L was calculated for each possible combination of geometry parameters at a constant Hertzian pressure p_H. The feed rate f is determined by twice the half-width of the contact ellipse formed during the process.

The values chosen for the geometry parameters are based on the dimensions of typical railway components and the tools employed in their manufacturing. It is worth noting that the calculated cold rolling forces may exceed the capabilities of modern machines. However, they are utilized in order to better illustrate the influences on the residual stress state.

The evaluation of the first parameter study has revealed that the wheelset axle diameter D_{axle} and work roller diameter D_{WR} have no significant influence on the distribution of residual stresses. Conversely, the work roller edge radius R_{WR} has been found to have a significant impact. The calculation of the load L and feed f for Table 3.3 has already demonstrated

that the contact ellipse changes significantly when the roller edge radius is varied while maintaining the same Hertzian pressure. These findings have motivated the need for a further parameter study.

The second study followed the same procedure as the first, but this time the values for the wheelset axle diameter (D_{axle}) and work roller diameter (D_{WR}) were kept constant. These values are based on existing component and tool geometries. The variable parameters in this study are the Hertzian pressure (p_H) and the work roller edge radius (R_{WR}). Similar to the first study, the load and feed are calculated using the process model developed by Maierhofer et al. An overview of all the values is provided in Table 3.4.

varied values		constant values		calculated values	
p_H	R_{WR}	D_{axle}	D_{WR}	L	f
(MPa)	(mm)	(mm)	(mm)	(N)	(mm/rev)
3 500	7.5	180	150	min 4 200	min 0,26
4 000	15			max 76 000	max 1,20
4 500	22.5				
5 000					
5 500					
6 000					

Table 3.4: Values of the second parameter study for the influencing factors of the cold rolling process.

The results of the parameter study demonstrate the influence of the Hertzian pressure (p_H) and the work roller edge radius (R_{WR}) on the distribution of residual stresses. To enhance computational efficiency, the model geometry was adjusted based on the process parameters. Lower work roller edge radii and corresponding lower loads result in a smaller contact ellipse, leading to a reduced feed and a shorter rolled-over length in the z-direction. Each simulation is conducted for 50 cycles or overrolls, which is determined empirically as a point where the stress curve in the region of interest (ROI) changes minimally. The model length in the z-direction is adjusted to cover the central 80% of the rolled-over region. Furthermore, small loads with a small contact ellipse only require an opening angle of 3° of the model, for large loads this is doubled to 12°.

In summary, the results of the parameter studies indicate the following: The depth of penetration of compressive stresses increases with higher Hertzian pressure and larger work roller edge radius. The overall shape of the stress distribution remains relatively consistent, regardless of the specific geometry parameters. Even with low loads, the cold rolling process generates significant compressive stresses at the surface, approaching the tensile strength of EA4T (around 700 MPa). The maximum compressive stress at the surface remains relatively constant, even with higher loads, due to limitations imposed by the chosen material model. It is important to exercise caution when implementing extremely high Hertzian pressures, as the simulations indicate the occurrence of high plastic strains that could potentially lead to surface damage. A Hertzian pressure of 5000 MPa is recommended as a safe limit. The

simulations also reveal that the increasing penetration depth of compressive stresses results in a higher peak of tensile stresses within the wheelset axle. This effect helps to maintain stress equilibrium. However, the influence of this increase in tensile stresses is not particularly significant. The study determined a maximum penetration depth of 8.7 mm, achieved with a Hertzian pressure of 6000 MPa and a work roller edge radius of 22.5 mm. The tensile stresses generated at depth remain below 100 MPa, which is considered sufficiently low for ensuring fatigue resistance in the design of EA4T.

The findings of this investigation have been consolidated into a user-friendly software tool called PRESD, which stands for Prediction of Residual Stress Distribution. The graphical user interface (GUI) of the program is depicted in Figure 3.17. The left side of the interface allows users to input the desired Hertzian pressure and tool geometry parameters. In the middle section, all the calculated results are displayed, including visualizations of stress and strain components. Users have the flexibility to examine stress values for specific strains or vice versa. On the right side, the input history and corresponding results are presented. The software utilizes data from the discussed FE simulations, with intermediate values being interpolated between the obtained results. With PRESD, estimating the distribution of residual stresses induced by cold rolling becomes a straightforward and efficient process.The following subsection deals with the question how the simulation results compare to experimental results.

Figure 3.17: GUI of the PRESD software tool. Left: definition of input parameters. Center: stress and strain distributions. Right: input and result history.

3.5.4 Comparison of Simulation and Experimental Results

This section discusses the comparison of simulation results with measurements. On the one hand, it is examined whether the FE simulations can reproduce the stress behavior at the actual component size and, on the other hand, whether the model is also suitable for small specimens that are orders of magnitude smaller. The measurements on the actual wheelset axles are further categorized into pure laboratory tests and wheelset axles that were installed in a train after being cold rolled and then running in regular ÖBB service.

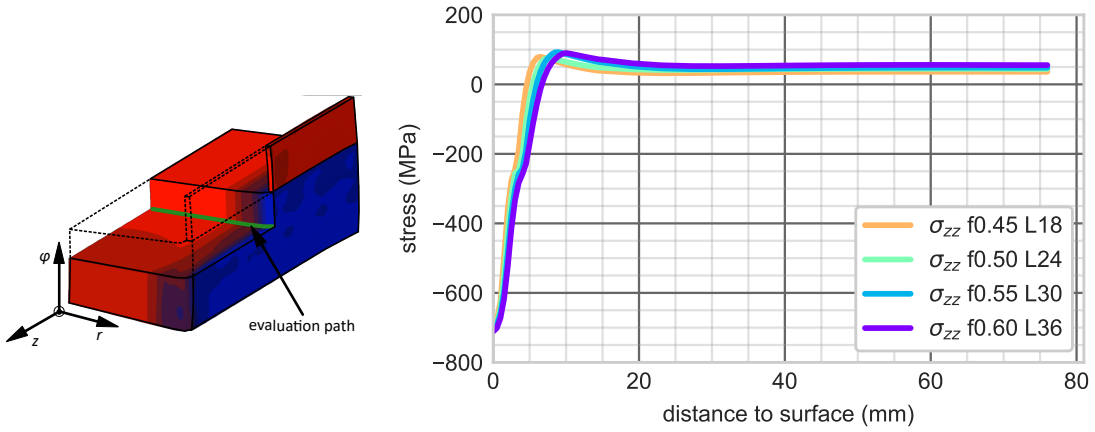
For the laboratory tests, a wheelset axle was cold rolled with four different parameter combinations by the well-known machine tool manufacturer HEGENSCHEIDT-MFD. This time the focus of these investigations was the cold rolling force L and the feed f. The selected parameters are shown in Table 3.5.

varied values		constant values			calculated values
L	f	D_{axle}	D_{WR}	R_{WR}	p_H
(N)	(mm/rev)	(mm)	(mm)	(mm)	(MPa)
18 000	0.45	182	155	15	4 300
24 000	0.50	182	155	15	4 700
30 000	0.55	182	155	15	5 100
36 000	0.60	182	155	15	5 400

Table 3.5: Cold rolling parameters used for laboratory test on full scale wheelset axles.

The force values chosen for the laboratory tests encompassed both values below and above the conventionally used range. The feed, on the other hand, was determined based on empirical values and was slightly lower than the prediction derived from the Maierhofer et al. process model.

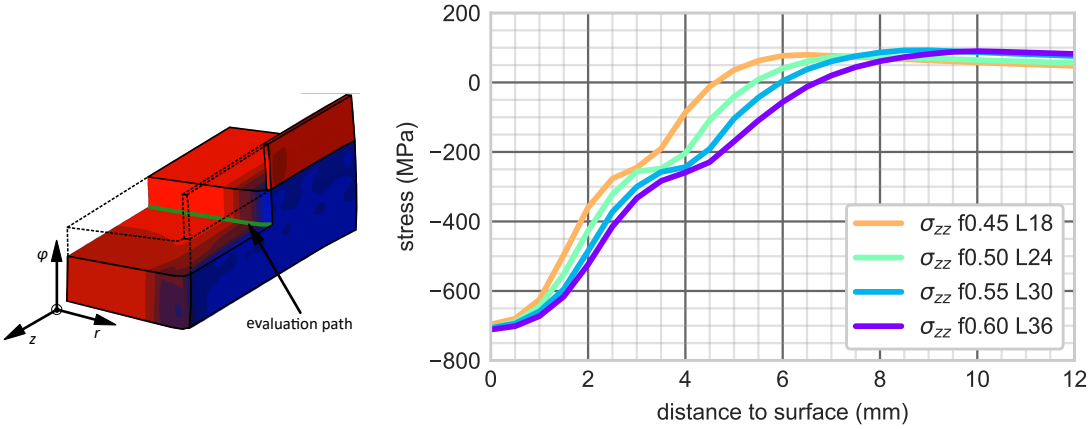
Figure 3.18 presents the simulation results for these four parameter combinations across the entire radius of the wheelset axle. For a more detailed view of the near-surface region, Figure 3.19 zooms in on the same configurations.



(a) Isometric view of cold rolling model.

(b) Distribution of the normal stress component σ_{zz} in radial direction.

Figure 3.18: Distribution of the normal stress component σ_{zz} over entire axle geometry of the cold rolled FE model with different combinations of the process parameters feed f and load L.



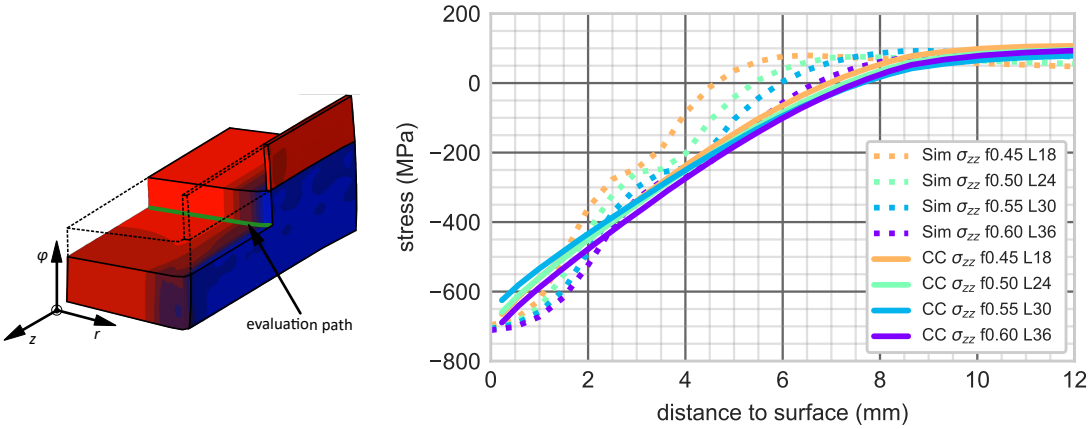
(a) Isometric view of cold rolling model.

(b) Distribution of the normal stress component σ_{zz} in radial direction.

Figure 3.19: Distribution of the normal stress component σ_{zz} in the near surface region of the cold rolled FE model with different combinations of the process parameters feed f and load L.

Furthermore, the results demonstrate that as the load increases, the depth of penetration of the residual compressive stresses also increases. All parameter combinations reach the maximum near-surface stress, which once again corresponds to the absolute maximum within the tensile strength range of the material.

Figure 3.20 presents a comparison between the simulation and measured results. It is important to note that in this figure, the measured results are displayed in the foreground, while the previously shown solid line representing the simulation results is now depicted as a dotted line. The color coding has been maintained.



(a) Isometric view of cold rolling model.

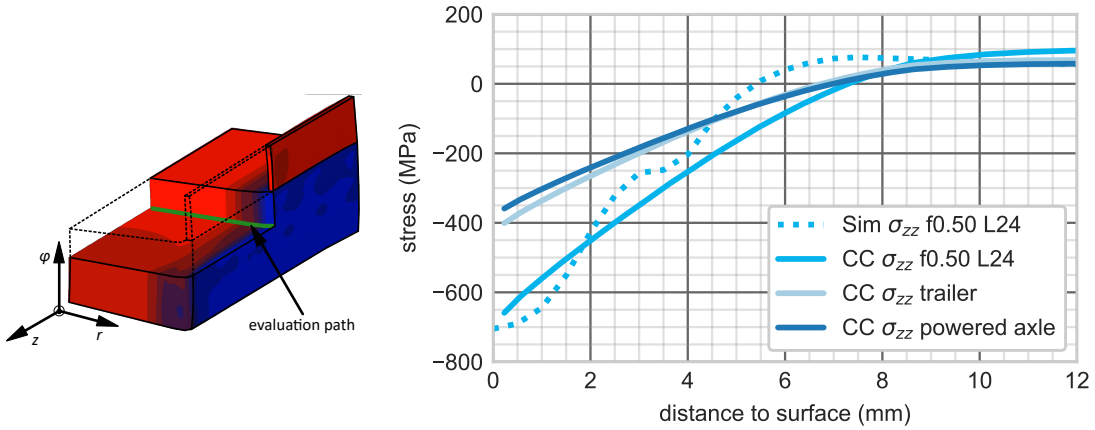
(b) Distribution of the normal stress component σ_{zz} of CC measurement and FE simulation in radial direction.

Figure 3.20: Comparison of CC measurement and FE simulation of four different combinations of the process parameters feed f and load L.

The measurement was conducted using the cut compliance method, which is described in detail in Section 2.3.2. The same wheelset axle was subjected to all four parameter combinations to optimize the utilization of the available surface. The length of each parameter combination exceeded twice the axle diameter, and a sufficient distance from the shaft shoulders was maintained to mitigate potential edge effects. Strain gauges were applied to the center of the respective regions for subsequent measurement. The wheelset axle was then cut using a conventional band saw, piece by piece, and the resulting strain from stress redistribution was measured.

The measurement results exhibit good agreement with the simulation. A trend towards increased depth of penetration with higher loads can be observed, although it is not as prominent as in the simulation. Both the surface stress values and the peak tensile stress in the interior show good agreement. These measurement results serve as verification of the accuracy of the FE model while also highlighting the superiority of simulation in predicting deep residual stress fields.

In the second investigation using full-scale wheelset axles, an in-service test was conducted at OEBB. Two wheelset axles were cold rolled with parameters of $L = 30$ kN and $f = 0.50$ mm. One axle was installed as a powered wheelset axle, and the other as a trailer wheelset axle, and both were used in regular service for several months. After the axles were removed from the train, cut compliance measurements were performed, and the results are shown in Figure 3.21. To facilitate comparison, the results of the simulation and the corresponding cut compliance measurements immediately after cold rolling are presented in the same graph.



(a) Isometric view of cold rolling model.

(b) Distribution of the normal stress component σ_{zz} of CC measurement and FE simulation in radial direction.

Figure 3.21: Comparison of CC measurement of a cold rolled powered axle and a cold rolled trailer axle.

The results of the in-service test on the full-scale wheelset axles indicate a significant redistribution of stresses during service. The compressive stresses at the surface are reduced by approximately 50% compared to the stresses introduced by cold rolling, while the depth of penetration remains unchanged. In addition, a slight decrease in tensile stresses in the interior

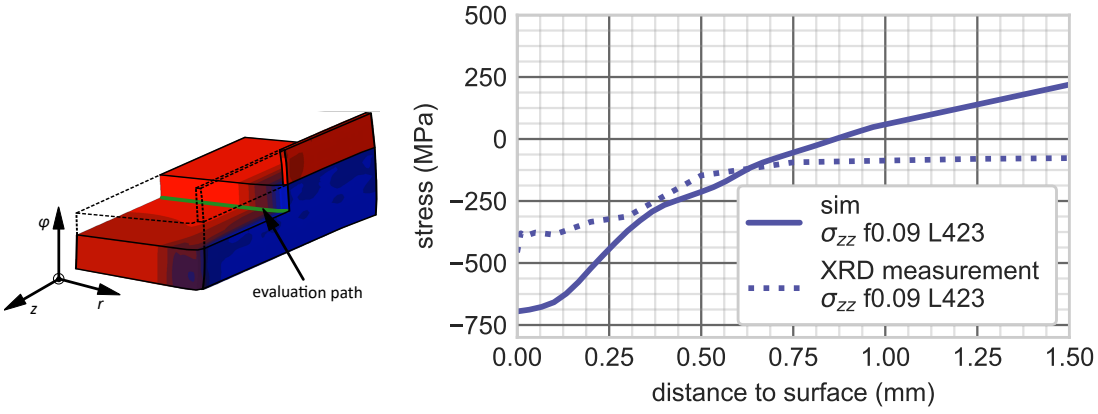
can be observed, as dictated by the prevailing stress equilibrium. Notably, no significant difference is observed between the powered and trailer axles.

In addition to the full-scale axles, small-scale specimens were manufactured and cold rolled for various rotating bending tests using the EA4T material. The cold rolling parameters employed for these tests are provided in Table 3.6.

varied values		constant values			calculated values
L	f	D_{axle}	D_{WR}	R_{WR}	p_H
(N)	(mm/rev)	(mm)	(mm)	(mm)	(MPa)
423	0.09	6.67	40	4	4 700

Table 3.6: Cold rolling parameters used for laboratory test on small scale specimens.

The cold rolling of the small specimens was also calculated using the FE model. The obtained stress curve is shown in Figure 3.22. The measurement of the residual stress distribution was performed using XRD. On the one hand, the size of the specimen is small, and on the other hand, the penetration depth of the stresses to be measured is not too large, and it is expected that better results can be obtained with the XRD measurement.



(a) Isometric view of cold rolling model.

(b) Distribution of the normal stress component σ_{zz} of XRD measurement and FE simulation in radial direction.

Figure 3.22: Comparison of XRD measurement and FE simulation on a cold rolled small scale specimen.

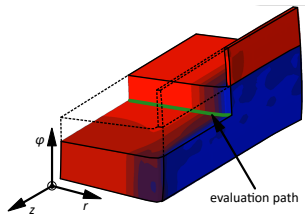
The simulation results again demonstrate the presence of compressive stresses at the surface, approaching the tensile strength of the material. The progression of stress into the depth is also similar to previous findings. However, it is important to note that the depth of penetration of the compressive stresses has a different proportion relative to the overall component geometry. In the full-scale tests, the depth of penetration ranged from 5 mm to 7 mm, which is below 10% of the radius of the wheelset axle. In contrast, for the small specimens, the depth of penetration is around 30% of the radius. Moreover, the effect of increasing tensile stresses

due to deeper penetrating compressive stresses is clearly observed. The comparison with the XRD measurement shows an acceptable agreement. However, more credibility is given to the simulation results as they align well with the previous process model by Maierhofer et al. Additionally, the rotational bending tests indicate that the compressive stresses at the surface are indeed high. If the values measured by XRD are accurate, there is potential to utilize them to enhance the fatigue strength of such components, considering the presence of significant compressive stresses.

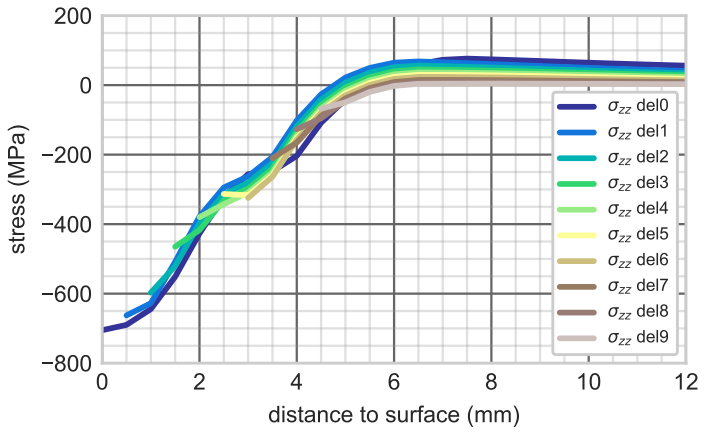
3.5.5 Stress Redistribution

This section presents the results of stress redistribution for cold rolled wheelset axles. Unlike the stress redistribution discussed in the previous section, the results here refer to predictions based on FE calculations.

At the end of the simulation of the cold rolling process, the surface up to and including the tenth row of elements is removed, which measures approximately 5 mm, depending on the model and consequently on the element size. Figure 3.23 shows an example of how stresses are redistributed when near-surface layers are removed, as it may occur during maintenance operations. Note that the shown results are qualitative only, as they do not account for many of the effects that may occur. However, they are a useful indication of the changing behavior of the stresses introduced into the material.



(a) Isometric view of cold rolling model.



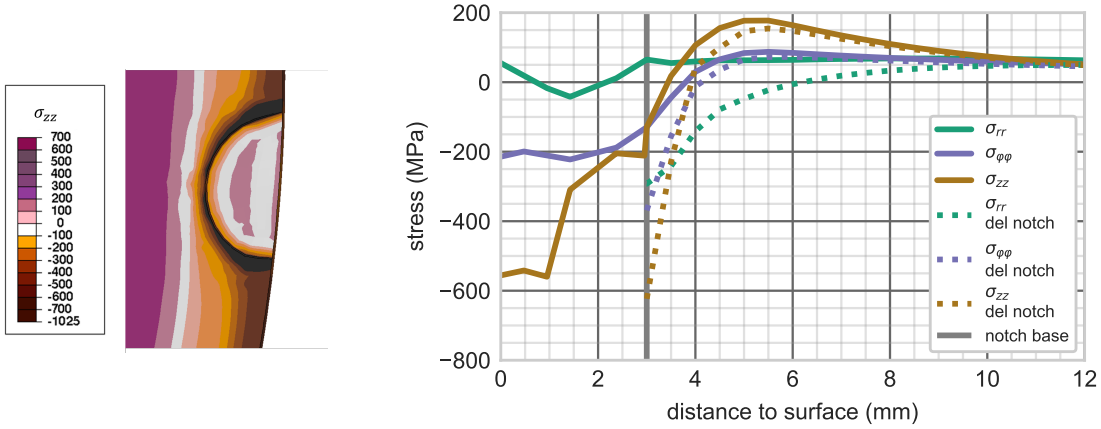
(b) Changes of the distribution of the normal stresses of a cold rolled wheelset axles due to the removal of surface layers similar to maintenance procedures.

Figure 3.23: Stress redistribution due to the removal of surface layers.

A noticeable trend is that the existing compressive stresses do not disappear when the layer containing them is removed. Instead, they shift at least partially to deeper levels, thereby also shifting the penetration depth of residual compressive stresses. As more and more of the surface is removed, the compressive stresses will eventually decrease, but not below the value originally present at that point. It should be noted that the results shown in Figure 3.23 are based on the original geometry and that the curve starts at the newly created surface in each

case. As the compressive stresses decrease, the tensile stresses inside also decrease. These stress redistribution calculations are also part of the PRESD software tool.

Finally, the simulation results with the introduced notch from the corrosion fatigue tests on full-scale wheelset axles are discussed. The question here is what is the residual compressive stress in the vicinity of the notch after it has been introduced. As in the previous example, the material is removed after the cold rolling process. Since the notch involves significantly less material, the results can also be interpreted quantitatively. Figure 3.24 (a) shows the section through the notch after stress redistribution. Figure 3.24 (b) shows the already known distribution of the three normal stress components in the r-direction before and after removal of the notch.



(a) Contour plot of the σ_{zz} stresses in a sectional view of a cold rolled wheelset axle after removal of the notch.

(b) Distribution of the normal stresses in the vicinity of the notch on radial paths around the notch.

Figure 3.24: Stress redistribution due to the removal of a notch in a cold rolled wheelset axle.

After removing the notch, a stress state with high compressive stresses is observed at the base of the notch. Figure 3.24 (a) presents a contour plot illustrating how the initially homogeneously distributed stresses in the r direction align along the notch base. Additionally, it can be observed that the stress magnitudes directly at the bottom of the notch exhibit higher compressive stress values. Two overlapping effects contribute to this behavior. Firstly, as seen in the previous example, the stresses are shifted to lower regions. Secondly, the removal of the sharp geometric feature allows the near-surface layer to deform more freely in the z direction . As the near-surface σ_{zz} stresses are compressive, the notch experiences tension at the surface and compression at the bottom.

Furthermore, it is worth noting that although the stresses shift in depth, the stress gradient increases significantly. The penetration depth of the compressive stresses remains unchanged at a depth $>$ 5 mm (or 2 mm below the notch base), indicating that the notch no longer influences the residual stresses.

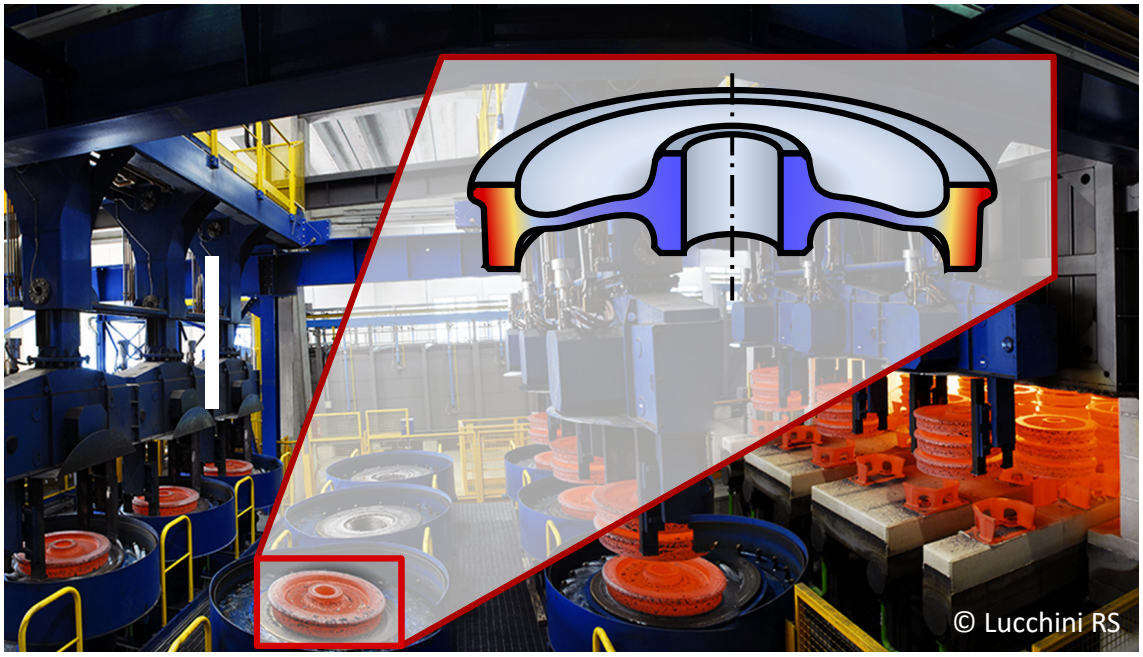
3.6 Conclusions

This thesis presents the development of a process model specifically designed for the cold rolling process of wheelset axles. The model incorporates innovative boundary conditions that accurately capture the effects of this manufacturing process on the component properties. These boundary conditions not only ensure precise representation of the cold rolling process but also enable a significant reduction in model size without compromising result accuracy. This reduction in size also improves computational efficiency, facilitating the exploration of various parameters influencing the cold rolling process.

A parameter investigation demonstrates that the component and tool geometry have minimal influence on the residual stress profile, while the Hertzian pressure arising from the contact between the toll and the component emerges as the most influential process parameter. Controlling this parameter allows for an effective design of the penetration depth of the favorable residual compressive stresses. Moreover, the developed process model enables an accurate assessment of the residual stress distribution throughout the entire component's cross section.

Building upon the results of this process model, a dedicated software tool has been created to predict residual stresses induced by cold rolling. This tool proves valuable in component design, eliminating the need for time-consuming trial-and-error approaches. Additionally, the process model demonstrates its efficacy in simulating stress redistribution during maintenance operations. These simulations offer qualitative insights into changes in favorable residual stresses and serve as indicators for determining when renewed surface treatment via cold rolling becomes necessary to ensure component safety.

4 Heat Treatment of Railway Wheels



4.1 The Component Railway Wheel

Railway wheels, similar to wheelset axles, are critical components that must meet stringent safety requirements in the railway industry. These components are subjected to various challenges, including the high speeds involved in railway operations and the constantly increasing accelerations experienced during starting and braking.

4.1.1 Requirements for Wheels

The product requirements for wheels in the railroad sector are defined in the standard EN13262 [84]. This standard specifically applies to forged, rolled solid wheels made of vacuum-degassed steel. It sets criteria for various aspects of the wheel, including the wheel rim and tread, which must undergo quenching and tempering processes to achieve increased hardness and residual compressive stresses. These requirements ensure the strength and durability of the wheels.

In addition to product requirements, there are other regulations that govern the approval of railway wheels for traffic. The UIC510-5 [85] and EN13979 [86] standards provide guidelines

and procedures for the approval process. These standards ensure that railway wheels meet the necessary safety and performance criteria before they can be used in railway systems.

The design process for railway wheels follows similar steps as described for wheelset axles in Subsection 3.1.1. It involves considering various requirements, including geometric interchangeability, thermomechanical aspects to control deformations and prevent wheel fractures during braking, mechanical considerations to evaluate service life and prevent fatigue cracks, and acoustic factors to ensure ride comfort for passengers and minimize noise pollution caused by passing trains.

Figure 3.1 in Section 3.1 *The Component Wheelset Axle* depicts the individual components of a railway wheel, such as the tread, rim, web, and hub. These components play crucial roles in the overall performance of the wheel. Modern wheel design adheres to the aforementioned requirements, considering factors related to geometry, thermomechanical behavior, mechanical strength, and acoustic performance.

The design aspects of a wheel also include the way in which the train moves through the rail network. This results in the geometry of the wheel rim with associated flange. Wheels are fixed to the axle. Therefore, it is not possible for wheels to rotate independently of each other. On a straight track, this is nothing special, but when cornering, one wheel has to travel a longer distance than the other. Most vehicles solve this problem with the help of a differential, which decouples the rotation of the individual wheels from each other. For railway wheels this is not possible. [87, 88]

Railway wheels also do not have a flat running surface but an inclined one. When cornering, centrifugal forces push the entire wheel set outward, causing the rails to contact the wheels at different positions. The conical geometry enables the wheelset to travel different distances at the same rotational speed. Another function of the conical shape of the treads is self-centering in the event of lateral misalignment on a straight track. Figure 4.1 shows this effect in simplified form. The flange serves here only as an additional protective measure. Slipping off the rail is prevented by the conical geometry alone. The only exception is when rails are changed with the aid of switches. Here, contact with the flange can also occur. [87, 88]

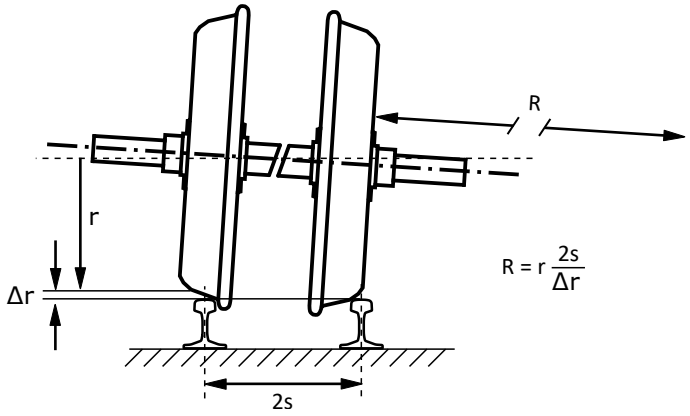


Figure 4.1: Steering effect of a slip-free wheelset on a tight curve (radius R) related to the inner rail. Figure and caption taken from [87].

4.1.2 Manufacturing of Wheels

The manufacturing process of railway wheels begins with a cylindrical ingot, which is cut into processable-sized blocks using a cold-cutting machine. The blocks are then heated in a pusher furnace to forging temperature. Once the desired temperature is reached, a press is used to shape the blocks into wheels through a multi-step process: (1) Presetting: The preheated and descaled ingot is compressed by one third using the press. (2) Upsetting: The press forms a flat round component called a pancake, which already has a preprofiled top and bottom. (3) Final upsetting: The preform undergoes further shaping to achieve the final contour. (4) Punching: A hole is punched in the center of the component using a pin. The partially formed wheel is then transferred to a ring rolling mill, where rotating rolls reshape the disk, gradually enlarging the part. Afterward, the wheel undergoes additional shaping in another press to achieve its final contour. Once the wheel has reached its final shape, it enters the heat treatment shop, which includes an austenitizing furnace, quenching tanks, and a tempering furnace. This heat treatment process gives the wheel the desired microstructure. A machining center for metal cutting brings the wheel to final size before it is ready for shipment. After thorough inspection throughout the entire manufacturing process, any defective components are immediately rejected. Figure 4.2 provides an overview of the necessary steps involved in manufacturing a railway wheel, from the ingot to shipment. [88]

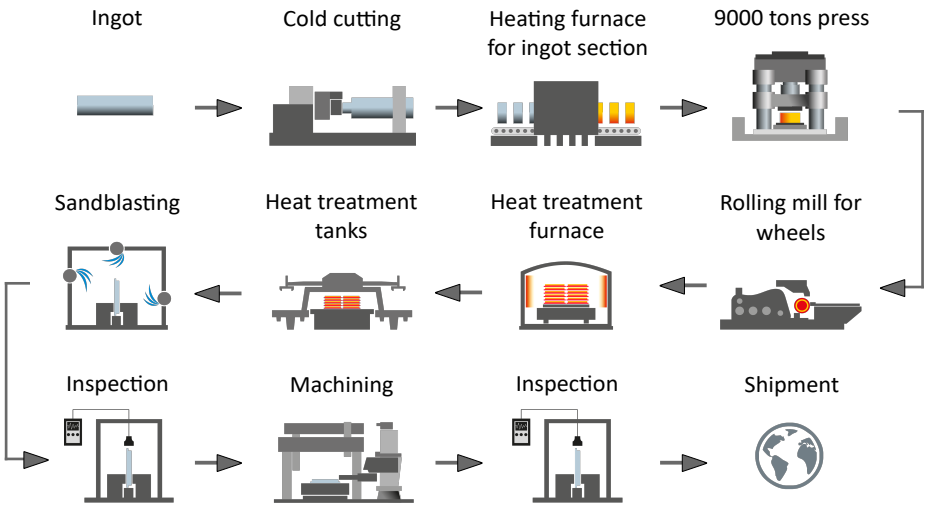


Figure 4.2: Manufacturing of wheels. Figure adapted from [48].

4.2 Heat Treatment Process

Another essential manufacturing process in this work is the heat treatment. This subsection gives a brief overview of the state of the art of such a process in combination with the material steel. In general, heat treatment is a process in which the physical and chemical properties of a material can be influenced by specific temperature control. Different types of heat treatment

are normalizing, quenching, annealing, tempering, carburizing, surface hardening or precipitation strengthening. For the particular railway wheel in this thesis, only normalizing, quenching and tempering are of interest.

For a railway wheel, the heat treatment starts after the wheel has been forged into its final shape. For this purpose, it is first austenitized, i.e. the temperature of the entire wheel is increased until the phases formed in previous process steps can be transformed into austenite. This process is also called normalizing and has the purpose of relieving the stresses of previous manufacturing steps such as forging. Subsequent process steps introduce stresses into the component again. Subsequent manufacturing processes reintroduce stresses to the component. Therefore, it is advantageous for process control and product quality to initiate the actual heat treatment with a stress-free component. Then the wheel is quenched and tempered, i.e. in the first step it is quenched in a tank with water and then tempered in the furnace. In this way, it is possible to obtain a wear-resistant and hard tread without sacrificing the damping and fatigue properties inside the wheel. [89]

4.3 Modeling of Heat Treatment

Heat treatment processes are intricate, involving multiple simultaneous physical phenomena that partly influence each other. Despite decades of research and efforts to develop comprehensive models, no single model can capture all these phenomena at once. Analytical methods have their limitations in handling such complexity, leading to the adoption of numerical methods like the FEM to solve heat treatment problems. This section gives an overview of the possibilities for modeling heat treatment processes and then goes into detail about the state of the art for modeling quenching processes for the component railway wheel. Regarding the material, the focus here is on low-alloyed steel that is used to manufacture railway wheels. [24, 90, 91]

The simulation of heat treatment processes, especially quenching, is a multiphysics and multiscale problem. The physical fields involved and their interactions are shown schematically in Figure 4.3. [24, 90, 91]

It is clear from this figure that the physical fields are interrelated and influence each other. Not only that, but they influence each other in both directions. From this multitude of possible interactions, one can already see the difficulties that can arise in such models.

In [91] these problems are defined as:

- *Multi-scale/multi-physics processes: The model must deal with couplings between different physical events such as heat/mass transfer, mechanical interactions and phase transformations. Some of those couplings may require multi-scale treatment due to mechanisms operating at different length and time scales.*
- *Strong material non-linearity: The model may require dealing with highly nonlinear material properties since the material properties usually have a pronounced variation with temperature, microstructural constitution, stress and concentration.*
- *Complex boundary conditions: The model may require dealing with complex boundary conditions such as nonlinear and moving boundary conditions.*

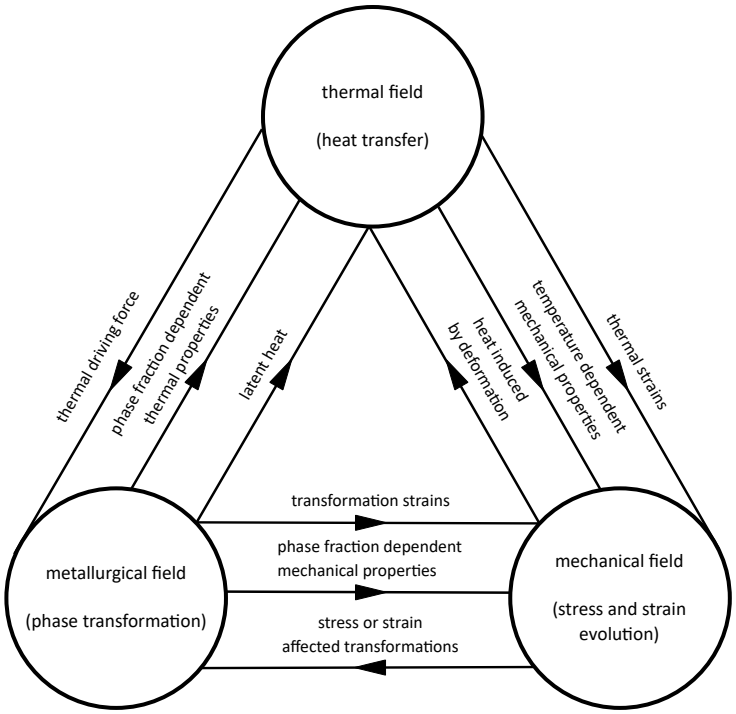


Figure 4.3: Physical fields and coupling interactions involved in heat treatment processes. Figure and caption adapted from [91].

- *Complex geometries: The model may necessitate handling of complex 3-D geometries since most of the critical engineering components have complex geometries.*
- *Thermophysical events and energy sources: The model may necessitate handling of thermophysical events driving the heat transfer and different energy sources ranging from conventional heating to induction or laser, etc.*

The first studies in this area date back to the 1970s. At that time, only the simplest phenomena, such as thermal expansion, could be applied to simple geometries. In the 1980's and 1990's, these models advanced rapidly to the point where they could be used to predict the microstructure, at least to some extent. The constitutive equations of individual phenomena such as TRIP could also be implemented in the first simulations. With the rapidly increasing power of modern computers these development processes accelerated even further, and by the end of the 1990s the focus was on process optimization using the simulations developed until then. In the 2000s, the focus shifted to working across computational methods like for example computational fluid dynamics (CFD) and FEM. The goal of these simulations was to better understand the interaction between the heat treated components and their surrounding. [24, 90, 91]

Since then, these simulations have evolved in several directions. A strong focus is on multiscale simulation, starting from the atomic scale up to the scale of entire plants in heat treatment shops. In this context, FEM is suitable for multiscale simulations starting from the mesoscopic scale. In contrast, atomistic modeling uses other methods such as Monte Carlo

simulations or cellular automata. Existing FEM models are now able to interact from one scale to the next. What is still missing are models that work across more than two scales. Most models in the field of heat treatment processes are mathematical rather than physical, which adds to the difficulty of parameter determination. Mathematical models focus on describing and simulating system behavior using mathematical equations, but the variables and relationships may not always have a direct physical interpretation. In contrast, physical models aim to capture the real-world meaning of variables and relationships, providing a deeper understanding of the phenomena involved. Mathematical models are particularly parameter sensitive, which also makes it difficult to find stable sets of parameters for a more general use. The problem with these models goes even further. In some cases they cannot be applied to similar problems without additional effort, since they are developed for specific applications. Thus, even after decades of research in this field, one still faces problems related to nonlinearities, numerical instabilities, multiscale effects, and the interaction of different computational methods. [24, 90, 91]

A wide range of phase transformation models are available for implementation in FEM software. While it is not possible to provide a comprehensive overview of all these models here, the focus will be on previous works that share a similar framework in terms of coding subroutines for phase transformation modeling. Schemmel et al. developed a model for predicting residual stresses in hot-working steel components that includes both phase transformation and transformation plasticity [92, 93]. Based on this work, Brunbauer et al. developed a model for the evaluation of residual stresses during spray cooling of seamless tubes, which considers both phase transformation and the complex heat transfer during spray cooling [94]. Further development of this work is focused on heat treatment design where the developed material models are used [95]. Another application of this material model can be found in [96]. The focus here is on resistance spot welding with rapid heating and cooling phases and phase transformations that occur differently when heating or cooling. The resulting residual stresses are used to address other effects, such as the damage behavior of the material due to liquid metal embrittlement [97]. The most recent work in this area is focused on cross-scale effects, starting with stresses in the microscopic range, as they occur in the vicinity of inclusions, up to the macroscopic view on component size [98]. The framework to build the material model that is the heart of the heat treatment simulation in this thesis builds on the latter works [92–98].

4.3.1 Heat Treatment Simulations of Railway Wheels

The railway wheel component is of great interest to both industry and research. In the field of simulation alone, the works go in many different directions, such as manufacturing, including forming processes and heat treatment, but also in service, such as fatigue, crack propagation or rolling contact fatigue.

There are numerous works in the field of heat treatment simulation of railway wheels alone. While these papers deal with different aspects of heat treatment, the focus is always on describing or predicting residual stresses due to production. The key elements leading to

a quantitative description of the residual stress state are the material model and the heat transfer.

Lingamanaik et al. [99] developed a thermo-mechanical model of the quenching process using the DANTE software. Dilatometer measurements were performed to account for the phase transformation behavior, which in turn were used to find parameters using a fitting tool built into DANTE. Although the model is still quite simple, since not all effects have been taken into account, both in the material model and in terms of heat transfer, it already provides good initial indications of the distribution of production-related residual stresses in railway wheels. In [100], the phase transformation behavior was improved by considering both martensite and pearlite. In addition, the heat transfer could be described more precisely by temperature measurements in a specially designed laboratory dip tank. However, the present results raise the question of whether the input variables have been well calibrated, since unusually high phase fractions occur at rather unexpected locations and predict a strongly martensitic wheel web.

Recognizing the large influence of heat transfer on residual stresses, Brunel et al. [101] focused their model on the calibration of the heat transfer. By iteratively matching measurements and numerical calculations, they were able to determine a temperature-dependent heat transfer coefficient for the water-quenched wheel tread. Simple material models were then used to investigate the influence of heat transfer on residual stress behavior.

Another simulation shows the importance of temperature dependent parameters. Here, temperature-dependent data for both the heat transfer and the material itself are entered, using only a simple elastic-plastic material model. On the other hand, the process control is well represented according to common standards for the heat treatment of railway wheels. [102]

A recent work in this area is that of Tian et al. The focus here is on a quantity not yet mentioned, namely hardness. In order to describe and later predict the distribution of hardness in the wheel, both martensitic and bainitic phase transformation are considered and a temperature dependent heat transfer is modeled. Finally, the simulation is verified with a series of hardness measurements and microstructure images, and an optimization strategy for the heat treatment is proposed. [103]

In summary, the following can be said about existing simulations in this field: (1) temperature dependent process and material parameters are required, (2) the process parameters need to be verified by experiments, since there is no satisfactory method for predicting the heat transfer between very hot components and cold water, (3) the material parameters must be measured according to the real heat treatment process and should have the same range of temperature, temperature change and strain rate, (4) for a qualitative prediction of residual stresses, complex processes such as phase transformations and transformation induced plasticity must be taken into account, (5) this type of simulation requires a detailed comparison with experiments on several stages to ensure the quality of the results.

4.3.2 Materials and Material Models for Heat Treatment Simulations

In principle, railway wheels are made of unalloyed or low-alloyed steel. The focus is always on a high degree of purity, since even small deviations from the chemical composition influence the mechanical properties. Since a railway wheel is a component that is produced in very large quantities, these tolerances must also be kept tight. The steels defined in EN 13262 [84] are called ER6 up to ER9, where the higher the number, the higher the strength of the material.

A key element of every process simulation is certainly the material used. Many properties and results depend exclusively on the selected material, although the process itself can also be influenced by it.

ER7, the material used for heat treatment simulations in this thesis is a standardized quenched and tempered steel for railway wheels (ER7). The chemical composition (c.f. Table 4.1), mechanical properties and the necessary test methods are regulated in EN13262 [84].

C	Si	Mn	P	S	Cr	Cu	Mo	Ni	V
0.52	0.40	0.80	0.02	0.015	0.30	0.30	0.08	0.30	0.06

Table 4.1: Maximum percentage content of the various specified elements of ER7 according to [84].

This material has been established for years (or even decades) as a material for railway wheels. It shows good behavior in terms of service life, including thermo-mechanical fatigue, wear, but also in terms of damping properties and ride comfort. In addition, the long experience of operators supports the use of ER7 wheels.

The materials used in the simulations are either based on the behavior of low-alloy steels or, as in most publications, experimentally derived from standardized ER7 and ER8 steels.

The development of material models goes in two directions. They differ in their application and thus fundamentally in the underlying constitutive laws. On the one hand, material models are being developed for the heat treatment process, which exhibits temperature and phase dependent plastic behavior, and on the other hand, research is going in the direction of service life behavior, where the focus is on cyclic plasticity.

The requirements for a future material model can be summarized as follows: (1) all material parameters must be temperature-dependent and determined over the entire spectrum of the process, (2) it must be possible for different phases to form, (3) the mechanical behavior of these phases must be represented, (4) effects such as transformation plasticity must be taken into account, (5) since heat treatment in the course of production is not a cyclic process in terms of fatigue, the cyclic material behavior can be reduced to modeling cooling with subsequent reheating.

4.4 Material Characterization Experiments

As described in the previous section, several measurements are required for a material model. They should be adapted to the process at hand, and the resulting material model is then only valid for certain applications. However, this ensures that the results of the process models are of a quantitative nature. This section goes into detail about the experimental procedure needed to create a material model for the heat treatment of ER7 steel.

4.4.1 Continuous Cooling Transformation Phase Diagram

A good first overview of the transformation behavior of a material can be obtained with transformation phase diagrams. A distinction is made between time-temperature transformation (TTT) diagrams and continuous cooling transformation (CCT) diagrams. Both show the relationship between time and temperature and the resulting solid phase fractions of a material, but they differ in their creation and validity. The TTT is also called an isothermal transformation diagram and, as the name implies, is generated at a constant temperature. The CCT, on the other hand, is generated with continuous temperature changes and thus better represents the quenching process.

Before the dilatometer measurements required for a CCT can be made, the heat treatment process is simulated using a first simple FE model. This initial simulation is used to estimate the cooling rates that occur in the process. The CCT is then generated based on these estimates. The results of this simulation already show one of the main difficulties of such calculations. On the one hand, the component railway wheel is very large and cools down slowly on the inside, but on the other hand, the outer layers cool down very quickly. The full range of cooling rates must be considered in the CCT.

Table 4.2 shows the experimental plan for the dilatometer measurement to obtain a CCT for the material ER7.

λ (hs)	0.013	0.035	0.070	0.150	0.300	0.600	1.250	2.500	5.000
$\dot{\theta}$ ($^K/_s$)	230	85	42.5	20	10	5	2.4	1.2	0.6

Table 4.2: Cooling rates for generating the CCT for the ER7 steel in two different representations: λ is the time in hs for cooling from 800 °C to 500 °C, $\dot{\theta}$ is the corresponding rate of temperature change in K/s.

The test specimens are cylinders of 10 mm length and 4 mm diameter. They are taken from the wheel rim made of ER7 at the same point in the circumferential direction and austenitized at 850 °C for 30 minutes prior to testing in order to minimize possible influences of the actual heat treatment of the wheel and the sampling point in the wheel. 850 °C was chosen as the starting temperature, since this is also the starting temperature of the actual heat treatment. In addition, 850 °C is above the Ac3 temperature at which the transformation to austenite ends. Metallographic examinations and hardness measurements to determine the proportions of the phase fractions follow the dilatometer measurements. The final CCT is presented in subsection 4.7.1, which also includes a discussion of the results.

4.4.2 Hot Tensile Tests

The CCT from Subsection 4.7.1 shows that it can be expected that the phases martensite, bainite, ferrite and pearlite can be formed during the heat treatment of a railway wheel made of ER7. One purpose of the simulation is to reproduce the plastic behavior of these different phases. For their determination, mechanical experiments are performed on pure phases at different temperature levels.

In order to measure the pure phase behavior, the specimens are first prepared in such a way that this behavior can be measured directly. For this purpose, standardized cylindrical tensile specimens are again produced from the wheel rim and then heat-treated in such a way that the corresponding microstructure is established in the specimen. As in the dilatometer tests, the specimens are austenitized at 850 °C and then continuously cooled at different cooling rates.

According to the CCT, an almost 100% martensitic specimen can be produced if cooling is fast enough. The ferrite and pearlite phases are combined into one phase for simulation purposes, as they are assumed to have similar mechanical behavior. Furthermore, when evaluating the dilatometer signal for the CCT measurements, no significant volume jump was observed during the conversion from ferrite to pearlite. In the following, the ferrite and pearlite phases will be referred to as pearlite only. In the case of very slow cooling, 100% pearlitic samples can be produced.

Unfortunately, this is not possible with bainite. It is infeasible to produce a 100% bainitic microstructure with continuous cooling. The CCT shows here a maximum of 35% of phase fraction for cooling with λ of 0.15 hs. However, the best way to measure the mechanical properties of bainite is to prepare specimens according to this cooling route.

The mechanical test is again preceded by a simulation. It is used to estimate the strain rates and strains that occur. Based on the results obtained, the strain rate dependence is not implemented and all tests are performed at a strain rate of 10^{-4} s^{-1}. Since strains are expected in both tension and compression, a combined testing scheme is performed to minimize the testing effort. In this scheme, a specimen is extended to 1% strain, then unloaded, then compressed in the opposite direction to 1% compressive strain, and then extended again to failure. Using this scheme, all hot tensile tests are performed at the respective temperature levels.

This wide range of test temperatures ensures that the plastic behavior of the ER7 phases formed during cooling can be reproduced in the simulation. Measurements are also made above the temperatures at which the phases first appear in order to describe their behavior during the subsequent annealing of the wheels.

The hot tensile test of austenite is a special case. Since this phase does not exhibit stable behavior, it is not possible to make specimens from it as is the case with the other phases. In this test, the specimen is austenitized, then cooled to the test temperature and tested directly. In order to exclude isothermal transformations into other phases, a test temperature of 700 °C and a cooling rate of $\lambda = 1.250$ and faster are chosen. At lower temperatures the available time window is too small to guarantee that no isothermal transformation will occur during the measurement. A summary of the test plan is shown in Table 4.3.

measurement objective	cooling rate (hs)	testing temperatures (°C)
martensite	$\lambda = 0.013$	RT 250 450
bainite	$\lambda = 0.150$	RT 150 300 450
pearlite	$\lambda = 5.000$	RT 150 300 450 600 700
austenite	$\lambda < 1.250$	700

Table 4.3: Experimental design for the hot tensile tests of ER7.

4.4.3 Deformation Dilatometry

TRIP is a phenomenon that occurs during the transformation of one solid phase to another. During the transformation with an additional load, plastic strains develop in the softer phase even though the yield strength is not reached. These plastic strains remain present at room temperature even after removal of the applied load. There are several mathematical models to describe this effect. The one by Leblond in [104] gives a linear relationship between the TRIP strains just mentioned and the applied external stresses. The experiments in this section are used to determine the linear coefficient K_i that relates $\dot{\boldsymbol{\varepsilon}}_i^{tp}$ and $\boldsymbol{S}$, as given in equation 4.6, to model TRIP in FE simulations. [104–106]

For this reason, the experiments are again performed on a dilatometer. However, this time it is necessary to apply additional loads to the specimens, which is why the experiments are carried out on a DSI Gleeble 3800 thermomechanical forming simulator. Figure 4.4 schematically shows the results to be expected as well as the critical adjustment variables of this measurement.

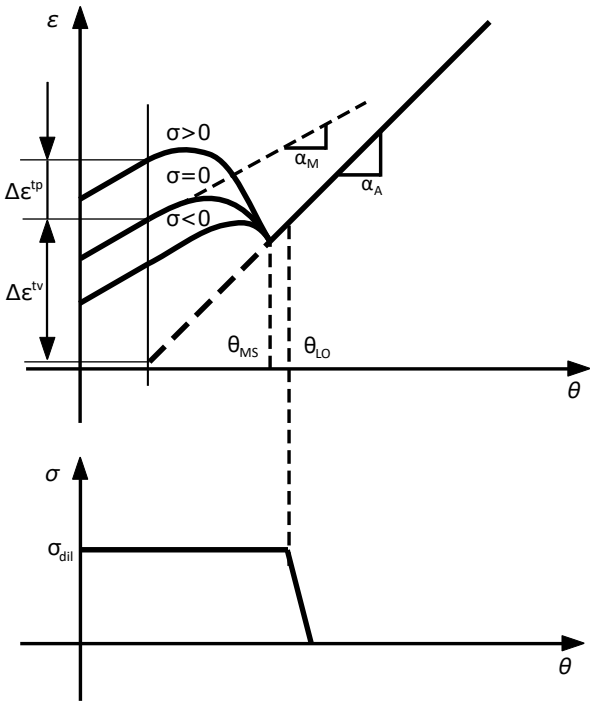


Figure 4.4: Top: Schematic representation of the dilatometer curve used to determine the TRIP strain with the use of different load levels, showing the onset of phase transformation and the onset of load application, as well as the changing thermal expansion of the different phases.
Bottom: Corresponding profile of the additional load. Figure adapted from [105].

The test procedure is based on the tests performed so far in this thesis. Basically, the experiment runs along the same cooling curves as in the previous experiments to ensure that the transformation start of the respective phases remains unchanged. Since there will be an additional load applied during the test, the timing of the load application must be determined. According to Besserdich et al. [106] additional effects are minimized if the load σ_{dil} is applied just above the transformation start temperature θ_{MS}. This is the best way to isolate the TRIP effect and to easily compare the results of the loaded and unloaded samples. Every test is performed according to the following scheme: (1) heating the sample and holding at 850 °C to ensure 100% austenitization, (2) cooling at a defined cooling rate to the load onset temperature θ_{LO}, (3) holding at this temperature, (4) applying the additional load σ_{dil}, (5) continuous cooling to room temperature, (6) unloading the specimen at room temperature.

The main difficulty of these experiments is to define the level of the additional load σ_{dil}. As mentioned above, it is necessary to ensure that the yield strength of the softer phase is not exceeded. Also, the TRIP effect is greater as the load increases. Ideally, the load should be as high as possible, but not so high as to cause additional plasticization of the softer phase. Austenite, which is also the softest phase, is the origin of all phases formed during transformation. The yield strength of austenite at a temperature just above the martensite start temperature θ_{MS} is a necessary design variable for the experiment. However, due to the very short time window available, it is not possible to perform a measurement of this value directly. Nevertheless, in order to estimate the plastic behavior of the austenite at lower temperatures, the results of the hot tensile test are extrapolated and a safety factor is applied to ensure that the material does not plasticize due to the additional load during the test.

In order to determine the parameter K_i, it has proven useful to perform measurements under both tensile and compressive loading. An unloaded measurement always serves as a reference. The applied load σ_{dil} is in each case 75% and 37.5% of the previously extrapolated yield strength of the austenite, taking into account the introduced safety factor. σ_{max} takes into account both the yield strength of the austenite and the safety factor. Table 4.4 summarizes the overall test plan of the deformation dilatometry.

transformation	θ_{LO} (°C)	σ_{max} (MPa)	σ_{dil}
austenite to martensite	350	250	
austenite to bainite	600	100	0.0% ± 37.5% ± 75.0%
austenite to pearlite	700	50	

Table 4.4: Experimental design for the deformation dilatometry tests of ER7.

4.5 Modeling of the Material Behavior

This section deals with the modeling of the material behavior. On the one hand, mathematical models are used to describe complex processes such as transformation kinetics or TRIP. On the other hand, it describes how the measured data are processed in order to use them as a material model in the FE calculation.

4.5.1 Modeling of Phase Transformation Kinetics

To model phase transformation one must first describe the two mechanisms responsible for the transformation. The driving force for transformation from austenite to martensite is supercooling and thus exclusively dependent on temperature. The other transformations into the phases bainite and pearlite are diffusion-driven and are described by different kinetics than the martensite transformation.

Koistinen and Marburger first described the austenite to martensite transformation in [107]. The fraction of martensite ξ_M is expressed in equation 4.1,

$$\xi_M = 1 - exp(-\alpha(\theta_{MS} - \theta)) \quad , \tag{4.1}$$

where α is a constant that represents the velocity of the phase transformation, θ_{MS} represents the martensite start temperature and θ the current temperature. Consequently, $(\theta_{MS}-\theta)$ results in supercooling below the martensite start temperature.

Modeling the diffusive transformation in this framework faces the challenge of fitting a multitude of parameters compared to the diffusionless and athermal transformation of austenite to martensite. One model for the diffusive phase transformation was given by Garrett et al. [108] and later refined by Mahnken et al. [109]. The underlying assumption is that in order for one phase to form from another, a nucleus of a critical size must first form and then grow at a certain rate. The necessary calculations can also be divided in this way, namely into a nucleation phase and a growth phase.

The transformation of one phase into another does not start immediately with the onset of cooling of the material, but with a time delay. This time delay is called incubation time. As can be seen in a time-transformation diagram, this time changes depend both on the cooling rate and the current temperature. In order to describe such a phenomenon, a model is necessary that takes into account the different cooling rates and determines the start of the transformation at different temperatures and times. One idea to calculate this incubation time is the nucleation model of Garrett et al. [108]. The idea behind the nucleation model is this: the new phase begins to grow only when the nucleus reaches a critical size. Until then, only the nucleus grows and no new phase is formed. [108, 109]

The underlying equations of the model have their origin in thermodynamics. For a thermodynamic process to start, a certain activation energy must be overcome. For the present case, this is described by equation 4.2, which is a slightly simplified form the equation presented in [108].

$$G^* = \frac{A_3 \Delta\theta^{1.5} \theta_0^2}{\Delta\theta Q_a} \tag{4.2}$$

In this equation G^* represents the Gibbs free energy at its peak, A_3 is a material constant, $\Delta\theta$ is the undercooling, θ_0 the equilibrium temperature at which the parent phase has the same free energy as the product phase, and Q_a is the activation energy.

The critical nucleation radius corresponding to the energy in equation 4.2 is r^* in equation 4.3. This critical nucleation radius strongly depends on the temperature. The so far

unmentioned quantity of this equation is A_1, which is a material constant used for modeling purposes, that is influenced by the surface free energy. [108]

$$r^* = \frac{A_1 \theta_0 \Delta\theta^{0.5}}{\theta \Delta\theta Q_a} \tag{4.3}$$

Finally, the differential equation for the growth of the nucleus can be defined as

$$\frac{dr_n}{dt} = \alpha_n \, exp\left(-\frac{|\theta - \theta^*|}{B}\right) \quad , \tag{4.4}$$

where B and α_n are again material constants used for modeling purposes of the growth rate. The required initial condition of this differential equation is $r(t = 0) = 0$. The nucleus radius r_n grows until it reaches the critical value r^*, at which point the actual phase growth can begin. The time taken to reach r^* is the aforementioned incubation time. The temperature θ^* represents a special temperature value, i.e. the temperature corresponding to the shortest possible incubation time of a given phase.

Using equations 4.2-4.4 it is possible to model the incubation time and thus the start of the phase transformation. This model is used for both the bainite and pearlite phase. Once the onset of transformation is determined, the rate at which the phase grows is required. Equation 4.5 describes the growth of the product phase in the form of a differential equation which is also temperature sensitive,

$$\frac{d\xi_f}{dt} = A_5 \, exp\left(\frac{\theta^* - \theta}{B_1}\right) \left(\frac{r - r^*}{r}\right)^n (1 - x_f)^\gamma \, exp\left(-\frac{G^*}{RT}\right) \quad , \tag{4.5}$$

where A_5, B_1, n and γ are material constants and R ist the universal gas constant.

The actual modeling work lies in the determination of the numerous parameters. Again, the problem is divided into two parts, first the nucleation and then the phase growth. The parameters are then optimized using a py software until a complete set of parameters is available to describe the transformation kinetics of bainite and pearlite, respectively. In order to use the model in a finite element simulation, the whole set of equations including the found material parameters has to be implemented with the help of user subroutines. The respective phase fraction is defined as a field variable on which other material properties depend.

4.5.2 Modeling of Plastic Material Behavior

This subsection describes the experimental setup of the hot tensile tests and how to process the data to make it available for use in an FE simulation.

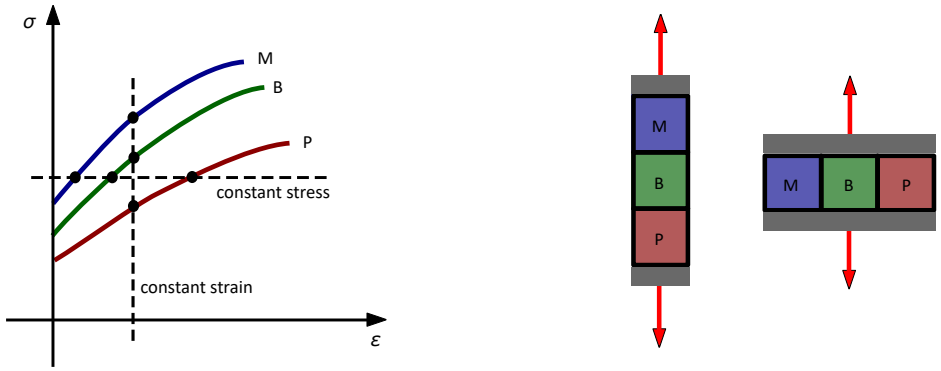
The first step is to define how the plastic law is to be implemented. The easiest way is to enter the flow curves in table form. The dependency of different influencing variables such as temperature, strain rate or any other field variable can also be taken into account easily. Unfortunately, the measurement results cannot be transferred directly to the FE software. They need to be processed first before being used. The main goal is to generate a data set of flow curves that represent the pure phase behavior of ER7 as a function of temperature.

In order to achieve this, it is first necessary to distinguish the elastic from the plastic part of the stress strain curve, or in other words, to find the yield strength of the individual measurement. For the purposes of the simulation, it is assumed that the elastic behavior of each phase is the same and therefore they all have the same Young's[1] modulus. However, the measurement shows deviations of about 5% of Young's modulus for the different measurements. It is well known that a tensile test is not the most precise method for determining Young's modulus. Nevertheless, the measurements can be used to determine the flow curves because the variations in Young's modulus are within a reasonable range. Therefore, only the plastic parts of the measurement are processed, assuming a constant modulus. To determine the end of the elastic and the beginning of the plastic behavior, a regression line is plotted through each of the measured points. The points used for this are determined individually for each measurement in such way that they obey Hooke's law. At a 0.2% strain deviation from this line, the plastic region begins. All flow curves are determined in this way. In the next step, they are checked for plausibility and adjusted if necessary. An example of this is the removal of minimal overlaps of flow curves of the same phase at different temperatures. Such an input is not tolerated by the FE software.

With the method described so far, the flow curves for martensite and pearlite can be determined because the tensile specimens consist of only one phase each. This is not possible for the bainite flow curve. According to CCT, the specimen cooled at $\lambda = 0.15$ has a martensite:bainite:pearlite ratio of 35:35:30. To determine the pure bainite behavior, the linear mixing rule is applied first. For this purpose, a linear system of equations is created from the already known flow curves and the known mixing ratio, and thus the unknown flow curve of the bainite is determined. For such a system of equations, another assumption must be made, namely whether the mixing of the mechanical properties takes place with constant strain or with constant stress. Figure 4.5 shows this consideration graphically, on the one hand with the help of a stress-strain diagram and on the other hand with different spatial arrangements of the different phases. The resulting bainite flow curves are then analyzed. First, there is a large deviation between the two different methods, and second, the results do not pass the plausibility check. At constant stress, bainite shows higher strength than martensite, and at constant strain, the flow curve of bainite is below that of pearlite. The true flow curve must be somewhere in between. This is the reason for a simulation assisted approach.

In this approach, the hot tensile test is modeled using Finite Element (FE) software. For this purpose, a so-called representative volume element (RVE) is used, as described in detail in Subsection 4.6.1. The loading conditions in the simulation are identical to those in the experiment, with 1% tension and 1% compression. The material data required for the FE simulation are stored separately for each phase in tabular form, and initially, the data for bainite is estimated. The primary objective of this simulation is to predict the mechanical behavior of bainite and compare it to the experimental results. To account for the mixing ratio of the phases, a user subroutine is programmed, enabling the representation of different mechanical behaviors within the RVE model. Each element in the model is assigned a phase with its

[1]Thomas Young (13 June 1773 – 10 May 1829): British mathematician.



(a) Flow curves in a stress strain diagram with a representation of the mixture rule at a constant strain or stress value.

(b) Schematic of a tensile test with different spatial distributions of phases.

Figure 4.5: Different approaches for the mixing rule.

individual mechanical properties. The stress-strain diagrams calculated in the simulation are then compared with the measured data, and if necessary, adjustments are made to the bainite flow curve. This iterative process continues until a satisfactory agreement is achieved between the measured and simulated mixed flow curves. To provide a visual representation of this procedure, Figure 4.6 illustrates the measured flow curve, including all three phases, the assumed bainite flow curve, and the resulting mixed flow curve obtained from the simulation. The flow curves generated through this methodology are presented and discussed in Subsection 4.7.2.

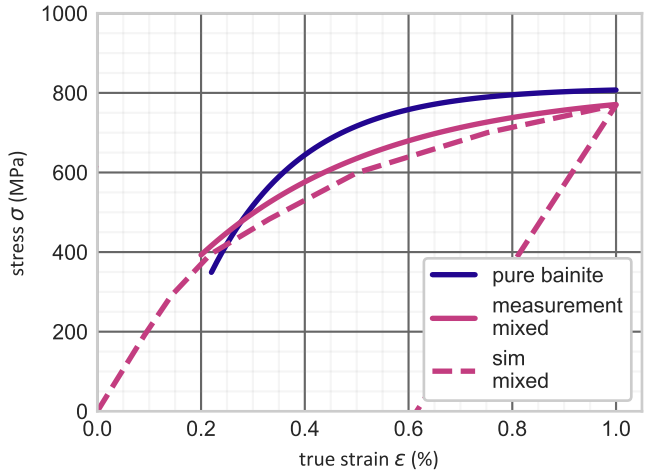


Figure 4.6: Comparison of measured and simulated plastic material behavior of ER7 steel with a composition of 35:35:30 of martensite:bainite:pearlite.

ER7 exhibits yet another noteworthy feature. At low temperatures, some specimens exhibit a pronounced yield strength. This behavior is particularly challenging to model. For the purpose of this work, the representation of the pronounced yield strength is omitted. However, this means that the material behavior must be modeled in a different way. In a first attempt

to create a continuous flow curve from the measured data, it was difficult to determine where the elastic region ends and the plastic region begins. In addition, it was unclear what shape the yield curve would take if it did not show a pronounced yield strength behavior. To solve this problem, a material behavior that is otherwise of interest in fatigue is used, namely, the Masing behavior. This means that a stress or strain amplitude that is equal to twice the value of the amplitude of the stress or strain of the initial load curve is applied upon reloading. Figure 4.7 shows this relationship in the form of a cyclic stress-strain diagram.

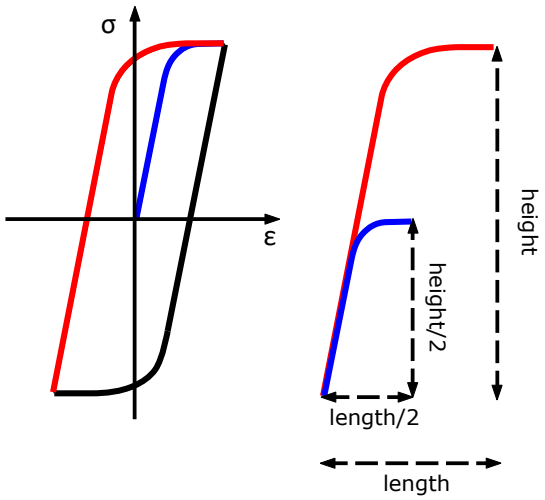


Figure 4.7: Masing hypothesis: the stress strain curve of the first cycle doubles for the second cycle.

With respect to the experiment to be modeled, the blue curve corresponds to the first tensile load, the black curve to the reverse load to 1% compression, and the red curve to the second tensile load. Here, in contrast to Masing's idea, the first load is not used as a basis for the subsequent loads, but the second load is used as a basis for the first load. In this way, an experimentally based material behavior can be derived and implemented in the FEM.

4.5.3 Modeling of Transformation Induced Plasticity

Multiple formulations for the transformation induced plastic strain rate exist. For numerical simulations of heat treatment processes, it has proven useful to describe the resulting strain due to the transformation using Leblond's law [21, 104, 106]. In its general form it reads:

$$\dot{\boldsymbol{\varepsilon}}_i^{tp} = \frac{3}{2} K_i f'(\xi_i) \dot{\xi}_i \boldsymbol{S} \quad , \tag{4.6}$$

where the index i represents the individual phase, $\dot{\boldsymbol{\varepsilon}}_i^{tp}$ represents the transformation induced strain rate, K_i the Greenwood-Johnson (GWJ) transformation coefficient, $f'(\xi_i)$ the derivative of a saturation function, ξ the volume fraction of the product phase satisfying $0 \leq \xi \leq 1$, $\dot{\xi}$ the time derivative of ξ and $\boldsymbol{S}$ the deviatoric part of the stress tensor. There are also many formulations for the saturation function. One of the most commonly used is:

$$f(\xi_i) = (2 - \xi_i)\,\xi_i \quad , \tag{4.7}$$

and consequently its derivative is:

$$f'(\xi_i) = 2\,(1 - \xi_i) \quad . \tag{4.8}$$

To implement this law in the FE software Abaqus, the user subroutine CREEP is used, but instead of a creep strain, the TRIP strain $\dot{\varepsilon}_i^{tp}$ is returned. This was put into practice by [21]. To model the TRIP behavior of ER7, only the GWJ transformation coefficient K_i needs to be determined for each phase transformation. The results of K_i and a comparison of the experimental and simulated TRIP behavior can be found in subsection 4.7.3.

4.6 Finite Element Models for the Heat Treatment of Railway Wheels

This section deals with the finite element models developed and used in this thesis. First, a brief overview of the RVE used to model all experiments is given. This is followed by a subsection describing the entire FE model developed here for the heat treatment of railway wheels.

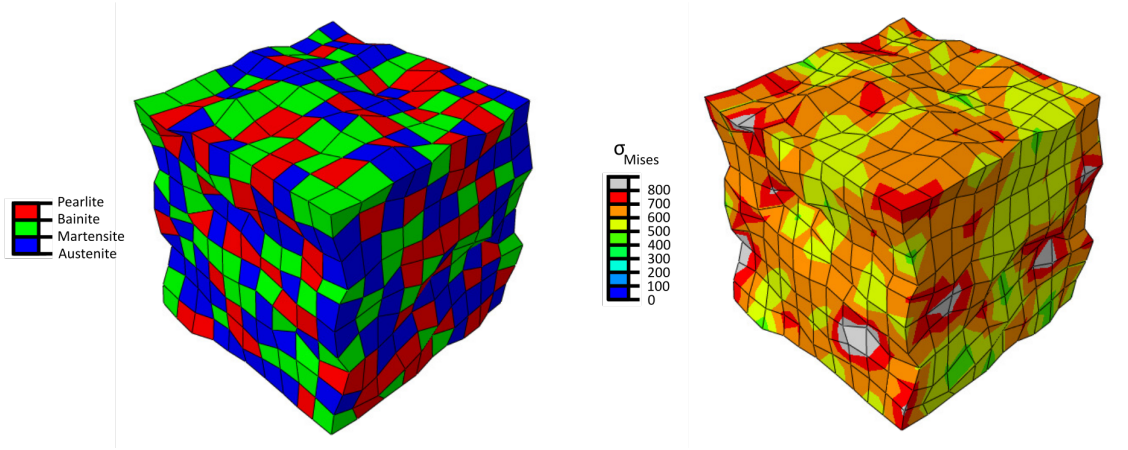
4.6.1 Representative Volume Element for Material Modeling

A representative volume element is a periodic unit cell. One common definition is that it is the smallest possible cell for which tests can be performed that are representative of the entire material. In continuum mechanics, RVEs are used to represent heterogeneous material behavior at the scale above. Here, the heterogeneous behavior originates from the different phases. [110]

The key to such an RVE is the boundary conditions and couplings. The surfaces are coupled in such a way that a periodic arrangement of this cell is possible in any spatial direction. The edges and corners make it possible to represent true shear with a slightly modified coupling. The RVE used in this thesis was adopted from previous works, e.g. it was used by [21].

Figure 4.8 shows an RVE with a heterogeneous distribution of different phases, as they can occur during heat treatment, and the resulting stresses.

A py script creates the entire geometry including all the couplings. See Appendix A.1 for the code of the py script for Abaqus used to generate the RVE models. The number of elements is arbitrary, but a size of 8x8x8 has proven suitable for the representation of metallurgical processes. Here, the size has been increased to 10x10x10 for an easier representation of the relationship of phase fractions and the number of transformed elements. The smallest possible number of elements is 1, which is used in the single-element test for the modeling of the TRIP behavior that is later implemented in the process model. All simulation results performed in parallel to the experimental work were generated with these RVEs and can be found in section 4.7.



(a) Heterogeneous distribution of stresses in an RVE caused by the different mechanical properties of the various phases.

(b) Distribution of different phases of the ER7 steel in an RVE.

Figure 4.8: Hot tensile test performed with an RVE.

4.6.2 Process Model for the Heat Treatment of Wheels

This subsection describes the setup of the FE model for the simulation of the heat treatment process of a railway wheel made of ER7. Special emphasis is placed on the modeling of the heat treatment process itself with all its boundary conditions and settings based on the actual process.

Process models always focus on the interaction of the process with the associated component. The component in this work is a railway wheel, specifically a solid wheel made of ER7 with a diameter of 850 mm and a profile of S1002/h28/e32.5/6.7% according to the EN13715 standard [111]. Such wheels are installed on the Desiro ML ÖBB Cityjet vehicle from Siemens. They are used for all wheelsets of the train, both powered and non-powered (also called trailer wheelsets).

Such a wheel is manufactured by the steps described in Subsection 4.1.2, where the steps considered in this process simulation are those of quenching and subsequent tempering. After forging, the wheel is placed in a pusher furnace for normalizing. During this process, the wheel is heated to 850 °C, which allows sufficient time for the microstructure to completely transform to a low-stress austenitic microstructure. Leaving this furnace represents the beginning of the process simulation and thus the stress free reference state.

Quenching Process for Railway Wheels

From this furnace, the wheel enters a water tank for quenching. This is a special quenching process in which only the tread of the wheel is quenched with water. Normally, a spray cooling process is used for this type of quenching, in which the tread is targeted with nozzles to get the water exactly where it is needed. A modified form of this cooling is used here. The wheels are submerged in a tank. In order to guarantee the contact of the water with the tread of the wheel, but at the same time to exclude the contact of the water with the rest of the wheel, a

specially designed water tank is used. Figure 4.9 shows a schematic of such a quenching tank with a submerged wheel.

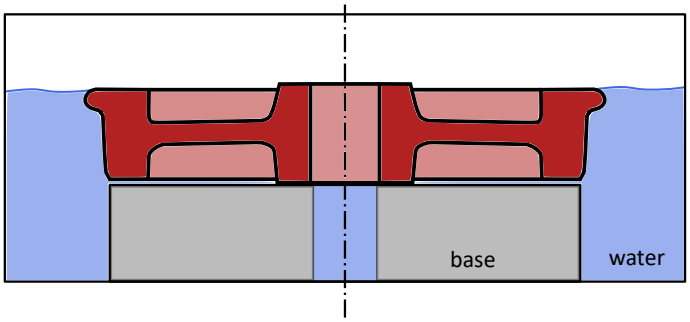


Figure 4.9: Schematic of the quenching tank with a submerged wheel.

Inside the tank there is a base that serves as a support for the wheel. This base also serves as a water drain that regulates the water level of the tank by allowing the water to run off which made it either over the flange of the wheel or between the wheel and the base. This method ensures that only the tread of the wheel is in contact with water. The amount of water that reaches the other parts of the wheel is negligible. The water in the tank rotates by means of angled nozzles so that fresh, cold water always reaches the wheel's tread. For the purpose of the simulation, it is assumed that fresh water is brought to the surface by movement and that there is no formation of a vapour film. This means that there is no insulating vapour layer on the surface of the component, as would be the case if the wheel were submerged in still water.

Temperature Measurement with an Instrumented Wheel

As the previous section shows, the quenching process is subject to many uncertainties. In order to correctly represent the heat transfer many different effects must be taken into account, i.e. the heat transfer by radiation, conduction and convection. For many forms of heat transfer, some of which are complex, the literature offers ways to calculate the heat transfer coefficient. However, for a forced flow with temperature differences above 800 K between a glowing steel surface and water, there is no satisfactory approach. For this reason, the heat transfer coefficient is calibrated by means of a temperature measurement during the process.

For this purpose, a wheel equipped with thermocouples passes through the entire heat treatment process. Temperature measurements are taken at multiple positions on the wheel to better distinguish between the different contributions to the heat transfer. The positions of the thermocouples in the wheel are shown in Figure 4.10.

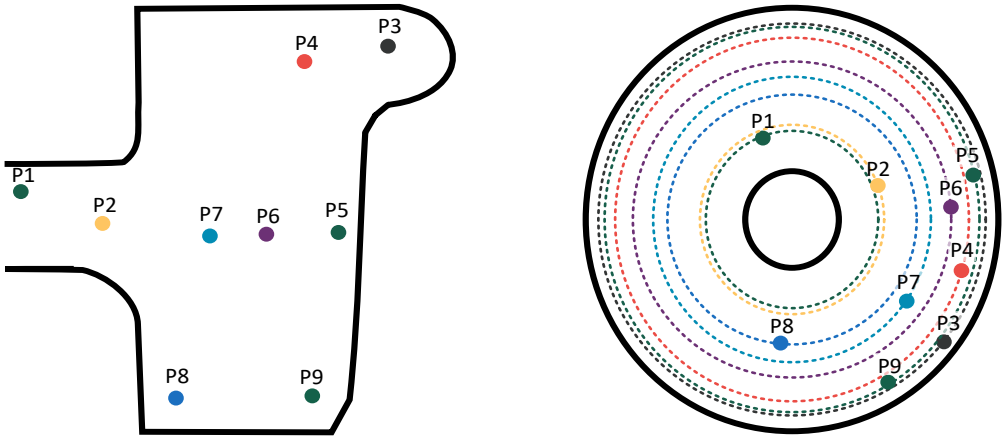


Figure 4.10: Positions of thermocouples in instrumented railway wheel.

Heat Transfer Modeling

The temperature measurements of the instrumented wheel are used as the basis for calibrating the heat transfer coefficients and thus the total heat transfer. Similar to the calibration of the bainite flow curves, the heat transfer calibration is inversely fit to the experimental results. The largest influence on the heat transfer is the water quenching at the tread. For this reason, the remaining contributions to heat transfer are determined in advance of the calibration. The contributions to heat transfer are: (1) conductivity, (2) radiation and (3) convection.

(1) In principle, it would be possible to determine the thermal conductivity of ER7 from these measurements. However, there are specific measurements carried out to determine the thermal conductivity in a previous work. Thermal conductivity is not a parameter that varies greatly from batch to batch and can therefore be safely adopted. A number of other material parameters are required for heat transfer calculations. All the temperature dependent physical material parameters required for this have also been taken from previous work and can be found in Table 4.5, where c denotes the specific heat capacity, α_{th} the coefficient of thermal expansion, ρ the mass density and λ the thermal conductivity. It should be noted that the parameters c, ρ, and λ are assumed to be constant across all phases, while α_{th} is replaced with phase-dependent values in subsequent calculations. For the initial calibration calculations, the values in Table 4.5 are used. Furthermore, these values are used exclusively for the calculation of the heat conduction. Other effects that may cause changes in density, such as volume changes during transformation (i.e. transformation expansion ε^{tv}), are considered at a later stage of the model.

(2) The contribution to the heat flux due to thermal radiation is a function of two natural constants, namely the temperature difference between the component and its surroundings and the emissivity ε of the material. Emissivity is a quantity that can vary widely for the same material. In some cases it can vary by a factor of 100 depending on the surface characteristics. For this reason, a plausible value for the emissivity must be assumed. In this thesis, a value of ε=0.29 is used to represent a surface that is not perfectly smooth during heat treatment. If not

θ	c	α_{th}	ρ	λ
°C	$^{kJ}/_{kgK}$	$^{10^{-6}}/_{K}$	$^{kg}/_{m^3}$	$^{W}/_{mK}$
20	0.470		7836	42.2
100	0.500	11.363	7815	42.9
200	0.534	12.344	7784	42.3
300	0.567	13.107	7750	40.5
400	0.606	13.650	7715	38.4
500	0.658	14.250	7677	36.4
600	0.732	15.252	7632	33.9
700	0.856	15.662	7591	30.4
750	0.604	13.611	7607	25.0
800	0.609	12.841	7605	25.7
850	0.614	13.410	7580	26.3

Table 4.5: Temperature-dependent thermo-physical data of ER7.

chosen correctly this value has no significant effect on the rest of the calculation. The remaining contribution to the heat transport, namely convection, is adjusted to the measurement in the next step, thus including any missing parts of the heat transfer by radiation. Nevertheless, a small deviation is unavoidable, since the temperature is of first order in Newton's[1] law for conductive heat transfer and of fourth order in the Stefan[2]-Boltzmann[3] law for thermal radiation, and thus an incorrectly chosen value for the emissivity has a greater impact at higher temperatures.

(3) The contribution of convection to the overall heat transfer in the wheel is divided into two components. One component accounts for the interaction with water at the tread of the wheel, which is calibrated through measurements. The other component represents the heat transfer at the remaining parts of the wheel, involving the interaction with air. This component can be calculated in advance. Temperature and geometry-dependent heat transfer coefficients are determined for Newton's law by considering the complex interaction between the air and the surface of the wheel. This involves using dimensionless quantities and temperature-dependent material parameters of the air. The detailed procedure is described in [112, 113]. The calculated heat transfer coefficients, which vary with temperature and geometry, are presented in Table 4.6. The different geometries considered include the top and bottom of horizontal surfaces, as well as vertical cylindrical surfaces, which are further influenced by their diameter and height. The corresponding surfaces with their respective heat transfer coefficients are shown in Figure 4.11 through color coding.

The second contribution to convective heat transfer is that of the water-cooled tread. To determine this, an FE simulation is set up to represent the first stage of the heat treatment, the quenching. The thermal boundary conditions are implemented according to the previously

[1] Isaac Newton (4 January 1643 - 31 March 1727): English mathematician, physicist and astronomer.
[2] Josef Stefan (24 March 1835 - 7 January 1893): Carinthian Slovenian physicist, mathematician and poet.
[3] Ludwig Boltzmann (20 February 1844 - 5 September 1906): Austrian physicist and philosopher.

temperature θ in °C	heat transfer coefficients α_k in $^{W}/_{m^2K}$		
	horizontal top	horizontal bottom	vertical cylinder
25	0.000	0.000	1.335
50	3.753	1.753	5.033
100	5.125	2.169	6.620
200	6.168	2.537	7.887
300	6.587	2.744	8.460
400	6.782	2.890	8.781
500	6.864	3.002	8.973
600	6.891	3.093	9.098
700	6.893	3.172	9.190
800	6.856	3.236	9.232
850	6.842	3.268	9.258

Table 4.6: Temperature and geometry dependent heat transfer coefficient α_k.

discussed points. Figure 4.11 illustrates these boundary conditions at the corresponding surfaces of the wheel. To evaluate the simulation, points are defined at the same positions on the wheel where the thermocouples were placed in the experiment, cf. Figure 4.10. As a starting point for the calibration of the heat transfer coefficient, an analytical method is again used to determine a temperature and pressure dependent heat transfer coefficient for nucleate boiling of water. The exact procedure can be found in [38, 39] with material data taken from [113]. Similar to the calculation for convective heat transfer in air, dimensionless quantities and functions are derived from tabulated material parameters of boiling water to finally derive this heat transfer coefficient. These calculated values are the starting point for the calibration.

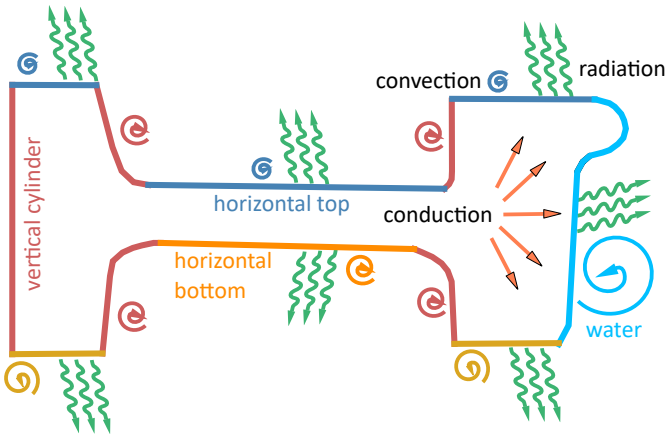


Figure 4.11: Boundary conditions of the heat treatment model during the water quenching step.

The calibration itself is now carried out in an iterative process where the heat transfer coefficient independent of temperature is fitted to the measurement. As soon as a good agreement is achieved, the temperature dependence is implemented in a second phase based

on trends from literature. The values obtained with this method are shown in Table 4.7. Note that a temperature of 162.5 °C is necessary for the calculation of the reference Nusselt number for the nucleate boiling heat transfer coefficient.

θ	°C	25	50	100	162.5	850
α_k	W/m^2K	30	209	385	2380	2380

Table 4.7: Temperature-dependent heat transfer coefficient for a water quenched surface.

The other heat treatment steps, such as tempering, cooling and transport between treatments, also require temperature and geometry dependent heat transfer coefficients. Since the surrounding medium is always air, the calculated values can be taken from Table 4.6. In the simulation, only the ambient temperature is adjusted.

Process Modeling

To determine the stress state of the wheel after heat treatment, sequentially coupled heat transfer and mechanical analyses are performed. This means that the heat transfer analysis is calculated first and the resulting temperature field is used as an input in the subsequent stress analysis.

The process model itself is axially symmetric. Although the web of the wheel has bores that only a 3D model can represent correctly, the advantages of a fine-meshed, axially symmetrical model outweigh the disadvantages of a 3D model that considers the bores. Also, the bores on the web are in a zone where there are only moderate temperature gradients. The chosen element type is a quadrilateral axially symmetric element with first order shape functions. These elements are particularly well suited to thermal analysis because the heat conduction equation for constant parameters yields only a linear temperature distribution within an element.

The core of this process model is the implementation of the material model. For the thermal analysis, the temperature-dependent thermophysical parameters are used, but also parts of the user subroutine are already required. User subroutines are used to describe a material behavior that exceeds the standard capabilities of commercially available finite element software. The transformation behavior is implemented as a purely thermal problem, so that a prediction of the distribution of the individual phases is already possible after the thermal analysis. The mechanical analysis then uses the full set of phase and temperature dependent flow curves. Again, user subroutines have to be coded to describe the complex mechanical material behavior such as e.g. the TRIP effect.

The boundary conditions for heat transfer are selected according to the steps of the actual heat treatment. These include the quenching, tempering and subsequent cooling before further manufacturing steps. The transports before and between the treatments are also modeled. With this setup it is possible to get accurate insights into the processes inside a heat treated railway wheel during the heat treatment process and also obtain information from inside the wheel that would not be accessible with any other method than simulation.

The process model generation is again fully automated using a py script. See Appendix A.3 for more details on the code.

4.6.3 Model Extension: Block Braking

The present model is designed specifically for heat treatment process modeling, yet it holds potential for addressing thermal issues related to braking as well. Consequently, the heat treatment models developed in this work can be utilized to describe the heat input resulting from block braking.

Note, that the model and the corresponding results presented in Subsection 4.7.6 were developed within the framework of a master's thesis [114] under my supervision.

Block brakes, the oldest braking system employed in trains, continue to be utilized in freight traffic to this day. This system involves pressing a brake block against the wheel's tread using a braking force. As the train's kinetic energy is transformed into heat, both the wheel and the brake block experience heating. Effective dissipation of this heat is crucial to prevent thermal overload, which can lead to excessive temperature rise in the components. The advantages of block brakes are apparent, as they boast a simple design and have a long-standing track record of reliability. Additionally, the contact between the brake pad and the tread serves the secondary purpose of cleaning the tread, thereby enhancing contact and power transmission with the rail. However, it is important to consider the drawbacks, such as the already high stress experienced by the wheel as a critical component. Consequently, in the design of these components, the additional thermal load, particularly at vulnerable locations like the tread, must be taken into account in addition to mechanical loads associated with supporting and guiding the vehicle on the rails. [88, 115, 116]

The UIC 510-5 standard [85] serves as a crucial reference for the design of components in the railway industry, particularly in relation to block brakes. According to this standard, a solid wheel intended for freight cars must meet specific requirements, including the ability to withstand a demanding braking scenario. Specifically, the wheel should be capable of enduring 45 minutes of continuous braking at a speed of 60 km/h, with a braking power of 50 kW. This particular case represents an extreme condition within the European railway system, simulating a journey over the Gotthard ramp. To ensure optimal performance, the design of the wheel must be meticulously engineered to withstand the associated thermal load imposed by this intense braking scenario. Adhering to the UIC 510-5 standard guarantees that the wheel meets the necessary criteria for safe and reliable operation under such challenging conditions.

The braking scenario described represents a combination of both thermal and mechanical loads that impact the performance of railway wheels. The wheel hub absorbs the axle loads, which are subsequently transmitted through the web to the rim and the tread and ultimately transferred to the contact point with the rail. Block braking adds an additional mechanical load onto the wheel as the brake pushes against it. However, this study focuses exclusively on the thermal effects caused by the block brake, specifically analyzing the resulting stresses affected by the thermal contribution of the brake. The investigation does not consider the influence of other concurrent mechanical loads on the wheel. By isolating and examining the thermal contribution of the block brake, the study aims to gain insights into the thermal behavior and stress distribution within the wheel during this specific braking scenario.

Block Braking Model

The modeling of the block braking process is accomplished by utilizing the existing heat treatment model and expanding it to meet the specific requirements of the braking scenario. A standardized brake test is employed as a reference to simulate the conditions experienced during the traversal of the Gotthard ramp. In this test, a railway wheel is mounted in a test stand, with a rigid axle connected to the wheel and being driven while simultaneously applying hydraulic braking force via a brake block. To capture the thermal behavior accurately, thermocouples are installed on both the wheel and the test stand, enabling the recording of temperature profiles. These temperature curves are then utilized to calibrate the thermal boundary conditions in the finite element model. It should be noted that the brake test primarily focuses on the thermal load imposed on the wheel and neglects the additional mechanical loads, aligning with the specific objective of investigating the thermal effects induced by block braking.

Various thermal boundary conditions can be applied to the edges of the FE model, allowing for the representation of heat transfer both from the component to its surroundings and the reverse case, such as heat input into the wheel during braking. In this study, the braking power is converted into a heat flux density and applied to a specific section of the wheel's tread that is about 80 mm wide. However, the temperature increase resulting from the braking power specified in the standard may not correspond exactly to the measured values in an idealized model. To address this, the heat flux density is adjusted inversely based on temperature measurements, providing a more accurate representation that accounts for previously unconsidered losses and heat transfer to the brake pad. The remaining edges of the FE model incorporate convection for heat transfer, considering both the geometric factors and the influence of the wheel's rotation, which affects the convective heat transfer by a moving air flow.

The simulation process for the thermal analysis follows the same steps as the mechanical analysis. Prior to simulating the braking process, a heat treatment is applied to the wheel under investigation in order to establish an initial stress field. Initially, a 45-minute period of braking is simulated, during which the wheel rotates. This is followed by a 4-hour cooling period without rotation. The heat transfer coefficients are adjusted accordingly to simulate these processes accurately. To capture the stress redistributions that occur during operation and to ensure the component design meets the required criteria, the simulation is repeated 30 times until a stabilized stress-strain state is achieved. It is important to note that the employed material model in this process is not the same as for the heat treatment simulation. It can effectively simulate cyclic plasticity in a temperature-dependent manner, albeit without incorporating the effects of phase transformation and TRIP.

In contrast to the previous calculations utilizing the heat treatment process model, the present study employs a different wheel geometry known as the SURA wheel, which is specifically used for freight transport and block braking applications. This SURA wheel features a curved web and a diameter of 920 mm, distinguishing it from the Cityjet wheel geometry. The focus of this investigation lies in examining the impact of thermal loads induced by block braking on critical locations of the web, which is of particular interest to railway operators. The results obtained using this process model are presented in Subsection 4.7.6, shedding light on the effects and behavior observed under these specific conditions.

4.7 Results and Discussion

The results section focuses on the material model for the ER7 steel. Firstly, the results of the experimental work are presented and the material parameters derived from them. The structure is similar to the modeling section with subsections on CCT diagram, hot tensile tests and forming dilatometry. A subsection is then devoted to the developed material model and summarizes all the parameters required for it. The results section concludes with the results of the process model and the special case of the heat treatment, namely the block braking.

4.7.1 Continuous Cooling Transformation Phase Diagram

With the dilatometer experiments described in subsection 4.4.1, it was possible to create a CCT. The CCT with the corresponding cooling curves from the dilatometer experiments is shown in Figure 4.12. It shows the different transformation behavior of ER7 at different cooling rates. It also shows the onset of transformation of each phase as well as the time at which each phase ceases to form.

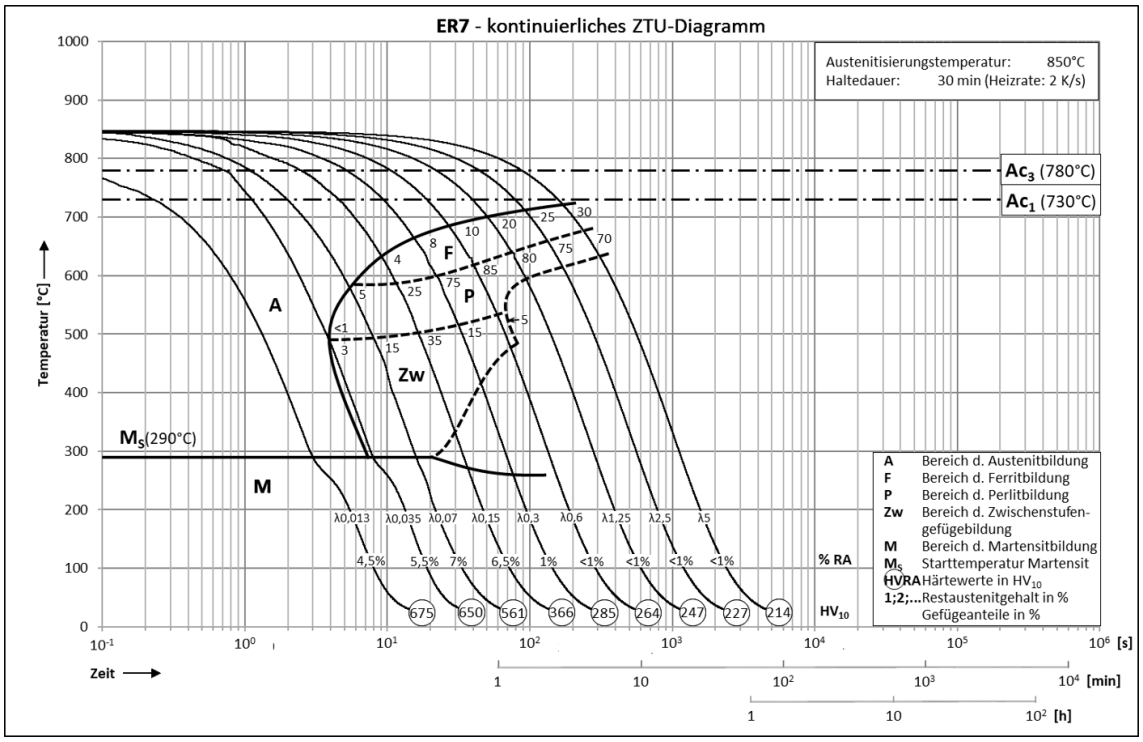


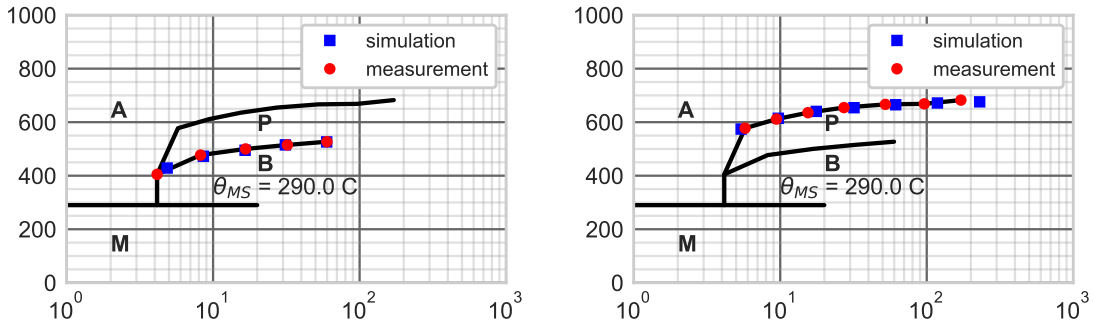
Figure 4.12: CCT diagram of the ER7 steel.

The martensite start temperature θ_{MS} is constant at 290 °C and decreases slightly at lower cooling rates, although this slight decrease is not accounted for in the material model. In general, well-defined phases of bainite, pearlite and ferrite are formed at moderate cooling rates, however the case is slightly different for the low alloyed ER7 steel. As seen in the CCT, the ferrite, pearlite, and bainite phases merge. Only the beginning of the transformation into one of these phases is well-defined. The transition from ferrite to pearlite and then to bainite is

not clear from the dilatometer tests alone. This is also the reason for the dashed representation of the phase boundaries in this CCT. The phase fractions shown in the figure were determined by metallographic examination. This includes the retained austenite content, shown as %RA in the diagram. An additional verification of the phase composition is the hardness test performed on the specimens. It shows high values for the pure martensitic microstructure at the highest cooling rate and decreases with increasing pearlite and ferrite phase content. In the diagram the values are marked with HV_{10}.

Furthermore, it can be seen in the CCT that the ferrite, pearlite and bainite region is very far on the left side of the diagram compared to high-alloyed steels, which means that the transformation into these phases occurs very early. As a consequence, extreme cooling rates are required to obtain a pure martensitic microstructure. In other words, if cooling from 850 °C to 500 °C takes longer than 4 seconds, some bainite has already formed. It is precisely this small time window that makes the modeling of the process and the experiments difficult. However, the first simulation models show that these high cooling rates can occur directly on the tread of the railway wheel. For this reason, also the highest cooling rates were taken into account in all experiments and simulations.

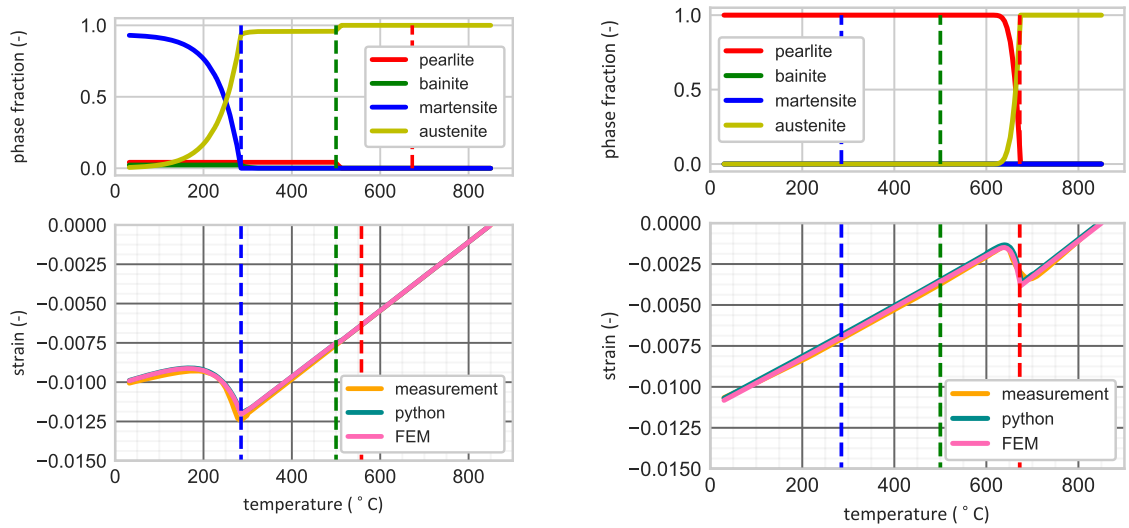
Using the nucleation model described in Subsection 4.5.1, it is possible to model the transformation behavior. A py routine was used to determine the material parameters from equations 4.2 through 4.4. In the routine, the transformation start of the respective phase and the cooling rate are entered and the differential equations are solved. The parameters found are then varied with the help of an optimizer until there is good agreement between the experimentally determined values and the simulated values. A comparison of these points is shown in Figure 4.13. Note that the phase labeled P already represents both pearlite and ferrite.



(a) Bainite **(b)** Pearlite

Figure 4.13: CCT of ER7: comparison between measurement and simulation of the transformation times of bainite and pearlite.

The comparison of experiment and simulation in Figure 4.13 shows almost perfect agreement for modeling the onset of the bainite transformation. The only noticeable deviation occurs at the point with the most rapid cooling rate. However, given that the abscissa is logarithmically scaled, this deviation is minor, registering within the range of one second. In addition, the modeling of the pearlitic transformation demonstrates a favorable correspondence between experimental and simulated results. Notably, the agreement is more pronounced for



(a) Cooling with $\lambda = 0.013$ hs triggers a transformation into martensite.

(b) Cooling with $\lambda = 5.000$ hs triggers a transformation into pearlite.

Figure 4.14: Comparison of measured and modeled dilatometry signal and corresponding evolution of the phase fraction for martensite and pearlite. The vertical lines represent the onset of the respective phase transformation.
Top: phase fraction evolution.
Bottom: dilatometry signals of the measurement, the py and the FEM simulation.

the more rapid cooling rates, while the temperature level of the transformation onset remains within acceptable limits for all cooling rates. For the slower cooling rates there is a deviation in time, but it is again within reasonable limits. The experimentally determined CCT shows that only a pearlitic microstructure is formed in this range, rendering the observed deviation negligible.

The second part of the modeling of the transformation kinetics is the growth of the respective phase. Again, a py script is used to solve the differential equations and optimize the desired material parameters from equation 4.5. This time the optimization is done by matching the entire dilatometer curves of the experiment and the simulation, as can be seen in the two lower plots of Figure 4.14.

For this purpose, the linear thermal expansion coefficients of the individual phases were first determined from the experimental dilatometer curves. For austenite, martensite and pearlite, these could be read directly at the cooling rates at which they predominantly occur. For bainite, the slope of the dilatometer curve of $\lambda = 0.15$ hs after complete transformation was measured and the linear mixing rule was applied again to separate the pure thermal expansion coefficient.

Initially, the martensitic transformation behavior is fitted by establishing the onset of phase growth, which occurs when both the temperature falls below the martensite start temperature and while the austenite phase is available for the transformation. The transformation rate of martensite is governed by the variable α in equation 4.1, which is determined via parameter fitting. Additionally, the transformation strain of martensite is determined by a linear function

of temperature. Figure 4.14 (a) exhibits excellent agreement between the experimental and simulation curves, indicating successful determination of the three parameters governing the martensitic transformation. The top part of Figure 4.14 shows the corresponding temperature dependent evolution of the respective phase fractions. Given the exponential nature of the process, it can be observed that 80% of the martensite phase is formed upon reaching the 200 °C limit. In comparison to the cooling curve depicted in the CCT diagram, this finding suggests that the time required to reach the aforementioned point is less than 2 seconds, while the complete transformation process takes less than 10 seconds.

The transformation strains for both pearlite and bainite are determined based on the experimental curves and incorporated into the py routine. In this step, the optimizer assesses the entire dilatometer curve for all cooling rates, instead of individual points, which presents a considerable challenge. Moreover, the calculation for pearlite and bainite must be performed concurrently, doubling the number of parameters to be optimized. It should be noted that all newly formed phases must originate from the austenite phase, and any pre-existing transformation in the austenite will impact the transformation rate of all other phases.

In addition, the simulation results agree well with the experimental data for the lower cooling rate of $\lambda = 1.25$ hs. The transformation strain is accurately predicted, but the onset of pearlite phase growth occurs much faster than in the experiment, which is evidenced by the sharp edge at the pearlite start temperature of 685 °C. The phase fraction evolution depicted in Figure 4.14 (b) indicates an abrupt transformation process, but it should be noted that the transformation is taking place within a narrow temperature range at a cooling rate that is several orders of magnitude lower than that necessary to form martensite. Therefore, while the transformation may appear rapid, in a diagram with the temperature scale as abscissa it is moderately slow in terms of its temporal evolution.

A comprehensive summary of all the values obtained in this study is provided in subsection 4.7.4.

4.7.2 Hot Tensile Tests

For the hot tensile tests, specimens were prepared for the respective phase and temperature level following the method outlined in Subsection 4.4.2. The standard LCF specimens were manufactured from the rim of the railway wheel and heat treated to obtain the desired phase compositions. The manufacturing was successful for all specimens, except for those requiring the highest cooling rates of $\lambda = 0.013$ hs. Reaching such cooling rates is at the limits of controllable conditions, and quenching these specimens leads to distortions. To rectify these distortions, the affected specimens were turned on a lathe to attain a smaller dimension, ultimately regaining a perfectly axially symmetrical geometry that is well suited for the combined tension-compression tests.

The hot tensile tests were conducted as per the planned protocol, where each specimen was subjected to a 1% tensile strain, followed by 1% compression, and again to failure under tension. The stress-strain diagrams obtained from these tests show hysteresis loops as expected. All tests were carried out in duplicate to identify and eliminate any measurement outliers. The

results of these tests show a high level of agreement, with the curves aligning almost perfectly within the range of $\pm 1\%$.

At both room temperature and 150 °C, pearlitic specimens show a pronounced yield strength, which is illustrated by the raw data presented in Figure 4.15 in a dark green color. It can be seen that there is always a slight drop in stress during the strain controlled loading. This effect is particularly strong when the yield point is reached for the first time.

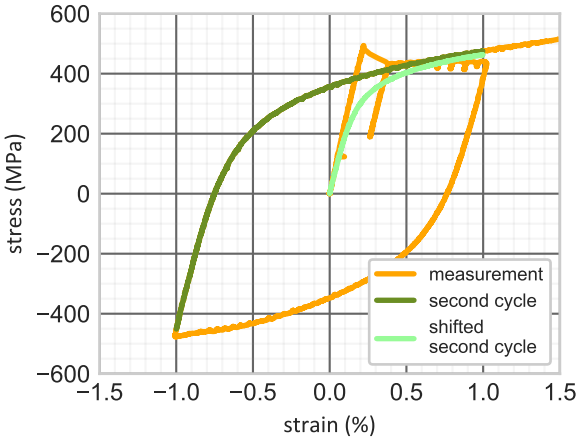
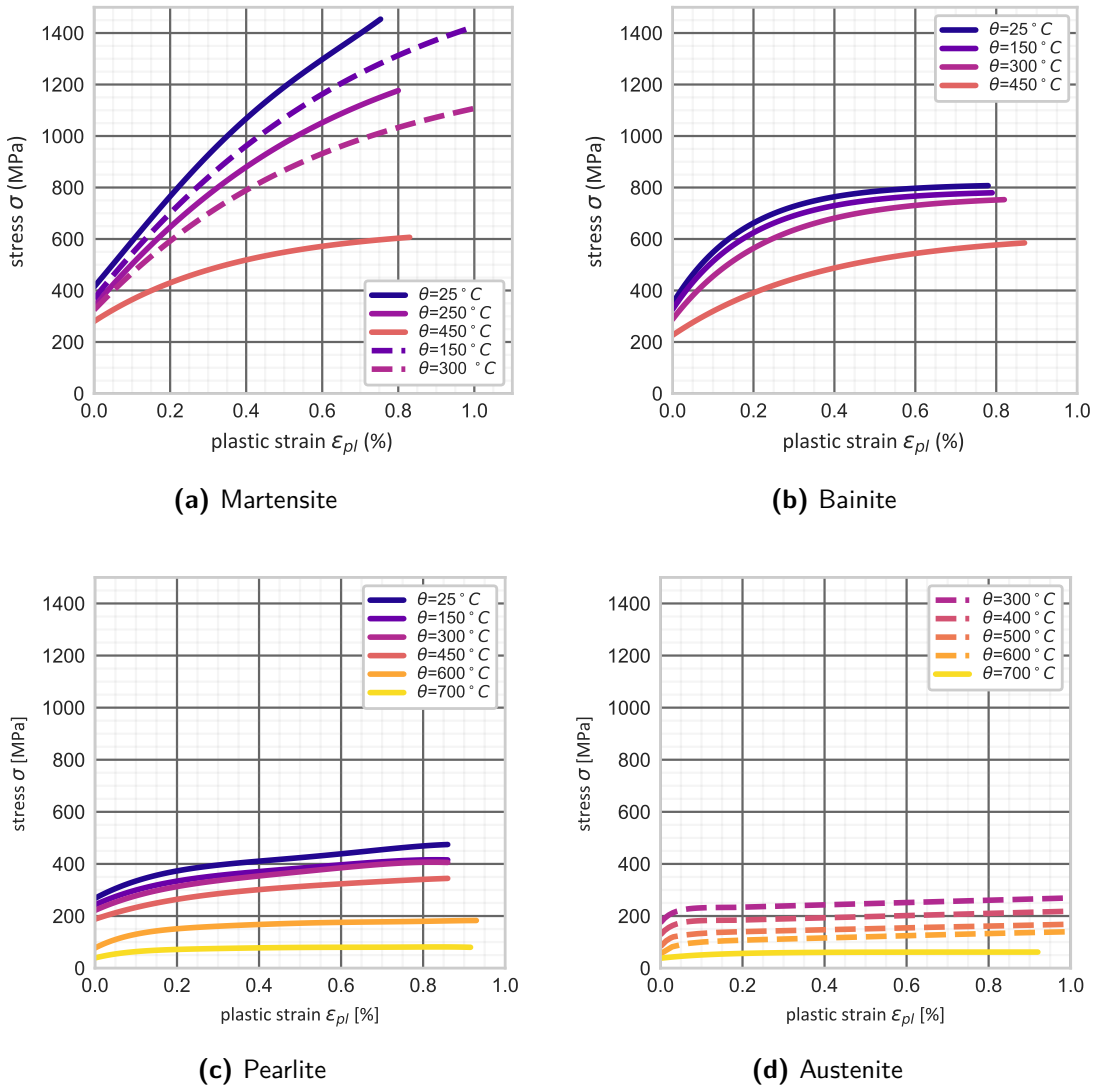


Figure 4.15: Processing of measurement data with pronounced yield strength.

To avoid having to model this effect, but still be able to use the measurements for simulation purposes, the Masing rule is applied as described in Subsection 4.5.2. The second half of the hysteresis loop, depicted in olive, is shifted to the origin and reduced by half. The outcome, displayed in pale green, is then separated into elastic and plastic components, enabling the utilization of these measurements for simulation purposes. This approach has been employed to process all data sets featuring pronounced yield strength. For the remaining measurements, solely the elastic and plastic components were separated, and the results were thinned and smoothed. A comprehensive summary of all the phase and temperature dependent flow curves can be found in Figure 4.16.

The 4 plots in Figure 4.16 show that the strength of ER7 decreases as expected from martensite through bainite and pearlite to austenite. The flow curves of martensite and pearlite differ by about a factor of 3, which is also a reason why the linear mixing rule used to calculate the bainite flow curve failed. The bainite flow curves shown here are those determined inversely using the RVE. The martensite and austenite data sets have another peculiarity. The dashed curves are linearly interpolated values since no measurements were taken at these temperature levels. In the case of austenite the entries marked with dashed lines are extrapolated values based on measurements found in literature.

In the process of modeling the bainite flow curves using the RVE, a discrepancy was found between the experimental and simulated results. To accurately represent the mixed structure of the sample, each phase is assigned a corresponding number of elements in the RVE. The flow curves of martensite and pearlite were obtained from the measurements, but a reliable estimation for bainite had to be made. The simulation result showed that the experimental



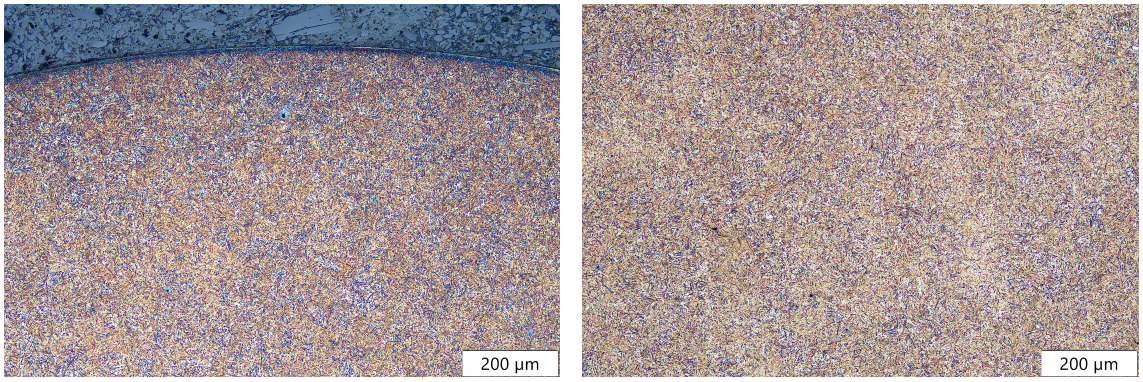
(a) Martensite **(b)** Bainite

(c) Pearlite **(d)** Austenite

Figure 4.16: Temperature dependent plastic material behavior of single phases of ER7 steel.

curve had a higher strength than the simulated curve. Theoretically, increasing the strength of pure bainite would result in perfect agreement between experiment and simulation. However, this approach would lead to a higher strength for bainite than martensite, which is not plausible. Therefore, the closest bainite flow curves to the experimental results were chosen for the analysis, resulting in a reasonable agreement between experiment and simulation. It is worth noting that Figure 4.6 does not show perfect agreement between the two curves due to this limitation.

Another potential cause of error for the determination of the bainite flow curves could be the manufacturing of the pure martensitic specimens. They were cooled at the maximum possible rate, which still may have been too slow to produce a 100% pure martensite specimen. If the interior of the specimen cools more slowly than the areas near the surface, this will lead to an inhomogeneous phase distribution. To test this hypothesis, a microsection of the martensitic specimen was prepared and examined under a light microscope. Figure 4.17 illustrates a comparison of the center of the specimen with the edge region, showing an



(a) Edge **(b)** Center

Figure 4.17: Microsections on two distinct positions of a heat treated specimen. Based on the conformity of the microsections both at the edge (a) and in the center (b) of the specimen, a uniform distribution of martensite over the entire specimen cross-section can be assumed.

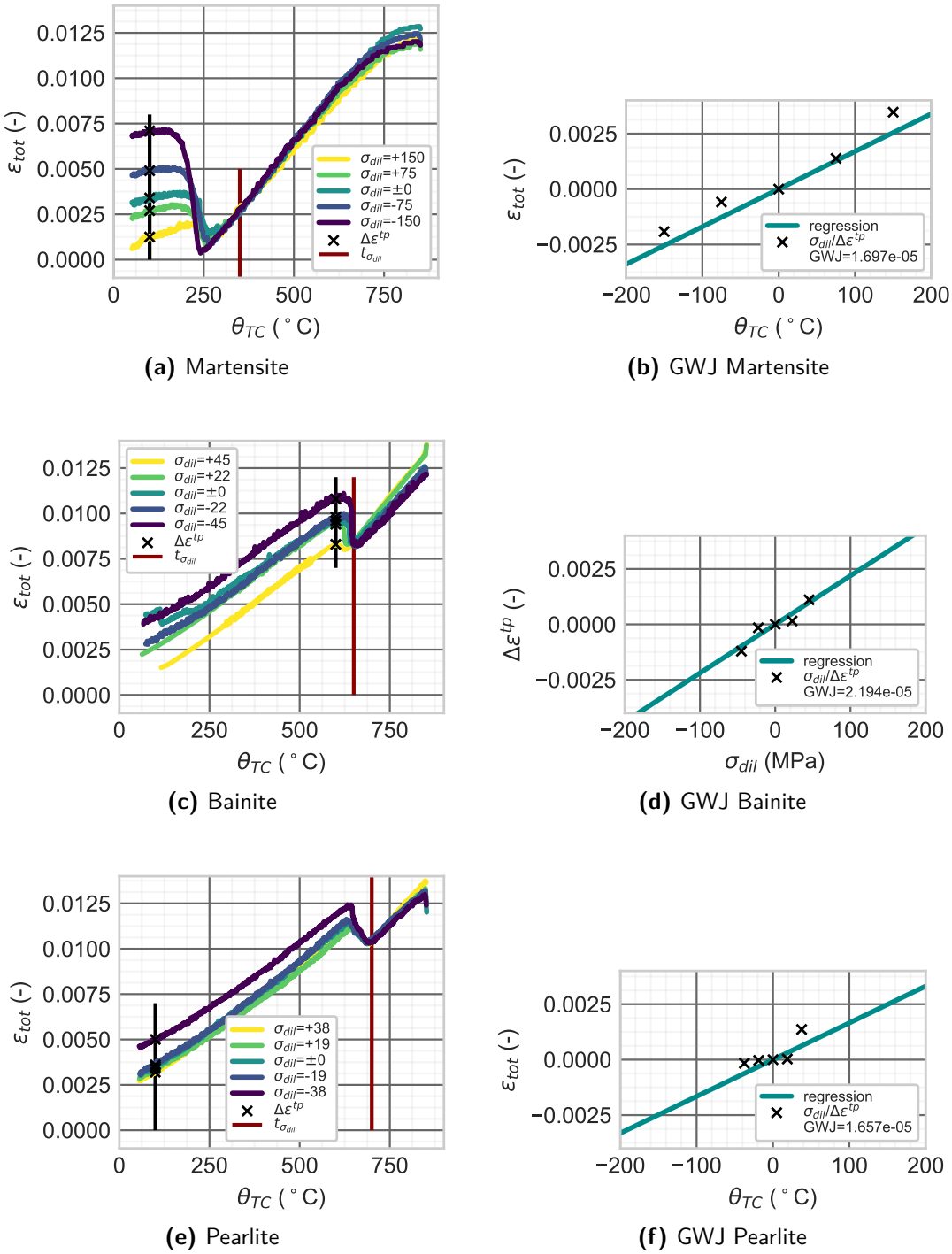
absolutely homogeneous microstructure over the entire specimen cross-section, confirming that the microstructure is pure martensite. Therefore, it is concluded that the inhomogeneous martensite sample assumption can be rejected and the bainite flow curves are determined as discussed earlier.

4.7.3 Deformation Dilatometry

The deformation dilatometry tests were conducted using the DSI Gleeble 3800 thermomechanical forming simulator. The specimens used in these tests were cylindrical in shape, and were subjected to different cooling rates and external loads as defined in the experimental design of Table 4.4 in Subsection 4.4.3. The change in length of the specimen was measured perpendicular to the loading direction by means of a laser, while temperature was monitored through a thermocouple affixed to the specimens. Force-controlled load application was carried out, taking into account the material's thermal expansion to convert the required stress into the corresponding force.

The plots on the left side of Figure 4.18 show the results of the dilatometer tests. They each consist of a reference measurement without additional load and then two measurements each under tension and compression. Note that the signal was recorded perpendicular to the load, and thus experiences negative strain under tension and positive strain under compression.

In the dilatometry plots of Figure 4.18, the vertical red line marks the time of load application. Note that for the martensite measurement with the highest cooling rate the time of load application is not immediately before the transformation temperature due to the gradual load application of the system. Instead, a small safety margin is provided from the transformation temperature. At the high cooling rate, it was crucial to ensure that the entire load was applied to the specimen before transformation occurred. In the subsequent measurements, sufficient time was available for full load application. The vertical black line serves as a reference for the measurement of the total strain ε, which comprises the elastic ε^{el}, plastic ε^{pl}, transformation expansion ε^{tv}, thermal ε^{θ}, and TRIP components ε^{tp}. The plastic, transformation expansion and thermal components are already included in the reference measurement. Thus, the



(a) Martensite **(b)** GWJ Martensite

(c) Bainite **(d)** GWJ Bainite

(e) Pearlite **(f)** GWJ Pearlite

Figure 4.18: Determination of the GWJ parameter for multiple solid phase transformations of ER7 steel using dilatometry with additional load application during cooling.

TRIP strain ε^{tp} can be determined directly by subtracting the elastic portion resulting from the applied load.

These TRIP strains obtained are then plotted as a function of applied stress in the right-hand plots of Figure 4.18, where the slope of each regression line represents the GWJ transformation coefficient K_i being sought.

4.7.4 ER7 Material Model for the Implementation in a Finite Element Software

For a better overview the Table 4.8 in this subsection presents a summary of all the material parameters required to model the heat treatment process of a railway wheel made of ER7.

parameters		austenite	martensite	bainite	pearlite
thermal expansion					
α_{th}	$10^{-5} \cdot \mathrm{K}^{-1}$	2.1818	0.8824	1.5035	$1.43 + 0.0002 \cdot \theta$
transformation expansion					
ε^{tv}	$1 \cdot 10^{-2}$	-	$0.95 + 7.978 \cdot 10^{-4} \cdot \theta$	0.4696	0.3100
displacive transformation kinetics					
θ_{MS}	°C	-	290	-	-
α	-	-	0.02	-	-
diffusive transformation kinetics					
α_n	s^{-1}	-	-	0.9202	1.0053
θ^*	°C	-	-	495	565
B	°C	-	-	255	245
A_1	m J $\sqrt{\mathrm{K}}$	-	-	6.8769	6.3636
θ_0	°C	-	-	825	955
A_3	$\mathrm{J}^3\sqrt{\mathrm{K}}$	-	-	0.0003	0.0003
A_5	-	-	-	1.9	1.8
B_1	°C	-	-	17	100
γ	-	-	-	0.2	0.2
n	-	-	-	0.03	0.03
transformation induced plasticity (GWJ)					
K_i	MPa^{-1}	-	$1.697 \cdot 10^{-5}$	$2.194 \cdot 10^{-5}$	$1.657 \cdot 10^{-5}$

Table 4.8: Required material parameters to model the heat treatment process of a railway wheel made of ER7.

4.7.5 Process Model for Heat Treatment of Wheels

This subsection describes the results of the process model described in Subsection 4.6.2. The following topics are addressed: (1) the calibration of the heat transfer coefficient on the water-quenched tread of the railway wheel, (2) the presentation and discussion of the phase distribution in the wheel, (3) and a stress analysis of this process.

Heat transfer coefficient

Figure 4.19 displays the temperature distribution of the instrumented wheel during heat treatment. The upper section of the figure depicts a close-up of the temperature evolution for a point located close to the surface during the quenching. The lower left section portrays the temperature profiles of all the instrumented points throughout the entire heat treatment

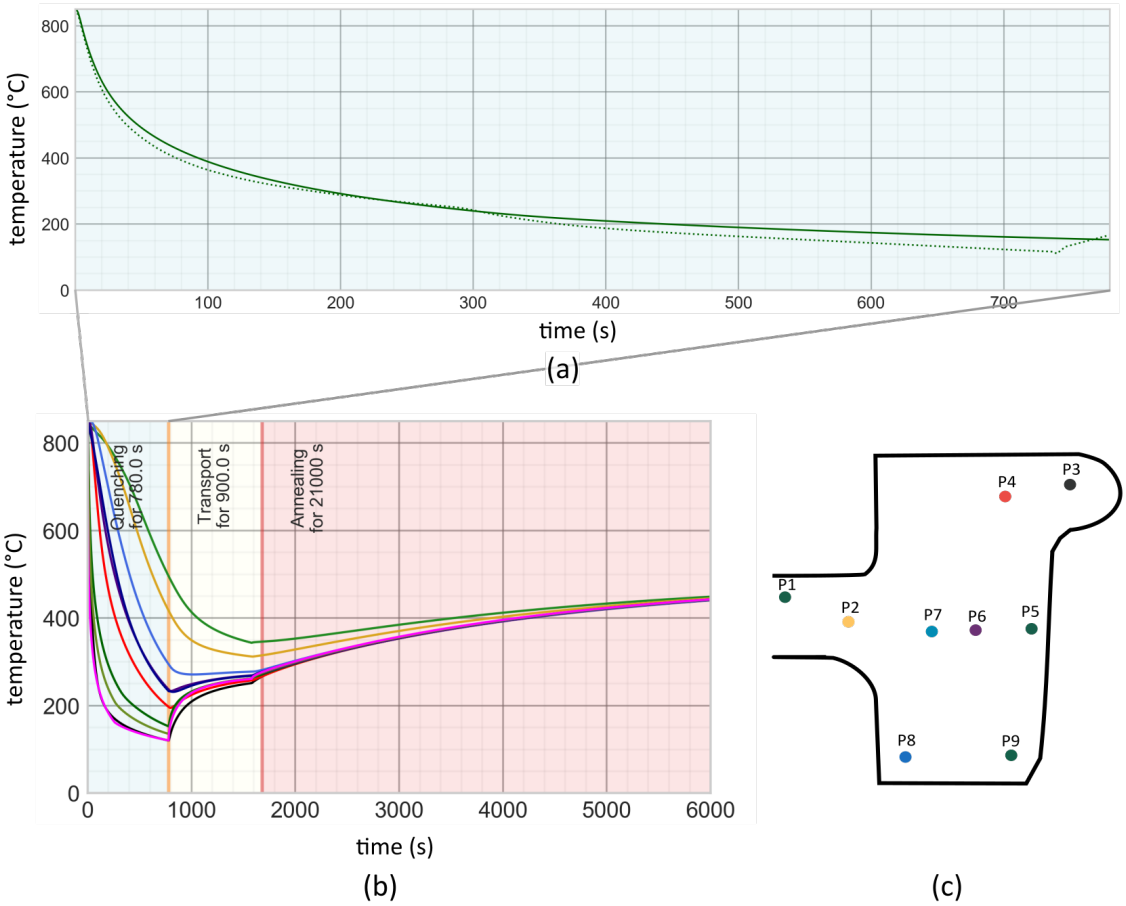


Figure 4.19: Measurement of temperature at specific positions in the railway wheel during heat treatment and comparison with results from simulation.

process. On the lower right, the positions of the points where the temperature was measured in the experiment and analyzed in the simulation are presented for an additional orientation.

The top part of Figure 4.19 illustrates the comparison between the temperature evolution obtained from measurements and simulations. A point near the surface of the instrumented wheel was chosen as an example, as the heat transfer at the tread surface has the greatest impact at this point. The graph displays an almost perfect agreement between the temperature curve obtained from measurement and the one obtained from simulation. This confirms the choice of the temperature dependent heat transfer coefficient from Table 4.7. This is not surprising since this agreement was exactly the target value for the calibration of the heat transfer coefficient. Nevertheless, this agreement shows exactly how accurate the heat transfer coefficient was determined. For all other points where temperature measurements were taken, the agreement is just as good, confirming both the choice of heat transfer coefficients on the remaining surfaces of the wheel and the temperature-dependent thermal conductivity of the material used as an invariant quantity in the calibration.

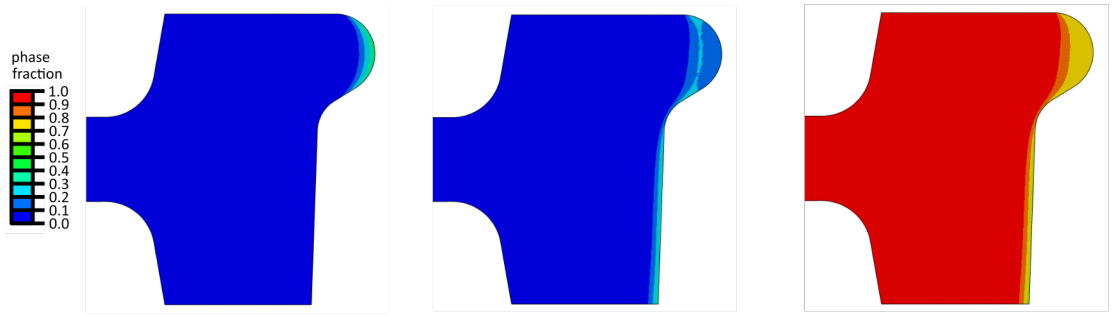
In the lower left part of the Figure 4.19 shows the temperature curves at different points in the wheel. Here only the simulation results of the temperature curves are shown. This

is because at this resolution, the simulations and the measurements coincide perfectly. In contrast to the temperature drop during quenching, it can be seen that the heat flux is already reversed at the point where transport to the tempering furnace begins (shaded in yellow), and most of the points heat up again. This happens even though heat is still being dissipated over the entire surface of the wheel. This result shows that the heat stored in the wheel prevails over the heat transferred to the air. Only the points farthest from the surface cool down further during the transport. The heating in the tempering furnace (shaded in red) is moderate and takes a relatively long time to completely heat the wheel.

The slope of the temperature profiles during quenching can be used to estimate the cooling rates required for the experiments to determine the phase transformation kinetics. The points used for the evaluation here reach a maximum cooling rate of 50 K/s (corresponding to $\lambda = 0.06$), but if points are evaluated directly on the surface, the cooling rate reaches values up to 300 K/s (corresponding to $\lambda = 0.01$). To consider phase transformations at this positions and consequently the resulting local property changes, the experiments for the material model thus must also include these highest cooling rates.

Solid phase distribution

With the material model developed in Subsection 4.5.1, it is feasible to analyze the phase composition during heat treatment. Figure 4.20 illustrates the distribution of the martensitic, bainitic, and pearlitic phases after the heat treatment process and Figure 4.21 shows the distribution after the final machining to the required size of the rim of the railway wheel. The proportion of the phase fraction is presented as a gradient ranging from 0% (in blue) to 100% (in red).

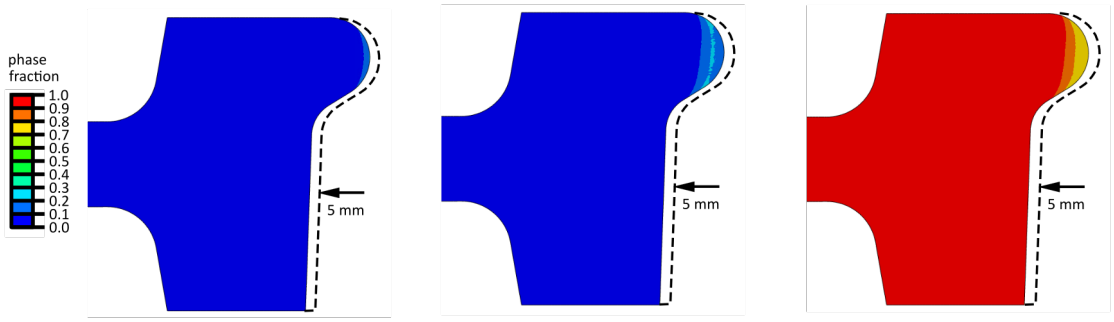


(a) Martensite **(b)** Bainite **(c)** Pearlite

Figure 4.20: Distribution of individual phases in a railway wheel after the heat treatment.

It is apparent that a small amount of martensite and bainite is formed during the heat treatment. Instead, a pearlitic microstructure is dominant throughout the wheel. Martensite and bainite are present only in a thin layer directly on the tread of the wheel.

During the modeling and meshing of the wheel, various partitions were defined to control mesh fineness at specific parts of the wheel and enable model features like element removal, as it was already used in the extension of the model of the wheelset axle described in Subsection 3.4.8. These partitions allow for a very fine mesh near the tread area. However, it



(a) Martensite **(b)** Bainite **(c)** Pearlite

Figure 4.21: Distribution of individual phases in a railway wheel after final machining.

is possible that the mesh fineness plays a role in the low amount of martensite observed. To investigate this, simulations were performed with a mesh twice as fine, but the same phase distribution was obtained, indicating that the selected elements were adequate to reproduce the temperature gradients accurately.

Figures 4.20 and 4.21 display the distribution of the phase fractions in the wheel before and after final machining, with partitioning used to delete elements at the relevant positions. The machining process removes approximately 5 mm of material relative to the original wheel radius, with the ultimate goal of achieving a final diameter of 850 mm. The majority of the machining occurs on the tread of the wheel and removes the higher strength phase components in this region, resulting in a fully pearlitic microstructure throughout the wheel.

The low amount of martensite and bainite phases formed during heat treatment was unexpected, given initial calculations that suggested a deeper region of high martensite content directly at the tread. However, by accurately modeling the heat transfer using temperature measurements of the instrumented wheel, the cooling rate was found to be reduced enough to almost completely suppress the formation of these higher-strength phases. To confirm the microstructure formed at and just below the surface, a fully fabricated wheel was cut and metallographic sections were taken. This investigation verified the results of the FE model and revealed a pearlitic-ferritic microstructure just beneath the surface of the tread, as depicted in the optical microscope image of Figure 4.22.

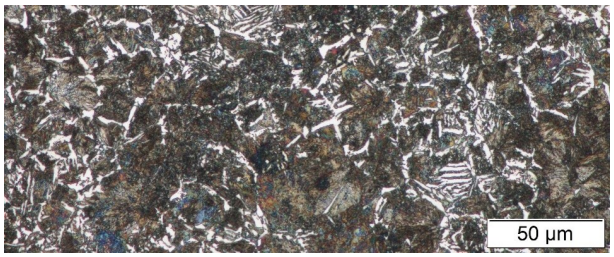
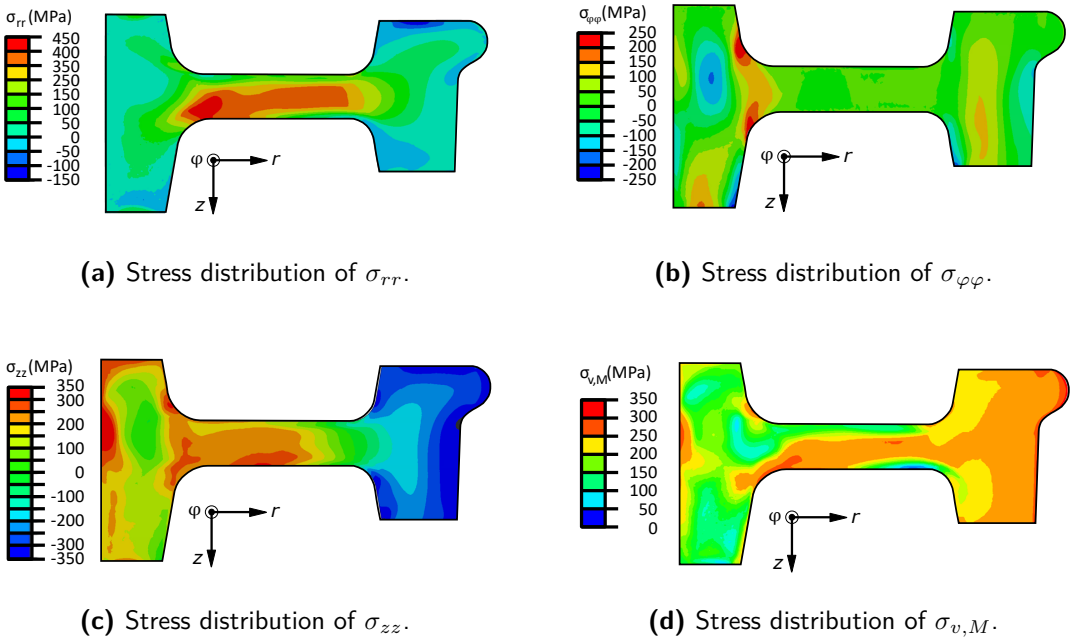


Figure 4.22: Optical microscope image of the microstructure just beneath the tread of a fully fabricated railway wheel made of ER7.

Stress analysis

The previously discussed results focus primarily on the thermal effects of the analysis. However, the ultimate goal of this model is to assess the stress state, which is strongly influenced by the thermal behavior of the whole system. Therefore, it is crucial to also consider the mechanical aspects of the problem. Figure 4.23 shows contour plots of the distributions of individual stress components of the railway wheel after the heat treatment.



(a) Stress distribution of σ_{rr}.

(b) Stress distribution of $\sigma_{\varphi\varphi}$.

(c) Stress distribution of σ_{zz}.

(d) Stress distribution of $\sigma_{v,M}$.

Figure 4.23: Distribution of residual stresses in a railway wheel after the heat treatment.

In a theoretical ideal scenario, residual stress formation and evolution in a thermally stressed cylinder proceed as follows. During the cooling and contraction of the surface, the core remains at a relatively high temperature, leading to the development of tensile stresses in the outer shell and compressive stresses in the core. As the core remains at high temperatures, its yield point is lower than that of the cooled shell, leading to compression and subsequently to plastic compressive stresses. Further cooling causes the core to contract, but the still non-plastically deformed outer shell inhibits this expansion, resulting in a stress reversal. As a consequence, the outer shell experiences compressive stresses while the core reverses to tensile stresses. The observed trend of stress development in the railway wheel aligns with the theoretical ideal case of residual stress formation in a thermally stressed cylinder.

The results of the finite element analysis (FEA) of the railway wheel show the development of compressive stresses on the tread in both radial and axial directions, with the highest magnitude occurring directly at the transition of the flange to the straight part of the tread. Although the axial compressive stresses are in the range of 350 MPa and may further increase during operation due to superposition with the rolling contact stresses, they do not pose a threat to operation. These compressive stresses have positive crack closure effects and counteract the crack growth of so called head checks, a typical damage mode of rolling contact. In the tangential direction the development of any stresses is minimal. Only tensile stresses

with peaks up to 250 MPa appear at the transition from the hub to the hub seat in the radii. On the other hand, the tensile stresses in radial and axial directions concentrate also around the hub of the railway wheel, with the radii at the transition to the hub seat exhibiting the highest values of up to 400 MPa in the radial direction and in the range of 300 MPa in the axial direction. The contour plot of the von Mises equivalent stresses in the lower right plot of Figure 4.23 confirms these results and shows stress maxima at the same radii. Furthermore, stresses at the tread represent another maximum.

The analysis of the stress distribution in the railway wheel reveals that the two radii at the transition between the hub and the hub seat are the most critical areas in terms of safety. The overlapping tensile stress components from several directions at these points could initiate cracks and accelerate crack growth. To enhance the safety of these components, measures must be taken to reduce these tensile stresses. The process model developed in this work can be used to simulate different heat treatment routes that can further optimize the currently used heat treatment process with respect to these local stresses. Alternatively, subsequent localized heat treatments can be applied to treat only the high-risk areas. These approaches can help to mitigate the risks associated with these stress concentrations and improve the overall safety of the railway wheel.

4.7.6 Block Braking of Railway Wheels

The analysis of block brakes in this work focuses specifically on the curved web of the railway wheel. At the start of the braking process, the wheel already has a residual stress state resulting from the preceding heat treatment. Figure 4.24 displays the web section chosen for further analysis, depicting the distribution of σ_{rr} stresses following the heat treatment. This stress state serves as the initial condition for the braking simulation. As shown in the figure, tensile stresses tend to manifest on the inside radii of the web, while compressive stresses tend to occur on the outside radii. Notably, these stress values are already relatively high. However, their impact remains benign as they counterbalance the stresses induced by the rolling of the wheel under the weight of the train in service. This arises from the fact that the displacement in r-direction induces compressive stresses at the inside radii and tensile stresses at the outside radii.

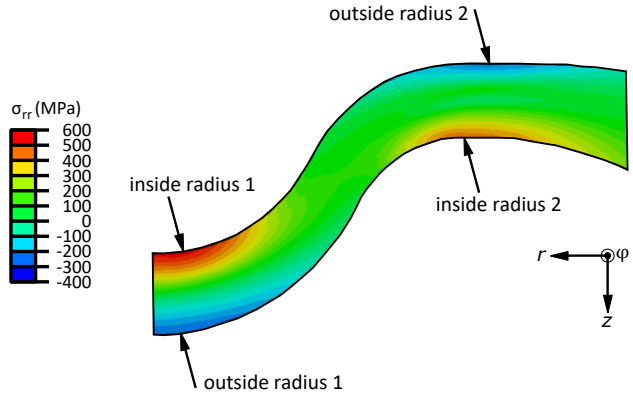
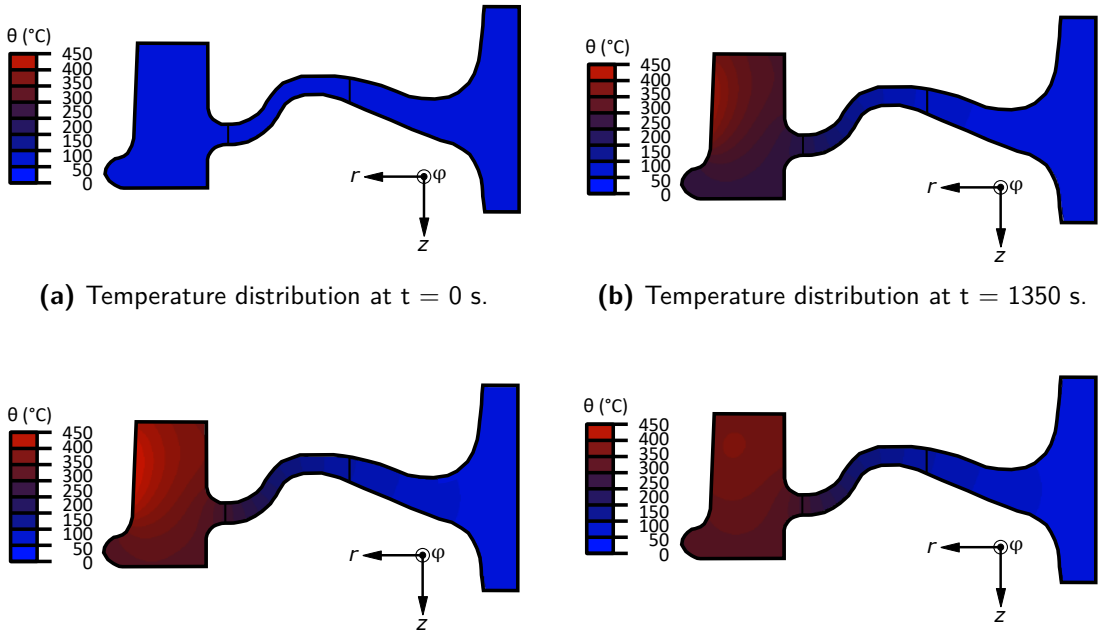


Figure 4.24: Distribution of residual stresses in the web of a SURA railway wheel after the heat treatment. Figure taken from [114].

Figure 4.25 serves to illustrate the heat input and the temperature evolution at important time points during the process. Initially, the wheel is at room temperature. During block braking, heat is introduced into the wheel through the tread, gradually increasing until reaching its peak after 2700 s. At this point, the braking process ends, and the wheel subsequently cools across all surfaces. The depth of heat penetration into the wheel can be clearly observed. Despite the rim's substantial mass, the extreme braking process leads to a significant temperature rise, particularly at the surface. Moreover, the web exhibits a comparatively moderate temperature increase, which holds significance for the subsequent stress analysis.



(a) Temperature distribution at t = 0 s.

(b) Temperature distribution at t = 1350 s.

(c) Temperature distribution at t = 2700 s.

(d) Temperature distribution at t = 2800 s.

Figure 4.25: Temperature distribution in a railway wheel at various stages of the block braking simulation. Figure taken from [114].

Figure 4.26 shows the temperature evolution at the wheel's tread. The red x's indicate temperatures measured during the experimental setup, serving as target values for calibrating the heat transfer coefficients and heat input of the block braking. It is evident that the temperature after braking aligns precisely with the target value, and the temperature after cooling exhibits minimal deviation. These findings validate the selected boundary conditions of the process model.

Subsequently, the stress analysis focuses on the web, specifically the point displaying the highest tensile stress in Figure 4.24. This point serves as a representative position for the analysis due to its exposure to the most significant stresses and stress redistributions. Figure 4.27 illustrates the stress-strain hysteresis of the σ_{rr} stress component over 30 braking cycles, each consisting of 45 minutes of brake-induced heat input followed by 4 hours of cooling. Initially, the stress is tensile, with an approximate magnitude of 550 MPa. Throughout the braking process, the stress continues to rise, reaching a peak value of around 700 MPa. Only during the cooling phase does the stress at this point diminish, eventually transitioning into the

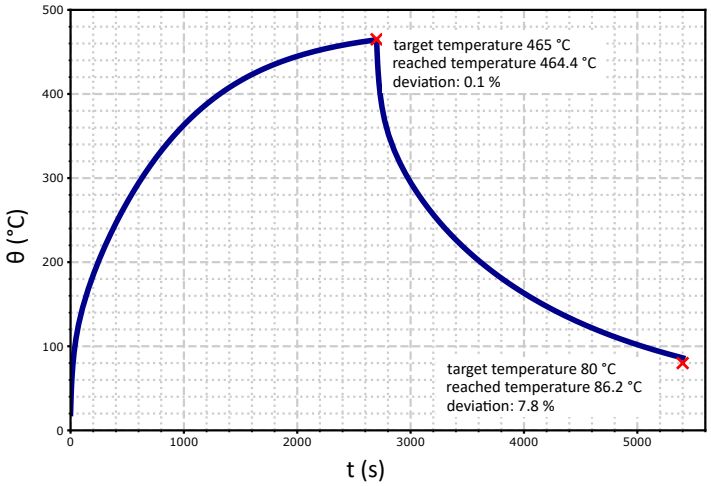


Figure 4.26: Temperature evolution at the tread of a railway wheel during block braking. Figure taken from [114].

compressive range as the wheel cools. This pattern is consistent over the subsequent braking cycles, where the stresses in the web change sign during each cycle. As long as the tread, and thus the wheel rim is subjected to heat, the tensile stresses rise in the web. At the end of each braking (at the maximum temperature), the web exhibits a tensile stress maximum. When the wheel cools again the stresses transition into the compressive regime with notable high values. As the number of cycles increases, the overall hysteresis stabilizes, featuring a stress ratio of -1 and maximum stress values ranging between +400 and -400 MPa.

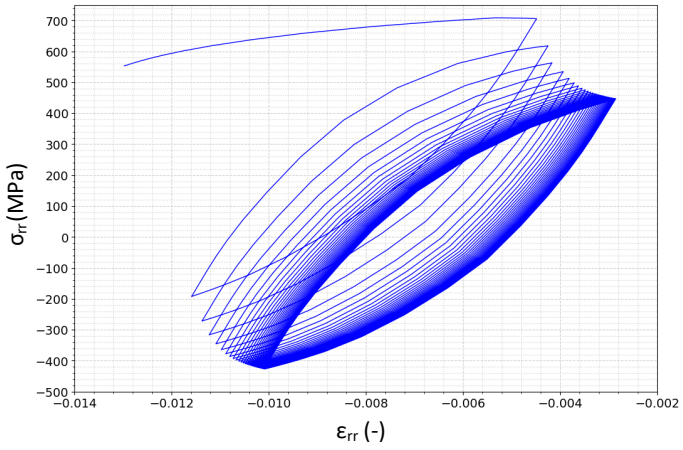


Figure 4.27: Stress-strain evolution at a critical position of the web of a railway wheel during block braking. Figure taken from [114].

Similar behavior is observed at other points on the web, albeit with lower stress values. These positions also undergo a sign change in stresses, where points initially exhibiting compressive stresses after heat treatment transition into the tensile regime. It is important to note that the simulated Gotthard ramp scenario represents an extreme and extraordinary thermal load for a railway wheel. Nevertheless, the results demonstrate that such a braking process

induces significant changes in the stress state of the railway wheel. Interestingly, the temperature increase in the web is moderate compared to the substantial impact on the stresses at these positions. This phenomenon is likely explained by the expansion of the wheel's rim, which absorbs the majority of the heat.

Stress redistributions yield further consequences. Localized property changes can be employed to design components that for example counteract the stresses experienced in service. However, in the case of block brakes, it is revealed that the stresses introduced during heat treatment are not merely eliminated but actually reversed. Consequently, additional surface treatments of the rim would lose its effectiveness following the initial high heat input of block braking in service.

4.8 Conclusions

This part of the thesis focuses on the development of a process model of the heat treatment process of railway wheels. The model accurately simulates the heat transfer phenomena that occur during various heating and cooling conditions. It successfully accounts for complex physical processes, including quenching at temperature differences of up to 800 °C, which result in the formation of high-strength phases on the wheel's surface. The material model developed for this process model is a key component of this research, as it accurately predicts the formation of different phases in the steel and considers other complex effect such as TRIP.

The development of such a material model necessitates careful preparation of the experimental work. The rapid transformation behavior of the ER7 material posed a significant challenge, requiring meticulous sample preparation and pushing the limits of the test rigs employed. However, the resulting material model is highly versatile and capable of accommodating more intricate heat treatment processes beyond those investigated in this work. An initial step in this direction has been taken by extending the model to incorporate block braking calculations. The findings highlight the crucial role of thermal stress in railway wheel performance and emphasize the need for continued research in this area.

5 Summary

The primary aim of this thesis is to develop comprehensive process models for the manufacturing of railway components. These models are designed to encompass localized property variations by both the material properties and those properties influenced by the manufacturing process itself. These localized property variations affect the overall behavior of the railway components. Specifically, for the cold rolling process employed in wheelset axle manufacturing, the objective is to accurately quantify the stress distribution within the components and identify the key parameters influencing it. Additionally, the developed model aims to enable the evaluation of the stress redistribution during service and maintenance operations. Regarding the railway wheels, a significant focus is placed on addressing the localized property changes arising from the non-uniform distribution of solid phases of steel affected by the heat treatment. Consequently, an advanced material model is developed to account for this influence and enhance the predictive capabilities of the model.

Scientific Contribution and Conclusion

The scientific contribution of this thesis is focused on the development of models for the manufacturing of railway components, with a particular emphasis on residual stresses and stress redistribution. It offers a deeper understanding of the evolution of production-induced residual stresses in these components, providing novel insights into the underlying mechanisms and influential factors during the manufacturing process of steel components. Regarding the cold rolling of wheelset axles the main attainments are:

- development of a novel, versatile and efficient finite element process model specifically designed for the cold rolling of wheelset axles
- implementation of new types of periodic boundary conditions and model features, which restore the continuity of the cold rolling process in a discontinuous reduced model geometry
- increase in accuracy for determining the residual stress profile throughout the entire component geometry of large components, such as wheelset axles
- increase in computation efficiency allows large parameter studies to be carried out, with the results then being used to create software tools, such as PRESD
- extension of the model's capabilities beyond manufacturing, as it allows for the examination of stress redistribution in cold rolled components under maintenance and in-service conditions

For the subject area of heat treatment of railway wheels, the most important findings are:

- investigation of the influence of inhomogeneously distributed solid phases in the manufacturing process of railway wheels and their impact on the overall component behavior
- development of an advanced material model specifically designed to accurately capture the unique characteristics of rapidly transforming materials like ER7 steel
- enhancement of the model's precision in representing the material's response under extreme and varying process conditions, including localized quenching during the heat treatment stage
- incorporation of multiple solid phases in the material model, encompassing a wide range of mechanical and metallurgical properties
- analysis of the significant changes in the residual stress field of the railway wheel resulting from the heat input during block braking

The outcomes of this research pave the way for enhanced manufacturing processes, improved component design, and more effective maintenance practices in the railway industry. By incorporating the developed models and insights into industry practices, manufacturers and engineers can make informed decisions to optimize the manufacturing process, improve component performance, and ensure safe and reliable operation of railway components.

Outlook

The process models developed in this work hold great potential to further advance the process optimization and component design in the railway industry. Integrating the calculated stress fields into further analyses is a crucial next step, to enable for example the development of models to predict crack propagation under corrosive conditions and facilitating the implementation of condition monitoring techniques. Continual research in this field is essential to enhance the attractiveness and modernization of rail transportation. By leveraging the insights gained from this thesis, future work can further optimize processes, improve component performance, and contribute to the advancement of the railway industry.

List of Figures

List of Tables

Bibliography

[1] Ö.-H. AG, "Zahlen, Daten, Fakten. Heute. Für morgen. Für uns.," 2020. Last visited: 2022-07-05.

[2] Umweltbundesamt GmbH, "Treibhausgase." `https://www.umweltbundesamt.at/klima/treibhausgase`, . Last visited: 2022-07-04.

[3] ÖBB-Holding-AG, "ÖBB Klimaschutzstrategie 2030," 2019. Last visited: 2022-07-05.

[4] U. Zerbst and C. Klinger, "Anmerkungen zur auslegung und zum sicheren betrieb von radsatzwellen aus der sicht von betriebsfestigkeit und bruchmechanik," in *40. Tagung des DVM-Arbeitskreises Betriebsfestigkeit-Die Betriebsfestigkeit als eine Schlüsselfunktion für die Mobilität der Zukunft*, vol. 140, pp. 69–80, 2013.

[5] F. Dunne and N. Petrinic, *Introduction to computational plasticity*. OUP Oxford, 2005.

[6] J. Besson, G. Cailletaud, J.-L. Chaboche, and S. Forest, *Non-linear mechanics of materials*, vol. 167. Springer Science & Business Media, 2009.

[7] J. Chaboche, "A review of some plasticity and viscoplasticity constitutive theories," *International Journal of Plasticity*, vol. 24, no. 10, pp. 1642–1693, 2008. Special Issue in Honor of Jean-Louis Chaboche.

[8] J.-L. Chaboche, "Constitutive equations for cyclic plasticity and cyclic viscoplasticity," *International journal of plasticity*, vol. 5, no. 3, pp. 247–302, 1989.

[9] Dassault Systèmes, "Simulia user assistance 2019." `https://help.3ds.com`, . Last visited: 2022-07-28.

[10] G. Gottstein, *Materialwissenschaft und Werkstofftechnik: Physikalische Grundlagen*. Springer-Verlag, 2013.

[11] W. Callister and D. Rethwisch, *Materials science and engineering*. Wiley, 2011.

[12] G. Totten, *Steel heat treatment: metallurgy and technologies*. CRC press, 2006.

[13] W. Bleck, *Werkstoffkunde Stahl für Studium und Praxis*. Verlag Mainz, 2010.

[14] S. Banerjee and P. Mukhopadhyay, *Phase Transformations: Examples from Titanium and Zirconium Alloys*. Elsevier, 2010.

[15] H. Dubbel and K.-H. Grote, *DUBBEL: Taschenbuch für den Maschinenbau*. Springer Berlin Heidelberg, 2007.

[16] Materials Science and Engineering Student, "Crystal structure, properties, interstitial sites, and examples." `https://msestudent.com`, . Last visited: 2023-04-25.

[17] H. Bhadeshia and R. Honeycombe, *Steels: microstructure and properties*. Butterworth-Heinemann, 2017.

[18] J. Beddoes and J. G. Parr, "Introduction to stainless steels, 3," 1999.

[19] J.-B. Leblond, G. Mottet, and J. Devaux, "A theoretical and numerical approach to the plastic behaviour of steels during phase transformations—i. derivation of general relations," *Journal of the Mechanics and Physics of Solids*, vol. 34, no. 4, pp. 395–409, 1986.

[20] G. W. Greenwood and R. Johnson, "The deformation of metals under small stresses during phase transformations," *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 283, no. 1394, pp. 403–422, 1965.

[21] T. Antretter, D. Zhang, and E. Parteder, "Modelling transformation induced plasticity–an application to heavy steel plates," *steel research international*, vol. 81, no. 8, pp. 675–680, 2010.

[22] R. Mahnken, A. Schneidt, and T. Antretter, "Macro modelling and homogenization for transformation induced plasticity of a low-alloy steel," *International Journal of Plasticity*, vol. 25, no. 2, pp. 183–204, 2009.

[23] T. Antretter, *Experimental and numerical investigations of the mechanics of martensitic transformation*. PhD thesis, University of Leoben, 2003.

[24] B. Liscic, H. M. Tensi, L. C. Canale, and G. E. Totten, *Quenching theory and technology*. CRC Press, 2010.

[25] G. E. Totten, *Handbook of residual stress and deformation of steel*. ASM international, 2002.

[26] C. Ruud, "A review of selected non-destructive methods for residual stress measurement," *NDT international*, vol. 15, no. 1, pp. 15–23, 1982.

[27] V. Dive and S. Lakade, "Recent research progress on residual stress measurement using non-destructive testing," *Materials Today: Proceedings*, vol. 47, pp. 3282–3287, 2021.

[28] B. Scholtes, *Eigenspannungen in mechanisch randschichtverformten Werkstoffzuständen: Ursachen, Ermittlung und Bewertung*. DGM Informationsgesellschaft, 1991.

[29] M. Lindgren and T. Lepistö, "Relation between residual stress and barkhausen noise in a duplex steel," *NDT & E International*, vol. 36, no. 5, pp. 279–288, 2003.

[30] S. Santa-Aho, A. Sorsa, A. Nurmikolu, and M. Vippola, "Review of railway track applications of barkhausen noise and other magnetic testing methods," *Insight-Non-Destructive Testing and Condition Monitoring*, vol. 56, no. 12, pp. 657–663, 2014.

[31] N. Rossini, M. Dassisti, K. Benyounis, and A.-G. Olabi, "Methods of measuring residual stresses in components," *Materials & Design*, vol. 35, pp. 572–588, 2012.

[32] F. Kandil, J. Lord, A. T. Fry, and P. Grant, "A review of residual stress measurement methods-a guide to technique selection.," *NPL MaterialsCentre QueensRoad*, 2001.

[33] H.-J. Schindler and P. Bertschinger, "Some steps towards automation of the crack compliance method to measure residual stress distributions," in *5th International Conference on Residual Stresses, Sweden*, pp. 682–687, 1997.

[34] H.-J. Schindler, "Experimental determination of crack closure by the cut compliance technique," *ASTM Special Technical Publication*, vol. 1343, pp. 175–190, 1999.

[35] H.-P. Gänser, "Entwicklung einer Methode zur Bestimmung von Eigenspannungen in Radsatzwellen mit Hilfe des Cut-Compliance-Verfahrens," *Interner Projektbericht Materials Center Leoben Forschung GmbH*, 2015.

[36] M. R. Hill, "The slitting method," *Practical residual stress measurement methods*, pp. 89–108, 2013.

[37] G. S. Schajer and M. B. Prime, "Use of inverse solutions for residual stress measurements," *Journal of engineering materials and technology*, vol. 128, no. 3, pp. 375–382, 2006.

[38] H. Baehr and K. Stephan, *Wärme und Stoffübertragung*. Springer-Verlag, 2008.

[39] P. Böckh and T. Wetzel, *Wärmeübertragung: Grundlagen und Praxis*. Springer-Verlag, 2018.

[40] M. Sander, *Sicherheit und Betriebsfestigkeit von Maschinen und Anlagen*. Berlin Heidelberg: Springer-Verlag, 2008.

[41] U. Zerbst, S. Beretta, G. Köhler, A. Lawton, M. Vormwald, H. T. Beier, C. Klinger, I. Černý, J. Rudlin, T. Heckel, and D. Klingbeil, "Safe life and damage tolerance aspects of railway axles - A review," *Engineering Fracture Mechanics*, vol. 98, no. 1, pp. 214–271, 2013.

[42] U. Zerbst and K. Mädler, "Bruchmechanische Bewertungskonzepte für Bahnkomponenten," *Materials Testing*, vol. 46, no. 7-8, pp. 354–362, 2004.

[43] T. Nguyen-Tajan and X. Lorang, "Euraxles–a global approach for design, production and maintenance of railway axles: Wp1–advances in fatigue load analysis and reliability assessment of railway axles," *Materialwissenschaft und Werkstofftechnik*, vol. 48, no. 7, pp. 666–686, 2017.

[44] J. Hug, V. Runzer, M. Traupe, H. Zenner, and A. Esderts, "Dauerfestigkeit von Radsatzwellen und Eisenbahnrädern," *Materials Testing*, vol. 46, no. 1-2, pp. 27–32, 2004.

[45] DIN Deutsches Institut für Normung e.V., "EN13103: Railway applications-Wheelsets and bogies-Non powered axles-Design method," 2009.

[46] DIN Deutsches Institut für Normung e.V., "EN13104: Railway applications-Wheelsets and bogies-Powered axles-Design method," 2009.

[47] DIN Deutsches Institut für Normung e.V., "EN13261: Railway applications-Wheelsets and bogies-Axles-Product requirements," 2009.

[48] Lucchini RS S.p.A, "Lucchini-lrs-axles-process," 2022.

[49] Lucchini RS S.p.A, "Lucchini-10-Hollow-Bored-Axle," .

[50] D. Regazzi, S. Cantini, S. Cervello, S. Foletti, A. Pourheidar, and S. Beretta, "Improving fatigue resistance of railway axles by cold rolling: Process optimisation and new experimental evidences," *International Journal of Fatigue*, vol. 137, p. 105603, 2020.

[51] P. Delgado, I. Cuesta, J. Alegre, and A. Díaz, "State of the art of Deep Rolling," *Precision Engineering*, vol. 46, pp. 1–10, 2016.

[52] F. Klocke and S. Mader, "Fundamentals of the deep rolling of compressor blades for turbo aircraft engines," *Steel research international*, vol. 76, no. 2-3, pp. 229–235, 2005.

[53] R. Nalla, I. Altenberger, U. Noster, G. Liu, B. Scholtes, and R. Ritchie, "On the influence of mechanical surface treatments—deep rolling and laser shock peening—on the fatigue behavior of ti–6al–4v at ambient and elevated temperatures," *Materials Science and Engineering: A*, vol. 355, no. 1-2, pp. 216–230, 2003.

[54] G. Majzoobi, K. Azadikhah, and J. Nemati, "The effects of deep rolling and shot peening on fretting fatigue resistance of Aluminum-7075-T6," *Materials Science and Engineering: A*, vol. 516, no. 1-2, pp. 235–247, 2009.

[55] F. Henning and E. Moeller, *Handbuch Leichtbau: Methoden, Werkstoffe, Fertigung*. Carl Hanser Verlag GmbH Co KG, 2020.

[56] B. Denkena, J. Dege, and C. Müller, "Beeinflussung der Randzoneneigenschaften des Werkstoffs 42CrMo4 durch einen geregelten Festwalzprozess," *HTM Journal of Heat Treatment and Materials*, vol. 61, no. 1, pp. 43–46, 2006.

[57] K. Röttger, G. Wilcke, and S. Mader, "Festwalzen–eine Technologie für effizienten Leichtbau," *Materialwissenschaft und Werkstofftechnik: Entwicklung, Fertigung, Prüfung, Eigenschaften und Anwendungen technischer Werkstoffe*, vol. 36, no. 6, pp. 270–274, 2005.

[58] J. Maierhofer, H.-P. Gänser, and R. Pippan, "Prozessmodell zum Einbringen von Eigenspannungen durch Festwalzen," *Materialwissenschaft und Werkstofftechnik*, vol. 11, no. 45, pp. 982–989, 2014.

[59] K. Choi and J. Pan, "Simulations of stress distributions in crankshaft sections under fillet rolling and bending fatigue tests," *International Journal of Fatigue*, vol. 31, no. 3, pp. 544–557, 2009.

[60] Y. Yen, P. Sartkulvanich, and T. Altan, "Finite element modeling of roller burnishing process," *CIRP annals*, vol. 54, no. 1, pp. 237–240, 2005.

[61] D. Mombeini and A. Atrian, "Investigation of deep cold rolling effects on the bending fatigue of brass C38500," *Latin American Journal of Solids and Structures*, vol. 15, 2018.

[62] F. Klocke, V. Bäcker, A. Timmer, and H. Wegner, "Innovative FE-analysis of the roller burnishing process for different geometries," *ed. E Oñate, Barcelona: CIMNE*, pp. 1–4, 2009.

[63] A. Manouchehrifar and K. Alasvand, "Finite element simulation of deep rolling and evaluate the influence of parameters on residual stress," *Recent researches in applied mechanics*, pp. 121–127, 2009.

[64] F. Mohammadi, R. Sedaghati, and A. Bonakdar, "Finite element analysis and design optimization of low plasticity burnishing process," *The International Journal of Advanced Manufacturing Technology*, vol. 70, no. 5, pp. 1337–1354, 2014.

[65] Z. Liu, C. Fu, M. Sealy, and Y. Guo, "Prediction and analysis of residual stress and deflections of almen strip by burnishing," *Production Engineering*, vol. 11, no. 3, pp. 265–274, 2017.

[66] M. Uddin, C. Hall, R. Hooper, E. Charrault, P. Murphy, and V. Santos, "Finite element analysis of surface integrity in deep ball-burnishing of a biodegradable AZ31B Mg alloy," *Metals*, vol. 8, no. 2, p. 136, 2018.

[67] A. Hadadian and R. Sedaghati, "Investigation on thermal relaxation of residual stresses induced in deep cold rolling of Ti–6Al–4V alloy," *The International Journal of Advanced Manufacturing Technology*, vol. 100, no. 1, pp. 877–893, 2019.

[68] S. Hassani-Gangaraj, M. Carboni, and M. Guagliano, "Finite element approach toward an advanced understanding of deep rolling induced residual stresses, and an application to railway axles," *Materials & Design*, vol. 83, pp. 689–703, 2015.

[69] P. Balland, L. Tabourot, F. Degre, and V. Moreau, "An investigation of the mechanics of roller burnishing through finite element simulation and experiments," *International Journal of Machine tools and manufacture*, vol. 65, pp. 29–36, 2013.

[70] M. Sayahi, S. Sghaier, and H. Belhadjsalah, "Finite element analysis of ball burnishing process: comparisons between numerical results and experiments," *The International Journal of Advanced Manufacturing Technology*, vol. 67, no. 5, pp. 1665–1673, 2013.

[71] J. Perenda, J. Trajkovski, A. Žerovnik, and I. Prebil, "Residual stresses after deep rolling of a torsion bar made from high strength steel," *Journal of Materials Processing Technology*, vol. 218, pp. 89–98, 2015.

[72] G. Majzoobi, F. Zare Jouneghani, and E. Khademi, "Experimental and numerical studies on the effect of deep rolling on bending fretting fatigue resistance of Al7075," *The International Journal of Advanced Manufacturing Technology*, vol. 82, no. 9, pp. 2137–2148, 2016.

[73] J. Perenda, J. Trajkovski, A. Žerovnik, and I. Prebil, "Modeling and experimental validation of the surface residual stresses induced by deep rolling and presetting of a torsion bar," *International Journal of Material Forming*, vol. 9, no. 4, pp. 435–448, 2016.

[74] F. Klocke, V. Bäcker, H. Wegner, and M. Zimmermann, "Finite element analysis of the roller burnishing process for fatigue resistance increase of engine components," *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 225, no. 1, pp. 2–11, 2011.

[75] V. Bäcker, F. Klocke, H. Wegner, A. Timmer, R. Grzhibovskis, and S. Rjasanow, "Analysis of the deep rolling process on turbine blades using the FEM/BEM-coupling," in *IOP Conference Series: Materials Science and Engineering*, vol. 10, p. 012134, IOP Publishing, 2010.

[76] L. Fonseca, A. Cantisano, and A. Faria, "Numerical modelling of deep rolling influence over crankshaft bending and correlation with fatigue behaviour," *Fatigue & Fracture of Engineering Materials & Structures*, vol. 43, no. 4, pp. 672–683, 2020.

[77] M. Traupe and A. Landaberea, "EURAXLES–A global approach for design, production and maintenance of railway axles: WP2–development of numerical models for the analysis of railway axles," *Materialwissenschaft und Werkstofftechnik*, vol. 48, pp. 687–698, 2017.

[78] D. Trauth, F. Klocke, P. Mattfeld, and A. Klink, "Time-efficient prediction of the surface layer state after deep rolling using similarity mechanics approach," *Procedia CIRP*, vol. 9, pp. 29–34, 2013.

[79] G. Küffner, "Achsentausch beim ICE: Der Eisenbahn-Stahl kehrt zurück," *Frankfurter Allgemeine*, 2010. URL: https://www.faz.net/-gyg-16g93, Last visited: 2022-07-20.

[80] K. A. Meyer, R. Skrypnyk, and M. Pletz, "Efficient 3d finite element modeling of cyclic elasto-plastic rolling contact," *Tribology International*, vol. 161, p. 107053, 2021.

[81] U. Zerbst, C. Klinger, and D. Klingbeil, "Structural assessment of railway axles–a critical review," *Engineering failure analysis*, vol. 35, pp. 54–65, 2013.

[82] A. Pourheidar, L. Patriarca, S. Beretta, and D. Regazzi, "Investigation of fatigue crack growth in full-scale railway axles subjected to service load spectra: Experiments and predictive models," *Metals*, vol. 11, no. 9, p. 1427, 2021.

[83] J. Bialowas, M. Pletz, S. Gapp, and J. Maierhofer, "A method to reduce computation time in finite element simulations of deep rolling," *Procedia structural integrity*, 2023.

[84] DIN Deutsches Institut für Normung e.V., "EN13262: Railway applications-Wheelsets and bogies-Wheels-Product requirements," 2009.

[85] Union Internationale der Chemins de Fer, "UIC510-5: Technische Zulassung von Vollrädern-Anwendungsdokument für die EN13979-1," 2007.

[86] DIN Deutsches Institut für Normung e.V., "EN13979: Railway applications-Wheelsets and bogies-Monobloc wheels-Technical approval procedure-Part1: Forged and rolled wheels," 2010.

[87] K. Grote and E. Antonsson, *Springer handbook of mechanical engineering*, vol. 10. Springer, 2009.

[88] J. Ihme, *Schienenfahrzeugtechnik*. Springer, 2016.

[89] J. Ahlström and B. Karlsson, "Modified Railway Wheel Steels: Production and Evaluation of Mechanical Properties with Emphasis on Low-Cycle Fatigue Behavior," *Metallurgical and Materials Transactions A*, vol. 40, no. 7, pp. 1557–1567, 2009.

[90] D. Mazumdar and J. W. Evans, *Modeling of steelmaking processes*. CRC press, 2009.

[91] C. Şimşir and C. H. Gür, "A fem based framework for simulation of thermal treatments: Application to steel quenching," *Computational Materials Science*, vol. 44, no. 2, pp. 588–600, 2008.

[92] M. Schemmel, P. Prevedel, R. Schöngrundner, W. Ecker, and T. Antretter, "Modelling of phase transformations and residual stress formation in hot-work tool steel components," in *European Conference of Heat Treatment and 21st IFHTSE Congress*, pp. 285–292, 2014.

[93] M. Schemmel, P. Prevedel, R. Schöngrundner, W. Ecker, and T. Antretter, "Size effects in residual stress formation during quenching of cylinders made of hot-work tool steel," *Advances in Materials Science and Engineering*, vol. 2015, 2015.

[94] S. Brunbauer, G. Winter, T. Antretter, P. Staron, and W. Ecker, "Residual stress and microstructure evolution in steel tubes for different cooling conditions–simulation and verification," *Materials Science and Engineering: A*, vol. 747, pp. 73–79, 2019.

[95] S. Leitner, G. Winter, J. Klarner, T. Antretter, and W. Ecker, "Model-based residual stress design in multiphase seamless steel tubes," *Materials*, vol. 13, no. 2, p. 439, 2020.

[96] K. Prabitz, M. Pichler, T. Antretter, H. Schubert, B. Hilpert, M. Gruber, R. Sierlinger, and W. Ecker, "Validated multi-physical finite element modelling of the spot welding process of the advanced high strength steel dp1200hd," *Materials*, vol. 14, no. 18, p. 5411, 2021.

[97] K. M. Prabitz, M. Z. Asadzadeh, M. Pichler, T. Antretter, C. Beal, H. Schubert, B. Hilpert, M. Gruber, R. Sierlinger, and W. Ecker, "Liquid metal embrittlement of advanced high strength steel: Experiments and damage modeling," *Materials*, vol. 14, no. 18, p. 5451, 2021.

[98] S. Leitner, G. Winter, J. Klarner, T. Antretter, and W. Ecker, "Residual stress evolution in low-alloyed steel at three different length scales," *Materials*, vol. 16, no. 7, 2023.

[99] S. N. Lingamanaik and B. K. Chen, "Thermo-mechanical modelling of residual stresses induced by martensitic phase transformation and cooling during quenching of railway wheels," *Journal of Materials Processing Technology*, vol. 211, no. 9, pp. 1547–1552, 2011.

[100] S. N. Lingamanaik and B. K. Chen, "Prediction of residual stresses in low carbon bainitic–martensitic railway wheels using heat transfer coefficients derived from quenching experiments," *Computational materials science*, vol. 77, pp. 153–160, 2013.

[101] F. Brunel, J.-F. Brunel, P. Dufrénoy, and F. Demilly, "Prediction of the initial residual stresses in railway wheels induced by manufacturing," *Journal of thermal stresses*, vol. 36, no. 1, pp. 37–55, 2013.

[102] M. Milošević, A. Miltenović, M. Banić, and M. Tomić, "Determination of residual stress in the rail wheel during quenching process by fem simulation," *Facta Universitatis, Series: Mechanical Engineering*, vol. 15, no. 3, pp. 413–425, 2017.

[103] Y. Tian, Z. Tan, J. Wang, R. Wang, Y. Liu, and M. Zhang, "Experiment and finite element analysis of asymmetrical hardness induced by quenching in railway wheel," *Engineering Failure Analysis*, vol. 133, p. 105959, 2022.

[104] J.-B. Leblond, J. Devaux, and J. Devaux, "Mathematical modelling of transformation plasticity in steels i: Case of ideal-plastic phases," *International journal of plasticity*, vol. 5, no. 6, pp. 551–572, 1989.

[105] S. Neubert, A. Pittner, and M. Rethmeier, "Experimental determination of trip-parameter k for mild-and high-strength low-alloy steels and a super martensitic filler material," *SpringerPlus*, vol. 5, pp. 1–16, 2016.

[106] G. Besserdich, B. Scholtes, H. Müller, and E. Macherauch, "Consequences of transformation plasticity on the development of residual stresses and distortions during martensitic hardening of sae 4140 steel cylinders," *Steel research*, vol. 65, no. 1, pp. 41–46, 1994.

[107] D. P. Koistinen, "A general equation prescribing the extent of the austenite-martensite transformation in pure iron-carbon alloys and plain carbon steels," *Acta metallurgica*, vol. 7, pp. 59–60, 1959.

[108] R. Garrett, S. Xu, J. Lin, and T. Dean, "A model for predicting austenite to bainite phase transformation in producing dual phase steels," *International Journal of Machine Tools and Manufacture*, vol. 44, no. 7-8, pp. 831–837, 2004.

[109] R. Mahnken, A. Schneidt, S. Tschumak, and H. Maier, "On the simulation of austenite to bainite phase transformation," *Computational Materials Science*, vol. 50, no. 6, pp. 1823–1829, 2011.

[110] R. Hill, "Elastic properties of reinforced solids: some theoretical principles," *Journal of the Mechanics and Physics of Solids*, vol. 11, no. 5, pp. 357–372, 1963.

[111] "DIN EN 13715:2020-10, Railway applications-Wheelsets and bogies-Wheels-Tread profile," 2006.

[112] R. Marek and K. Nitsche, "Praxis der wärmeübertragung," 2015.

[113] V. D. I. V.-G. V. und Chemieingenieurwesen (GVC), *Vdi-Wärmeatlas*. Springer, 2006.

[114] D. Scheinast, "Simulation der thermischen belastung blockgebremster eienbahnräder," 2021.

[115] B. Breuer and K. Bill, "Brake manual. fundamentals, components, systems, vehicle dynamics; bremsenhandbuch. grundlagen, komponenten, systeme, fahrdynamik," 2003.

[116] P. Berger, R. Rau, J. Galander, and R. Loebner, "Bremssysteme von schienenfahrzeugen," *Bremsenhandbuch: Grundlagen, Komponenten, Systeme, Fahrdynamik*, pp. 389–410, 2017.

A Appendix

A.1 RVE py-Script

```
# -*- coding: mbcs -*-
'''
Author:
J. Bialowas
'''
from abaqus import *
from mesh import *
from abaqusConstants import *
from caeModules import *
import os
import numpy as np
import math

session.journalOptions.setValues(replayGeometry=COORDINATE, recoverGeometry=COORDINATE)
TOL = 1e-6
workDir =os.path.abspath('')


jobName = 'test_of_sub'
odb_name = 'test_of_sub'
odbPath = workDir + '\\' + jobName +'.odb'
rptPath = workDir + jobName +'.rpt'

Lambda = [0.013, 0.150,  5.000,]
LambdaStr = ['0p013', '0p150', '5p000',]

# +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# General purpose functions
# +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
def calc_true_strain(engineering_strain):
true_strain = math.log(1+engineering_strain)
return true_strain
# -------------------------------------------------------------------------------
def calc_true_stress(engineering_stress, engineering_strain):
true_stress = engineering_stress * (1+engineering_strain)
return true_stress
# -------------------------------------------------------------------------------
# +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Functions for Abaqus
# +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
def import_inp():
m = mdb.ModelFromInputFile(inputFileName='output10.inp',
name='output10')
# define physical constants of the model
# absolute zero in C
# Stefan Boltzmann in t-mm-s
m.setValues(absoluteZero = 0.0,
stefanBoltzmann = 5.670371e-11)

m = mdb.models['output10']
m.setValues(noPartsInputFile=ON)
p = m.parts['PART-1']
print 'model from inp file imported'
return m, p

# -------------------------------------------------------------------------------
def create_geometry(m, l):
s = m.ConstrainedSketch(name='__profile__', sheetSize=200.0)
s.rectangle(point1=(0, 0), point2=(l, l))
p = m.Part(dimensionality=THREE_D, name='Part-RVE', type=DEFORMABLE_BODY)
p.BaseSolidExtrude(depth=l, sketch=s)
del s
return p
```

```
#-----------------------------------------------------------------------------
def create_assembly(m, p, m_name):
m.rootAssembly.DatumCsysByDefault(CARTESIAN)
a = mdb.models[m_name].rootAssembly
inst = m.rootAssembly.Instance(dependent=ON, name='Part-RVE-1', part=p)
return a, inst

#-----------------------------------------------------------------------------
def create_step(m, kelvin):
# Loadtxt of temperature amplitude with distinct lambda value
# and define step time with last entry
amp_Temp = np.loadtxt(os.path.join(os.getcwd(),'Temp_Lambda_0p03'),
skiprows=0,
delimiter=',',
unpack=True)
amp_Temp[1][:] = amp_Temp[1][:] + kelvin
# Define Quench-step
m.StaticStep(initialInc=0.01,
maxInc=12.0,
maxNumInc=10000,
minInc=1e-07,
name='Step-Quench',
nlgeom=ON,
previous='Initial',
timePeriod=amp_Temp[0][-1])

return amp_Temp

#-----------------------------------------------------------------------------
def create_Amplitude(m, amp_Temp):
m.TabularAmplitude(data=((0.0, 0.0),
(1.0, 1.0),
(2.0, 0.0),
(3.0, -1.0),
(4.0, 0.0)),
name='Amp-Tens-Comp',
smooth=SOLVER_DEFAULT,
timeSpan=STEP)

m.TabularAmplitude(data=(tuple(map(tuple, zip(*amp_Temp)))),
name='Amp-Temp',
smooth=SOLVER_DEFAULT,
timeSpan=STEP)
return

#-----------------------------------------------------------------------------
def create_material(m, p, kelvin):
mat = m.Material(name='ER7-TD')

# Import material properties from files

# USER DEFINED material properties
mat.expansion.setValues(type=ISOTROPIC, userSubroutine=ON)
mat.Creep(law=USER, table=())
mat.HeatGeneration()


# Create and assign Section
m.HomogeneousSolidSection(material='ER7-TD',
name='Section-RVE', thickness=None)
p.Set(cells=p.cells[:], name='Set_all')
p.SectionAssignment(offset=0.0, offsetField='',
offsetType=MIDDLE_SURFACE,
region=p.sets['Set_all'],
sectionName='Section-RVE',
thicknessAssignment=FROM_SECTION)

#-----------------------------------------------------------------------------
def create_pl_material():
'''
with open('ER7_plastic_TD', 'w') as f:
f.write('Yield Stress [MPa], Plastic Strain [-], Temperature [C], A phase, M phase, B phase, P, phase\n')
f.write('1000.0, 0.0, 20.0, 0.0, 1.0, 0.0, 0.0\n')
f.write('2000.0, 0.02, 20.0, 0.0, 1.0, 0.0,0.0\n')
'''

#-----------------------------------------------------------------------------
def create_RPs(m,a):
REFD = a.ReferencePoint(point=(0.0, 0.0, 0.0))
a.features.changeKey(fromName='RP-1', toName='REFD')
```

```
a.Set(name='Set-REFD', referencePoints=(a.referencePoints[REFD.id], ))
REFS = a.ReferencePoint(point=(0.0, 0.0, 0.0))
a.features.changeKey(fromName='RP-1', toName='REFS')
a.Set(name='Set-REFS', referencePoints=(a.referencePoints[REFS.id], ))

# ----------------------------------------------------------------------
def create_mesh(p, l, nr_elements):
# Assign Element Type
p.setElementType(elemTypes=(ElemType(elemCode=C3D8, elemLibrary=STANDARD),
ElemType(elemCode=C3D6,elemLibrary=STANDARD),
ElemType(elemCode=C3D4, elemLibrary=STANDARD)),
regions=(p.sets['Set_all']) )

p.seedPart(deviationFactor=0.1, minSizeFactor=0.1, size=1./nr_elements*l)
p.seedEdgeByNumber(constraint=FINER,
edges=p.edges.findAt(((0.0, l, 0.25*l), )),
number=nr_elements)
p.generateMesh()

# ----------------------------------------------------------------------
def create_p_sets(p, inst, l):
# Corners
pset_geom_C_000 = p.Set(name='pSet-C_000', vertices=p.vertices.findAt(((0.0, 0.0, 0.0), )))
pset_geom_C_L00 = p.Set(name='pSet-C_L00', vertices=p.vertices.findAt(((l, 0.0, 0.0), )))
pset_geom_C_0L0 = p.Set(name='pSet-C_0L0', vertices=p.vertices.findAt(((0.0, l, 0.0), )))
pset_geom_C_00L = p.Set(name='pSet-C_00L', vertices=p.vertices.findAt(((0.0, 0.0, l), )))
pset_geom_C_L0L = p.Set(name='pSet-C_L0L', vertices=p.vertices.findAt(((l, 0.0, l), )))
pset_geom_C_LL0 = p.Set(name='pSet-C_LL0', vertices=p.vertices.findAt(((l, l, 0.0), )))
pset_geom_C_LLL = p.Set(name='pSet-C_LLL', vertices=p.vertices.findAt(((l, l, l), )))
pset_geom_C_0LL = p.Set(name='pSet-C_0LL', vertices=p.vertices.findAt(((0.0, l, l), )))

# Edges without vertices
pset_geom_E_X00 = p.Set(edges=p.edges.findAt(((l/2, 0.0, 0.0), )), name='pSet-E_X00',
xVertices=p.vertices.findAt(((0.0, 0.0, 0.0), ), ((l, 0.0, 0.0), ), ))
pset_geom_E_L0X = p.Set(edges=p.edges.findAt(((l, 0.0, l/2), )), name='pSet-E_L0X',
xVertices=p.vertices.findAt(((l, 0.0, 0.0), ), ((l, 0.0, l), ), ))
pset_geom_E_X0L = p.Set(edges=p.edges.findAt(((l/2, 0.0, l), )), name='pSet-E_X0L',
xVertices=p.vertices.findAt(((0.0, 0.0, l), ), ((l, 0.0, l), ), ))
pset_geom_E_00X = p.Set(edges=p.edges.findAt(((0.0, 0.0, l/2), )), name='pSet-E_00X',
xVertices=p.vertices.findAt(((0.0, 0.0, 0.0), ), ((0.0, 0.0, l), ), ))
pset_geom_E_0X0 = p.Set(edges=p.edges.findAt(((0.0, l/2, 0), )), name='pSet-E_0X0',
xVertices=p.vertices.findAt(((0.0, 0.0, 0.0), ), ((0.0, l, 0.0), ), ))
pset_geom_E_LX0 = p.Set(edges=p.edges.findAt(((l, l/2, 0), )), name='pSet-E_LX0',
xVertices=p.vertices.findAt(((l, 0.0, 0.0), ), ((l, l, 0.0), ), ))
pset_geom_E_LXL = p.Set(edges=p.edges.findAt(((l, l/2, l), )), name='pSet-E_LXL',
xVertices=p.vertices.findAt(((l, 0.0, l), ), ((l, l, l), ), ))
pset_geom_E_0XL = p.Set(edges=p.edges.findAt(((0.0, l/2, l), )), name='pSet-E_0XL',
xVertices=p.vertices.findAt(((0.0, 0.0, l), ), ((0.0, l, l), ), ))
pset_geom_E_XL0 = p.Set(edges=p.edges.findAt(((l/2, l, 0), )), name='pSet-E_XL0',
xVertices=p.vertices.findAt(((0.0, l, 0.0), ), ((l, l, 0.0), ), ))
pset_geom_E_LLX = p.Set(edges=p.edges.findAt(((l, l, l/2), )), name='pSet-E_LLX',
xVertices=p.vertices.findAt(((l, l, 0.0), ), ((l, l, l), ), ))
pset_geom_E_XLL = p.Set(edges=p.edges.findAt(((l/2, l, l), )), name='pSet-E_XLL',
xVertices=p.vertices.findAt(((0.0, l, l), ), ((l, l, l), ), ))
pset_geom_E_0LX = p.Set(edges=p.edges.findAt(((0.0, l, l/2), )), name='pSet-E_0LX',
xVertices=p.vertices.findAt(((0.0, l, 0.0), ), ((0.0, l, l), ), ))

# Faces without edges
pset_geom_F_X0X = p.Set(faces=p.faces.findAt(((l/2, 0.0, l/2), )), name='pSet-F_X0X',
xEdges=p.edges.findAt(
((0.0, 0.0, l/2), ),
((l, 0.0, l/2), ),
((l/2, 0.0, l), ),
((l/2, 0.0, 0.0), ), ))
pset_geom_F_XX0 = p.Set(faces=p.faces.findAt(((l/2, l/2, 0.0), )), name='pSet-F_XX0',
xEdges=p.edges.findAt(
((0.0, l/2, 0.0), ),
((l/2, l, 0.0), ),
((l, l/2, 0.0), ),
((l/2, 0.0, 0.0), ), ))
pset_geom_F_0XX = p.Set(faces=p.faces.findAt(((0.0, l/2, l/2), )), name='pSet-F_0XX',
xEdges=p.edges.findAt(
((0.0, l/2, l), ),
((0.0, l, l/2), ),
((0.0, l/2, 0.0), ),
((0.0, 0.0, l/2), ), ))
pset_geom_F_XLX = p.Set(faces=p.faces.findAt(((l/2, l, l/2), )), name='pSet-F_XLX',
xEdges=p.edges.findAt(
((0.0, l, l/2), ),
((l, l, l/2), ),
```

```
((l/2, l, l), ),
((l/2, l, 0.0), ), ))
pset_geom_F_XXL = p.Set(faces=p.faces.findAt(((l/2, l/2, l), )), name='pSet-F_XXL',
xEdges=p.edges.findAt(
((0.0, l/2, l), ),
((l/2, l, l), ),
((l, l/2, l), ),
((l/2, 0.0, l), ), ))
pset_geom_F_LXX = p.Set(faces=p.faces.findAt(((l, l/2, l/2), )), name='pSet-F_LXX',
xEdges=p.edges.findAt(
((l, l/2, l), ),
((l, l, l/2), ),
((l, l/2, 0.0), ),
((l, 0.0, l/2), ), ))
# magic
p.regenerate()

#-------------------------------------------------------------------------
def create_a_sets(a, inst, l):
# Corners
set_geom_C_000 = a.Set(name='Set-C_000', vertices=inst.vertices.findAt(((0.0, 0.0, 0.0), )))
set_geom_C_L00 = a.Set(name='Set-C_L00', vertices=inst.vertices.findAt(((l, 0.0, 0.0), )))
set_geom_C_0L0 = a.Set(name='Set-C_0L0', vertices=inst.vertices.findAt(((0.0, l, 0.0), )))
set_geom_C_00L = a.Set(name='Set-C_00L', vertices=inst.vertices.findAt(((0.0, 0.0, l), )))
set_geom_C_L0L = a.Set(name='Set-C_L0L', vertices=inst.vertices.findAt(((l, 0.0, l), )))
set_geom_C_LL0 = a.Set(name='Set-C_LL0', vertices=inst.vertices.findAt(((l, l, 0.0), )))
set_geom_C_LLL = a.Set(name='Set-C_LLL', vertices=inst.vertices.findAt(((l, l, l), )))
set_geom_C_0LL = a.Set(name='Set-C_0LL', vertices=inst.vertices.findAt(((0.0, l, l), )))

# Edges without vertices
set_geom_E_X00 = a.Set(edges=inst.edges.findAt(((l/2, 0.0, 0.0), )), name='Set-E_X00',
xVertices=inst.vertices.findAt(((0.0, 0.0, 0.0), ), ((l, 0.0, 0.0), ), ))
set_geom_E_L0X = a.Set(edges=inst.edges.findAt(((l, 0.0, l/2), )), name='Set-E_L0X',
xVertices=inst.vertices.findAt(((l, 0.0, 0.0), ), ((l, 0.0, l), ), ))
set_geom_E_X0L = a.Set(edges=inst.edges.findAt(((l/2, 0.0, l), )), name='Set-E_X0L',
xVertices=inst.vertices.findAt(((0.0, 0.0, l), ), ((l, 0.0, l), ), ))
set_geom_E_00X = a.Set(edges=inst.edges.findAt(((0.0, 0.0, l/2), )), name='Set-E_00X',
xVertices=inst.vertices.findAt(((0.0, 0.0, 0.0), ), ((0.0, 0.0, l), ), ))
set_geom_E_0X0 = a.Set(edges=inst.edges.findAt(((0.0, l/2, 0), )), name='Set-E_0X0',
xVertices=inst.vertices.findAt(((0.0, 0.0, 0.0), ), ((0.0, l, 0.0), ), ))
set_geom_E_LX0 = a.Set(edges=inst.edges.findAt(((l, l/2, 0), )), name='Set-E_LX0',
xVertices=inst.vertices.findAt(((l, 0.0, 0.0), ), ((l, l, 0.0), ), ))
set_geom_E_LXL = a.Set(edges=inst.edges.findAt(((l, l/2, l), )), name='Set-E_LXL',
xVertices=inst.vertices.findAt(((l, 0.0, l), ), ((l, l, l), ), ))
set_geom_E_0XL = a.Set(edges=inst.edges.findAt(((0.0, l/2, l), )), name='Set-E_0XL',
xVertices=inst.vertices.findAt(((0.0, 0.0, l), ), ((0.0, l, l), ), ))
set_geom_E_XL0 = a.Set(edges=inst.edges.findAt(((l/2, l, 0), )), name='Set-E_XL0',
xVertices=inst.vertices.findAt(((0.0, l, 0.0), ), ((l, l, 0.0), ), ))
set_geom_E_LLX = a.Set(edges=inst.edges.findAt(((l, l, l/2), )), name='Set-E_LLX',
xVertices=inst.vertices.findAt(((l, l, 0.0), ), ((l, l, l), ), ))
set_geom_E_XLL = a.Set(edges=inst.edges.findAt(((l/2, l, l), )), name='Set-E_XLL',
xVertices=inst.vertices.findAt(((0.0, l, l), ), ((l, l, l), ), ))
set_geom_E_0LX = a.Set(edges=inst.edges.findAt(((0.0, l, l/2), )), name='Set-E_0LX',
xVertices=inst.vertices.findAt(((0.0, l, 0.0), ), ((0.0, l, l), ), ))

# Faces without edges
set_geom_F_X0X = a.Set(faces=inst.faces.findAt(((l/2, 0.0, l/2), )), name='Set-F_X0X',
xEdges=inst.edges.findAt(
((0.0, 0.0, l/2), ),
((l, 0.0, l/2), ),
((l/2, 0.0, l), ),
((l/2, 0.0, 0.0), ), ))
set_geom_F_XX0 = a.Set(faces=inst.faces.findAt(((l/2, l/2, 0.0), )), name='Set-F_XX0',
xEdges=inst.edges.findAt(
((0.0, l/2, 0.0), ),
((l/2, l, 0.0), ),
((l, l/2, 0.0), ),
((l/2, 0.0, 0.0), ), ))
set_geom_F_0XX = a.Set(faces=inst.faces.findAt(((0.0, l/2, l/2), )), name='Set-F_0XX',
xEdges=inst.edges.findAt(
((0.0, l/2, l), ),
((0.0, l, l/2), ),
((0.0, l/2, 0.0), ),
((0.0, 0.0, l/2), ), ))
set_geom_F_XLX = a.Set(faces=inst.faces.findAt(((l/2, l, l/2), )), name='Set-F_XLX',
xEdges=inst.edges.findAt(
((0.0, l, l/2), ),
((l, l, l/2), ),
((l/2, l, l), ),
((l/2, l, 0.0), ), ))
```

```
set_geom_F_XXL = a.Set(faces=inst.faces.findAt(((l/2, l/2, l), )), name='Set-F_XXL',
xEdges=inst.edges.findAt(
((0.0, l/2, l), ),
((l/2, l, l), ),
((l, l/2, l), ),
((l/2, 0.0, l), ), ))
set_geom_F_LXX = a.Set(faces=inst.faces.findAt(((l, l/2, l/2), )), name='Set-F_LXX',
xEdges=inst.edges.findAt(
((l, l/2, l), ),
((l, l, l/2), ),
((l, l/2, 0.0), ),
((l, 0.0, l/2), ), ))

# magic
p.regenerate()
a.regenerate()

#node sets from geometry sets
# Vertices
set_node_C_000 = a.Set(
nodes=set_geom_C_000.nodes,
name='N_Set-C_000')
set_node_C_L00 = a.Set(
nodes=set_geom_C_L00.nodes,
name='N_Set-C_L00')
set_node_C_0L0 = a.Set(
nodes=set_geom_C_0L0.nodes,
name='N_Set-C_0L0')
set_node_C_00L = a.Set(
nodes=set_geom_C_00L.nodes,
name='N_Set-C_00L')
set_node_C_L0L = a.Set(
nodes=set_geom_C_L0L.nodes,
name='N_Set-C_L0L')
set_node_C_LL0 = a.Set(
nodes=set_geom_C_LL0.nodes,
name='N_Set-C_LL0')
set_node_C_LLL = a.Set(
nodes=set_geom_C_LLL.nodes,
name='N_Set-C_LLL')
set_node_C_0LL = a.Set(
nodes=set_geom_C_0LL.nodes,
name='N_Set-C_0LL')
# Edges
set_node_E_X00 = a.Set(nodes=set_geom_E_X00.nodes,
name='N_Set-E_X00')
set_node_E_L0X = a.Set(nodes=set_geom_E_L0X.nodes,
name='N_Set-E_L0X')
set_node_E_X0L = a.Set(nodes=set_geom_E_X0L.nodes,
name='N_Set-E_X0L')
set_node_E_00X = a.Set(nodes=set_geom_E_00X.nodes,
name='N_Set-E_00X')
set_node_E_0X0 = a.Set(nodes=set_geom_E_0X0.nodes,
name='N_Set-E_0X0')
set_node_E_LX0 = a.Set(nodes=set_geom_E_LX0.nodes,
name='N_Set-E_LX0')
set_node_E_LXL = a.Set(nodes=set_geom_E_LXL.nodes,
name='N_Set-E_LXL')
set_node_E_0XL = a.Set(nodes=set_geom_E_0XL.nodes,
name='N_Set-E_0XL')
set_node_E_XL0 = a.Set(nodes=set_geom_E_XL0.nodes,
name='N_Set-E_XL0')
set_node_E_LLX = a.Set(nodes=set_geom_E_LLX.nodes,
name='N_Set-E_LLX')
set_node_E_XLL = a.Set(nodes=set_geom_E_XLL.nodes,
name='N_Set-E_XLL')
set_node_E_0LX = a.Set(nodes=set_geom_E_0LX.nodes,
name='N_Set-E_0LX')
#Faces
set_node_F_X0X = a.Set(
nodes=set_geom_F_X0X.nodes,
name='N_Set-F_X0X')
set_node_F_XX0 = a.Set(
nodes=set_geom_F_XX0.nodes,
name='N_Set-F_XX0')
set_node_F_0XX = a.Set(
nodes=set_geom_F_0XX.nodes,
name='N_Set-F_0XX')
set_node_F_XLX = a.Set(
nodes=set_geom_F_XLX.nodes,
```

```
name='N_Set-F_XLX')
set_node_F_XXL = a.Set(
nodes=set_geom_F_XXL.nodes,
name='N_Set-F_XXL')
set_node_F_LXX = a.Set(
nodes=set_geom_F_LXX.nodes,
name='N_Set-F_LXX')

# Unsorted nodes sets
# faces
nodes_F_0XX = [x.label for x in a.sets['Set-F_0XX'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-F_0XX', nodeLabels=(('Part-RVE-1', nodes_F_0XX), ), unsorted=True)
nodes_F_LXX = [x.label for x in a.sets['Set-F_LXX'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-F_LXX', nodeLabels=(('Part-RVE-1', nodes_F_LXX), ), unsorted=True)
nodes_F_X0X = [x.label for x in a.sets['Set-F_X0X'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-F_X0X', nodeLabels=(('Part-RVE-1', nodes_F_X0X), ), unsorted=True)
nodes_F_XLX = [x.label for x in a.sets['Set-F_XLX'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-F_XLX', nodeLabels=(('Part-RVE-1', nodes_F_XLX), ), unsorted=True)
nodes_F_XX0 = [x.label for x in a.sets['Set-F_XX0'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-F_XX0', nodeLabels=(('Part-RVE-1', nodes_F_XX0), ), unsorted=True)
nodes_F_XXL = [x.label for x in a.sets['Set-F_XXL'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-F_XXL', nodeLabels=(('Part-RVE-1', nodes_F_XXL), ), unsorted=True)

#edges
nodes_E_X00 = [x.label for x in a.sets['Set-E_X00'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-E_X00', nodeLabels=(('Part-RVE-1', nodes_E_X00), ), unsorted=True)
nodes_E_L0X = [x.label for x in a.sets['Set-E_L0X'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-E_L0X', nodeLabels=(('Part-RVE-1', nodes_E_L0X), ), unsorted=True)
nodes_E_X0L = [x.label for x in a.sets['Set-E_X0L'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-E_X0L', nodeLabels=(('Part-RVE-1', nodes_E_X0L), ), unsorted=True)
nodes_E_00X = [x.label for x in a.sets['Set-E_00X'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-E_00X', nodeLabels=(('Part-RVE-1', nodes_E_00X), ), unsorted=True)
nodes_E_0X0 = [x.label for x in a.sets['Set-E_0X0'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-E_0X0', nodeLabels=(('Part-RVE-1', nodes_E_0X0), ), unsorted=True)
nodes_E_LX0 = [x.label for x in a.sets['Set-E_LX0'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-E_LX0', nodeLabels=(('Part-RVE-1', nodes_E_LX0), ), unsorted=True)
nodes_E_LXL = [x.label for x in a.sets['Set-E_LXL'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-E_LXL', nodeLabels=(('Part-RVE-1', nodes_E_LXL), ), unsorted=True)
nodes_E_0XL = [x.label for x in a.sets['Set-E_0XL'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-E_0XL', nodeLabels=(('Part-RVE-1', nodes_E_0XL), ), unsorted=True)
nodes_E_XL0 = [x.label for x in a.sets['Set-E_XL0'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-E_XL0', nodeLabels=(('Part-RVE-1', nodes_E_XL0), ), unsorted=True)
nodes_E_LLX = [x.label for x in a.sets['Set-E_LLX'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-E_LLX', nodeLabels=(('Part-RVE-1', nodes_E_LLX), ), unsorted=True)
nodes_E_XLL = [x.label for x in a.sets['Set-E_XLL'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-E_XLL', nodeLabels=(('Part-RVE-1', nodes_E_XLL), ), unsorted=True)
nodes_E_0LX = [x.label for x in a.sets['Set-E_0LX'].nodes]
a.SetFromNodeLabels(name='Unsorted-Set-E_0LX', nodeLabels=(('Part-RVE-1', nodes_E_0LX), ), unsorted=True)

#**************************************************************************
def create_equations(m):
# equations for faces and reference points
m.Equation(name='Eq-F_LXX-DOF-1',
terms=((1.0, 'Unsorted-Set-F_LXX', 1),
(-1.0, 'Unsorted-Set-F_0XX', 1),
(-1.0, 'Set-REFD', 1)))
m.Equation(name='Eq-F_LXX-DOF-2',
terms=((1.0, 'Unsorted-Set-F_LXX', 2),
(-1.0, 'Unsorted-Set-F_0XX', 2),
(-1.0, 'Set-REFS', 1)))
m.Equation(name='Eq-F_LXX-DOF-3',
terms=((1.0, 'Unsorted-Set-F_LXX', 3),
(-1.0, 'Unsorted-Set-F_0XX', 3),
(-1.0, 'Set-REFS', 3)))
m.Equation(name='Eq-F_XLX-DOF-1',
terms=((1.0, 'Unsorted-Set-F_XLX', 1),
(-1.0, 'Unsorted-Set-F_X0X', 1),
(-1.0, 'Set-REFS', 1)))
m.Equation(name='Eq-F_XLX-DOF-2',
terms=((1.0, 'Unsorted-Set-F_XLX', 2),
(-1.0, 'Unsorted-Set-F_X0X', 2),
(-1.0, 'Set-REFD', 2)))
m.Equation(name='Eq-F_XLX-DOF-3',
terms=((1.0, 'Unsorted-Set-F_XLX', 3),
(-1.0, 'Unsorted-Set-F_X0X', 3),
(-1.0, 'Set-REFS', 2)))
m.Equation(name='Eq-F_XXL-DOF-1',
terms=((1.0, 'Unsorted-Set-F_XXL', 1),
(-1.0, 'Unsorted-Set-F_XX0', 1),
(-1.0, 'Set-REFS', 3)))
```

```
m.Equation(name='Eq-F_XXL-DOF-2',
terms=((1.0, 'Unsorted-Set-F_XXL', 2),
(-1.0, 'Unsorted-Set-F_XX0', 2),
(-1.0, 'Set-REFS', 2)))
m.Equation(name='Eq-F_XXL-DOF-3',
terms=((1.0, 'Unsorted-Set-F_XXL', 3),
(-1.0, 'Unsorted-Set-F_XX0', 3),
(-1.0, 'Set-REFD', 3)))

# equations for edges and reference points
m.Equation(name='Eq-E_XL0-DOF-1',
terms=((1.0, 'Unsorted-Set-E_XL0', 1),
(-1.0, 'Unsorted-Set-E_X00', 1),
(-1.0, 'Set-REFS', 1)))
m.Equation(name='Eq-E_XL0-DOF-2',
terms=((1.0, 'Unsorted-Set-E_XL0', 2),
(-1.0, 'Unsorted-Set-E_X00', 2),
(-1.0, 'Set-REFD', 2)))
m.Equation(name='Eq-E_XL0-DOF-3',
terms=((1.0, 'Unsorted-Set-E_XL0', 3),
(-1.0, 'Unsorted-Set-E_X00', 3),
(-1.0, 'Set-REFS', 2)))
m.Equation(name='Eq-E_X0L-DOF-1',
terms=((1.0, 'Unsorted-Set-E_X0L', 1),
(-1.0, 'Unsorted-Set-E_X00', 1),
(-1.0, 'Set-REFS', 3)))
m.Equation(name='Eq-E_X0L-DOF-2',
terms=((1.0, 'Unsorted-Set-E_X0L', 2),
(-1.0, 'Unsorted-Set-E_X00', 2),
(-1.0, 'Set-REFS', 2)))
m.Equation(name='Eq-E_X0L-DOF-3',
terms=((1.0, 'Unsorted-Set-E_X0L', 3),
(-1.0, 'Unsorted-Set-E_X00', 3),
(-1.0, 'Set-REFD', 3)))
m.Equation(name='Eq-E_XLL-DOF-1',
terms=((1.0, 'Unsorted-Set-E_XLL', 1),
(-1.0, 'Unsorted-Set-E_X00', 1),
(-1.0, 'Set-REFS', 1),
(-1.0, 'Set-REFS', 3)))
m.Equation(name='Eq-E_XLL-DOF-2',
terms=((1.0, 'Unsorted-Set-E_XLL', 2),
(-1.0, 'Unsorted-Set-E_X00', 2),
(-1.0, 'Set-REFD', 2),
(-1.0, 'Set-REFS', 2)))
m.Equation(name='Eq-E_XLL-DOF-3',
terms=((1.0, 'Unsorted-Set-E_XLL', 3),
(-1.0, 'Unsorted-Set-E_X00', 3),
(-1.0, 'Set-REFS', 2),
(-1.0, 'Set-REFD', 3)))
m.Equation(name='Eq-E_0XL-DOF-1',
terms=((1.0, 'Unsorted-Set-E_0XL', 1),
(-1.0, 'Unsorted-Set-E_0X0', 1),
(-1.0, 'Set-REFS', 3)))
m.Equation(name='Eq-E_0XL-DOF-2',
terms=((1.0, 'Unsorted-Set-E_0XL', 2),
(-1.0, 'Unsorted-Set-E_0X0', 2),
(-1.0, 'Set-REFS', 2)))
m.Equation(name='Eq-E_0XL-DOF-3',
terms=((1.0, 'Unsorted-Set-E_0XL', 3),
(-1.0, 'Unsorted-Set-E_0X0', 3),
(-1.0, 'Set-REFD', 3)))
m.Equation(name='Eq-E_LX0-DOF-1',
terms=((1.0, 'Unsorted-Set-E_LX0', 1),
(-1.0, 'Unsorted-Set-E_0X0', 1),
(-1.0, 'Set-REFD', 1)))
m.Equation(name='Eq-E_LX0-DOF-2',
terms=((1.0, 'Unsorted-Set-E_LX0', 2),
(-1.0, 'Unsorted-Set-E_0X0', 2),
(-1.0, 'Set-REFS', 1)))
m.Equation(name='Eq-E_LX0-DOF-3',
terms=((1.0, 'Unsorted-Set-E_LX0', 3),
(-1.0, 'Unsorted-Set-E_0X0', 3),
(-1.0, 'Set-REFS', 3)))
m.Equation(name='Eq-E_LXL-DOF-1',
terms=((1.0, 'Unsorted-Set-E_LXL', 1),
(-1.0, 'Unsorted-Set-E_0X0', 1),
(-1.0, 'Set-REFD', 1),
(-1.0, 'Set-REFS', 3)))
m.Equation(name='Eq-E_LXL-DOF-2',
terms=((1.0, 'Unsorted-Set-E_LXL', 2),
```

```
(-1.0, 'Unsorted-Set-E_0X0', 2),
(-1.0, 'Set-REFS', 1),
(-1.0, 'Set-REFS', 2)))
m.Equation(name='Eq-E_LXL-DOF-3',
terms=((1.0, 'Unsorted-Set-E_LXL', 3),
(-1.0, 'Unsorted-Set-E_0X0', 3),
(-1.0, 'Set-REFS', 3),
(-1.0, 'Set-REFD', 3)))
m.Equation(name='Eq-E_L0X-DOF-1',
terms=((1.0, 'Unsorted-Set-E_L0X', 1),
(-1.0, 'Unsorted-Set-E_00X', 1),
(-1.0, 'Set-REFD', 1)))
m.Equation(name='Eq-E_L0X-DOF-2',
terms=((1.0, 'Unsorted-Set-E_L0X', 2),
(-1.0, 'Unsorted-Set-E_00X', 2),
(-1.0, 'Set-REFS', 1)))
m.Equation(name='Eq-E_L0X-DOF-3',
terms=((1.0, 'Unsorted-Set-E_L0X', 3),
(-1.0, 'Unsorted-Set-E_00X', 3),
(-1.0, 'Set-REFS', 3)))
m.Equation(name='Eq-E_0LX-DOF-1',
terms=((1.0, 'Unsorted-Set-E_0LX', 1),
(-1.0, 'Unsorted-Set-E_00X', 1),
(-1.0, 'Set-REFS', 1)))
m.Equation(name='Eq-E_0LX-DOF-2',
terms=((1.0, 'Unsorted-Set-E_0LX', 2),
(-1.0, 'Unsorted-Set-E_00X', 2),
(-1.0, 'Set-REFD', 2)))
m.Equation(name='Eq-E_0LX-DOF-3',
terms=((1.0, 'Unsorted-Set-E_0LX', 3),
(-1.0, 'Unsorted-Set-E_00X', 3),
(-1.0, 'Set-REFS', 2)))
m.Equation(name='Eq-E_LLX-DOF-1',
terms=((1.0, 'Unsorted-Set-E_LLX', 1),
(-1.0, 'Unsorted-Set-E_00X', 1),
(-1.0, 'Set-REFD', 1),
(-1.0, 'Set-REFS', 1)))
m.Equation(name='Eq-E_LLX-DOF-2',
terms=((1.0, 'Unsorted-Set-E_LLX', 2),
(-1.0, 'Unsorted-Set-E_00X', 2),
(-1.0, 'Set-REFS', 1),
(-1.0, 'Set-REFD', 2)))
m.Equation(name='Eq-E_LLX-DOF-3',
terms=((1.0, 'Unsorted-Set-E_LLX', 3),
(-1.0, 'Unsorted-Set-E_00X', 3),
(-1.0, 'Set-REFS', 3),
(-1.0, 'Set-REFS', 2)))

# equations for corners and reference points
m.Equation(name='Eq-C_L00-DOF-1',
terms=((1.0, 'N_Set-C_L00', 1),
(-1.0, 'N_Set-C_000', 1),
(-1.0, 'Set-REFD', 1)))
m.Equation(name='Eq-C_L00-DOF-2',
terms=((1.0, 'N_Set-C_L00', 2),
(-1.0, 'N_Set-C_000', 2),
(-1.0, 'Set-REFS', 1)))
m.Equation(name='Eq-C_L00-DOF-3',
terms=((1.0, 'N_Set-C_L00', 3),
(-1.0, 'N_Set-C_000', 3),
(-1.0, 'Set-REFS', 3)))
m.Equation(name='Eq-C_0L0-DOF-1',
terms=((1.0, 'N_Set-C_0L0', 1),
(-1.0, 'N_Set-C_000', 1),
(-1.0, 'Set-REFS', 1)))
m.Equation(name='Eq-C_0L0-DOF-2',
terms=((1.0, 'N_Set-C_0L0', 2),
(-1.0, 'N_Set-C_000', 2),
(-1.0, 'Set-REFD', 2)))
m.Equation(name='Eq-C_0L0-DOF-3',
terms=((1.0, 'N_Set-C_0L0', 3),
(-1.0, 'N_Set-C_000', 3),
(-1.0, 'Set-REFS', 2)))
m.Equation(name='Eq-C_00L-DOF-1',
terms=((1.0, 'N_Set-C_00L', 1),
(-1.0, 'N_Set-C_000', 1),
(-1.0, 'Set-REFS', 3)))
m.Equation(name='Eq-C_00L-DOF-2',
terms=((1.0, 'N_Set-C_00L', 2),
(-1.0, 'N_Set-C_000', 2),
```

```
(-1.0, 'Set-REFS', 2)))
m.Equation(name='Eq-C_00L-DOF-3',
terms=((1.0, 'N_Set-C_00L', 3),
(-1.0, 'N_Set-C_000', 3),
(-1.0, 'Set-REFD', 3)))
m.Equation(name='Eq-C_LL0-DOF-1',
terms=((1.0, 'N_Set-C_LL0', 1),
(-1.0, 'N_Set-C_000', 1),
(-1.0, 'Set-REFD', 1),
(-1.0, 'Set-REFS', 1)))
m.Equation(name='Eq-C_LL0-DOF-2',
terms=((1.0, 'N_Set-C_LL0', 2),
(-1.0, 'N_Set-C_000', 2),
(-1.0, 'Set-REFS', 1),
(-1.0, 'Set-REFD', 2)))
m.Equation(name='Eq-C_LL0-DOF-3',
terms=((1.0, 'N_Set-C_LL0', 3),
(-1.0, 'N_Set-C_000', 3),
(-1.0, 'Set-REFS', 3),
(-1.0, 'Set-REFS', 2)))
m.Equation(name='Eq-C_L0L-DOF-1',
terms=((1.0, 'N_Set-C_L0L', 1),
(-1.0, 'N_Set-C_000', 1),
(-1.0, 'Set-REFD', 1),
(-1.0, 'Set-REFS', 3)))
m.Equation(name='Eq-C_L0L-DOF-2',
terms=((1.0, 'N_Set-C_L0L', 2),
(-1.0, 'N_Set-C_000', 2),
(-1.0, 'Set-REFS', 1),
(-1.0, 'Set-REFS', 2)))
m.Equation(name='Eq-C_L0L-DOF-3',
terms=((1.0, 'N_Set-C_L0L', 3),
(-1.0, 'N_Set-C_000', 3),
(-1.0, 'Set-REFS', 3),
(-1.0, 'Set-REFD', 3)))
m.Equation(name='Eq-C_0LL-DOF-1',
terms=((1.0, 'N_Set-C_0LL', 1),
(-1.0, 'N_Set-C_000', 1),
(-1.0, 'Set-REFS', 1),
(-1.0, 'Set-REFS', 3)))
m.Equation(name='Eq-C_0LL-DOF-2',
terms=((1.0, 'N_Set-C_0LL', 2),
(-1.0, 'N_Set-C_000', 2),
(-1.0, 'Set-REFD', 2),
(-1.0, 'Set-REFS', 2)))
m.Equation(name='Eq-C_0LL-DOF-3',
terms=((1.0, 'N_Set-C_0LL', 3),
(-1.0, 'N_Set-C_000', 3),
(-1.0, 'Set-REFS', 2),
(-1.0, 'Set-REFD', 3)))
m.Equation(name='Eq-C_LLL-DOF-1',
terms=((1.0, 'N_Set-C_LLL', 1),
(-1.0, 'N_Set-C_000', 1),
(-1.0, 'Set-REFD', 1),
(-1.0, 'Set-REFS', 1),
(-1.0, 'Set-REFS', 3)))
m.Equation(name='Eq-C_LLL-DOF-2',
terms=((1.0, 'N_Set-C_LLL', 2),
(-1.0, 'N_Set-C_000', 2),
(-1.0, 'Set-REFS', 1),
(-1.0, 'Set-REFD', 2),
(-1.0, 'Set-REFS', 2)))
m.Equation(name='Eq-C_LLL-DOF-3',
terms=((1.0, 'N_Set-C_LLL', 3),
(-1.0, 'N_Set-C_000', 3),
(-1.0, 'Set-REFS', 3),
(-1.0, 'Set-REFS', 2),
(-1.0, 'Set-REFD', 3)))

#-----------------------------------------------------------------------
def create_boundary(m, a):
# Initial step
m.DisplacementBC(amplitude=UNSET, createStepName='Initial',
distributionType=UNIFORM, fieldName='', localCsys=None, name='BC-C000',
region=a.sets['N_Set-C_000'],
u1=SET, u2=SET, u3=SET, ur1=UNSET, ur2=UNSET, ur3=UNSET)
'''
# Load step
amp = 0.01
m.DisplacementBC(amplitude='Amp-Tens-Comp',
```

```
createStepName='Step-Hot-Tensile-Test',
distributionType=UNIFORM,
fieldName='', fixed=OFF,
localCsys=None,
name='BC-Tens-Comp',
region=m.rootAssembly.sets['Set-REFD'],
u1=UNSET, u2=amp, u3=UNSET, ur1=UNSET, ur2=UNSET, ur3=UNSET)
'''

# ------------------------------------------------------------------------
def create_predefined_field(m, a, Temp_850):
# Initial Temperature
m.Temperature(createStepName='Initial',
crossSectionDistribution=CONSTANT_THROUGH_THICKNESS,
distributionType=UNIFORM,
magnitudes=(Temp_850, ),
name='Predefined Field-Initial_Temp',
region=a.instances['Part-RVE-1'].sets['Set_all'])
m.Temperature(amplitude='Amp-Temp',
createStepName='Step-Quench',
crossSectionDistribution=CONSTANT_THROUGH_THICKNESS,
distributionType=UNIFORM,
magnitudes=(1.0, ),
name='Predefined Field-Temp',
region=a.instances['Part-RVE-1'].sets['Set_all'])

# ------------------------------------------------------------------------
def create_job(m, jobName):
# create field output
m.fieldOutputRequests['F-Output-1'].setValues(
variables=('FV','SDV','S','PE', 'PEEQ', 'PEMAG', 'LE', 'U', 'RF',
'CSTRESS', 'CDISP', 'NT', 'TEMP'))

#create job
mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
explicitPrecision=SINGLE, getMemoryFromAnalysis=True, historyPrint=OFF,
memory=90, memoryUnits=PERCENTAGE, model='Model-1', modelPrint=OFF,
multiprocessingMode=DEFAULT, name=jobName, nodalOutputPrecision=FULL,
numCpus=1, numDomains=1, numGPUs=0, queue=None, resultsFormat=ODB, scratch='',
type=ANALYSIS, userSubroutine='Sub_Quench_ER7_Toggling.f', waitHours=0, waitMinutes=0)

# adding keywords for additional output variables
m.keywordBlock.setValues(edited=0)
m.keywordBlock.synchVersions(storeNodesAndElements=False)
block_list = m.keywordBlock.sieBlocks
pos_i = [i for i, j in enumerate(block_list) if 'Output, field' in j]
str_out = '**\n*Element Output, directions=YES\nCOORD, IVOL, \n** \n*Node Output \nCOORD, '
# String einfuegen!
m.keywordBlock.insert(pos_i[0], str_out)

# adding keywords for use of subroutines
pos_i = [i for i, j in enumerate(block_list) if '*User Defined Field' in j]
str_out = '\n*Initial Conditions, type=SOLUTION, user\n*Initial Conditions, type=FIELD'
# String einfuegen!
m.keywordBlock.insert(pos_i[0], str_out)

# adding new names for state variables
pos_i = [i for i, j in enumerate(block_list) if '*Depvar' in j]
str_out = '1,TEMP,TEMP\
\n2,AUSTENITE,AUSTENITE\
\n3,MARTENSITE,MARTENSITE\
\n4,BAINITE, BAINITE\
\n5,PEARLITE,PEARLITE\
\n6,DMARTENSITE,DMARTENSITE\
\n7,DBAINITE,DBAINITE\
\n8,DPEARLITE,DPEARLITE\
\n9,DTEMP,DTEMP\
\n10,RBAINITE,RBAINITE\
\n11,RPERLITE,RPERLITE\
\n12,NELEMENTS,NELEMENTS\
\n13,NAUSTENITE,NAUSTENITE\
\n14,NMARTENSITE,NMARTENSITE\
\n15,NBAINITE,NBAINITE\
\n16,NPERLITE,NPERLITE\
\n17,RKRITBAIN,RKRITBAIN\
\n18,RKRITPERL,RKRITPERL\
\n19,NOEL,NOEL\
\n20,PHASE,PHASE'
# String einfuegen!
m.keywordBlock.insert(pos_i[0], str_out)
```

```
# set general solution controls to allow more cutbacks
m.steps['Step-Quench'].control.setValues(
 allowPropagation=OFF,
 timeIncrementation=(8.0, 10.0, 9.0, 16.0, 10.0, 4.0, 12.0, 25.0, 6.0, 3.0, 50.0))
 '''
m.steps['Step-Quench'].control.setValues(
 allowPropagation=OFF,
 resetDefaultValues=OFF,
 timeIncrementation=(8.0, 10.0))
 '''
# write input
mdb.jobs[jobName].writeInput(consistencyChecking=OFF)

# +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# running program
# +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
if __name__ == '__main__':
# Model and part
m_name='Model-1'
m = mdb.models[m_name]
m.setValues(noPartsInputFile=ON)
l = 1.

# ------------------------------------------------------------------------------
# define geometry for RVE
l = 1.
nr_elements = 10
# ------------------------------------------------------------------------------

# define start temperature for tests
Temp_850 = 850. + kelvin

p = create_geometry(m, l)
amp_Temp = create_step(m, kelvin)
create_Amplitude(m, amp_Temp)
create_pl_material()
create_material(m, p, kelvin)
a, inst = create_assembly(m, p, m_name)
create_RPs(m,a)
create_mesh(p, l, nr_elements)
create_p_sets(p, inst, l)
create_a_sets(a, inst, l)
create_equations(m)
create_boundary(m, a)
create_predefined_field(m, a, Temp_850)
create_job(m, jobName)
```

A.2 Cold Rolling py-Script

```
# DR_axle_script Deep-Rolling-3D-Python-Script
# ------------------------------------------------------------------------
# This script generates and calculates an ABAQUS model for the deep rolling
# process of a train axle.
# The axle subdivides into a fine meshed segment and a surface of dummy elements
# to fasten the calculation of the rolling process.
#
# ------------------------------------------------------------------------
# 2021-09-06
# ------------------------------------------------------------------------
# J. Bialowas, S. Gapp, D. Otipka
# ------------------------------------------------------------------------
#
# Import of packages
from abaqus import *
from abaqusConstants import *
from caeModules import *
import numpy as np
import os
import math
import shutil
from connectorBehavior import *

session.journalOptions.setValues(replayGeometry=COORDINATE,
recoverGeometry=COORDINATE)
TOL = 1e-6

global testing
testing = True
coarse_bias = False # true if the coarse meshed (inside) partition of the axle is bias meshed

# Different variants of boundary conditions
# first two BC can be both active (Variant_3)
front_z_loose = True # True if the Boundary Condition is fixed(back) - loose(front) in z-direction
hollow_encastre = False # True if the Boundary Condition is Axle_hollow fixed

dir0=os.path.abspath('')

Mdb()

# Perspective off
session.viewports['Viewport: 1'].view.setProjection(projection=PARALLEL)

# +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Definition of Functions
# +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

# ------------------------------------------------------------------------
def get_geometry_parameters():
# Model parameters (N-mm-s)

# ------------------------------------------------------------------
# Parameters for parameter study
# ------------------------------------------------------------------
r_Axle = 180./2. # Outside radius of the axle
r_Axle_hollow = 15. # !do not use values below 0.1 mm!
# use 0.1 mm for full axle
t_Axle_sector = r_Axle - r_Axle_hollow # Thickness of axle sector
r_WRoller = 30. # Contour radius of work roller
R_WRoller = 150./2. # Outside radius of work roller
h_WRoller = r_WRoller-1. # Heigth of contour of work roller
#Set feed and load in function get_process_parameters according to Excel tool
# ------------------------------------------------------------------

# Axle
l_Axle_sector = 50. # Length of axle sector
aa_Axle_sector = 6./360.*2.*math.pi # Aperture angle of axle sector in radiant
ta_Axle_sector = 1./360.*2.*math.pi # Tilting angle of axle sector in radiant
#ta_Axle_sector = aa_Axle_sector/2.
# Hollow axle
r_Axle_hollow = r_Axle - t_Axle_sector # Inside radius of the hollow axle
a_WR_start = 2./360.*2.*math.pi # Angle where the WR starts, measured form the sector;
# If <0 it's beneath the sector; also defines the angle
# that the WR is rolling on the dummy elements

scale_analytical = 1.1 # Scale factor to compensate for non-deformable behaviour
# of analytical rigid body
```

```
# Dummy
a_dummy_min = 4./360.*2.*math.pi # minimum Angle above the axle sector that shall be covered with dummy elements
# a_dummy_min must big greater than a_WR_start and smaller than aa_Axle_sector!!
nr_layer_dummy = 1 # number of dummy layers

# Translator
distance_centerpoint = r_Axle + R_WRoller*1.1
lowerbound_Trans = distance_centerpoint - 0.5
upperbound_Trans = distance_centerpoint + 1.1

dtype = [('Name', (np.str_, 20)), ('Parameter', np.float64)]
geom_par=np.array([('r_Axle',r_Axle),('t_Axle_sector',t_Axle_sector),
('l_Axle',l_Axle_sector),('aa_Axle_sector',aa_Axle_sector),
('aa_Axle', aa_Axle_sector), ('t_Axle',t_Axle_sector),
('ta_Axle_sector',ta_Axle_sector),('r_WRoller',r_WRoller),
('R_WRoller',R_WRoller),('h_WRoller',h_WRoller),
('scale_analytical',scale_analytical),
('r_Axle_hollow', r_Axle_hollow),
('a_WR_start', a_WR_start), ('nr_layer_dummy', nr_layer_dummy),
('lowerbound_Trans',lowerbound_Trans),
('upperbound_Trans',upperbound_Trans)
]
,dtype=dtype)

return (r_Axle, t_Axle_sector, l_Axle_sector, aa_Axle_sector, ta_Axle_sector,
r_Axle_hollow, r_WRoller, R_WRoller, h_WRoller, scale_analytical,
geom_par, a_WR_start, a_dummy_min, nr_layer_dummy,
lowerbound_Trans, upperbound_Trans)
#-----------------------------------------------------------------------------
def get_process_parameters(aa_Axle_sector, ta_Axle_sector, a_WR_start,
a_dummy_min, nr_layer_dummy):
# Process parameter
#-----------------------------------------------------------------------------
# Element deletion
nr_el_del = 10 # number of element rows that shall be deleted
nr_dels_per_cyc = 1 # number of elements that shall be deleted per one turning cycle
nr_dels = int(math.ceil(float(nr_el_del)/nr_dels_per_cyc)) # number of necessary deletion processes

nr_cycles = 25 # number of turns of the deep rolling process
i_models = nr_cycles
#-----------------------------------------------------------------------------
feed = 1.16 # distance covered in axial direction by the work roller in
#0 one turn (cycle)
load_WR = -44000. # operating force of the work roller
indent_WR = (l_Axle_sector-nr_cycles*feed)/2. # indention of WR from right bottom corner of the axle
#-----------------------------------------------------------------------------
rpm = 220 # rotational velocity of the deep rolling process in
# rounds per minute
rad_per_s = rpm*2*math.pi/60 # rotational velocity in radiant per second
rolling_angle =  aa_Axle_sector #3./360.*2.*math.pi # angle of the model covered
# by the work roller in radiant
feed_v = feed/(60./rpm) # velocity of the feed dependent on rpm in mm/s
movement_type = 'controlled' # 'controlled'=rotation of the workroller is
# controlled by a boundary condition
# 'free'=rotation of the workroller is free and
# enforced due to friction and the load

# Mesh
mesh_num = 10 # variable for element numbers on the axle sector
mesh_ref_angle = 6./360.*2.*math.pi # reference angle for the element number at the edges
if testing:
#rolling_angle = 13. / 360. * 2. * math.pi
nr_cycles = 3
nr_el_del = 1
nr_dels = 1
mesh_num = 6
mesh_num_edges = int(round(mesh_num * 2. * aa_Axle_sector/mesh_ref_angle)) #*4 # element number at the edges
mesh_num_depth = int(round(mesh_num * 3.)) #*1 # element number from the surface to depth of t_Partition
mesh_num_side = int(round(mesh_num * 8.))  #*2 # element number in Z-direction
mesh_bias_ratio = 5 # bias ratio for the coarse partition

# Partition
t_Partition = 15. # depth of finely meshed section of the

# Element Type
reduced = True # True if reduced integration

# Calculation
num_cpus = 8
```

```
# Dummy
angle_per_el = aa_Axle_sector/mesh_num_edges # angle that is covered by one element
nr_dummy = int(math.ceil(a_dummy_min/angle_per_el) ) # number of dummy rows
a_dummy = angle_per_el*nr_dummy # angle of dummy elements above the axle sector
depth_per_el = t_Partition/mesh_num_depth # depth of one element
depth_dummy = nr_layer_dummy*depth_per_el # total depth of the dummy elements

# Displacement during preload-step
preload = -(1.)
# Saving parameters for future availability
dtype = [('Name', (np.str_, 20)), ('Parameter', np.float64)]
process_par = np.array([('nr_cycles', nr_cycles), ('i_models', i_models),
('feed', feed),
('nr_el_del', nr_el_del), ('nr_dels_per_cyc',nr_dels_per_cyc),
('rpm', rpm),('rad_per_s', rad_per_s),
('load_WR', load_WR),('rolling_angle', rolling_angle),
('feed_v', feed_v), ('t_Partition', t_Partition),
('num_cpus', num_cpus), ('mesh_num', mesh_num),
('coarse_bias',coarse_bias), ('mesh_bias_ratio', mesh_bias_ratio),
('nr_dummy', nr_dummy), ('a_dummy', a_dummy)],
dtype=dtype)

return nr_cycles, i_models, feed, rpm, rad_per_s, load_WR, indent_WR, rolling_angle,\
feed_v, movement_type, t_Partition, num_cpus, process_par, \
nr_el_del, nr_dels_per_cyc, nr_dels, mesh_num, mesh_num_edges, mesh_num_depth, \
mesh_num_side, mesh_bias_ratio, reduced, nr_dummy, a_dummy, depth_dummy, preload

#-----------------------------------------------------------------------------
def save_parameter(geom_par,process_par,mat_par,name_geom='geometry_parameters',
name_process='process_parameters',name_mat='material_parameters'):
par={name_geom:geom_par,name_process:process_par,name_mat:mat_par}
for i_name,i_par in par.items():
np.savetxt(i_name+'.csv', i_par, delimiter=',', fmt=['%s', '%f'])
return
#-----------------------------------------------------------------------------
def set_directory():
dir_name='DR_DAxle_{}_WR_D{}R{}_feed_{}_load_{}_nr_cycles{}'.format(str(int(r_Axle*2)),
str(int(R_WRoller*2)).replace('.','p'),str(int(r_WRoller)).replace('.','p'),str(feed).replace('.','p'),
str(abs(load_WR/1000)).replace('.','p'),str(int(nr_cycles)))

# deletes the old directory
if os.path.exists(dir_name):
# shutil.rmtree(dir_name)
os.chdir(dir_name)
dir_calc = os.path.abspath('')
if not os.path.exists('Results'):
os.mkdir('Results')
else:
# create a new directory
os.mkdir(dir_name)
# change directory
os.chdir(dir_name)
dir_calc = os.path.abspath('')
os.mkdir('Results')
return dir_name, dir_calc

#-----------------------------------------------------------------------------
def scale_WRoller(r_WRoller,R_WRoller,h_WRoller,scale_analytical):
# Scaling the geometry to compensate for non-deformable behaviour
# of analytical rigid body
r_WRoller = r_WRoller * scale_analytical
R_WRoller = R_WRoller * scale_analytical
h_WRoller = h_WRoller * scale_analytical
return r_WRoller, R_WRoller, h_WRoller

#-----------------------------------------------------------------------------
def make_node_sets(m, a, p_Axle_sector):

# This function creates the node set for the boundary condition (fixed end)
# It contains the Axle-hollow Set without the edges, because the edges nodes are coupled

n_Set_1 = p_Axle_sector.sets['Set-Axle_hollow'].nodes
p_Axle_sector.Set(name='N_Set-Axle_hollow', nodes = n_Set_1)

n_Set_2 = p_Axle_sector.sets['Set-Axle_sector_edges_hollow']. nodes
p_Axle_sector.Set(name='N_Set-Axle_hollow_edges', nodes = n_Set_2)

p_Axle_sector.SetByBoolean(name='Encastre_Set', operation=DIFFERENCE,
sets=(p_Axle_sector.sets['N_Set-Axle_hollow'],
p_Axle_sector.sets['N_Set-Axle_hollow_edges']))
```

```
# ------------------------------------------------------------------------
def make_geometry(m,r_Axle,t_Axle_sector,l_Axle_sector,aa_Axle_sector,ta_Axle_sector,
r_WRoller,R_WRoller,h_WRoller, r_Axle_hollow, a_dummy, depth_dummy):
r_dummy = r_Axle - depth_dummy
if ta_Axle_sector == aa_Axle_sector:
a_half = -ta_Axle_sector/2
else:
a_half = (aa_Axle_sector - ta_Axle_sector)/ 2
# Axle sector
s_Axle_sector = m.ConstrainedSketch(name='s_Axle_sector', sheetSize=200.0)
s_Axle_sector.ArcByCenterEnds(center=(0.0, 0.0), direction=CLOCKWISE,
point1=(math.cos(aa_Axle_sector - ta_Axle_sector) *
(r_Axle - t_Axle_sector),
math.sin(aa_Axle_sector - ta_Axle_sector) *
(r_Axle - t_Axle_sector)),
point2=(math.cos(ta_Axle_sector) * (r_Axle - t_Axle_sector),
(-1) * math.sin(ta_Axle_sector) *
(r_Axle - t_Axle_sector)))
s_Axle_sector.Line(point1=(math.cos(ta_Axle_sector) * r_Axle,
(-1) * math.sin(ta_Axle_sector) * r_Axle),
point2=(math.cos(ta_Axle_sector) * (r_Axle - t_Axle_sector),
(-1) * math.sin(ta_Axle_sector) *
(r_Axle - t_Axle_sector)))

s_Axle_sector.ArcByCenterEnds(center=(0.0, 0.0), direction=CLOCKWISE,
point1=(math.cos(aa_Axle_sector - ta_Axle_sector + a_dummy) * r_Axle,
math.sin(aa_Axle_sector - ta_Axle_sector + a_dummy) * r_Axle),
point2=(math.cos(ta_Axle_sector) * r_Axle,
(-1) * math.sin(ta_Axle_sector) * r_Axle))
s_Axle_sector.ArcByCenterEnds(center=(0.0, 0.0), direction=CLOCKWISE,
point1=(math.cos(aa_Axle_sector - ta_Axle_sector + a_dummy) * r_dummy,
math.sin(aa_Axle_sector - ta_Axle_sector + a_dummy) * r_dummy),
point2=(math.cos(aa_Axle_sector - ta_Axle_sector) * r_dummy,
math.sin(aa_Axle_sector - ta_Axle_sector) * r_dummy))

s_Axle_sector.Line(point1=(math.cos(aa_Axle_sector - ta_Axle_sector + a_dummy) * r_Axle,
math.sin(aa_Axle_sector - ta_Axle_sector + a_dummy) * r_Axle),
point2=(math.cos(aa_Axle_sector - ta_Axle_sector + a_dummy) * r_dummy,
math.sin(aa_Axle_sector - ta_Axle_sector + a_dummy) * r_dummy))
s_Axle_sector.Line(point1=(math.cos(aa_Axle_sector - ta_Axle_sector) * r_dummy,
math.sin(aa_Axle_sector - ta_Axle_sector) * r_dummy),
point2=(math.cos(aa_Axle_sector - ta_Axle_sector) * r_Axle_hollow,
math.sin(aa_Axle_sector - ta_Axle_sector) * r_Axle_hollow))

p_Axle_sector = m.Part(dimensionality=THREE_D, name='p_Axle_sector',
type=DEFORMABLE_BODY)
p_Axle_sector.BaseSolidExtrude(depth=l_Axle_sector, sketch=s_Axle_sector)
del s_Axle_sector

# Partitions
s_Partition = m.ConstrainedSketch(name='partition_sketch',
sheetSize=100.0,
transform=p_Axle_sector.MakeSketchTransform(
sketchPlane=p_Axle_sector.faces.findAt(
((r_Axle - t_Axle_sector / 2) * math.cos(a_half),
(r_Axle - t_Axle_sector / 2) * math.sin(a_half),
0.0), ),
sketchPlaneSide=SIDE1,
sketchUpEdge=p_Axle_sector.edges.findAt(((r_Axle - t_Axle_sector / 2) *
math.cos(aa_Axle_sector - ta_Axle_sector),
(r_Axle - t_Axle_sector / 2) * math.sin(
aa_Axle_sector - ta_Axle_sector),
0.0), ),
sketchOrientation=RIGHT,
origin=(0.0, 0.0, 0.0)))
p_Axle_sector.projectReferencesOntoSketch(filter=COPLANAR_EDGES,
sketch=m.sketches['partition_sketch'])
s_Partition.ArcByCenterEnds(center=(0.0, 0.0), direction=CLOCKWISE,
point1=(math.cos(aa_Axle_sector - ta_Axle_sector) * (r_Axle - t_Partition),
math.sin(aa_Axle_sector - ta_Axle_sector) * (r_Axle - t_Partition)),
point2=((-1) * math.cos(ta_Axle_sector) * (r_Axle - t_Partition),
(-1) * math.sin(ta_Axle_sector) * (r_Axle - t_Partition)))

p_Axle_sector.PartitionFaceBySketch(faces=p_Axle_sector.faces.findAt(
(((r_Axle - t_Axle_sector / 2) * math.cos(a_half),
(r_Axle - t_Axle_sector / 2) * math.sin(a_half),
0.0),)),
sketch=m.sketches['partition_sketch'],
sketchUpEdge=p_Axle_sector.edges.findAt(((r_Axle - t_Axle_sector / 2) *
```

```
math.cos(aa_Axle_sector - ta_Axle_sector),
(r_Axle - t_Axle_sector / 2) * math.sin(
aa_Axle_sector - ta_Axle_sector),
0.0), ))
del s_Partition

p_Axle_sector.PartitionCellByExtrudeEdge(cells=p_Axle_sector.cells.findAt(
((r_Axle - 1., 0.0, 0.0),)),
edges=(p_Axle_sector.edges.findAt(((r_Axle - t_Partition) * math.cos(a_half),
(r_Axle - t_Partition) * math.sin(a_half),
0.0), ),),
line=p_Axle_sector.edges.findAt((r_Axle * math.cos(ta_Axle_sector),
(-1) * r_Axle * math.sin(ta_Axle_sector),
l_Axle_sector / 3), ),
sense=FORWARD)

s_Partition_dummy = m.ConstrainedSketch(name='partition_dummy_sketch',
sheetSize=100.0, transform=p_Axle_sector.MakeSketchTransform(
sketchPlane=p_Axle_sector.faces.findAt(((r_Axle - t_Partition / 2) * math.cos(a_half),
(r_Axle - t_Partition / 2) * math.sin(a_half),
l_Axle_sector), ),
sketchPlaneSide=SIDE1,
sketchUpEdge=p_Axle_sector.edges.findAt((r_Axle, 0., l_Axle_sector), ),
sketchOrientation=RIGHT,
origin=(0.0, 0.0, 0.0)))
p_Axle_sector.projectReferencesOntoSketch(filter=COPLANAR_EDGES,
sketch=m.sketches['partition_dummy_sketch'])
s_Partition_dummy.Line(point1=(math.cos(aa_Axle_sector - ta_Axle_sector) * r_Axle,
math.sin(aa_Axle_sector - ta_Axle_sector) * r_Axle),
point2=(math.cos(aa_Axle_sector - ta_Axle_sector) * r_dummy,
math.sin(aa_Axle_sector - ta_Axle_sector) * r_dummy))

p_Axle_sector.PartitionFaceBySketch(faces=p_Axle_sector.faces.findAt(
(((r_Axle - t_Partition / 2) * math.cos(a_half),
(r_Axle - t_Partition / 2) * math.sin(a_half),
l_Axle_sector),)),
sketch=m.sketches['partition_dummy_sketch'],
sketchUpEdge=p_Axle_sector.edges.findAt((r_Axle, 0., l_Axle_sector), ), )
del s_Partition_dummy

p_Axle_sector.PartitionCellByExtrudeEdge(cells=p_Axle_sector.cells.findAt(
((r_Axle-1./1000.*r_Axle, 0.0, 0.0), )),
edges=(p_Axle_sector.edges.findAt((
math.cos(aa_Axle_sector - ta_Axle_sector) * (r_Axle + r_dummy)/2.,
math.sin(aa_Axle_sector - ta_Axle_sector) * (r_Axle + r_dummy)/2.,
l_Axle_sector), ), ),
line=p_Axle_sector.edges.findAt((r_Axle*math.cos(ta_Axle_sector),
(-1)*r_Axle*math.sin(ta_Axle_sector),
l_Axle_sector/3), ),
sense=REVERSE)

# Work Roller
s_WRoller = m.ConstrainedSketch(name='s_WRoller', sheetSize=200.0)
# Construction of work roller geometry
s_WRoller.ConstructionLine(point1=(0.0, -R_WRoller*2),
point2=(0.0, R_WRoller*2))
s_WRoller.geometry.findAt((0.0, 0.0))
s_WRoller.FixedConstraint(entity=s_WRoller.geometry.findAt((0.0, 0.0), ))
s_WRoller.ArcByCenterEnds(center=(R_WRoller - r_WRoller, 0.0),
direction=CLOCKWISE,
point1=(R_WRoller - r_WRoller + math.sqrt(r_WRoller *
r_WRoller - h_WRoller * h_WRoller), h_WRoller),
point2=(R_WRoller - r_WRoller + math.sqrt(r_WRoller *
r_WRoller - h_WRoller * h_WRoller), -h_WRoller))
p_WRoller = m.Part(dimensionality=THREE_D, name='p_WRoller',
type=ANALYTIC_RIGID_SURFACE)
p_WRoller.AnalyticRigidSurfRevolve(sketch=s_WRoller)
del s_WRoller

return p_Axle_sector, p_WRoller
#-------------------------------------------------------------------------
def make_feed_compensation(m, p_Axle_sector, r_Axle, l_Axle_sector, ta_Axle_sector,
aa_Axle_sector, a_dummy, feed):
# cut sides for the compensation of the feed
datum_front = p_Axle_sector.DatumPlaneByPrincipalPlane(offset=r_Axle + 1.,
principalPlane=YZPLANE)
l_comp = feed * (1 - (aa_Axle_sector + a_dummy) / (2 * math.pi))
h_all = r_Axle * (math.sin(aa_Axle_sector + a_dummy - ta_Axle_sector) - math.sin(-ta_Axle_sector))
a_comp = math.atan(l_comp/h_all)
s_Axle_feed_comp = m.ConstrainedSketch(name='s_Axle_feed_compensation',
```

```
sheetSize=200.,
transform=p_Axle_sector.MakeSketchTransform(
sketchPlane=p_Axle_sector.datums[datum_front.id],
sketchPlaneSide=SIDE1,
sketchUpEdge=p_Axle_sector.edges.findAt((r_Axle, 0., 0.0), ),
sketchOrientation=RIGHT,
origin=(r_Axle, 0, l_Axle_sector / 2)))
p_Axle_sector.projectReferencesOntoSketch(filter=COPLANAR_EDGES,
sketch=s_Axle_feed_comp)

s_Axle_feed_comp.Line(point1=(l_Axle_sector / 2,
r_Axle * math.sin(-ta_Axle_sector + aa_Axle_sector + a_dummy)),
point2=(l_Axle_sector / 2,
r_Axle * math.sin(-ta_Axle_sector)))
s_Axle_feed_comp.Line(point1=(l_Axle_sector / 2,
r_Axle * math.sin(-ta_Axle_sector)),
point2=(l_Axle_sector / 2 - l_comp,
r_Axle * math.sin(-ta_Axle_sector)))
s_Axle_feed_comp.Line(point1=(l_Axle_sector / 2 - l_comp,
r_Axle * math.sin(-ta_Axle_sector)),
point2=(l_Axle_sector / 2,
r_Axle * math.sin(-ta_Axle_sector + aa_Axle_sector + a_dummy)))

s_Axle_feed_comp.Line(point1=(-l_Axle_sector / 2,
r_Axle * math.sin(-ta_Axle_sector + aa_Axle_sector + a_dummy)),
point2=(-l_Axle_sector / 2,
r_Axle * math.sin(-ta_Axle_sector)))
s_Axle_feed_comp.Line(point1=(-l_Axle_sector / 2,
r_Axle * math.sin(-ta_Axle_sector)),
point2=(-l_Axle_sector / 2 + l_comp,
r_Axle * math.sin(-ta_Axle_sector + aa_Axle_sector + a_dummy)))
s_Axle_feed_comp.Line(point1=(-l_Axle_sector / 2 + l_comp,
r_Axle * math.sin(-ta_Axle_sector + aa_Axle_sector + a_dummy)),
point2=(-l_Axle_sector / 2,
r_Axle * math.sin(-ta_Axle_sector + aa_Axle_sector + a_dummy)))

p_Axle_sector.CutExtrude(flipExtrudeDirection=OFF,
sketch=s_Axle_feed_comp,
sketchOrientation=RIGHT,
sketchPlane=p_Axle_sector.datums[datum_front.id],
sketchPlaneSide=SIDE1,
sketchUpEdge=p_Axle_sector.edges.findAt((r_Axle, 0., 0.0), ))
return l_comp, h_all, a_comp
#-----------------------------------------------------------------------------
def make_datum_planes(p_WRoller, h_WRoller, p_Axle_sector, indent_WR, l_comp):
# Assigning datum planes to variables for further use
# (eg. for reading 'id' of instances)
dat_WR_XZ_h = p_WRoller.DatumPlaneByPrincipalPlane(offset=indent_WR+l_comp,
principalPlane=XZPLANE)
dat_Ax_XY_0 = p_Axle_sector.DatumPlaneByPrincipalPlane(offset=0.0,
principalPlane=XYPLANE)
return dat_WR_XZ_h, dat_Ax_XY_0
#-----------------------------------------------------------------------------
def make_instances(m, p_Axle_sector, p_WRoller, r_Axle):
# Instances
a = m.rootAssembly
a.DatumCsysByDefault(CARTESIAN)
inst_Axle_sector = a.Instance(dependent=ON, name='p_Axle_sector-1', part=p_Axle_sector)
inst_WRoller = a.Instance(dependent=ON, name='p_WRoller-1', part=p_WRoller)
a.features['p_Axle_sector-1'].resume()

# Create Datum Axes
dat_axis_x = a.DatumAxisByPrincipalAxis(principalAxis=XAXIS)
dat_axis_y = a.DatumAxisByPrincipalAxis(principalAxis=YAXIS)
dat_axis_z = a.DatumAxisByPrincipalAxis(principalAxis=ZAXIS)

# Surfaces on Sector
a.Surface(name='Surf_Sector_outside',
side1Faces=inst_Axle_sector.faces.findAt(((math.cos((aa_Axle_sector - ta_Axle_sector) / 2.) * (r_Axle),
math.sin((aa_Axle_sector - ta_Axle_sector) / 2.) * (r_Axle),
l_Axle_sector / 2.),),
((math.cos(aa_Axle_sector - ta_Axle_sector + a_dummy/2.) * (r_Axle),
math.sin(aa_Axle_sector - ta_Axle_sector + a_dummy/2.) * (r_Axle),
l_Axle_sector / 2.),),
))

# Surface on WRoller
a.Surface(name='Surf_WRoller',
side1Faces=a.instances['p_WRoller-1'].faces.findAt(((
R_WRoller,
```

```
0.0,
0.0), )))
return a, inst_Axle_sector, inst_WRoller, dat_axis_x, dat_axis_y, dat_axis_z
# ------------------------------------------------------------------------
def get_edges(part, list_coords):
counter = 0
for coords in list_coords:
edge = part.edges.getClosest((coords,),)[0][0]
index = edge.index
if counter==0:
edges = part.edges[index:index+1]
else:
edges = edges.__add__(part.edges[index:index+1])
counter = counter + 1
return edges
# ------------------------------------------------------------------------
def get_faces(part, list_coords):
counter = 0
for coords in list_coords:
face = part.faces.getClosest((coords,),)[0][0]
index = face.index
if counter==0:
faces = part.faces[index:index+1]
else:
faces = faces.__add__(part.faces[index:index+1])
counter = counter + 1
return faces
# ------------------------------------------------------------------------
def make_p_sets(p_Axle_sector, r_Axle, t_Axle_sector, l_Axle_sector,
a_dummy, depth_dummy, l_comp, h_all, a_comp):
if ta_Axle_sector == aa_Axle_sector:
a_half = -ta_Axle_sector/2
else:
a_half = (aa_Axle_sector - ta_Axle_sector)/ 2.

# Compensation in z-direction of tilting angle
# and radial dependence
h_top = r_Axle*math.tan(a_dummy+aa_Axle_sector)
z_comp_edges = l_comp*math.tan(a_half)/h_top
z_comp_r = l_comp/h_top*(h_top-(r_Axle)*math.tan(a_half))
z_comp_r_t = l_comp/h_top*(h_top-(r_Axle - t_Axle_sector)*math.tan(a_half))
z_comp_r_p = l_comp/h_top*(h_top-(r_Axle-t_Partition)*math.tan(a_half))

z_comp_t_Axle_sector = l_comp/h_top*(h_top-(r_Axle)*math.tan(aa_Axle_sector - ta_Axle_sector))

# Sets
p_Axle_sector.Set(cells=p_Axle_sector.cells[:], name='all_Axle_sector')
p_Axle_sector.Set(cells=p_Axle_sector.cells.findAt(
((r_Axle * math.cos(a_half),
r_Axle * math.sin(a_half),
l_Axle_sector/2.), ),
((r_Axle_hollow * math.cos(a_half),
r_Axle_hollow * math.sin(a_half),
l_Axle_sector/2.), ), ),
name='Set-Axle_sector')
p_Axle_sector.Set(cells=p_Axle_sector.cells.findAt(
(((r_Axle_hollow + 1./1000.*r_Axle_hollow) * math.cos(a_half),
(r_Axle_hollow + 1./1000.*r_Axle_hollow) * math.sin(a_half),
l_Axle_sector / 2.),),
),
name='Set-Axle_elastic')
p_Axle_sector.Set(cells=p_Axle_sector.cells.findAt(
(((r_Axle - 1./1000.*r_Axle) * math.cos(a_half),
(r_Axle - 1./1000.*r_Axle)  * math.sin(a_half),
l_Axle_sector / 2.),),
),
name='Set-Axle_elastic_plastic')

list_coords_side_back = (
((r_Axle - 1./1000.*r_Axle) * math.cos(a_half),
(r_Axle - 1./1000.*r_Axle) * math.sin(a_half),
0.),
((r_Axle_hollow + 1./1000.*r_Axle_hollow) * math.cos(a_half),
(r_Axle_hollow + 1./1000.*r_Axle_hollow) * math.sin(a_half),
0.)
)
faces_back = get_faces(p_Axle_sector, list_coords_side_back)
set_surf_back = p_Axle_sector.Set(
faces=faces_back,
name='Set-Axle_side_back')
```

```
list_coords_side_front = (
((r_Axle - 1./1000.*r_Axle) * math.cos(a_half),
(r_Axle - 1./1000.*r_Axle) * math.sin(a_half),
l_Axle_sector),
((r_Axle_hollow + 1./1000.*r_Axle_hollow) * math.cos(a_half),
(r_Axle_hollow + 1./1000.*r_Axle_hollow) * math.sin(a_half),
l_Axle_sector)
)
faces_front = get_faces(p_Axle_sector, list_coords_side_front)
set_surf_front = p_Axle_sector.Set(
faces=faces_front,
name='Set-Axle_side_front')

p_Axle_sector.Set(cells=p_Axle_sector.cells.findAt(
((r_Axle * math.cos(aa_Axle_sector - ta_Axle_sector + a_dummy),
r_Axle * math.sin(aa_Axle_sector - ta_Axle_sector + a_dummy),
l_Axle_sector/2.), ), ),
name='Set-Dummy')

list_coords_bottom = (
(math.cos(ta_Axle_sector) * (r_Axle-t_Partition/3),
(-1)*math.sin(ta_Axle_sector) * (r_Axle-t_Partition/3),
l_Axle_sector),
(math.cos(ta_Axle_sector) * (r_Axle-t_Partition/3),
(-1)*math.sin(ta_Axle_sector) * (r_Axle-t_Partition/3),
0.0)
)
edges_bottom = get_edges(p_Axle_sector, list_coords_bottom)
p_Axle_sector.Set(edges=edges_bottom,
name='Set-Axle_t_sector_bottom')

list_coords_top = (
(math.cos(aa_Axle_sector - ta_Axle_sector) *(r_Axle - t_Partition / 3),
math.sin(aa_Axle_sector - ta_Axle_sector) * (r_Axle - t_Partition / 3),
l_Axle_sector),
(math.cos(aa_Axle_sector - ta_Axle_sector) * (r_Axle - t_Partition / 3),
math.sin(aa_Axle_sector - ta_Axle_sector) * (r_Axle - t_Partition / 3),
0.0),
)
edges_top = get_edges(p_Axle_sector, list_coords_top)
p_Axle_sector.Set(edges=edges_top,
name='Set-Axle_t_sector_top')

list_coords_bias_sides = (
(math.cos(aa_Axle_sector - ta_Axle_sector) * r_Axle,
math.sin(aa_Axle_sector - ta_Axle_sector) * r_Axle,
l_Axle_sector / 2),
(math.cos(aa_Axle_sector - ta_Axle_sector) * (r_Axle - t_Partition),
math.sin(aa_Axle_sector - ta_Axle_sector) * (r_Axle - t_Partition),
l_Axle_sector / 2),
(math.cos(aa_Axle_sector - ta_Axle_sector) * (r_Axle - t_Axle_sector),
math.sin(aa_Axle_sector - ta_Axle_sector) * (r_Axle - t_Axle_sector),
l_Axle_sector / 2),
(math.cos(ta_Axle_sector) * r_Axle,
(-1) * math.sin(ta_Axle_sector) * r_Axle,
l_Axle_sector / 2),
(math.cos(ta_Axle_sector) * (r_Axle - t_Partition),
(-1) * math.sin(ta_Axle_sector) * (r_Axle - t_Partition),
l_Axle_sector / 2),
(math.cos(ta_Axle_sector) * (r_Axle - t_Axle_sector),
(-1) * math.sin(ta_Axle_sector) * (r_Axle - t_Axle_sector),
l_Axle_sector / 2),
(math.cos(aa_Axle_sector - ta_Axle_sector + a_dummy) * r_Axle,
math.sin(aa_Axle_sector - ta_Axle_sector + a_dummy) * r_Axle,
l_Axle_sector / 2),
(math.cos(aa_Axle_sector - ta_Axle_sector + a_dummy) * (r_Axle - depth_dummy),
math.sin(aa_Axle_sector - ta_Axle_sector + a_dummy) * (r_Axle - depth_dummy),
l_Axle_sector / 2),
(math.cos(aa_Axle_sector - ta_Axle_sector) * (r_Axle - depth_dummy),
math.sin(aa_Axle_sector - ta_Axle_sector) * (r_Axle - depth_dummy),
l_Axle_sector / 2),
)
edges_bias_sides = get_edges(p_Axle_sector, list_coords_bias_sides)
p_Axle_sector.Set(edges=edges_bias_sides,
name='Set-Axle_bias_sides')

list_coords_bias_edges = (
(math.cos(a_half) * r_Axle,
math.sin(a_half) * r_Axle,
z_comp_r),
```

```
(math.cos(a_half) * r_Axle,
math.sin(a_half) * r_Axle,
l_Axle_sector-l_comp +z_comp_r),
(math.cos(a_half) * (r_Axle - t_Axle_sector),
math.sin(a_half) * (r_Axle - t_Axle_sector),
z_comp_r_t),
(math.cos(a_half) * (r_Axle - t_Axle_sector),
math.sin(a_half) * (r_Axle - t_Axle_sector),
l_Axle_sector-l_comp +z_comp_r_t),
(math.cos(a_half) * (r_Axle - t_Partition),
math.sin(a_half) * (r_Axle - t_Partition),
z_comp_r_p),
(math.cos(a_half) * (r_Axle - t_Partition),
math.sin(a_half) * (r_Axle - t_Partition),
l_Axle_sector-l_comp +z_comp_r_p),
)

edges_bias_edges = get_edges(p_Axle_sector, list_coords_bias_edges)
p_Axle_sector.Set(edges=edges_bias_edges,
name='Set-Axle_bias_edges')

list_coords_bias_coarse = (
(math.cos(aa_Axle_sector - ta_Axle_sector) * (
r_Axle - (2 * t_Axle_sector + t_Partition) / 3),
math.sin(aa_Axle_sector - ta_Axle_sector) * (
r_Axle - (2 * t_Axle_sector + t_Partition) / 3),
l_Axle_sector),
(math.cos(aa_Axle_sector - ta_Axle_sector) * (r_Axle - (2 * t_Axle_sector + t_Partition) / 3),
math.sin(aa_Axle_sector - ta_Axle_sector) * (r_Axle - (2 * t_Axle_sector + t_Partition) / 3),
0.0),
(math.cos(ta_Axle_sector) * (r_Axle - (2 * t_Axle_sector + t_Partition) / 3),
(-1) * math.sin(ta_Axle_sector) * (r_Axle - (2 * t_Axle_sector + t_Partition) / 3),
0.0),
(math.cos(ta_Axle_sector) * (r_Axle - (2 * t_Axle_sector + t_Partition) / 3),
(-1) * math.sin(ta_Axle_sector) * (r_Axle - (2 * t_Axle_sector + t_Partition) / 3),
l_Axle_sector),
)
edges_bias_coarse = get_edges(p_Axle_sector, list_coords_bias_coarse)
p_Axle_sector.Set(edges=edges_bias_coarse,
name='Set-Axle_bias_coarse')

list_coords_Dummy_bias_edges = (
(math.cos(aa_Axle_sector - ta_Axle_sector + a_dummy / 2) * r_Axle,
math.sin(aa_Axle_sector - ta_Axle_sector + a_dummy / 2) * r_Axle,
0.0),
(math.cos(aa_Axle_sector - ta_Axle_sector + a_dummy / 2) * r_Axle,
math.sin(aa_Axle_sector - ta_Axle_sector + a_dummy / 2) * r_Axle,
l_Axle_sector),
(math.cos(aa_Axle_sector - ta_Axle_sector + a_dummy / 2) * (r_Axle - depth_dummy),
math.sin(aa_Axle_sector - ta_Axle_sector + a_dummy / 2) * (r_Axle - depth_dummy),
0.0),
(math.cos(aa_Axle_sector - ta_Axle_sector + a_dummy / 2) * (r_Axle - depth_dummy),
math.sin(aa_Axle_sector - ta_Axle_sector + a_dummy / 2) * (r_Axle - depth_dummy),
l_Axle_sector),
)
edges_Dummy_bias_edges = get_edges(p_Axle_sector, list_coords_Dummy_bias_edges)
p_Axle_sector.Set(edges=edges_Dummy_bias_edges,
name='Set-Dummy_bias_edges')

list_coords_Dummy_t_sector = (
(math.cos(aa_Axle_sector - ta_Axle_sector) * (r_Axle - depth_dummy / 3),
math.sin(aa_Axle_sector - ta_Axle_sector) * (r_Axle - depth_dummy / 3),
l_Axle_sector - l_comp + z_comp_t_Axle_sector),
(math.cos(aa_Axle_sector - ta_Axle_sector) * (r_Axle - depth_dummy / 3),
math.sin(aa_Axle_sector - ta_Axle_sector) * (r_Axle - depth_dummy / 3),
z_comp_t_Axle_sector),
)
edges_Dummy_t_sector = get_edges(p_Axle_sector, list_coords_Dummy_t_sector)
p_Axle_sector.Set(edges=edges_Dummy_t_sector,
name='Set-Dummy_t_sector')
return

# ----------------------------------------------------------------------
def make_a_sets(a, inst_Axle_sector, p_WRoller,r_Axle,R_WRoller,h_WRoller,
ta_Axle_sector, a_WR_start, dat_Ax_XY_0,dat_WR_XZ_h,
indent_WR, l_comp):
# Assembly sets
# Reference point definition
rp_RC = a.ReferencePoint(point=(0.0, 0.0, 0.0))
a.features.changeKey(fromName=rp_RC.name, toName='RP-Rolling-Centre')
```

```
rp_H = a.ReferencePoint(point=((r_Axle + R_WRoller)*cos(a_WR_start-ta_Axle_sector),
(r_Axle + R_WRoller)*sin(a_WR_start-ta_Axle_sector),
0.0))
a.features.changeKey(fromName=rp_H.name, toName='RP-Hinge')
rp_WR = a.ReferencePoint(point=((r_Axle + R_WRoller)*cos(a_WR_start-ta_Axle_sector),
(r_Axle + R_WRoller)*sin(a_WR_start-ta_Axle_sector),
h_WRoller * 2))
a.features.changeKey(fromName=rp_WR.name, toName='RP-WR')
set_Rolling_Centre = a.Set(name='Set-RP-Rolling-Centre',
referencePoints=(a.referencePoints[rp_RC.id],))
# Creating sets of reference points
a.Set(name='Set-RP-WR',
referencePoints=(a.referencePoints[rp_WR.id],))
a.Set(name='Set-RP-Hinge',
referencePoints=(a.referencePoints[rp_H.id],))
dat_plane_WR=a.DatumPlaneByThreePoints(point1=a.referencePoints[rp_RC.id],
point2=a.referencePoints[rp_WR.id],
point3=a.referencePoints[rp_H.id])
a.features.changeKey(fromName=dat_plane_WR.name, toName='Datum-WR')

#-------------------------------------------------------------------------
# Positioning of instances
a.FaceToFace(clearance=0.0,
fixedPlane=inst_Axle_sector.datums[dat_Ax_XY_0.id], flip=ON,
movablePlane=inst_WRoller.datums[dat_WR_XZ_h.id])
pt_out_center_WR = p_WRoller.DatumPointByCoordinate(coords=(0.0, indent_WR+l_comp, 0.0))
a.CoincidentPoint(fixedPoint=a.referencePoints[rp_H.id],
movablePoint=a.instances['p_WRoller-1'].datums[pt_out_center_WR.id])

# Axle sector
if ta_Axle_sector == aa_Axle_sector:
a_half = -ta_Axle_sector/2
else:
a_half = (aa_Axle_sector - ta_Axle_sector)/ 2
list_coords_inside_back = (
(r_Axle_hollow * math.cos(a_half),
r_Axle_hollow * math.sin(a_half),
0.),
)
edges_inside_back = get_edges(inst_Axle_sector, list_coords_inside_back)
set_geom_inside_back = a.Set(edges=edges_inside_back,
name='Set-Sector_inside_back')

list_coords_inside_front = (
(r_Axle_hollow * math.cos(a_half),
r_Axle_hollow * math.sin(a_half),
l_Axle_sector),
)
edges_inside_front = get_edges(inst_Axle_sector, list_coords_inside_front)
set_geom_inside_front = a.Set(edges=edges_inside_front,
name='Set-Sector_inside_front')

set_geom_angle_top = a.Set(
faces=inst_Axle_sector.faces.findAt(
(((r_Axle - depth_dummy - 1./1000.*r_Axle) * math.cos(aa_Axle_sector - ta_Axle_sector),
(r_Axle -depth_dummy - 1./1000.*r_Axle) * math.sin(aa_Axle_sector - ta_Axle_sector),
l_Axle_sector/2.),),
(((r_Axle_hollow + 1./1000.*r_Axle_hollow) * math.cos(aa_Axle_sector - ta_Axle_sector),
(r_Axle_hollow + 1./1000.*r_Axle_hollow) * math.sin(aa_Axle_sector - ta_Axle_sector),
l_Axle_sector / 2.),),),
name='Set-Sector_angle_top')
set_geom_angle_bottom = a.Set(
faces=inst_Axle_sector.faces.findAt(
(((r_Axle - 1./1000.*r_Axle) * math.cos(-ta_Axle_sector),
(r_Axle - 1./1000.*r_Axle) * math.sin(-ta_Axle_sector),
l_Axle_sector/2.),),
(((r_Axle_hollow + 1./1000.*r_Axle_hollow) * math.cos(-ta_Axle_sector),
(r_Axle_hollow + 1./1000.*r_Axle_hollow) * math.sin(-ta_Axle_sector),
l_Axle_sector / 2.),),),
name='Set-Sector_angle_bottom')

set_node_inside_back = a.Set(
nodes=set_geom_inside_back.nodes,
name='N_Set-inside_back')
set_node_inside_front = a.Set(
nodes=set_geom_inside_front.nodes,
name='N_Set-inside_front')
set_node_angle_top = a.Set(
nodes=set_geom_angle_top.nodes,
name='N_Set-angle_top')
```

```
set_node_angle_bottom = a.Set(
nodes=set_geom_angle_bottom.nodes,
name='N_Set-angle_bottom')

a.SetByBoolean(name='N_Set-Sector_angle_top',
operation=DIFFERENCE,
sets=(set_node_angle_top,
set_node_inside_back,
set_node_inside_front))
a.SetByBoolean(name='N_Set-Sector_angle_bottom',
operation=DIFFERENCE,
sets=(set_node_angle_bottom,
set_node_inside_back,
set_node_inside_front))
a.deleteSets(setNames=('N_Set-inside_back', 'N_Set-inside_front',
'N_Set-angle_top', 'N_Set-angle_bottom'))

return rp_RC, rp_H, rp_WR
# ------------------------------------------------------------------------
def make_mat_sections(m, mat_name, mat_name_dummy, youngs_modulus,
youngs_modulus_dummy, nu, nu_dummy,
yield_stress, c1, gamma1, c2,
gamma2, c3, gamma3, equiv_stress,
q_infinity, hardening_b):
mat_name_elastic = mat_name + '_elastic'
mat_name_ideal_plastic = mat_name + '_ideal_plastic'

# Material
mat = m.Material(name=mat_name)
mat_ideal_pl = m.Material(name=mat_name_ideal_plastic)
mat_el = m.Material(name=mat_name_elastic)
mat_dummy = m.Material(name=mat_name_dummy)

# elastic
mat.Elastic(table=((youngs_modulus, nu),))
mat_ideal_pl.Elastic(table=((youngs_modulus, nu),))
mat_el.Elastic(table=((youngs_modulus, nu),))
mat_dummy.Elastic(table=((youngs_modulus_dummy, nu_dummy),))

# plastic
mat.Plastic(dataType=PARAMETERS,
hardening=COMBINED,
numBackstresses=3,
table=((yield_stress, c1, gamma1,
c2, gamma2, c3, gamma3),))
mat.plastic.CyclicHardening(parameters=ON,
table=((equiv_stress, q_infinity, hardening_b),))
mat_ideal_pl.Plastic(table=((yield_stress, 0.0),
(yield_stress, 0.1)))

# Section and material assignment
# chaboche type section
m.HomogeneousSolidSection(material=mat_name,
name='Section-EA4T',
thickness=None)
m.HomogeneousSolidSection(material=mat_name_elastic,
name='Section-EA4T_elastic',
thickness=None)
m.HomogeneousSolidSection(material=mat_name_ideal_plastic,
name='Section-ideal_plastic',
thickness=None)
p_Axle_sector.SectionAssignment(offset=0.0,
offsetField='', offsetType=MIDDLE_SURFACE,
region=p_Axle_sector.sets['Set-Axle_elastic_plastic'],
sectionName='Section-EA4T',
thicknessAssignment=FROM_SECTION)
p_Axle_sector.SectionAssignment(offset=0.0,
offsetField='', offsetType=MIDDLE_SURFACE,
region=p_Axle_sector.sets['Set-Axle_elastic'],
sectionName='Section-EA4T_elastic',
thicknessAssignment=FROM_SECTION)

m.HomogeneousSolidSection(material=mat_name_dummy,
name='Section-Dummy',
thickness=None)
p_Axle_sector.SectionAssignment(offset=0.0,
offsetField='', offsetType=MIDDLE_SURFACE,
region=p_Axle_sector.sets['Set-Dummy'],
sectionName='Section-Dummy',
thicknessAssignment=FROM_SECTION)
```

```
    return

# ------------------------------------------------------------------------------
def make_rigid(a, m, inst_WRoller):
# Rigid body constraint
m.RigidBody(name='Constraint-analytical-rigid',
refPointRegion=a.sets['Set-RP-WR'],
surfaceRegion=a.surfaces['Surf_WRoller'])
return

# ------------------------------------------------------------------------------
def make_connectors(a, m, rp_RC, rp_H, rp_WR, r_Axle, R_WRoller, h_WRoller,
ta_Axle_sector, a_WR_start, lowerbound_Trans, upperbound_Trans):
# Connectors
# wire definition
a.WirePolyLine(mergeType=IMPRINT,
meshable=OFF,
points=((a.referencePoints[rp_RC.id],
a.referencePoints[rp_H.id]),))
a.Set(edges=a.edges.findAt((((r_Axle + R_WRoller - 1)*cos(a_WR_start-ta_Axle_sector),
(r_Axle + R_WRoller - 1)*sin(a_WR_start-ta_Axle_sector), 0.0),)),
name='Wire-Translator')

a.WirePolyLine(mergeType=IMPRINT, meshable=OFF,
points=((a.referencePoints[rp_H.id],
a.referencePoints[rp_WR.id]),))
a.Set(edges=a.edges.findAt((((r_Axle + R_WRoller)*cos(a_WR_start-ta_Axle_sector),
(r_Axle + R_WRoller)*sin(a_WR_start-ta_Axle_sector),
h_WRoller / 3),)),
name='Wire-Hinge')

# Connector definition and CSYS definition
m.ConnectorSection(assembledType=TRANSLATOR, name='ConnSect-Translator')
m.ConnectorSection(assembledType=HINGE, name='ConnSect-Hinge')
csys_translator = a.DatumCsysByThreePoints(coordSysType=
CARTESIAN, name='Datum␣csys-Translator',
origin=a.referencePoints[rp_RC.id],
point1=a.referencePoints[rp_H.id],
point2=a.referencePoints[rp_H2.id])
a.SectionAssignment(region=a.sets['Wire-Translator'],
sectionName='ConnSect-Translator')
a.ConnectorOrientation(localCsys1=a.datums[csys_translator.id],
region=a.allSets['Wire-Translator'])
csys_hinge = a.DatumCsysByThreePoints(coordSysType=
CARTESIAN, name='Datum␣csys-Hinge',
origin=a.referencePoints[rp_H.id],
point1=a.referencePoints[rp_H2.id],
point2=a.referencePoints[rp_RC.id])
a.SectionAssignment(region=a.sets['Wire-Hinge'],
sectionName='ConnSect-Hinge')
a.ConnectorOrientation(localCsys1=a.datums[csys_hinge.id],
region=a.allSets['Wire-Hinge'])
m.sections['ConnSect-Translator'].setValues(
behaviorOptions=(ConnectorDamping(table=((.03,),),
independentComponents=(), components=(1,)),
ConnectorStop(minMotion=lowerbound_Trans,
maxMotion=upperbound_Trans, components=(1, ))
))
m.sections['ConnSect-Translator'].behaviorOptions[0].ConnectorOptions()
return

# ------------------------------------------------------------------------------
def make_mesh(p_Axle_sector, mesh_num_edges, mesh_num_depth, mesh_num_side, mesh_bias_ratio, reduced):
# Mesh of sector
p_Axle_sector.seedEdgeByNumber(constraint=FIXED,
edges=p_Axle_sector.sets['Set-Axle_bias_sides'].edges,
number=mesh_num_side)
p_Axle_sector.seedEdgeByNumber(constraint=FIXED,
edges=p_Axle_sector.sets['Set-Axle_bias_edges'].edges,
number=mesh_num_edges)
p_Axle_sector.seedEdgeByNumber(constraint=FIXED,
edges=p_Axle_sector.sets['Set-Axle_t_sector_top'].edges,
number=mesh_num_depth - nr_layer_dummy)
p_Axle_sector.seedEdgeByNumber(constraint=FIXED,
edges=p_Axle_sector.sets['Set-Axle_t_sector_bottom'].edges,
number=mesh_num_depth)
if coarse_bias:
p_Axle_sector.seedEdgeByBias(biasMethod=SINGLE,
constraint=FIXED,
end1Edges=(p_Axle_sector.sets['Set-Axle_bias_coarse'].edges[0],
```

```
p_Axle_sector.sets['Set-Axle_bias_coarse'].edges[2]),
end2Edges=(p_Axle_sector.sets['Set-Axle_bias_coarse'].edges[1],
p_Axle_sector.sets['Set-Axle_bias_coarse'].edges[3]),
number=mesh_num_depth/2,
ratio=mesh_bias_ratio)
else:
p_Axle_sector.seedEdgeByNumber(constraint=FIXED,
edges=p_Axle_sector.sets['Set-Axle_bias_coarse'].edges,
number=mesh_num_depth/2)

p_Axle_sector.seedEdgeByNumber(constraint=FIXED,
edges=p_Axle_sector.sets['Set-Dummy_bias_edges'].edges,
number=nr_dummy)
p_Axle_sector.seedEdgeByNumber(constraint=FIXED,
edges=p_Axle_sector.sets['Set-Dummy_t_sector'].edges,
number=nr_layer_dummy)
p_Axle_sector.generateMesh()

# Assign Element Type
if reduced:
p_Axle_sector.setElementType(elemTypes=(
mesh.ElemType(elemCode=C3D8R, elemLibrary=STANDARD,
secondOrderAccuracy=OFF, distortionControl=DEFAULT),
mesh.ElemType(elemCode=C3D6, elemLibrary=STANDARD),
mesh.ElemType(elemCode=C3D4, elemLibrary=STANDARD)),
regions=(p_Axle_sector.sets['all_Axle_sector']))
else:
p_Axle_sector.setElementType(elemTypes=(
mesh.ElemType(elemCode=C3D8, elemLibrary=STANDARD,
secondOrderAccuracy=OFF, distortionControl=DEFAULT),
mesh.ElemType(elemCode=C3D6, elemLibrary=STANDARD),
mesh.ElemType(elemCode=C3D4, elemLibrary=STANDARD)),
regions=(p_Axle_sector.sets['all_Axle_sector']))
return

# ------------------------------------------------------------------------
def make_el_sets(p_Axle_sector, nr_el_del, nr_dels, mesh_num_depth, mesh_num_side,
l_comp, a_comp):
a_top = aa_Axle_sector - ta_Axle_sector + a_dummy # angle of the top edge
h = r_Axle * sin(a_top)
hyp1 = h * sin(a_comp)
z1 = hyp1 * cos(a_comp)

# Element Sets Axle_sector
el_size_z = (l_Axle_sector-l_comp) / mesh_num_side # element length in Z-direction
el_size_r = t_Partition/mesh_num_depth # radial height of one element
nr_dels_per_layer = mesh_num_side # number of deletions per layer

p_Axle_sector.Set(elements=p_Axle_sector.elements[:],
name='Set-all_el')

for i_del in range(1, nr_dels+1):
p_Axle_sector.Set(elements=p_Axle_sector.elements.getByBoundingCylinder(center1=(0., 0., 0.),
center2=(0., 0., l_Axle_sector),
radius=r_Axle-i_del*el_size_r+el_size_r*0.05),
name='Set-remaining_el_Layer_'+str(i_del).zfill(2))
if i_del==1:
p_Axle_sector.SetByBoolean(name='Set-el_del_Layer_'+str(i_del).zfill(2),
operation=DIFFERENCE,
sets=(p_Axle_sector.sets['Set-all_el'],
p_Axle_sector.sets['Set-remaining_el_Layer_'+str(i_del).zfill(2)]))
else:
p_Axle_sector.SetByBoolean(name='Set-el_del_Layer_' + str(i_del).zfill(2),
operation=DIFFERENCE,
sets=(p_Axle_sector.sets['Set-remaining_el_Layer_' + str(i_del-1).zfill(2)],
p_Axle_sector.sets['Set-remaining_el_Layer_' + str(i_del).zfill(2)]))

return nr_dels_per_layer

# ------------------------------------------------------------------------
def make_step(a,m,rpm, rad_per_s, feed_v, nr_cycles, rolling_angle, nr_dels_per_layer,
initialInc=5e-05, maxInc=0.1, maxNumInc=10000,
minInc=1e-07):
timePeriodperCycle = 60./rpm
timePeriod = timePeriodperCycle*nr_cycles
timePeriodperRoll = rolling_angle/(2.*math.pi)*timePeriodperCycle
timePeriodperReturn = (1-rolling_angle/(2.*math.pi))*60./rpm

# ------------------------------------------------------------------------
# Steps
```

```
for i_cycles in range(1,nr_cycles+1):
if i_cycles==1:
m.StaticStep(initialInc=0.02, maxInc=maxInc,
matrixStorage=UNSYMMETRIC,
maxNumInc=maxNumInc, minInc=minInc, name='Pre-Load'+str(i_cycles).zfill(2),
nlgeom=ON,
previous='Initial', timePeriod=0.1,
convertSDI=CONVERT_SDI_ON)

m.StaticStep(initialInc=initialInc*10., maxInc=maxInc,
matrixStorage=UNSYMMETRIC,
maxNumInc=maxNumInc, minInc=minInc,
name='Load' + str(i_cycles).zfill(2),
nlgeom=ON,
previous='Pre-Load'+str(i_cycles).zfill(2),
timePeriod=0.1,
convertSDI=CONVERT_SDI_ON)

m.StaticStep(initialInc=timePeriodperRoll*1e-02, maxInc=timePeriodperRoll*1e-01,
matrixStorage=UNSYMMETRIC,
maxNumInc=maxNumInc, minInc=minInc,
name='Cycles'+str(i_cycles).zfill(2),
nlgeom=ON,
previous='Load'+str(i_cycles).zfill(2),
timePeriod=timePeriodperRoll)

m.StaticStep(initialInc=initialInc*1000., matrixSolver=DIRECT,
matrixStorage=UNSYMMETRIC, maxInc=maxInc,
maxNumInc=maxNumInc, minInc=minInc,
name='Unload'+str(i_cycles).zfill(2),
previous='Cycles'+str(i_cycles).zfill(2),
timePeriod=timePeriodperCycle)

m.StaticStep(initialInc=maxInc, matrixSolver=DIRECT,
matrixStorage=UNSYMMETRIC, maxInc=maxInc,
maxNumInc=maxNumInc, minInc=minInc,
name='Return'+str(i_cycles).zfill(2),
previous='Unload'+str(i_cycles).zfill(2),
timePeriod=timePeriodperReturn)

#-------------------------------------------------------------------
# Controls (WARNING: This settings are for advanced users only!)
m.steps['Pre-Load' + str(i_cycles).zfill(2)].control.setValues(allowPropagation=OFF,
discontinuous=ON,
resetDefaultValues=OFF)
m.steps['Load' + str(i_cycles).zfill(2)].control.setValues(allowPropagation=OFF,
discontinuous=ON,
resetDefaultValues=OFF)
m.steps['Cycles'+str(i_cycles).zfill(2)].control.setValues(allowPropagation=OFF,
discontinuous=ON,
resetDefaultValues=OFF)
m.steps['Unload'+str(i_cycles).zfill(2)].control.setValues()
m.steps['Return'+str(i_cycles).zfill(2)].control.setValues()
else:
m.StaticStep(initialInc=0.02, maxInc=maxInc,
matrixStorage=UNSYMMETRIC,
maxNumInc=maxNumInc, minInc=minInc,
name='Pre-Load' + str(i_cycles).zfill(2),
nlgeom=ON,
previous='Return'+str(i_cycles-1).zfill(2),
timePeriod=0.1)

m.StaticStep(initialInc=initialInc, maxInc=maxInc,
matrixStorage=UNSYMMETRIC,
maxNumInc=maxNumInc, minInc=minInc, name='Load'+str(i_cycles).zfill(2),
nlgeom=ON,
previous='Pre-Load'+str(i_cycles).zfill(2),
timePeriod=0.1)

m.StaticStep(initialInc=initialInc, maxInc=maxInc*1e-02,
matrixStorage=UNSYMMETRIC,
maxNumInc=maxNumInc, minInc=minInc, name='Cycles'+str(i_cycles).zfill(2), nlgeom=ON,
previous='Load'+str(i_cycles).zfill(2), timePeriod=timePeriodperRoll)

m.StaticStep(initialInc=initialInc*1000., matrixSolver=DIRECT,
matrixStorage=UNSYMMETRIC, maxInc=maxInc, maxNumInc=maxNumInc,
minInc=minInc, name='Unload'+str(i_cycles).zfill(2),previous='Cycles'+str(i_cycles).zfill(2),
timePeriod=timePeriodperCycle)

m.StaticStep(initialInc=maxInc, matrixSolver=DIRECT,
```

```
matrixStorage=UNSYMMETRIC, maxInc=maxInc,
maxNumInc=maxNumInc, minInc=minInc,
name='Return'+str(i_cycles).zfill(2),
previous='Unload'+str(i_cycles).zfill(2),
timePeriod=timePeriodperReturn)
#-------------------------------------------------------------------------
# Controls (WARNING: This settings are for advanced users only!)
m.steps['Pre-Load' + str(i_cycles).zfill(2)].control.setValues(allowPropagation=OFF,
discontinuous=ON,
resetDefaultValues=OFF)
m.steps['Load' + str(i_cycles).zfill(2)].control.setValues(allowPropagation=OFF,
discontinuous=ON,
resetDefaultValues=OFF)
m.steps['Cycles'+str(i_cycles).zfill(2)].control.setValues(allowPropagation=OFF,
discontinuous=ON,
resetDefaultValues=OFF)
m.steps['Unload'+str(i_cycles).zfill(2)].control.setValues()
m.steps['Return'+str(i_cycles).zfill(2)].control.setValues()
#-------------------------------------------------------------------------
# Restart Option
if nr_cycles>=10:
m.steps['Return10'].Restart(
frequency=1, numberIntervals=0, overlay=ON, timeMarks=OFF)
if nr_cycles>=20:
m.steps['Return20'].Restart(
frequency=1, numberIntervals=0, overlay=ON, timeMarks=OFF)
if nr_cycles>=30:
m.steps['Return30'].Restart(
frequency=1, numberIntervals=0, overlay=ON, timeMarks=OFF)
#-------------------------------------------------------------------------
# Steps of Element Deletion
# layer-wise deletion
m.StaticStep(initialInc=1, matrixSolver=DIRECT,
matrixStorage=UNSYMMETRIC, maxInc=1,
maxNumInc=maxNumInc, minInc=minInc,
name='Step-Element_Deletion-Layer_01',
previous='Return' + str(nr_cycles).zfill(2))

for i_del in range(2, nr_dels+1):
m.StaticStep(initialInc=1, matrixSolver=DIRECT,
matrixStorage=UNSYMMETRIC, maxInc=1,
maxNumInc=maxNumInc, minInc=minInc,
name='Step-Element_Deletion-Layer_{}'.format(str(i_del).zfill(2)),
previous='Step-Element_Deletion-Layer_{}'.format(str(i_del-1).zfill(2)))
#-------------------------------------------------------------------------
# Restart Option for Restart Steps
for i_del_steps in range(1, nr_dels + 1):
#el_last_step = sets_el_del[i_del_steps-1].elements[-1].label
m.steps['Step-Element_Deletion-Layer_{}'.format(str(i_del_steps).zfill(2))].Restart(
frequency=1,
numberIntervals=0,
overlay=ON,
timeMarks=OFF)

return timePeriodperCycle, timePeriod, timePeriodperRoll, timePeriodperReturn

#-------------------------------------------------------------------------
def make_contact(m, movement_type, mu_r=0.3):
# Contact
m.ContactProperty('IntProp-Contact_penalty')
m.ContactProperty('IntProp-Contact_old')
m.ContactProperty('IntProp-Augmented_Lagrange')
tangential = False
if tangential == True:
if movement_type == 'controlled':
mu_r = 0.1
m.interactionProperties['IntProp-Contact_penalty'].TangentialBehavior(
dependencies=0, directionality=ISOTROPIC, elasticSlipStiffness=None,
formulation=PENALTY, fraction=0.005, maximumElasticSlip=FRACTION,
pressureDependency=OFF, shearStressLimit=None, slipRateDependency=OFF,
table=((mu_r,),), temperatureDependency=OFF)
m.interactionProperties['IntProp-Contact_old'].TangentialBehavior(
dependencies=0, directionality=ISOTROPIC, elasticSlipStiffness=None,
formulation=PENALTY, fraction=0.005, maximumElasticSlip=FRACTION,
pressureDependency=OFF, shearStressLimit=None, slipRateDependency=OFF,
table=((mu_r,),), temperatureDependency=OFF)
m.interactionProperties['IntProp-Augmented_Lagrange'].TangentialBehavior(
dependencies=0, directionality=ISOTROPIC, elasticSlipStiffness=None,
formulation=PENALTY, fraction=0.005, maximumElasticSlip=FRACTION,
pressureDependency=OFF, shearStressLimit=None, slipRateDependency=OFF,
```

```
table=((mu_r,),), temperatureDependency=OFF)
else:
m.interactionProperties['IntProp-Contact_penalty'].TangentialBehavior(
dependencies=0, directionality=ISOTROPIC, elasticSlipStiffness=None,
formulation=PENALTY, fraction=0.005, maximumElasticSlip=FRACTION,
pressureDependency=OFF, shearStressLimit=None, slipRateDependency=OFF,
table=((mu_r,),), temperatureDependency=OFF)
m.interactionProperties['IntProp-Contact_old'].TangentialBehavior(
dependencies=0, directionality=ISOTROPIC, elasticSlipStiffness=None,
formulation=PENALTY, fraction=0.005, maximumElasticSlip=FRACTION,
pressureDependency=OFF, shearStressLimit=None, slipRateDependency=OFF,
table=((mu_r,),), temperatureDependency=OFF)
m.interactionProperties['IntProp-Augmented_Lagrange'].TangentialBehavior(
dependencies=0, directionality=ISOTROPIC, elasticSlipStiffness=None,
formulation=PENALTY, fraction=0.005, maximumElasticSlip=FRACTION,
pressureDependency=OFF, shearStressLimit=None, slipRateDependency=OFF,
table=((mu_r,),), temperatureDependency=OFF)

# with default constraint enforcement method (the stress from the dummy are not copied on the axle)
m.interactionProperties['IntProp-Contact_old'].NormalBehavior(
allowSeparation=ON, constraintEnforcementMethod=DEFAULT,
pressureOverclosure=HARD)

# with hard constraint enforcement method; the value of the contact stiffness must be evaluated with augment lagrange
m.interactionProperties['IntProp-Contact_penalty'].NormalBehavior(allowSeparation=ON,
clearanceAtZeroContactPressure=0.0,
constraintEnforcementMethod=PENALTY,
contactStiffness=500000.0,
contactStiffnessScaleFactor=1.0,
pressureOverclosure=HARD,
stiffnessBehavior=LINEAR)

# with augmented lagrange constraint enforcement method
m.interactionProperties['IntProp-Augmented_Lagrange'].NormalBehavior(allowSeparation=ON,
clearanceAtZeroContactPressure=0.0,
constraintEnforcementMethod=AUGMENTED_LAGRANGE,
contactStiffness=DEFAULT,
contactStiffnessScaleFactor=1.0,
pressureOverclosure=HARD)

m.SurfaceToSurfaceContactStd(adjustMethod=OVERCLOSED,
clearanceRegion=None, createStepName='Initial', datumAxis=None,
initialClearance=OMIT, interactionProperty='IntProp-Augmented_Lagrange', #'IntProp-Contact',
master=a.surfaces['Surf_WRoller'], name='Int-Surf_cont_Axle',
slave=a.surfaces['Surf_Sector_outside'],
sliding=FINITE, surfaceSmoothing=AUTOMATIC, thickness=ON)

# contact controlls for finding contact in Pre-Load Step
for i_cycles in range(1,nr_cycles+1):
m.StdContactControl(name='ContCtrl-FindingContact', stabilizeChoice=AUTOMATIC)
m.interactions['Int-Surf_cont_Axle'].setValuesInStep(
contactControls='ContCtrl-FindingContact',
stepName='Pre-Load' + str(i_cycles).zfill(2))
m.interactions['Int-Surf_cont_Axle'].setValuesInStep(
contactControls=FREED,
stepName='Load' + str(i_cycles).zfill(2))
return
#--------------------------------------------------------------------------
def make_equations(m, a, inst_Axle_sector, r_Axle, t_Partition, mesh_num_depth, mesh_num_side,
l_Axle_sector, aa_Axle_sector, ta_Axle_sector, a_dummy, l_comp, a_comp, mesh_bias_ratio):
# Node sets for the equations
r_list_fine = np.linspace(r_Axle, r_Axle-t_Partition, mesh_num_depth+1)

if coarse_bias:
bias_factor = np.array([mesh_bias_ratio**(float(i)/(mesh_num_depth/2-1)) for i in range(mesh_num_depth/2)])
l0 = (r_Axle - t_Partition - r_Axle_hollow)/np.sum(bias_factor)
r_list_coarse = r_Axle - t_Partition - l0 * np.cumsum(bias_factor)
else:
r_list_coarse = np.linspace(r_Axle-t_Partition, r_Axle_hollow, mesh_num_depth/2 + 1)[1:]
r_list = np.hstack((r_list_fine, r_list_coarse))
z_list = np.linspace(0., l_Axle_sector - l_comp, mesh_num_side+1)
h_axle = r_Axle * (math.sin(aa_Axle_sector - ta_Axle_sector) - math.sin(-ta_Axle_sector))
if a_comp==0:
h_all = h_axle
else:
h_all = l_comp / math.tan(a_comp)
z_list_bottom = np.linspace(l_comp,
l_Axle_sector,
mesh_num_side + 1)
```

```
z_list_top = np.linspace(l_comp*(1-h_axle/h_all),
l_Axle_sector - l_comp*h_axle/h_all,
mesh_num_side + 1)

h_axle_zero_top = h_all*(aa_Axle_sector-ta_Axle_sector)/(aa_Axle_sector+a_dummy)
h_axle_zero_bottom = h_all*(ta_Axle_sector)/(aa_Axle_sector+a_dummy)

row_counter = 1
node_counter = 1

for r_val in r_list:
node_counter = 1
r_comp_top = (r_Axle-r_val)/r_Axle*h_axle_zero_top*math.tan(a_comp)
r_comp_bottom = (r_Axle-r_val)/r_Axle*h_axle_zero_bottom*math.tan(a_comp)
for z_val_bottom in z_list_bottom:
# bottom Surface of the axle sector
node_label = inst_Axle_sector.nodes.getClosest((r_val * math.cos(-ta_Axle_sector),
r_val * math.sin(-ta_Axle_sector),
z_val_bottom-r_comp_bottom),).label
a.Set(name='N_Set-bottom_row_{}_node_{}'.format(str(row_counter).zfill(2),
str(node_counter).zfill(3)),
nodes=inst_Axle_sector.nodes[node_label-1:node_label])
node_counter = node_counter + 1

node_counter = 1
for z_val_top in z_list_top:
# top Surface of the axle sector
node_label = inst_Axle_sector.nodes.getClosest((r_val * math.cos(aa_Axle_sector-ta_Axle_sector),
r_val * math.sin(aa_Axle_sector-ta_Axle_sector),
z_val_top+r_comp_top), ).label
a.Set(name='N_Set-top_row_{}_node_{}'.format(str(row_counter).zfill(2),
str(node_counter).zfill(3)),
nodes=inst_Axle_sector.nodes[node_label - 1:node_label])

node_counter = node_counter + 1
row_counter = row_counter + 1

# Dummy
a_start_dummy = aa_Axle_sector-ta_Axle_sector
angle_list = np.linspace(0., a_dummy, nr_dummy+1)[1:]
row_counter_dummy = 1
node_counter_dummy = 1

for angle in angle_list:
node_counter_dummy = 1
h_val_bottom = r_Axle * (math.sin(angle - ta_Axle_sector) - math.sin(-ta_Axle_sector))
h_val_dummy = r_Axle * (math.sin(a_start_dummy + angle) - math.sin(-ta_Axle_sector))
for z_val in z_list:
z_val_bottom = z_val + l_comp * (1 - h_val_bottom/h_all)
z_val_dummy = z_val + l_comp * (1 - h_val_dummy / h_all)

# Nodes from the lower edge of the axle sector
node_label = inst_Axle_sector.nodes.getClosest((r_Axle * math.cos(-ta_Axle_sector + angle),
r_Axle * math.sin(-ta_Axle_sector + angle),
z_val_bottom), ).label
a.Set(name='N_Set-lower_edge_row_{}_node_{}'.format(str(row_counter_dummy).zfill(2),
str(node_counter_dummy).zfill(3)),
nodes=inst_Axle_sector.nodes[node_label - 1:node_label])

# Nodes on the dummy
node_label = inst_Axle_sector.nodes.getClosest((r_Axle * math.cos(a_start_dummy + angle),
r_Axle * math.sin(a_start_dummy + angle),
z_val_dummy), ).label
a.Set(name='N_Set-dummy_row_{}_node_{}'.format(str(row_counter_dummy).zfill(2),
str(node_counter_dummy).zfill(3)),
nodes=inst_Axle_sector.nodes[node_label - 1:node_label])

node_counter_dummy = node_counter_dummy + 1
row_counter_dummy = row_counter_dummy + 1

# front and back area coupling

z_list_bottom = np.linspace(l_comp,
l_Axle_sector,
mesh_num_side + 1)

z_list_top = np.linspace(l_comp*(1-h_axle/h_all),
l_Axle_sector - l_comp*h_axle/h_all,
mesh_num_side + 1)
```

```
angle_list_all_side = np.linspace(0, aa_Axle_sector, mesh_num_edges + 1)[1:-1]
r_list = np.hstack((r_list_fine, r_list_coarse))[1:]

row_counter_front = 1

if front_z_loose == False:

for r_val_2 in r_list:
node_counter_front = 1
counter_front = 0
counter_back = 0
r_comp_top_side = (r_Axle-r_val_2)/r_Axle*h_axle_zero_top*math.tan(a_comp)
r_comp_bottom_side = (r_Axle-r_val_2)/r_Axle*h_axle_zero_bottom*math.tan(a_comp)

for angle_2 in angle_list_all_side:

z_koordinate_1 = l_comp - r_comp_bottom_side
z_koordinate_2 = l_comp * (1-h_axle/h_all) + r_comp_top_side

phi_comp_list = np.linspace(z_koordinate_1, z_koordinate_2, mesh_num_edges +1)

# Front Surface of the axle sector
node_label_front = inst_Assembly.nodes.getClosest((r_val_2 * math.cos(-ta_Axle_sector + angle_2),
r_val_2 * math.sin(-ta_Axle_sector + angle_2),
phi_comp_list[counter_front]),).label
a.Set(name='N_Set-front_row_{}_node_{}'.format(str(row_counter_front).zfill(2),
str(node_counter_front).zfill(3)),
nodes=inst_Assembly.nodes[node_label_front - 1:node_label_front])

node_counter_front += 1
counter_front += 1

# Necessary for the back side sets (so the iteration of the nodes starts from 1)
node_counter_front = 1

for angle_2_back in angle_list_all_side:

# back Surface of the axle sector
node_label_back = inst_Assembly.nodes.getClosest((r_val_2 * math.cos(-ta_Axle_sector + angle_2_back),
r_val_2 * math.sin(-ta_Axle_sector + angle_2_back),
l_Axle_sector - l_comp +  phi_comp_list[counter_back]), ).label
a.Set(name='N_Set-back_row_{}_node_{}'.format(str(row_counter_front).zfill(2),
str(node_counter_front).zfill(3)),
nodes=inst_Assembly.nodes[node_label_back - 1:node_label_back])

node_counter_front += 1
counter_back += 1
row_counter_front += 1

# Internal coupling of the side edges
r_list = np.hstack((r_list_fine, r_list_coarse))[1:]
node_counter_edges = 1
row_counter_edges = 1

for r_val_side in r_list:

r_comp_top_side = (r_Axle-r_val_side)/r_Axle*h_axle_zero_top*math.tan(a_comp)
r_comp_bottom_side = (r_Axle-r_val_side)/r_Axle*h_axle_zero_bottom*math.tan(a_comp)

z_koordinate_bot= l_comp - r_comp_bottom_side
z_koordinate_top = l_comp * (1-h_axle/h_all) + r_comp_top_side

# Master edge of the assembly edges
node_label_master_edge = inst_Assembly.nodes.getClosest((r_val_side * math.cos(-ta_Axle_sector),
r_val_side * math.sin(-ta_Axle_sector),
z_koordinate_bot), ).label
a.Set(name='N_Set-Master_edge-row_{}_node_{}'.format(str(row_counter_edges).zfill(2),
str(node_counter_edges).zfill(3)),
nodes=inst_Assembly.nodes[node_label_master_edge - 1:node_label_master_edge])

# Slave nodes sets edges
node_label_slave_edge_I = inst_Assembly.nodes.getClosest((r_val_side * math.cos(-ta_Axle_sector),
r_val_side * math.sin(-ta_Axle_sector),
l_Axle_sector - l_comp  + z_koordinate_bot), ).label
node_label_slave_edge_II = inst_Assembly.nodes.getClosest((r_val_side * math.cos(-ta_Axle_sector + aa_Axle_sector),
r_val_side * math.sin(-ta_Axle_sector + aa_Axle_sector),
z_koordinate_top), ).label
node_label_slave_edge_III = inst_Assembly.nodes.getClosest((r_val_side * math.cos(-ta_Axle_sector + aa_Axle_sector),
r_val_side * math.sin(-ta_Axle_sector + aa_Axle_sector),
l_Axle_sector - l_comp + z_koordinate_top), ).label
```

```
a.Set(name='N_Set-Slave_edges_I_II_III-row_{}_node_{}'.format(str(row_counter_edges).zfill(2),
str(node_counter_edges).zfill(3)),
nodes = inst_Assembly.nodes[node_label_slave_edge_I - 1:node_label_slave_edge_I] +
inst_Assembly.nodes[node_label_slave_edge_II - 1:node_label_slave_edge_II] +
inst_Assembly.nodes[node_label_slave_edge_III - 1:node_label_slave_edge_III])

node_counter_edges += 1

print('Sets_done')

#-----------------------------------------------------------------------
# Local Coordinate Systems for DOF
dat_csys_cylindrical = a.DatumCsysByThreePoints(coordSysType=CYLINDRICAL,
line1=(1.0, 0.0, 0.0),
line2=(0.0, 1.0, 0.0),
name='Datum_Csys-Cylindrical',
origin=(0.0, 0.0, 0.0))

#-----------------------------------------------------------------------
# Definition of the equations
for dof in range(1,4):
for row in range(1, row_counter):
for node in range(1, node_counter):
m.Equation(name='Equ-Row_{}_node_{}_DOF_{}'.format(str(row).zfill(2),
str(node).zfill(3),
dof),
terms=((-1.0,'N_Set-top_row_{}_node_{}'.format(str(row).zfill(2),
str(node).zfill(3)),
dof, dat_csys_cylindrical.id), #dat_csys_top.id),
(1.0, 'N_Set-bottom_row_{}_node_{}'.format(str(row).zfill(2),
str(node).zfill(3)),
dof, dat_csys_cylindrical.id)))#dat_csys_bottom.id)))

for row in range(1, row_counter_dummy):
for node in range(1, node_counter_dummy):
m.Equation(name='Equ-Dummy_Row_{}_node_{}_DOF_{}'.format(str(row).zfill(2),
str(node).zfill(3),
dof),
terms=((-1.0, 'N_Set-dummy_row_{}_node_{}'.format(str(row).zfill(2),
str(node).zfill(3)),
dof, dat_csys_cylindrical.id),#dat_csys_top.id),
(1.0, 'N_Set-lower_edge_row_{}_node_{}'.format(str(row).zfill(2),
str(node).zfill(3)),
dof, dat_csys_cylindrical.id)))# dat_csys_bottom.id)))

for row in range(1, row_counter_front):
for node in range(1, node_counter_front):
m.Equation(name='Equ-front_back-Dummy_Row_{}_node_{}_DOF_{}'.format(str(row).zfill(2),
str(node).zfill(3),
dof),
terms=((-1.0, 'N_Set-back_row_{}_node_{}'.format(str(row).zfill(2),
str(node).zfill(3)),
dof, dat_csys_cylindrical.id),#dat_csys_top.id),
(1.0, 'N_Set-front_row_{}_node_{}'.format(str(row).zfill(2),
str(node).zfill(3)),
dof, dat_csys_cylindrical.id)))# dat_csys_bottom.id)))

for node in range(1, node_counter_edges):
m.Equation(name='Equ-edges_Row_{}_node_{}_DOF_{}'.format(str(row_counter_edges).zfill(2),
str(node).zfill(3),
dof),
terms=((-1.0, 'N_Set-Slave_edges_I_II_III-row_{}_node_{}'.format(str(row_counter_edges).zfill(2),
str(node).zfill(3)),
dof, dat_csys_cylindrical.id),#dat_csys_top.id),
(1.0, 'N_Set-Master_edge-row_{}_node_{}'.format(str(row_counter_edges).zfill(2),
str(node).zfill(3)),
dof, dat_csys_cylindrical.id)))# dat_csys_bottom.id)))

# exclude the nodes that bounded by a BC
# if side faces fix
for i_row in range(1, row_counter):
del m.constraints['Equ-Row_{}_node_{}_DOF_{}'.format(str(i_row).zfill(2), str(1).zfill(3), dof)]
del m.constraints['Equ-Row_{}_node_{}_DOF_{}'.format(str(i_row).zfill(2), str(node_counter - 1).zfill(3), dof)]

for i_row in range(1, row_counter_dummy):
del m.constraints['Equ-Dummy_Row_{}_node_{}_DOF_{}'.format(str(i_row).zfill(2), str(1).zfill(3), dof)]
del m.constraints['Equ-Dummy_Row_{}_node_{}_DOF_{}'.format(str(i_row).zfill(2), str(node_counter - 1).zfill(3), dof)]

return z_list_bottom, z_list_top, r_list, h_all, h_axle, h_axle_zero_top, h_axle_zero_bottom, r_comp_top, r_comp_bottom
#-----------------------------------------------------------------------
```

```
def make_load(m, load_WR, ta_Axle_sector, a_WR_start, nr_cycles, preload):
for i_cycles in range(1, nr_cycles+1):

m.ConcentratedForce(cf1=preload*cos(ta_Axle_sector-a_WR_start),
cf2=-preload*sin(ta_Axle_sector-a_WR_start),
createStepName='Pre-Load'+str(i_cycles).zfill(2),
name='Preload_WR_Hinge'+str(i_cycles).zfill(2),
region=a.sets['Set-RP-Hinge'],
follower=ON)
m.loads['Preload_WR_Hinge'+str(i_cycles).zfill(2)].deactivate('Load'+str(i_cycles).zfill(2))
# Load
m.ConcentratedForce(cf1=load_WR*cos(ta_Axle_sector-a_WR_start),
cf2=-load_WR*sin(ta_Axle_sector-a_WR_start),
createStepName='Load'+str(i_cycles).zfill(2),
name='load_WR_Hinge'+str(i_cycles).zfill(2),
region=a.sets['Set-RP-Hinge'],
follower=ON)
m.loads['load_WR_Hinge'+str(i_cycles).zfill(2)].deactivate('Unload'+str(i_cycles).zfill(2))

return
#-------------------------------------------------------------------------------
def make_boundary(m, a, rad_per_s, feed_v, aa_Axle_sector, ta_Axle_sector,
r_Axle_hollow, nr_cycles, timePeriodperCycle, timePeriod,
timePeriodperRoll, timePeriodperReturn, movement_type):
num_rev=(r_Axle+R_WRoller)/R_WRoller
rot_vel_WR=num_rev*2*pi/timePeriodperCycle-rad_per_s # rotational velocity of
# the WorkRoller in radiant
# per second
vr_return = rad_per_s*timePeriodperRoll/timePeriodperReturn

# Different variants of boundary conditions

# Variant_1: front side z-direction is loose
if front_z_loose == True:

m.DisplacementBC(createStepName='Initial',
name='BC-Axle_back_fix',
region=inst_Axle_sector.sets['Set-Axle_side_back'],
u1=SET, u2=SET, u3=SET, ur1=UNSET, ur2=UNSET, ur3=UNSET)
m.DisplacementBC(createStepName='Initial',
name='BC-Axle_front_Z_loose',
region=inst_Axle_sector.sets['Set-Axle_side_front'],
u1=SET, u2=SET, u3=UNSET, ur1=UNSET, ur2=UNSET, ur3=UNSET)

# Variant_2: hollow_axle is fixed
if hollow_encastre == True:

m.EncastreBC(createStepName = 'Initial',
name = 'BC-Axle_hollow_fix',
region = inst_Axle_sector.sets['Encastre_Set'])

#-------------------------------------------------------------------------------
m.VelocityBC(amplitude=UNSET, createStepName='Initial',
distributionType=UNIFORM, fieldName='', localCsys=None, name='WR_Movement',
region=a.sets['Set-RP-Rolling-Centre'],
v1=0.0, v2=0.0, v3=0.0, vr1=0.0, vr2=0.0, vr3=0.0)
for i_cycles in range(1, nr_cycles+1):
if movement_type=='free':
# Load-Step
if i_cycles==1:
m.ConnVelocityBC(amplitude=UNSET, createStepName='Pre-Load'+str(i_cycles).zfill(2),
distributionType=UNIFORM, name='ConnVel_Hinge_Load'+str(i_cycles).zfill(2),
region=a.sets['Wire-Hinge'],
v1=UNSET, v2=UNSET, v3=UNSET, vr1=0.0, vr2=UNSET, vr3=UNSET)
if i_cycles>1:
m.boundaryConditions['WR_Movement'].setValuesInStep(
stepName='Pre-Load'+str(i_cycles).zfill(2), v3=0.0, vr3=0.0)
m.boundaryConditions['ConnDisp_Trans_Unload'+str(i_cycles-1).zfill(2)].deactivate('Load'+str(i_cycles).zfill(2))

#-------------------------------------------------------------------------------
# Cycles-Step
if i_cycles==1:
m.boundaryConditions['ConnVel_Hinge_Load'+str(i_cycles).zfill(2)].deactivate('Cycles'+str(i_cycles).zfill(2))

m.boundaryConditions['WR_Movement'].setValuesInStep(
stepName='Cycles'+str(i_cycles).zfill(2), v3=feed_v, vr3=rad_per_s)

#-------------------------------------------------------------------------------
# Unload-Step
m.ConnVelocityBC(amplitude=UNSET, createStepName='Unload'+str(i_cycles).zfill(2),
```

```
distributionType=UNIFORM, name='ConnVel_Hinge_Unload'+str(i_cycles).zfill(2), region=
a.sets['Wire-Hinge'], v1=UNSET, v2=UNSET, v3=UNSET, vr1=0.0, vr2=UNSET, vr3=UNSET)
m.boundaryConditions['WR_Movement'].setValuesInStep(
stepName='Unload'+str(i_cycles).zfill(2), v3=0.0, vr3=0.0)
m.ConnDisplacementBC(amplitude=UNSET, createStepName='Unload'+str(i_cycles).zfill(2),
distributionType=UNIFORM, fixed=OFF,
name='ConnDisp_Trans_Unload'+str(i_cycles).zfill(2),
region=a.sets['Wire-Translator'],
u1=0.5, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET, ur3=UNSET)
if i_cycles<nr_cycles:
m.boundaryConditions['ConnVel_Hinge_Unload'+str(i_cycles).zfill(2)].deactivate('Cycles'+str(i_cycles+1).zfill(2))

# ------------------------------------------------------------------------
# Return-Step
m.boundaryConditions['WR_Movement'].setValuesInStep(
stepName='Return'+str(i_cycles).zfill(2), v3=feed_v, vr3=(-1)*vr_return)

if movement_type=='controlled':
# Load-Step
if i_cycles==1:
m.ConnVelocityBC(amplitude=UNSET, createStepName='Pre-Load'+str(i_cycles).zfill(2),
distributionType=UNIFORM, name='ConnVel_Hinge_Load',
region=a.sets['Wire-Hinge'],
v1=UNSET, v2=UNSET, v3=UNSET, vr1=0.0, vr2=UNSET, vr3=UNSET)
if i_cycles>1:
m.boundaryConditions['WR_Movement'].setValuesInStep(
stepName='Pre-Load'+str(i_cycles).zfill(2), v3=0.0, vr3=0.0)
m.boundaryConditions['ConnDisp_Trans_Unload'+str(i_cycles-1).zfill(2)].deactivate('Pre-Load'+str(i_cycles).zfill(2))

# ------------------------------------------------------------------------
# Cycles-Step
m.boundaryConditions['ConnVel_Hinge_Load'].setValuesInStep(
stepName='Cycles'+str(i_cycles).zfill(2),vr1=rot_vel_WR)
m.boundaryConditions['WR_Movement'].setValuesInStep(
stepName='Cycles'+str(i_cycles).zfill(2), v3=feed_v, vr3=rad_per_s)

# ------------------------------------------------------------------------
# Unload-Step
m.boundaryConditions['ConnVel_Hinge_Load'].setValuesInStep(
stepName='Unload'+str(i_cycles).zfill(2),vr1=0.0)
m.boundaryConditions['WR_Movement'].setValuesInStep(
stepName='Unload'+str(i_cycles).zfill(2), v3=0.0, vr3=0.0)
m.ConnDisplacementBC(amplitude=UNSET, createStepName='Unload'+str(i_cycles).zfill(2),
distributionType=UNIFORM, fixed=OFF,
name='ConnDisp_Trans_Unload'+str(i_cycles).zfill(2),
region=a.sets['Wire-Translator'],
u1=0.5, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET, ur3=UNSET)

# ------------------------------------------------------------------------
# Return-Step
m.boundaryConditions['WR_Movement'].setValuesInStep(
stepName='Return'+str(i_cycles).zfill(2), v3=feed_v, vr3=(-1)*vr_return)

# ------------------------------------------------------------------------
# Stop Movement for element-deletion steps
m.boundaryConditions['WR_Movement'].setValuesInStep(
stepName='Step-Element_Deletion-Layer_01',
v3=0.,
vr3=0.)

return
# ------------------------------------------------------------------------
def make_el_deletion(m, nr_dels_per_layer):
# Deactivate Contact interaction for element deletion steps
m.interactions['Int-Surf_cont_Axle'].deactivate('Step-Element_Deletion-Layer_01')

# Delete the elements in layers:
for i_del in range(1, nr_dels + 1):
m.ModelChange(activeInStep=False,
createStepName='Step-Element_Deletion-Layer_{}'.format(str(i_del).zfill(2)),
includeStrain=False,
name='Int-Element_Deletion-Layer_{}'.format(str(i_del).zfill(2)),
region=inst_Axle_sector.sets['Set-el_del_Layer_{}'.format(str(i_del).zfill(2))],
regionType=ELEMENTS)
return

# ------------------------------------------------------------------------
def create_job(m, dir_name):
'''
job_name='DR_DAxle_{}_WR_D{}R{}_feed_{}_load_{}'.format(str(int(r_Axle*2)).replace('.','p'),
```

```
str(int(round(R_WRoller*2/scale_analytical))).replace('.','p'),
str(int(round(r_WRoller/scale_analytical))).replace('.','p'),
str(feed).replace('.','p'),
str(abs(load_WR/1000)).replace('.','p'))
'''
job_name = dir_name
job_type = ANALYSIS

# allow Model change for subsequent restart simulations
m.ModelChange(createStepName='Pre-Load01',
isRestart=True,
name='Int-Model-Change')


job = mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
explicitPrecision=SINGLE, getMemoryFromAnalysis=True, historyPrint=OFF,
memory=90, memoryUnits=PERCENTAGE, model=m.name, modelPrint=OFF,
multiprocessingMode=THREADS, name=job_name, nodalOutputPrecision=SINGLE,
numCpus=num_cpus, numDomains=num_cpus, numGPUs=0, queue=None, resultsFormat=ODB, scratch='',
type=job_type, userSubroutine='', waitHours=0, waitMinutes=0)

# create field outputs
del m.fieldOutputRequests['F-Output-1']
m.FieldOutputRequest(createStepName='Pre-Load01',
name='F-Output', variables=PRESELECT)

# create history outputs
del m.historyOutputRequests['H-Output-1']
m.HistoryOutputRequest(createStepName='Pre-Load01',
name='H-Output', variables=PRESELECT)
m.HistoryOutputRequest(createStepName='Pre-Load01',
name='H-Output_contact',
variables=('CAREA', 'ALLAE', 'ALLCD', 'ALLDMD', 'ALLEE',
'ALLFD', 'ALLIE', 'ALLJD', 'ALLKE', 'ALLKL', 'ALLPD',
'ALLQB', 'ALLSE', 'ALLSD', 'ALLVD', 'ALLWK', 'ETOTAL'))

#-----------------------------------------------------------------------
# additional keywords for inp.-file
m.keywordBlock.synchVersions()
block_list = m.keywordBlock.sieBlocks
pos_i = [i for i, j in enumerate(block_list) if 'Output, field' in j]
str_out = '**\n*Element Output, directions=YES\nCOORD, IVOL\n**\*Node Output\nCOORD, '
m.keywordBlock.insert(pos_i[0], str_out)
m.keywordBlock.insert(pos_i[1]+1, str_out)
#-----------------------------------------------------------------------
# create inp.-file
job.writeInput()
# save CAE
mdb.saveAs(pathName=job_name+'.cae')
return
#-----------------------------------------------------------------------
def move_files(dir_calc,nr_cycles):
list_dir=os.listdir(dir_calc)
for files in list_dir:
for i_cycles in range(1,nr_cycles):
if (files.startswith('m-c' + str(i_cycles).zfill(2)) or
files.startswith('j-' + str(i_cycles).zfill(2))):
shutil.move(files, ('m-c' + str(i_cycles).zfill(2)))
return
#-----------------------------------------------------------------------

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# running program
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

if __name__ == '__main__':
# Models
m_name='DR_axle'
mdb.models.changeKey(fromName='Model-1', toName=m_name)
m = mdb.models[m_name]
m.setValues(noPartsInputFile=ON)

r_Axle, t_Axle_sector, l_Axle_sector, aa_Axle_sector, ta_Axle_sector, r_Axle_hollow,\
r_WRoller, R_WRoller, h_WRoller, scale_analytical, geom_par,\
a_WR_start, a_dummy_min, nr_layer_dummy, lowerbound_Trans, \
upperbound_Trans = get_geometry_parameters()

nr_cycles, i_models, feed, rpm, rad_per_s, load_WR, indent_WR, rolling_angle, feed_v, \
movement_type, t_Partition, num_cpus, process_par, \
nr_el_del, nr_dels_per_cyc, nr_dels, mesh_num, mesh_num_edges, mesh_num_depth, \
```

```
mesh_num_side, mesh_bias_ratio, reduced, nr_dummy, a_dummy, depth_dummy, preload=\
get_process_parameters(aa_Axle_sector, ta_Axle_sector, a_WR_start, a_dummy_min, nr_layer_dummy)

dir_name, dir_calc = set_directory()

mat_name, mat_name_dummy, youngs_modulus, nu, yield_stress, c1, gamma1, c2, gamma2, \
c3, gamma3, equiv_stress, q_infinity, hardening_b, \
youngs_modulus_dummy, nu_dummy, mat_par = \
get_material_parameters()

save_parameter(geom_par, process_par, mat_par)

r_WRoller,R_WRoller,h_WRoller = scale_WRoller(r_WRoller,R_WRoller,h_WRoller,scale_analytical)

p_Axle_sector,p_WRoller = make_geometry(m,r_Axle,t_Axle_sector,l_Axle_sector,aa_Axle_sector,
ta_Axle_sector,r_WRoller, R_WRoller, h_WRoller,
r_Axle_hollow, a_dummy, depth_dummy)

l_comp, h_all, a_comp = make_feed_compensation(m, p_Axle_sector, r_Axle, l_Axle_sector, ta_Axle_sector,
aa_Axle_sector, a_dummy, feed)


dat_WR_XZ_h, dat_Ax_XY_0 = make_datum_planes(p_WRoller, h_WRoller,
p_Axle_sector, indent_WR,
l_comp)

a, inst_Axle_sector, inst_WRoller, dat_axis_x, dat_axis_y, dat_axis_z = \
make_instances(m, p_Axle_sector, p_WRoller, r_Axle)

make_p_sets(p_Axle_sector, r_Axle, t_Axle_sector, l_Axle_sector, a_dummy, depth_dummy,
l_comp, h_all, a_comp)

make_mesh(p_Axle_sector, mesh_num_edges, mesh_num_depth, mesh_num_side, mesh_bias_ratio, reduced)

rp_RC, rp_H, rp_H2 = make_a_sets(a, inst_Axle_sector,p_WRoller, r_Axle,R_WRoller,
h_WRoller, ta_Axle_sector, a_WR_start,
dat_Ax_XY_0, dat_WR_XZ_h, indent_WR, l_comp)

make_mat_sections(m, mat_name, mat_name_dummy, youngs_modulus, youngs_modulus_dummy,
nu, nu_dummy, yield_stress, c1, gamma1, c2, gamma2, c3, gamma3,
equiv_stress, q_infinity, hardening_b)

make_rigid(a,m,inst_WRoller)

make_connectors(a,m,rp_RC,rp_H, rp_H2, r_Axle,R_WRoller,h_WRoller,ta_Axle_sector,a_WR_start,
lowerbound_Trans, upperbound_Trans)


nr_dels_per_layer = make_el_sets(p_Axle_sector, nr_el_del, nr_dels, mesh_num_depth,
mesh_num_side, l_comp, a_comp)


timePeriodperCycle, timePeriod, timePeriodperRoll, timePeriodperReturn = \
make_step(a, m, rpm, rad_per_s, feed_v, nr_cycles,
rolling_angle, nr_dels_per_layer)

make_contact(m, movement_type)

z_list_bottom, z_list_top, r_list, h_all, h_axle, h_axle_zero_top, h_axle_zero_bottom, r_comp_top, r_comp_bottom = make_equations(m, a, in
t_Partition, mesh_num_depth,
mesh_num_side,l_Axle_sector,
aa_Axle_sector, ta_Axle_sector,
a_dummy, l_comp, a_comp,
mesh_bias_ratio)

make_load(m, load_WR, ta_Axle_sector, a_WR_start, nr_cycles, preload)

make_boundary(m, a, rad_per_s, feed_v, aa_Axle_sector, ta_Axle_sector,
r_Axle_hollow, nr_cycles,timePeriodperCycle, timePeriod,
timePeriodperRoll, timePeriodperReturn, movement_type)


make_el_deletion(m, nr_dels_per_layer)

create_job(m, dir_name)

os.chdir(dir0)
```

A.3 Heat Treatment py-Script

```
# Heat-Treatment-Wheel-Axisymmetric-Python-Script
# 2023-04-25
# J. Bialowas, S. Gapp
# ----------------------------------------------------------------------
# Import of packages
from abaqus import *
from abaqusConstants import *
from caeModules import *
import numpy as np
import os
import math
import shutil
from connectorBehavior import *
# ----------------------------------------------------------------------
# Model parameters (N-mm-s-K)
# ----------------------------------------------------------------------

session.journalOptions.setValues(replayGeometry=COORDINATE,
recoverGeometry=COORDINATE)
TOL = 1e-6

dir=os.path.abspath('')
os.chdir(dir)

Mdb()

job_name = '_HT_brake'
dir_name = 'HT_wheel'+job_name
DIR0 = os.path.abspath('')

# Functions
# ----------------------------------------------------------------------
def get_geometry_parameters():
# Wheel
#
geom_name = 'CityJet'
# reference: wheel Lucchini
# radii of the wheel
r_inside = 182.1/2 # inside radius of the wheel
r_bottom = 300.3/2 # radius of the bottom of the wheel
r_head_inside = 722.8/2 # inside radius of the wheel head
r_head_left_outside = 872.1/2 # outside radius of the wheel head on the top left corner
r_head_transition_flank = 878.9/2 # radius at the transition of the head, where R18 is
#length
width_bottom = 192.9 # bottom width of the wheel
width_head = 147.2 # head width of the wheel
dist_left_head = 39.6 # horizontal distance from the left bottom to the left top edge
dist_web_left_bottom = 91.4 # horizontal distance from the left bottom edge
# to the left bottom corner of the web
dist_web_left_top = dist_left_head + 52.3  # horizontal distance from the left bottom
# edge to the left top corner of the web
dist_web_right_bottom = dist_web_left_bottom + 43.6 # horizontal distance from the left
# bottom edge to the right bottom corner of the web
dist_web_right_top = dist_web_left_top + 42.6  # horizontal distance from the left bottom
# edge to the left top corner of the web
height_flank = 29. # height of the flank
dist_flank_1 = dist_left_head + width_head - 49.2 # horizontal distance from the left bottom
# edge to the start of the flank
dist_flank_2 = dist_left_head + width_head - 31.4 # horizontal distance from the left bottom
# edge to the end of the straight flank
# angles:
angle_web = 10*pi/180 # transition angle at the web
angle_head_1 = 2*pi/180 # transition angle 1(left side) at the head
angle_head_2 = 31.5 * pi / 180  # transition angle 2(middle) at the head
# transition radii
tran_r_web_bottom = 34. # transition radius at the bottom of the web
tran_r_web_top = 25.  # transition radius at the top of the web
tran_r_head_1 = 18. # transition radius 1 at the head
tran_r_head_2 = 21.  # transition radius 1 at the head
tran_r_head_3 = 20.  # transition radius 1 at the head

# Saving parameters for future availability
dtype = [('Name', (np.str_, 20)), ('Parameter', np.float64)]
geom_par = np.array([('r_inside', r_inside), ('r_bottom', r_bottom),
('r_head_inside', r_head_inside),
('r_head_left_outside', r_head_left_outside),
('r_head_transition_flank', r_head_transition_flank),
('width_bottom', width_bottom), ('width_head', width_head),
```

```
('dist_left_head', dist_left_head),
('dist_web_left_bottom', dist_web_left_bottom),
('dist_web_left_top', dist_web_left_top),
('dist_web_right_bottom', dist_web_right_bottom),
('dist_web_right_top', dist_web_right_top),
('height_flank', height_flank), ('dist_flank_1', dist_flank_1),
('dist_flank_2', dist_flank_2), ('angle_web', angle_web),
('angle_head_1', angle_head_1), ('angle_head_2', angle_head_2),
('tran_r_web_bottom', tran_r_web_bottom),
('tran_r_web_top', tran_r_web_top),
('tran_r_head_1', tran_r_head_1), ('tran_r_head_2', tran_r_head_2),
('tran_r_head_3', tran_r_head_3),
], dtype=dtype)
return geom_name, r_inside, r_bottom, r_head_inside, r_head_left_outside, \
r_head_transition_flank, width_bottom, width_head, dist_left_head, \
dist_web_left_bottom, dist_web_left_top, dist_web_right_bottom, dist_web_right_top, \
height_flank, dist_flank_1, dist_flank_2, angle_web, angle_head_1, angle_head_2, \
tran_r_web_bottom, tran_r_web_top, tran_r_head_1, tran_r_head_2, tran_r_head_3, \
geom_par
#-----------------------------------------------------------------------------
def get_process_parameters(dist_left_head, r_head_left_outside, kelvin):
# User Subroutine
bool_user = True # bool that defines whether user subroutines are used
bool_HT = True # bool that defines whether the simulation is of type heat transfer
# Element size
el_size = 1.5 # normal element size
el_size_fine = 0.8 # fine element size

# Water Temperature during quenching
Temp_water = 25. + kelvin

# Annealing Temperature
Temp_annealing = 500. + kelvin

# Partition
t_Partition = 20.  # depth of finely meshed section, from the top left point

partition_offset = 5.

# Calculation with one cpu only because of subroutine usage
num_cpus = 1

# Saving parameters for future availability
dtype = [('Name', (np.str_, 20)), ('Parameter', np.float64)]
process_par = np.array([('bool_user', int(bool_user)),
('bool_HT', int(bool_HT)),
('el_size', el_size),
('el_size_fine', el_size_fine),
('Temp_water', Temp_water),
('Temp_annealing', Temp_annealing),
], dtype=dtype)
return bool_user, bool_HT, el_size, el_size_fine, Temp_water, Temp_annealing, \
t_Partition, partition_offset, num_cpus, process_par
#-----------------------------------------------------------------------------
def calc_true_strain(engineering_strain):
true_strain = math.log(1+engineering_strain)
return true_strain
#-----------------------------------------------------------------------------
def calc_true_stress(engineering_stress, engineering_strain):
true_stress = engineering_stress * (1+engineering_strain)
return true_stress
#-----------------------------------------------------------------------------
def create_material(m, p, kelvin):
mat = m.Material(name='ER7-TD')

mat.Depvar(n=25)
mat.UserDefinedField()

# density in kg/m^3 and degrees C
rho_conversion_fact = 1e-12

os.chdir('material_data')
density_data = np.loadtxt(os.path.join(os.getcwd(),
'ER7_density_TD.csv'),
skiprows=1,
delimiter=',',
unpack=True)
density_data[0][:] = density_data[0][:] * rho_conversion_fact
density_data[1][:] = density_data[1][:] + kelvin
os.chdir(DIR0)
```

```
# ------------------------------------------------------------------------
# Mechanical data
# elastic in Pa and degrees C
E_conversion_fact = 1e-6

os.chdir('material_data')
elastic_data = np.loadtxt(os.path.join(os.getcwd(),
'ER7_youngs_modulus_TD.csv'),
skiprows=1,
delimiter=',',
unpack=True)
elastic_data[0][:] = elastic_data[0][:] * E_conversion_fact
elastic_data[2][:] = elastic_data[2][:] + kelvin
os.chdir(DIR0)

# temperature dependent plastic material data for individual phases
pl_mat_table = []

# Austenite

temp_list_M = [300, 400., 500., 600.]
for temp in temp_list_M:
os.chdir('flow_curves')
pl_data_A = np.loadtxt(os.path.join(os.getcwd(),
'ER7_plastic_L_Austenite_T_{}_extrapolated.csv'.\
format(str(int(temp)))),
skiprows=0,
delimiter=',',
unpack=True)
os.chdir(DIR0)

for i in range(pl_data_A.shape[1]):
strain_A= (pl_data_A[0, i] -  pl_data_A[0, 0]) * 0.01
stress_A = pl_data_A[1, i]
pl_mat_table.append([stress_A, strain_A, temp + kelvin, 1, 0, 0, 0])

os.chdir('flow_curves')
pl_data_A = np.loadtxt(os.path.join(os.getcwd(),
'ER7_plastic_L_austenite_T_700.csv'),
skiprows=0,
delimiter=',',
unpack=True)
os.chdir(DIR0)

for i in range(pl_data_A.shape[1]):
strain_A = (pl_data_A[0, i] -  pl_data_A[0, 0]) * 0.01
stress_A = pl_data_A[1, i]
pl_mat_table.append([stress_A, strain_A, 700. + kelvin, 1, 0, 0, 0])

temp_list_M = [25., 150., 250., 300., 450.]
for temp in temp_list_M:
os.chdir('flow_curves')
pl_data_M = np.loadtxt(os.path.join(os.getcwd(),
'ER7_plastic_L_0p013_T_{}.csv'.format(str(int(temp)))),
skiprows=0,
delimiter=',',
unpack=True)
os.chdir(DIR0)

for i in range(pl_data_M.shape[1]):
strain_M = (pl_data_M[0, i] -  pl_data_M[0, 0]) * 0.01
stress_M = pl_data_M[1, i]
pl_mat_table.append([stress_M, strain_M, temp + kelvin, 0, 1, 0, 0])

temp_list_B = [25., 150., 300., 450.]
for temp in temp_list_B:
os.chdir('flow_curves')
pl_data_B = np.loadtxt(os.path.join(os.getcwd(),
'ER7_plastic_L_0p150_T_{}_pure_bainite.csv'\
.format(str(int(temp)))),
skiprows=0,
delimiter=',',
unpack=True)
os.chdir(DIR0)

for i in range(pl_data_B.shape[1]):
strain_B = (pl_data_B[0, i] -  pl_data_B[0, 0]) * 0.01
stress_B = pl_data_B[1, i]
pl_mat_table.append([stress_B, strain_B, temp + kelvin, 0, 0, 1, 0])
```

```
temp_list_P = [25., 150., 300., 450., 600., 700]
for temp in temp_list_P:
os.chdir('flow_curves')
pl_data_P = np.loadtxt(os.path.join(os.getcwd(),
'ER7_plastic_L_5p000_T_{}.csv'.format(str(int(temp)))),
skiprows=0,
delimiter=',',
unpack=True)
os.chdir(DIR0)

for i in range(pl_data_P.shape[1]):
strain_P = (pl_data_P[0, i] - pl_data_P[0, 0]) * 0.01
stress_P = pl_data_P[1, i]
pl_mat_table.append([stress_P, strain_P, temp + kelvin, 0, 0, 0, 1])

mat.Plastic(dependencies = 4,
table = pl_mat_table,
temperatureDependency = ON)

# ------------------------------------------------------------------------
# Thermophysical data
# Specific Heat in J/(kg K)
cp_conversion_fact = 1e+6
os.chdir('material_data')
specitic_heat_data = np.loadtxt(os.path.join(os.getcwd(),
'ER7_specific_heat_TD.csv'),
skiprows=1,
delimiter=',',
unpack=True)
specitic_heat_data[0][:] = specitic_heat_data[0][:] * cp_conversion_fact
specitic_heat_data[1][:] = specitic_heat_data[1][:] + kelvin
os.chdir(DIR0)

# Conductivity in W/(m K)
# Conversion into t-mm-s with factor 1
os.chdir('material_data')
conductivity_data = np.loadtxt(os.path.join(os.getcwd(),
'ER7_conductivity_TD.csv'),
skiprows=1,
delimiter=',',
unpack=True)
conductivity_data[0][:] = conductivity_data[0][:]
conductivity_data[1][:] = conductivity_data[1][:] + kelvin
os.chdir(DIR0)

# ------------------------------------------------------------------------
# Assign material data to material properties in Abaqus
mat.Density(dependencies=0,
table=(tuple(map(tuple,
zip(density_data[0][:],density_data[1][:])))),
temperatureDependency=ON)

mat.Elastic(dependencies=0,
table=(tuple(map(tuple,
zip(elastic_data[0][:],
elastic_data[1][:],
elastic_data[2][:])))),
temperatureDependency=ON)

mat.Conductivity(dependencies=0,
table=(tuple(map(tuple,
zip(conductivity_data[0][:],conductivity_data[1][:])))),
temperatureDependency=ON)

mat.SpecificHeat(dependencies=0,
table=(tuple(map(tuple,
zip(specitic_heat_data[0][:],specitic_heat_data[1][:])))),
temperatureDependency=ON)

# USER DEFINED material properties
mat.Expansion()
mat.expansion.setValues(type=ISOTROPIC, userSubroutine=ON)
mat.Creep(law=USER, table=())
# mat.HeatGeneration()

print 'material data is implemented'

# Section and material assignment
m.HomogeneousSolidSection(material='ER7-TD', name='Section-Wheel',
```

```
thickness=None)
p_Wheel.SectionAssignment(region=p_Wheel.sets['Set-all_Wheel'],
sectionName='Section-Wheel')

#-------------------------------------------------------------------------------
def set_directory():
# Set the working directory
job_name = job_name
dir_name = dir_name
DIR0 = os.path.abspath('')
'''
# chooses the directory if it exists already
if os.path.exists(dir_name):
# change directory
os.chdir(dir_name)
# save the directory path
DIR0 = os.path.abspath('')
if not os.path.exists('Results'):
# creates a new directory 'Results'
os.mkdir('Results')
# creates the directory
else:
# create a new directory
os.mkdir(dir_name)
# change directory
os.chdir(dir_name)
# save the directory path
DIR0 = os.path.abspath('')
# creates a new directory 'Results'
os.mkdir('Results')
'''
return dir_name, DIR0, job_name

#-------------------------------------------------------------------------------
def save_parameter(geom_par,
process_par,
name_geom='geometry_parameters',
name_process='process_parameters'):
par = {name_geom: geom_par,  name_process: process_par}
for i_name, i_par in par.items():
np.savetxt(i_name + '.csv', i_par, delimiter=',', fmt=['%s', '%f'])
return
#-------------------------------------------------------------------------------
def make_geometry(m, geom_name, r_inside, r_bottom, r_head_inside,
r_head_left_outside, r_head_transition_flank, width_bottom,
width_head, dist_left_head, dist_web_left_bottom,  dist_web_left_top,
dist_web_right_bottom, dist_web_right_top, height_flank, dist_flank_1,
dist_flank_2, angle_web, angle_head_1, angle_head_2, tran_r_web_bottom,
tran_r_web_top, tran_r_head_1, tran_r_head_2, tran_r_head_3, t_Partition):
# Wheel
# Load the step file as a sketch
# Attention: the sketch origin is not the left bottom point of the sketch because the
# wheel is hollow!
name_s_wheel = 's_wheel_' + geom_name
#mdb.openStep(path_geometry, scaleFromFile=OFF)
#m.ConstrainedSketchFromGeometryFile(geometryFile=mdb.acis, name='s_' + geom_name)
#s_wheel = m.ConstrainedSketch(name='__profile__', sheetSize=200.0)
#s_wheel.retrieveSketch(sketch=m.sketches['s_' + geom_name])
# sketch the wheel
dist_web_top = dist_web_left_top - dist_left_head # horizontal distance from the edge of
# the head to the web
# for the sketch of the flank some dimensions had to be measured from 2D-Step model:
end_straight = (r_head_transition_flank + 19.0462590450569,
dist_flank_1 + 11.6715625539186)
end_R21 = (r_head_transition_flank + 27.524216279494,
dist_flank_1 + 21.6598798928313)
s_wheel = m.ConstrainedSketch(name=name_s_wheel, sheetSize=200.0)
s_wheel.Line(point1=(r_inside, 0), point2=(r_bottom, 0))
s_wheel.Line(point1=(r_bottom, 0),
point2=(r_bottom + dist_web_left_bottom * tan(angle_web),
dist_web_left_bottom))
s_wheel.Line(point1=(r_bottom + dist_web_left_bottom * tan(angle_web),
dist_web_left_bottom),
point2=(r_head_inside - dist_web_top * tan(angle_web),
dist_web_left_top))
s_wheel.Line(point1=(r_head_inside - dist_web_top * tan(angle_web),
dist_web_left_top),
point2=(r_head_inside, dist_left_head))
s_wheel.Line(point1=(r_head_inside, dist_left_head),
point2=(r_head_left_outside, dist_left_head))
```

```
s_wheel.Line(point1=(r_head_left_outside, dist_left_head),
point2=(r_head_left_outside +
(dist_flank_1-dist_left_head) * tan(angle_head_1),
dist_flank_1))
s_wheel.Line(point1=(r_head_left_outside +
(dist_flank_1-dist_left_head) * tan(angle_head_1),
dist_flank_1),
point2=end_straight)
s_wheel.ArcByStartEndTangent(entity= s_wheel.geometry.findAt(end_straight, ),
point1=end_straight,
point2=end_R21)
s_wheel.ArcByStartEndTangent(entity= s_wheel.geometry.findAt(end_R21, ),
point1=end_R21,
point2=(r_head_transition_flank + height_flank -
tran_r_head_3,
dist_left_head + width_head))
s_wheel.Line(point1=(r_head_transition_flank + height_flank - tran_r_head_3,
dist_left_head + width_head),
point2=(r_head_inside, dist_left_head + width_head))
s_wheel.Line(point1=(r_head_inside, dist_left_head + width_head),
point2=(r_head_inside - dist_web_top * tan(angle_web),
dist_web_right_top))
s_wheel.Line(point1=(r_head_inside - dist_web_top * tan(angle_web),
dist_web_right_top),
point2=(r_bottom + (width_bottom - dist_web_right_bottom) * tan(angle_web),
dist_web_right_bottom))
s_wheel.Line(point1=(r_bottom + (width_bottom - dist_web_right_bottom) * tan(angle_web),
dist_web_right_bottom),
point2=(r_bottom, width_bottom))
s_wheel.Line(point1=(r_bottom, width_bottom),
point2=(r_inside, width_bottom))
s_wheel.Line(point1=(r_inside, width_bottom),
point2=(r_inside, 0.0))
# fillets:
# web bottom left
s_wheel.FilletByRadius(curve1=s_wheel.geometry.findAt((r_bottom + dist_web_left_bottom/2 *
tan(angle_web),
dist_web_left_bottom/2), ),
curve2=s_wheel.geometry.findAt((((r_head_inside - dist_web_top*tan(angle_web))
+(r_bottom + dist_web_left_bottom*tan(angle_web)))/2,
dist_web_left_bottom +
(dist_web_left_top-dist_web_left_bottom)/2), ),
nearPoint1=(r_bottom + dist_web_left_bottom/2*tan(angle_web),
dist_web_left_bottom/2),
nearPoint2=(r_bottom + tran_r_web_bottom,
dist_web_left_bottom),
radius=tran_r_web_bottom)
# web bottom right
dist_tran_web_r_bot = dist_web_right_bottom + tran_r_web_bottom # horizontal distance to
# define the fillet at the right bottom of the web
s_wheel.FilletByRadius(curve1=s_wheel.geometry.findAt((r_bottom + (width_bottom -
dist_tran_web_r_bot) *
tan(angle_web),
dist_tran_web_r_bot), ),
curve2=s_wheel.geometry.findAt((((r_head_inside - dist_web_top*tan(angle_web))
+(r_bottom + (width_bottom - dist_tran_web_r_bot) *
tan(angle_web)))/2,
dist_web_right_bottom +
(dist_web_right_top-dist_web_right_bottom)/2), ),
nearPoint1=(r_bottom + (width_bottom - dist_tran_web_r_bot) *
tan(angle_web),
dist_tran_web_r_bot),
nearPoint2=(r_bottom + tran_r_web_bottom,
dist_web_right_bottom),
radius=tran_r_web_bottom)
# web top left
s_wheel.FilletByRadius(curve1=s_wheel.geometry.findAt((r_head_inside - dist_web_top/2 *
tan(angle_web),
dist_left_head + dist_web_top/2), ),
curve2=s_wheel.geometry.findAt((((r_head_inside - dist_web_top*tan(angle_web))
+(r_bottom + dist_web_left_bottom*tan(angle_web)))/2,
dist_web_left_bottom +
(dist_web_left_top-dist_web_left_bottom)/2), ),
nearPoint1=(r_head_inside - (dist_left_head + dist_web_top / 2) *
tan(angle_web),
dist_left_head + dist_web_top/2),
nearPoint2=(r_head_inside - tran_r_web_top,
dist_web_left_top),
radius=tran_r_web_top)
# web top right
```

```
dist_tran_web_r_top = dist_web_right_top + tran_r_web_top # horizontal distance to
# define the fillet at the right bottom of the web
s_wheel.FilletByRadius(curve1=s_wheel.geometry.findAt((r_head_inside -
(dist_web_top - tran_r_web_top) *
tan(angle_web),
dist_tran_web_r_top), ),
curve2=s_wheel.geometry.findAt((((r_head_inside - dist_web_top*tan(angle_web))
+(r_bottom + (width_bottom - dist_tran_web_r_top) *
tan(angle_web)))/2,
dist_web_right_bottom +
(dist_web_right_top-dist_web_right_bottom)/2), ),
nearPoint1=(r_head_inside + (dist_web_top - tran_r_web_top) *
tan(angle_web),
dist_tran_web_r_top),
nearPoint2=(r_head_inside - tran_r_web_top,
dist_web_right_top),
radius=tran_r_web_top)
# flank
s_wheel.FilletByRadius(curve1=s_wheel.geometry.findAt((r_head_left_outside +
(dist_flank_1/2 - dist_left_head)*
tan(angle_head_1),
dist_flank_1/2), ),
curve2=s_wheel.geometry.findAt((r_head_transition_flank + height_flank/2,
dist_flank_1 + height_flank/2 *
tan(angle_head_2)), ),
nearPoint1=(r_head_left_outside + (dist_flank_1 - tran_r_head_1) *
tan(angle_head_1),
dist_flank_1 - tran_r_head_1),
nearPoint2=(r_head_transition_flank+height_flank/2,
dist_flank_1 + height_flank/2 *
tan(angle_head_2)),
radius=tran_r_head_1)

# Create the wheel part as axisymmetric part from the sketch s_wheel
s_wheel_AS= m.ConstrainedSketch(name='s_wheel_AS', sheetSize=200.0)
s_wheel_AS.sketchOptions.setValues(viewStyle=AXISYM)
s_wheel_AS.ConstructionLine(point1=(0.0, -100.0), point2=(0.0, 100.0))
s_wheel_AS.geometry.findAt((0.0, 0.0))
s_wheel_AS.FixedConstraint(entity=s_wheel_AS.geometry.findAt((0.0, 0.0), ))
s_wheel_AS.retrieveSketch(sketch=s_wheel)
p_Wheel=m.Part(dimensionality=AXISYMMETRIC,
name='p_Wheel', type=DEFORMABLE_BODY)
p_Wheel.BaseShell(sketch=s_wheel_AS)
return s_wheel_AS, s_wheel, p_Wheel

#-------------------------------------------------------------------------------
def make_thermocouples():
P1 = {
        "x" : 230.0,
        "y" : 65.0,
        "z" : 0.0
}

P2 = {
        "x" : 250.0,
        "y" : 80.0,
        "z" : 0.0
}

P3 = {
        "x" : 365.0,
        "y" : 30.0,
        "z" : 0.0
}

P4 = {
        "x" : 330.0,
        "y" : 30.0,
        "z" : 0.0
}

P5 = {
        "x" : 340.0,
        "y" : 80.0,
        "z" : 0.0
}

P6 = {
        "x" : 310.0,
        "y" : 80.0,
```

```
        "z" : 0.0
}

P7 = {
        "x" : 290.0,
        "y" : 80.0,
        "z" : 0.0
}

P8 = {
        "x" : 290.0,
        "y" : 145.0,
        "z" : 0.0
}

P9 = {
        "x" : 340.0,
        "y" : 140.0,
        "z" : 0.0
}

P10 = {
        "x" : 345.0,
        "y" : 153.3,
        "z" : 0.0
}

thermocouples = {
        "P1" : P1,
        "P2" : P2,
        "P3" : P3,
        "P4" : P4,
        "P5" : P5,
        "P6" : P6,
        "P7" : P7,
        "P8" : P8,
        "P9" : P9,
        "P10" : P10
}

# coordinate transformation to local coordinate system
coord_x = 91.05
coord_y = 192.9

for i in thermocouples.keys():
thermocouples[i]["x"] = thermocouples[i]["x"]+coord_x
thermocouples[i]["y"] = (-1)*thermocouples[i]["y"]+coord_y
return thermocouples

#---------------------------------------------------------------------------
def make_p_sets(p_Wheel, width_head, partition_offset):
# Sets
# Wheel
# Sets have to be adjusted for every change in geometry

p_Wheel.Set(faces=p_Wheel.faces[:], name='Set-all_Wheel')
p_Wheel.Set(edges=p_Wheel.edges.findAt(((r_inside, width_bottom/3, 0.0),)),
name='Set-inside_Wheel')
p_Wheel.Set(edges=p_Wheel.edges.findAt(((r_head_left_outside+(width_head/5)*
math.tan(angle_head_1),
width_head/5+dist_left_head,
0.0),),
((468.34844, 168.853046, 0.0), ),
((461.185779, 151.230374, 0.0), ),
((450.4073, 144.308848, 0.0), ),
((439.831446, 132.308944, 0.0), )
),
name='Set-water-quenching')
p_Wheel.Set(edges=p_Wheel.edges.findAt(
(((r_bottom-r_inside)/3+r_inside, 0.0, 0.0),),
(((r_head_left_outside-r_head_inside)/3+r_head_inside,
dist_left_head,
0.0),),
),
name='Set-ceramic-contact')
p_Wheel.Set(edges=p_Wheel.edges.findAt(
((437.85, 186.8, 0.0), ),
((394.8875, 186.8, 0.0), ),
((360.007614, 178.903384, 0.0), ),
((352.852745, 147.040634, 0.0), ),
```

```
((295.562119, 134.647577, 0.0), ),
((159.445882, 151.944907, 0.0), ),
((154.101266, 170.491258, 0.0), ),
((135.375, 192.9, 0.0), ),
((91.05, 144.675, 0.0), ),
((152.943931, 15.84517, 0.0), ),
((165.3529, 74.45684, 0.0), ),
((228.824143, 91.568246, 0.0), ),
((352.852793, 79.358073, 0.0), ),
((357.223071, 63.288542, 0.0), ), ),
name='air-contact')

p_Wheel.Set(edges=p_Wheel.edges.findAt(
((413.55, 39.6, 0.0), ),
((436.821227, 61.685044, 0.0), ),
((439.831446, 132.308944, 0.0), ),
((450.4073, 144.308848, 0.0), ),
((461.185779, 151.230374, 0.0), ),
((468.34844, 168.853046, 0.0), ),
((437.85, 186.8, 0.0), ),
((394.8875, 186.8, 0.0), ),
((360.007614, 178.903384, 0.0), ),
((352.852745, 147.040634, 0.0), ),
((295.562119, 134.647577, 0.0), ),
((159.445882, 151.944907, 0.0), ),
((154.101266, 170.491258, 0.0), ),
((135.375, 192.9, 0.0), ),
((91.05, 144.675, 0.0), ),
((105.825, 0.0, 0.0), ),
((152.943931, 15.84517, 0.0), ),
((165.3529, 74.45684, 0.0), ),
((228.824143, 91.568246, 0.0), ),
((352.852793, 79.358073, 0.0), ),
((357.223071, 63.288542, 0.0), ),
((372.5625, 39.6, 0.0), ), ),
name='Set-all_surf')
return

# ----------------------------------------------------------------------------
def make_p_surf(p_Wheel, width_head):
p_Wheel.Surface(side1Edges=p_Wheel.edges.findAt(
((r_head_left_outside+(width_head/5)*
math.tan(angle_head_1),
width_head/5+dist_left_head,
0.0),),
((468.34844, 168.853046, 0.0), ),
((461.185779, 151.230374, 0.0), ),
((450.4073, 144.308848, 0.0), ),
((439.831446, 132.308944, 0.0), )
),
name='Surf-water_contact')
p_Wheel.Surface(side1Edges=p_Wheel.edges.findAt(
(((r_bottom-r_inside)/3+r_inside, 0.0, 0.0),),
(((r_head_left_outside-r_head_inside)/3+r_head_inside,
dist_left_head,
0.0),),
),
name='Surf-ceramic_contact')

p_Wheel.Surface(name='Surf-air_contact_vertical',
side1Edges=p_Wheel.edges.findAt(
((360.007614, 178.903384, 0.0), ),
((352.852745, 147.040634, 0.0), ),
((159.445882, 151.944907, 0.0), ),
((154.101266, 170.491258, 0.0), ),
((91.05, 144.675, 0.0), ),
((152.943931, 15.84517, 0.0), ),
((165.3529, 74.45684, 0.0), ),
((352.852793, 79.358073, 0.0), ),
((357.223071, 63.288542, 0.0), ), ))

p_Wheel.Surface(name='Surf-air_contact-horizontal-top',
side1Edges=p_Wheel.edges.findAt(
((437.85, 186.8, 0.0), ),
((394.8875, 186.8, 0.0), ),
((135.375, 192.9, 0.0), ),
((295.562119, 134.647577, 0.0), )
))

p_Wheel.Surface( name='Surf-all',
```

```
side1Edges=p_Wheel.edges.findAt(
((413.55, 39.6, 0.0), ),
((436.821227, 61.685044, 0.0), ),
((439.831446, 132.308944, 0.0), ),
((450.4073, 144.308848, 0.0), ),
((461.185779, 151.230374, 0.0), ),
((468.34844, 168.853046, 0.0), ),
((437.85, 186.8, 0.0), ),
((394.8875, 186.8, 0.0), ),
((360.007614, 178.903384, 0.0), ),
((352.852745, 147.040634, 0.0), ),
((295.562119, 134.647577, 0.0), ),
((159.445882, 151.944907, 0.0), ),
((154.101266, 170.491258, 0.0), ),
((135.375, 192.9, 0.0), ),
((91.05, 144.675, 0.0), ),
((105.825, 0.0, 0.0), ),
((152.943931, 15.84517, 0.0), ),
((165.3529, 74.45684, 0.0), ),
((228.824143, 91.568246, 0.0), ),
((352.852793, 79.358073, 0.0), ),
((357.223071, 63.288542, 0.0), ),
((372.5625, 39.6, 0.0), )
))

p_Wheel.Surface(name='Surf-air_contact-horizontal-bottom',
side1Edges=p_Wheel.edges.findAt(((228.824143, 91.568246, 0.0), )))
return

#-----------------------------------------------------------------------------
def make_mat_sections(m, p_Wheel, mat_name, density_tbl, bool_user, bool_HT,
bool_plastic, youngs_modulus_TD_tbl, plastic_isotropic_TD_tbl,
specificheat_tbl, conductivity_tbl):
# Material
mat = m.Material(name=mat_name)

# general
mat.Density(table=density_tbl,
temperatureDependency=ON)

if not bool_HT:
# elastic
mat.Elastic(table=youngs_modulus_TD_tbl,temperatureDependency=ON)
# plastic
if bool_plastic:
mat.Plastic(dataType=PARAMETERS,
hardening=ISOTROPIC,
rate=OFF,
temperatureDependency=ON,
table=plastic_isotropic_TD_tbl)
else:
mat.Plastic(dataType=PARAMETERS,
hardening=COMBINED,
numBackstresses=3,
table=(plastic_isotropic_TD_tbl[1],)
)

# Section and material assignment
m.HomogeneousSolidSection(material=mat_name, name='Section-' + mat_name,
thickness=None)
p_Wheel.SectionAssignment(region=p_Wheel.sets['Set-all_Wheel'],
sectionName='Section-' + mat_name)


# Specific heat
mat.SpecificHeat(
table=specificheat_tbl,
temperatureDependency=ON)

# Conductivity
mat.Conductivity(
table = conductivity_tbl,
temperatureDependency=ON)

# Latent heat
# mat.LatentHeat(
#     table=((100.0, 600.0, 700.0), ))

# Variables controlled by user subroutine
if bool_user:
```

```
# depvar
mat.Depvar(n=25)

# UEXPAN
mat.Expansion(userSubroutine=ON)

# CREEP
#mat.Creep(law=USER, table=())

# user defienied field (USDFLD)
mat.UserDefinedField()

# HETVAL
# mat.HeatGeneration()
return

# ------------------------------------------------------------------------
def make_partition(p_Wheel, partition_offset=2):
# Partition of fine meshed surface zone
s_partition_fine=m.ConstrainedSketch(name='s_partition_fine', sheetSize=10.0,
transform=p_Wheel.MakeSketchTransform(
sketchPlane=p_Wheel.faces.findAt((r_inside+(r_bottom-r_inside)/2,
width_bottom/2,
0.0),
(0.0, 0.0, 1.0)),
sketchPlaneSide=SIDE1,
sketchOrientation=RIGHT,
origin=(0.0, 0.0, 0.0))
)
p_Wheel.projectReferencesOntoSketch(filter=
COPLANAR_EDGES, sketch=s_partition_fine)
s_partition_fine.Line(point1=(r_head_left_outside-t_Partition, dist_left_head),
point2=(r_head_left_outside-t_Partition, dist_left_head+width_head))
p_Wheel.PartitionFaceBySketch(faces=p_Wheel.faces.findAt((
(r_inside+(r_bottom-r_inside)/2, width_bottom/2, 0.0), )),
sketch=s_partition_fine)

# Partition for element deletion due to machining operations
t = p_Wheel.MakeSketchTransform(
sketchPlane=p_Wheel.faces[0],
sketchPlaneSide=SIDE1,
origin=(0.0, 0.0, 0.0))
s = m.ConstrainedSketch(name='s_wheel_partition',
sheetSize=1000., gridSpacing=25., transform=t)
edge_list = [edge for edge in p_Wheel.sets['Set-water-quenching'].edges]
p_Wheel.projectEdgesOntoSketch(
sketch=s,
edges=edge_list,
constrainToBackground=False)
s.move(vector=(-partition_offset, 0.0), objectList=[s.geometry[key] for key in s.geometry.keys()])
p_Wheel.PartitionFaceBySketch(faces=p_Wheel.faces, sketch=s)

# Additional partition necessary for meshing
s_part_circ = m.ConstrainedSketch(name='s_wheel_partition_circ',
sheetSize=1000.0,
transform=t)
p_Wheel.projectReferencesOntoSketch(filter=COPLANAR_EDGES,
sketch=s_part_circ)
s_part_circ.CircleByCenterPerimeter(center=(448.45 - partition_offset,186.8),
point1=(448.45, 186.8))
p_Wheel.PartitionFaceBySketch(faces=p_Wheel.faces.findAt((
(448.45 - 2*partition_offset, 186.8 - 2*partition_offset, 0.0), )),
sketch=s_part_circ)
# Create element set for element deletion
p_Wheel.Set(faces=p_Wheel.faces.findAt((
(436.05 - partition_offset/2,
39.6 + partition_offset/2,
0.), )),
name='Set-eldel')

# Partition for braking surface
s_part_brake = m.ConstrainedSketch(name='s_wheel_partition_brake',
sheetSize=1000.0,
transform=t)
p_Wheel.projectReferencesOntoSketch(filter=COPLANAR_EDGES,
sketch=s_part_brake)
s_part_brake.Line(
point1=(416.05, 39.6+15),
point2=(416.05 + 50, 39.6+15))
s_part_brake.Line(
```

```
point1=(416.05, 39.6+15+80),
point2=(416.05 + 50, 39.6+15+80))
p_Wheel.PartitionFaceBySketch(faces=p_Wheel.faces,
sketch=s_part_brake)

# Partition for mesh of web
s_part_hub = m.ConstrainedSketch(name='s_wheel_partition_hub',
sheetSize=1000.0,
transform=t)
p_Wheel.projectReferencesOntoSketch(filter=COPLANAR_EDGES,
sketch=s_part_hub)
s_part_hub.rectangle(
point1=(192.,145),
point2=(330,80))
p_Wheel.PartitionFaceBySketch(faces=p_Wheel.faces,
sketch=s_part_hub)

p_Wheel.Surface(name='Surf-Brake',
side1Edges=p_Wheel.edges.findAt((
(432.214085, 72.935044, 0.0), )))
return

#-----------------------------------------------------------------------
def make_mesh(p_Wheel, el_size, partition_offset):
# Mesh
p_Wheel.seedEdgeBySize(constraint=FINER,
edges=p_Wheel.edges.findAt(
((416.05,39.6+15+80+1,0.), ),
((436.948081, 136.991512, 0.0), ),
((441.948081, 136.991512, 0.0), ),
((416.05,113.2,0.), ),
((416.05,39.6+1,0.), ),
((436.05 - partition_offset*1.01,39.6,0.), ),
((436.821227, 61.685044, 0.0), ),
((439.831446, 132.308944, 0.0), ),
((450.4073, 144.308848, 0.0), ),
((461.185779, 151.230374, 0.0), ),
((468.34844, 168.853046, 0.0), ),
((437.85, 186.8, 0.0), ), ), size=el_size_fine)

p_Wheel.seedEdgeBySize(constraint=FINER,
edges=p_Wheel.edges.findAt(
((436.358504,48.434405,0.), ),
((436.358504- partition_offset,48.434405,0.), ),
((416.05,48.434405,0.), ),
((446.45 - partition_offset/2,186.8,0.), ),
((442.2, 186.8, 0.0), ),
((436.05 - 0.01, 39.6, 0.), ),
((436.821227 - partition_offset, 61.685044, 0.0), ),
((439.831446 - partition_offset, 132.308944, 0.0), ),
((450.4073 - partition_offset, 144.308848, 0.0), ),
((461.185779 - partition_offset, 151.230374, 0.0), ),
((468.34844 - partition_offset, 168.853046, 0.0), ),
((448.45 - 0.01, 186.8, 0.), ),
((437.85 - partition_offset, 186.8, 0.0), ), ), size=el_size_fine)

p_Wheel.seedPart(minSizeFactor=0.1, size=el_size)
mdb.models['m-wheel-01'].parts['p_Wheel'].setElementType(elemTypes=(mesh.ElemType(
elemCode=DCAX4, elemLibrary=STANDARD),
mesh.ElemType(elemCode=DCAX4, elemLibrary=STANDARD)),
regions=(p_Wheel.sets['Set-all_Wheel'] ))
p_Wheel.generateMesh()
return

#-----------------------------------------------------------------------
def make_instances(m, p_Wheel):
# Instances
a = m.rootAssembly
inst_Wheel = a.Instance(dependent=ON, name='p_Wheel-1', part=p_Wheel)
return a, inst_Wheel

#-----------------------------------------------------------------------
def make_p_sets_with_mesh(m, a, p_Wheel, thermocouples):
# Sets
# Wheel
# Sets have to be adjusted for every change in geometry
allNodes = p_Wheel.nodes
delta = 1.0e-0
for i in thermocouples.keys():
x, y, z = thermocouples[i]["x"], thermocouples[i]["y"], thermocouples[i]["z"]
```

```
myNodes = allNodes.getClosest((x, y, z),)
p_Wheel.Set(name='Set-'+str(i), nodes=allNodes[(myNodes.label-1):myNodes.label])
return myNodes

#-------------------------------------------------------------------------------
def make_steps(m):
# 50 seconds of transport
m.HeatTransferStep(deltmx=1.0, initialInc=0.00001, maxInc=10.0,
maxNumInc=10000, minInc=0.0000001, name='Step-Transport-1',
previous='Initial', timePeriod=50.0)
# 740 seconds of quenching
m.HeatTransferStep(deltmx=1.0, initialInc=0.00001, maxInc=10.0,
maxNumInc=10000, minInc=0.0000001, name='Step-Quenching',
previous='Step-Transport-1', timePeriod=740.0)
# 800 seconds of transport
m.HeatTransferStep(deltmx=1.0, initialInc=0.00001, maxInc=10.0,
maxNumInc=10000, minInc=0.0000001, name='Step-Transport-2',
previous='Step-Quenching', timePeriod=800.0)
# 6 hours of annealing
m.HeatTransferStep(deltmx=1.0, initialInc=0.01, maxInc=1000.0,
maxNumInc=10000, minInc=0.0000001, name='Step-Annealing',
previous='Step-Transport-2', timePeriod=21600.0)
# 24 hours of cooling to RT
m.HeatTransferStep(deltmx=100.0, initialInc=0.01,
maxInc=1000.0, maxNumInc=10000,
minInc=0.0001, name='Step-Cooling-RT',
previous='Step-Annealing', timePeriod=86400.0)
# Element deletion representing machining operation
m.HeatTransferStep(deltmx=100.0, initialInc=0.01,
maxInc=1.0, maxNumInc=10000,
minInc=0.0001, name='Step-Element-Deletion',
previous='Step-Cooling-RT', timePeriod=1.0)
# Brake Test for 45 minutes
m.HeatTransferStep(deltmx=100.0, initialInc=0.01,
maxInc=1000, maxNumInc=10000,
minInc=0.0001, name='Step-Brake-Test',
previous='Step-Element-Deletion', timePeriod=45*60)
# Cooling after brake test for 4 hours
m.HeatTransferStep(deltmx=100.0, initialInc=0.01,
maxInc=1000, maxNumInc=10000,
minInc=0.0001, name='Step-Cooling-after-Brake-Test',
previous='Step-Brake-Test', timePeriod=4*60*60)

# Restart options for further analysis
m.steps['Step-Transport-1'].Restart(
frequency=1, numberIntervals=0, overlay=ON, timeMarks=OFF)
m.steps['Step-Quenching'].Restart(
frequency=1, numberIntervals=0, overlay=ON, timeMarks=OFF)
m.steps['Step-Transport-2'].Restart(
frequency=1, numberIntervals=0, overlay=ON, timeMarks=OFF)
m.steps['Step-Annealing'].Restart(
frequency=1, numberIntervals=0, overlay=ON, timeMarks=OFF)
m.steps['Step-Cooling-RT'].Restart(
frequency=1, numberIntervals=0, overlay=ON, timeMarks=OFF)
m.steps['Step-Element-Deletion'].Restart(
frequency=1, numberIntervals=0, overlay=ON, timeMarks=OFF)
m.steps['Step-Brake-Test'].Restart(
frequency=1, numberIntervals=0, overlay=ON, timeMarks=OFF)
m.steps['Step-Cooling-after-Brake-Test'].Restart(
frequency=1, numberIntervals=0, overlay=ON, timeMarks=OFF)
return

#-------------------------------------------------------------------------------
def make_initial_conditions(m, kelvin):
m.Temperature(createStepName='Initial',
crossSectionDistribution=CONSTANT_THROUGH_THICKNESS, distributionType=
UNIFORM, magnitudes=(850.0 +kelvin, ), name='Predefined␣Field-Init-Temp',
region=inst_Wheel.sets['Set-all_Wheel'])
return
#-------------------------------------------------------------------------------
def make_boundary(a, m, inst_Wheel):
# Fixation of the bottom
m.DisplacementBC(createStepName='Initial',
name='BC-bottom_Wheel',
region=inst_Wheel.sets['Set-ceramic-contact'],
u1=UNSET, u2=SET, u3=UNSET, ur1=UNSET, ur2=UNSET, ur3=UNSET)
return
#-------------------------------------------------------------------------------
def make_interaction(m, inst_Wheel, Temp_water, Temp_annealing, kelvin):
# emissivity: data estimated
```

```
emissivity = 0.29

# data from https://doi.org/10.1016/j.jmrt.2019.09.024
h_water_tdep = ((0.07, 25.0),
(0.3, 150.0),
(1.2, 180.0),
(0.8, 200.0),
(0.6, 250.0),
(0.8, 350.0),
(0.8, 650.0),
(0.4, 800.0),
(0.4, 850.0),
(0.1, 950.0))

# calculated using Excel tool in W/(m^2 K)
h_water_tdep_calc = ((459.43, 0.0),
(30.834, 25.0),
(868.48, 50.0),
(1541.3, 100.0),
(9518.5, 162.5),
(9518.5, 850.0))

# estimated in W/(m^2 K)
h_ceramic_tdep_calc = ((459.43, 0.0),
(30.834, 25.0),
(868.48, 50.0),
(1541.3, 100.0),
(9518.5, 162.5),
(9518.5, 850.0))

# calculated using Excel tool in W/(m^2 K)
h_dry_air_horz_top_tdep_calc = ((4.03910, 0.0000E+00),
(0.00000, 2.5000E+01),
(3.75268, 5.0000E+01),
(5.12464, 1.0000E+02),
(5.77934, 1.5000E+02),
(6.16802, 2.0000E+02),
(6.41983, 2.5000E+02),
(6.58736, 3.0000E+02),
(6.70430, 3.5000E+02),
(6.78203, 4.0000E+02),
(6.83337, 4.5000E+02),
(6.86454, 5.0000E+02),
(6.88202, 5.5000E+02),
(6.89197, 6.0000E+02),
(6.89851, 6.5000E+02),
(6.89382, 7.0000E+02),
(6.87282, 7.5000E+02),
(6.85550, 8.0000E+02),
(6.84205, 8.5000E+02))

# calculated using Excel tool in W/(m^2 K)
h_dry_air_horz_bottom_tdep_calc = ((1.78321, 0.0000E+00),
(0.00000, 2.5000E+01),
(1.75308, 5.0000E+01),
(2.16861, 1.0000E+02),
(2.38670, 1.5000E+02),
(2.53664, 2.0000E+02),
(2.65149, 2.5000E+02),
(2.74421, 3.0000E+02),
(2.82299, 3.5000E+02),
(2.89020, 4.0000E+02),
(2.94935, 4.5000E+02),
(3.00190, 5.0000E+02),
(3.04946, 5.5000E+02),
(3.09335, 6.0000E+02),
(3.13471, 6.5000E+02),
(3.17228, 7.0000E+02),
(3.20456, 7.5000E+02),
(3.23626, 8.0000E+02),
(3.26755, 8.5000E+02))

# calculated using Excel tool in W/(m^2 K)
h_dry_air_vert_tdep_calc = ((5.3064, 0.0000),
(0.13345, 2.5000E+01),
(5.0326, 5.0000E+01),
(6.6195, 1.0000E+02),
(7.4006, 1.5000E+02),
(7.8867, 2.0000E+02),
(8.2207, 2.5000E+02),
```

```
(8.4607, 3.0000E+02),
(8.6437, 3.5000E+02),
(8.7818, 4.0000E+02),
(8.8895, 4.5000E+02),
(8.9735, 5.0000E+02),
(9.0411, 5.5000E+02),
(9.0987, 6.0000E+02),
(9.1509, 6.5000E+02),
(9.1900, 7.0000E+02),
(9.2101, 7.5000E+02),
(9.2329, 8.0000E+02),
(9.2585, 8.5000E+02))

# Conversion of h into mW/(m^2 K) and K
h_conversion_fact = 1e-3
h_water_tdep_calc = np.asarray(h_water_tdep_calc)
h_water_tdep_calc[:,0] *= (h_conversion_fact/4.)
h_water_tdep_calc[:,1] += kelvin
h_ceramic_tdep_calc = np.asarray(h_ceramic_tdep_calc)
h_ceramic_tdep_calc[:,0] *= h_conversion_fact*1e-2
h_ceramic_tdep_calc[:,1] += kelvin
h_dry_air_horz_top_tdep_calc = np.asarray(h_dry_air_horz_top_tdep_calc)
h_dry_air_horz_top_tdep_calc[:,0] *= h_conversion_fact
h_dry_air_horz_top_tdep_calc[:,1] += kelvin
h_dry_air_horz_bottom_tdep_calc = np.asarray(h_dry_air_horz_bottom_tdep_calc)
h_dry_air_horz_bottom_tdep_calc[:,0] *= h_conversion_fact
h_dry_air_horz_bottom_tdep_calc[:,1] += kelvin
h_dry_air_vert_tdep_calc = np.asarray(h_dry_air_vert_tdep_calc)
h_dry_air_vert_tdep_calc[:,0] *= h_conversion_fact
h_dry_air_vert_tdep_calc[:,1] += kelvin
h_water_10000 = 10. # turn temperature Dependency OFF
h_water_1000 = 1. # turn temperature Dependency OFF

# Film conditions for transport step
m.FilmConditionProp(dependencies=0,
name='IntProp-h-air-vertical-transport',
property=h_dry_air_vert_tdep_calc,
temperatureDependency=ON)
m.FilmCondition(createStepName='Step-Transport-1',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-vertical-transport',
name='Int-film-air-vertical-transport1-2',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-water_contact'])

m.FilmCondition(createStepName='Step-Transport-1',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-vertical-transport',
name='Int-film-air-vertical-transport1',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-air_contact_vertical'])

m.FilmConditionProp(dependencies=0,
name='IntProp-h-air-horizontal-bottom-transport',
property=h_dry_air_horz_bottom_tdep_calc,
temperatureDependency=ON)
m.FilmCondition(createStepName='Step-Transport-1',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-horizontal-bottom-transport',
name='Int-film-air-horizontal-bottom-transport1',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-ceramic_contact'])

m.FilmConditionProp(dependencies=0,
name='IntProp-h-air-horizontal-top-transport',
property=h_dry_air_horz_top_tdep_calc,
temperatureDependency=ON)
m.FilmCondition(createStepName='Step-Transport-1',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-horizontal-top-transport',
name='Int-film-air-horizontal-top-transport1',
```

```
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-air_contact-horizontal-top'])

m.FilmCondition(createStepName='Step-Transport-1',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-horizontal-bottom-transport',
name='Int-film-air-horizontal-bottom-transport1-2',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-air_contact-horizontal-bottom'])

# Film conditions for annealing step
m.interactions['Int-film-air-horizontal-top-transport1'].deactivate('Step-Quenching')
m.interactions['Int-film-air-vertical-transport1'].deactivate('Step-Quenching')
m.interactions['Int-film-air-vertical-transport1-2'].deactivate('Step-Quenching')
m.interactions['Int-film-air-horizontal-bottom-transport1'].deactivate('Step-Quenching')
m.interactions['Int-film-air-horizontal-bottom-transport1-2'].deactivate('Step-Quenching')

# Film conditions for quenching step
m.FilmConditionProp(dependencies=0,
name='IntProp-h-water',
property=h_water_tdep_calc,
temperatureDependency=ON)
m.FilmCondition(createStepName='Step-Quenching',
definition=PROPERTY_REF, interactionProperty='IntProp-h-water',
name='Int-film-water-quenching', sinkAmplitude='',
sinkDistributionType=UNIFORM, sinkFieldName='',
sinkTemperature=Temp_water, surface=inst_Wheel.surfaces['Surf-water_contact'])

m.FilmConditionProp(dependencies=0,
name='IntProp-h-ceramic',
property=h_ceramic_tdep_calc,
temperatureDependency=ON)
m.FilmCondition(createStepName='Step-Quenching',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-ceramic',
name='Int-film-ceramic-quenching', sinkAmplitude='',
sinkDistributionType=UNIFORM, sinkFieldName='',
sinkTemperature=Temp_water, surface=inst_Wheel.surfaces['Surf-ceramic_contact'])

m.FilmConditionProp(dependencies=0,
name='IntProp-h-air-vertical',
property=h_dry_air_vert_tdep_calc,
temperatureDependency=ON)
m.FilmCondition(createStepName='Step-Quenching',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-vertical',
name='Int-film-air-vertical-quenching',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-air_contact_vertical'])

m.FilmConditionProp(dependencies=0,
name='IntProp-h-air-horizontal-top',
property=h_dry_air_horz_top_tdep_calc,
temperatureDependency=ON)
m.FilmCondition(createStepName='Step-Quenching',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-horizontal-top',
name='Int-film-air-horizontal-top-quenching',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-air_contact-horizontal-top'])

m.FilmConditionProp(dependencies=0,
name='IntProp-h-air-horizontal-bottom',
property=h_dry_air_horz_bottom_tdep_calc,
temperatureDependency=ON)
m.FilmCondition(createStepName='Step-Quenching',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-horizontal-bottom',
```

```
name='Int-film-air-horizontal-bottom-quenching',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-air_contact-horizontal-bottom'])

# Film conditions for transport 2 step
m.interactions['Int-film-air-horizontal-top-quenching'].deactivate('Step-Transport-2')
m.interactions['Int-film-air-vertical-quenching'].deactivate('Step-Transport-2')
m.interactions['Int-film-water-quenching'].deactivate('Step-Transport-2')
m.interactions['Int-film-ceramic-quenching'].deactivate('Step-Transport-2')
m.interactions['Int-film-air-horizontal-bottom-quenching'].deactivate('Step-Transport-2')

# Film conditions for transport 2 step
m.FilmConditionProp(dependencies=0,
name='IntProp-h-air-vertical-transport',
property=h_dry_air_vert_tdep_calc,
temperatureDependency=ON)
m.FilmCondition(createStepName='Step-Transport-2',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-vertical-transport',
name='Int-film-air-vertical-transport-2',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-water_contact'])

m.FilmCondition(createStepName='Step-Transport-2',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-vertical-transport',
name='Int-film-air-vertical-transport',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-air_contact_vertical'])

m.FilmConditionProp(dependencies=0,
name='IntProp-h-air-horizontal-bottom-transport',
property=h_dry_air_horz_bottom_tdep_calc,
temperatureDependency=ON)
m.FilmCondition(createStepName='Step-Transport-2',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-horizontal-bottom-transport',
name='Int-film-air-horizontal-bottom-transport',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-ceramic_contact'])

m.FilmConditionProp(dependencies=0,
name='IntProp-h-air-horizontal-top-transport',
property=h_dry_air_horz_top_tdep_calc,
temperatureDependency=ON)
m.FilmCondition(createStepName='Step-Transport-2',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-horizontal-top-transport',
name='Int-film-air-horizontal-top-transport',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-air_contact-horizontal-top'])

m.FilmCondition(createStepName='Step-Transport-2',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-horizontal-bottom-transport',
name='Int-film-air-horizontal-bottom-transport-2',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-air_contact-horizontal-bottom'])

# Film conditions for annealing step
m.interactions['Int-film-air-horizontal-top-transport'].deactivate('Step-Annealing')
m.interactions['Int-film-air-vertical-transport'].deactivate('Step-Annealing')
```

```
m.interactions['Int-film-air-vertical-transport-2'].deactivate('Step-Annealing')
m.interactions['Int-film-air-horizontal-bottom-transport'].deactivate('Step-Annealing')
m.interactions['Int-film-air-horizontal-bottom-transport-2'].deactivate('Step-Annealing')

m.FilmCondition(createStepName='Step-Annealing',
 definition=PROPERTY_REF,
 interactionProperty='IntProp-h-air-horizontal-top',
 name='Int-film-air-horizontal-top-annealing',
 sinkAmplitude='',
 sinkDistributionType=
UNIFORM, sinkFieldName='',
 sinkTemperature=Temp_annealing,
 surface=inst_Wheel.surfaces['Surf-air_contact-horizontal-top'])
m.FilmCondition(createStepName='Step-Annealing',
 definition=PROPERTY_REF,
 interactionProperty='IntProp-h-air-horizontal-bottom',
 name='Int-film-air-horizontal-bottom-annealing',
 sinkAmplitude='',
 sinkDistributionType=UNIFORM,
 sinkFieldName='',
 sinkTemperature=Temp_annealing,
 surface=inst_Wheel.surfaces['Surf-air_contact-horizontal-bottom'])
m.FilmCondition(createStepName='Step-Annealing',
 definition=PROPERTY_REF,
 interactionProperty='IntProp-h-air-vertical',
 name='Int-film-air-vertical-annealing',
 sinkAmplitude='',
 sinkDistributionType=UNIFORM,
 sinkFieldName='',
 sinkTemperature=Temp_annealing,
 surface=inst_Wheel.surfaces['Surf-air_contact_vertical'])
m.FilmCondition(createStepName='Step-Annealing',
 definition=PROPERTY_REF,
 interactionProperty='IntProp-h-air-vertical',
 name='Int-film-air-vertical-annealing-tread',
 sinkAmplitude='',
 sinkDistributionType=UNIFORM,
 sinkFieldName='',
 sinkTemperature=Temp_annealing,
 surface=inst_Wheel.surfaces['Surf-water_contact'])

# Film conditions for cooling to RT step
m.interactions['Int-film-air-horizontal-top-annealing'].deactivate('Step-Cooling-RT')
m.interactions['Int-film-air-horizontal-bottom-annealing'].deactivate('Step-Cooling-RT')
m.interactions['Int-film-air-vertical-annealing'].deactivate('Step-Cooling-RT')
m.interactions['Int-film-air-vertical-annealing-tread'].deactivate('Step-Cooling-RT')

# Film conditions for cooling to RT step
m.FilmConditionProp(dependencies=0,
name='IntProp-h-air-vertical-transport',
property=h_dry_air_vert_tdep_calc,
temperatureDependency=ON)
m.FilmCondition(createStepName='Step-Cooling-RT',
 definition=PROPERTY_REF,
 interactionProperty='IntProp-h-air-vertical-transport',
 name='Int-film-air-vertical-cooling-RT',
 sinkAmplitude='',
 sinkDistributionType=UNIFORM,
 sinkFieldName='',
 sinkTemperature=Temp_water,
 surface=inst_Wheel.surfaces['Surf-water_contact'])

m.FilmCondition(createStepName='Step-Cooling-RT',
 definition=PROPERTY_REF,
 interactionProperty='IntProp-h-air-vertical-transport',
 name='Int-film-air-vertical-2-cooling-RT',
 sinkAmplitude='',
 sinkDistributionType=UNIFORM,
 sinkFieldName='',
 sinkTemperature=Temp_water,
 surface=inst_Wheel.surfaces['Surf-air_contact_vertical'])

m.FilmConditionProp(dependencies=0,
name='IntProp-h-air-horizontal-bottom-transport',
property=h_dry_air_horz_bottom_tdep_calc,
temperatureDependency=ON)
m.FilmCondition(createStepName='Step-Cooling-RT',
 definition=PROPERTY_REF,
 interactionProperty='IntProp-h-air-horizontal-bottom-transport',
 name='Int-film-air-horizontal-bottom-cooling-RT',
```

```
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-ceramic_contact'])

m.FilmConditionProp(dependencies=0,
name='IntProp-h-air-horizontal-top-transport',
property=h_dry_air_horz_top_tdep_calc,
temperatureDependency=ON)
m.FilmCondition(createStepName='Step-Cooling-RT',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-horizontal-top-transport',
name='Int-film-air-horizontal-top-cooling-RT',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-air_contact-horizontal-top'])

m.FilmCondition(createStepName='Step-Cooling-RT',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-horizontal-bottom-transport',
name='Int-film-air-horizontal-bottom-2-cooling-RT',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-air_contact-horizontal-bottom'])

# Surface radiation Transport1 step
m.RadiationToAmbient(
ambientTemperature=Temp_water,
createStepName='Step-Transport-1',
distributionType=UNIFORM,
emissivity = emissivity,
name='Int-rad-transport',
radiationType=AMBIENT,
surface=inst_Wheel.surfaces['Surf-all'])
m.interactions['Int-rad-transport'].deactivate('Step-Quenching')

# Surface radiation Quenching step
m.RadiationToAmbient(
ambientTemperature=Temp_water,
createStepName='Step-Quenching',
distributionType=UNIFORM,
emissivity = emissivity,
name='Int-rad-quenching',
radiationType=AMBIENT,
surface=inst_Wheel.surfaces['Surf-all'])
m.interactions['Int-rad-quenching'].deactivate('Step-Transport-2')

# Surface radiation Transport2 step
m.RadiationToAmbient(
ambientTemperature=Temp_water,
createStepName='Step-Transport-2',
distributionType=UNIFORM,
emissivity = emissivity,
name='Int-rad-transport',
radiationType=AMBIENT,
surface=inst_Wheel.surfaces['Surf-all'])
m.interactions['Int-rad-transport'].deactivate('Step-Annealing')

# Surface radiation Annealing step
m.RadiationToAmbient(
ambientTemperature=Temp_annealing,
createStepName='Step-Annealing',
distributionType=UNIFORM,
emissivity = emissivity,
name='Int-rad-annealing',
radiationType=AMBIENT,
surface=inst_Wheel.surfaces['Surf-all'])
m.interactions['Int-rad-annealing'].deactivate('Step-Cooling-RT')

# Surface radiation Cooling to RT step
m.RadiationToAmbient(
ambientTemperature=Temp_water,
createStepName='Step-Cooling-RT',
distributionType=UNIFORM,
emissivity = emissivity,
```

```
name='Int-rad-cooling-RT',
radiationType=AMBIENT,
surface=inst_Wheel.surfaces['Surf-all'])

m.interactions['Int-film-air-horizontal-bottom-2-cooling-RT'].deactivate('Step-Element-Deletion')
m.interactions['Int-film-air-horizontal-bottom-cooling-RT'].deactivate('Step-Element-Deletion')
m.interactions['Int-film-air-horizontal-top-cooling-RT'].deactivate('Step-Element-Deletion')
m.interactions['Int-film-air-vertical-2-cooling-RT'].deactivate('Step-Element-Deletion')
m.interactions['Int-film-air-vertical-cooling-RT'].deactivate('Step-Element-Deletion')
m.interactions['Int-rad-cooling-RT'].deactivate('Step-Element-Deletion')

m.ModelChange(activeInStep=False,
createStepName='Step-Element-Deletion',
includeStrain=False,
name='Int-Model-Change-Element-Deletion',
region=inst_Wheel.sets['Set-eldel'])

# Cooling after braking
# Surface definition for surfaces after partitioning
p_Wheel.Surface(name='Surf-Tread',
side2Edges= p_Wheel.edges.findAt(
((447.193015, 186.448482, 0.0), ),
((463.437552, 167.603143, 0.0), ),
((456.185779, 151.230374, 0.0), ),
((445.4073, 144.308848, 0.0), ),
((436.948081, 136.991512, 0.0), ),
((431.180953, 43.35, 0.0), ),
((432.214085, 72.935044, 0.0), ),
((434.27675, 129.65244, 0.0), ), ))

p_Wheel.Surface(name='Surf-Tread-without-brake',
side2Edges=p_Wheel.edges.findAt(
((447.193015, 186.448482, 0.0), ),
((456.185779, 151.230374, 0.0), ),
((445.4073, 144.308848, 0.0), ),
((436.948081, 136.991512, 0.0), ),
((431.180953, 43.35, 0.0), ),
((432.214085, 72.935044, 0.0), ),
((434.27675, 129.65244, 0.0), ), ))

m.FilmCondition(createStepName='Step-Cooling-after-Brake-Test',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-vertical-transport',
name='Int-film-air-vertical-Colling-braking',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-Tread-without-brake'])
m.RadiationToAmbient(
ambientTemperature=Temp_water,
createStepName='Step-Cooling-after-Brake-Test',
distributionType=UNIFORM,
emissivity = emissivity,
name='Int-rad-break-test-tread',
radiationType=AMBIENT,
surface=inst_Wheel.surfaces['Surf-Tread-without-brake'])

m.FilmCondition(createStepName='Step-Brake-Test',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-vertical-transport',
name='Int-film-air-vertical-2-cooling-RT',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-air_contact_vertical'])
m.RadiationToAmbient(
ambientTemperature=Temp_water,
createStepName='Step-Brake-Test',
distributionType=UNIFORM,
emissivity = emissivity,
name='Int-rad-break-test-vertical',
radiationType=AMBIENT,
surface=inst_Wheel.surfaces['Surf-air_contact_vertical'])

m.FilmCondition(createStepName='Step-Brake-Test',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-horizontal-bottom-transport',
name='Int-film-air-horizontal-bottom-cooling-RT',
```

```
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-ceramic_contact'])
m.RadiationToAmbient(
ambientTemperature=Temp_water,
createStepName='Step-Brake-Test',
distributionType=UNIFORM,
emissivity = emissivity,
name='Int-rad-break-test-ceramic',
radiationType=AMBIENT,
surface=inst_Wheel.surfaces['Surf-ceramic_contact'])

m.FilmCondition(createStepName='Step-Brake-Test',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-horizontal-top-transport',
name='Int-film-air-horizontal-top-cooling-RT',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-air_contact-horizontal-top'])
m.RadiationToAmbient(
ambientTemperature=Temp_water,
createStepName='Step-Brake-Test',
distributionType=UNIFORM,
emissivity = emissivity,
name='Int-rad-break-test-horizontal-top',
radiationType=AMBIENT,
surface=inst_Wheel.surfaces['Surf-air_contact-horizontal-top'])

m.FilmCondition(createStepName='Step-Brake-Test',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-horizontal-bottom-transport',
name='Int-film-air-horizontal-bottom-2-cooling-RT',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-air_contact-horizontal-bottom'])
m.RadiationToAmbient(
ambientTemperature=Temp_water,
createStepName='Step-Brake-Test',
distributionType=UNIFORM,
emissivity = emissivity,
name='Int-rad-break-test-horizontal-bottom',
radiationType=AMBIENT,
surface=inst_Wheel.surfaces['Surf-air_contact-horizontal-bottom'])

m.FilmCondition(createStepName='Step-Cooling-after-Brake-Test',
definition=PROPERTY_REF,
interactionProperty='IntProp-h-air-vertical-transport',
name='Int-film-air-vertical-Colling-after-braking',
sinkAmplitude='',
sinkDistributionType=UNIFORM,
sinkFieldName='',
sinkTemperature=Temp_water,
surface=inst_Wheel.surfaces['Surf-Tread'])
m.RadiationToAmbient(
ambientTemperature=Temp_water,
createStepName='Step-Cooling-after-Brake-Test',
distributionType=UNIFORM,
emissivity = emissivity,
name='Int-rad-after-break-test',
radiationType=AMBIENT,
surface=inst_Wheel.surfaces['Surf-Tread'])
return

#-----------------------------------------------------------------------
def make_load(m, heat_flux_braking):
m.SurfaceHeatFlux(createStepName = 'Step-Brake-Test',
magnitude = heat_flux_braking,
name = 'Load-Surface_Heat_Flux_Breaking',
region = inst_Wheel.surfaces['Surf-Brake'])
m.loads['Load-Surface_Heat_Flux_Breaking'].deactivate('Step-Cooling-after-Brake-Test')
return

#-----------------------------------------------------------------------
def make_job(m, job_name):
```

```
job = mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
explicitPrecision=SINGLE, getMemoryFromAnalysis=True, historyPrint=OFF,
memory=90, memoryUnits=PERCENTAGE, model='m-wheel-01', modelPrint=OFF,
multiprocessingMode=DEFAULT, name='Job'+job_name, nodalOutputPrecision=FULL,
numCpus=num_cpus, numDomains=num_cpus, numGPUs=0, queue=None, resultsFormat=ODB,
scratch='', type=ANALYSIS, userSubroutine='Sub_HT_wheel_process.f', waitHours=0, waitMinutes=0)

# create field outputs
m.FieldOutputRequest(createStepName='Step-Transport-1',
name='F-Output',
variables=('TEMP', 'HFL', 'NFLUX', 'FLUXS', 'FILMCOEF',
'SINKTEMP', 'FV','SDV','COORD', 'NT'),
frequency=1)

# create history outputs
m.HistoryOutputRequest(createStepName='Step-Transport-1',
name='H-Output', variables=PRESELECT,frequency=1)
'''
#-----------------------------------------------------------------------
# additional keywords for inp.-file
m.keywordBlock.synchVersions()
block_list = m.keywordBlock.sieBlocks
pos_i = [i for i, j in enumerate(block_list) if 'Output, field' in j]
str_out = '**\n*Element Output, directions=YES\nCOORD,IVOL\n**\n*Node Output\nCOORD,'
m.keywordBlock.insert(pos_i[0], str_out)
'''
#-----------------------------------------------------------------------
# additional keywords when usig user subroutines
if bool_user:
m.keywordBlock.synchVersions()
block_list = m.keywordBlock.sieBlocks
pos_i = [i for i, j in enumerate(block_list) if '*User Defined Field' in j]
str_out = '*Initial Conditions ,type=SOLUTION,user\n*Initial Conditions ,type=FIELD'
m.keywordBlock.insert(pos_i[0], str_out)

# adding new names for state variables
pos_i = [i for i, j in enumerate(block_list) if '*Depvar' in j]
str_out = '1,TEMP,TEMP\
\n2,AUSTENITE,AUSTENITE\
\n3,MARTENSITE,MARTENSITE\
\n4,BAINITE, BAINITE\
\n5,PEARLITE,PEARLITE\
\n6,DMARTENSITE,DMARTENSITE\
\n7,DBAINITE,DBAINITE\
\n8,DPEARLITE,DPEARLITE\
\n9,DTEMP,DTEMP\
\n10,RBAINITE,RBAINITE\
\n11,RPERLITE,RPERLITE\
\n12,NELEMENTS,NELEMENTS\
\n13,NAUSTENITE,NAUSTENITE\
\n14,NMARTENSITE,NMARTENSITE\
\n15,NBAINITE,NBAINITE\
\n16,NPERLITE,NPERLITE\
\n17,RKRITBAIN,RKRITBAIN\
\n18,RKRITPERL,RKRITPERL\
\n19,NOEL,NOEL\
\n20,EXPAN,EXPAN'
# String einfuegen!
m.keywordBlock.insert(pos_i[0], str_out)
#-----------------------------------------------------------------------
# create inp.-file
job.writeInput()
return
#-----------------------------------------------------------------------
# ***************************************************************************
# ***************************************************************************
#-----------------------------------------------------------------------
# define the model
mdb.models.changeKey(fromName='Model-1', toName='m-wheel-01')
m = mdb.models['m-wheel-01']
# define physical constants of the model
# absolute zero in C
# Stefan Boltzmann in t-mm-s
m.setValues(absoluteZero = 0,
stefanBoltzmann = 5.670371e-11)
m.setValues(noPartsInputFile = ON)
kelvin = 273.15

heat_flux_braking = 162.2 #mW/(mm^2)
#-----------------------------------------------------------------------
```

```
# Set the parameter
geom_name, r_inside, r_bottom, r_head_inside, r_head_left_outside, \
r_head_transition_flank, width_bottom, width_head, dist_left_head, dist_web_left_bottom, \
dist_web_left_top, dist_web_right_bottom, dist_web_right_top, height_flank, dist_flank_1, \
dist_flank_2, angle_web, angle_head_1, angle_head_2, tran_r_web_bottom, tran_r_web_top, \
tran_r_head_1, tran_r_head_2, tran_r_head_3, geom_par = \
get_geometry_parameters()

bool_user, bool_HT, el_size, el_size_fine, \
Temp_water, Temp_annealing, t_Partition, partition_offset, \
num_cpus, process_par = \
get_process_parameters(dist_left_head, r_head_left_outside, kelvin)

# ----------------------------------------------------------------------------
# Set the working directory and save the parameter
#dir_name, DIR0, job_name = set_directory()

save_parameter(geom_par, process_par)

# ----------------------------------------------------------------------------
# Load the wheel geometry and create the parts
s_wheel_AS, s_wheel, p_Wheel= make_geometry(m, geom_name, r_inside, r_bottom,
r_head_inside, r_head_left_outside,
r_head_transition_flank, width_bottom,
width_head, dist_left_head, dist_web_left_bottom,
dist_web_left_top, dist_web_right_bottom,
dist_web_right_top, height_flank, dist_flank_1,
dist_flank_2, angle_web, angle_head_1, angle_head_2,
tran_r_web_bottom, tran_r_web_top, tran_r_head_1,
tran_r_head_2, tran_r_head_3, t_Partition)

# ----------------------------------------------------------------------------
# Create part sets
make_p_sets(p_Wheel, width_head, partition_offset)

# ----------------------------------------------------------------------------
# Create dictionary with coordinates of thermocouples
thermocouples = make_thermocouples()

# ----------------------------------------------------------------------------
# Create part surfaces
make_p_surf(p_Wheel, width_head)

# ----------------------------------------------------------------------------
# Partition the wheel
make_partition(p_Wheel, partition_offset)
# ----------------------------------------------------------------------------
# Define and assign the material
create_material(m, p_Wheel, kelvin)
'''
make_mat_sections(m, p_Wheel, mat_name, density_tbl, bool_user, bool_HT, bool_plastic,
youngs_modulus_TD_tbl, plastic_isotropic_TD_tbl, specificheat_tbl,
conductivity_tbl)
'''
# ----------------------------------------------------------------------------
# Mesh the wheel
make_mesh(p_Wheel, el_size, partition_offset)

# ----------------------------------------------------------------------------
# Create the assembly
a, inst_Wheel= make_instances(m, p_Wheel)

# ----------------------------------------------------------------------------
# Create part sets with mesh
myNodes = make_p_sets_with_mesh(m, a, p_Wheel, thermocouples)

# ----------------------------------------------------------------------------
# Create steps
make_steps(m)

# ----------------------------------------------------------------------------
# Create the initial conditions
make_initial_conditions(m, kelvin)

# ----------------------------------------------------------------------------
# Create the boundary conditions
make_boundary(a, m, inst_Wheel)

# ----------------------------------------------------------------------------
# Create interaction
```

```
make_interaction(m, inst_Wheel, Temp_water, Temp_annealing, kelvin)

#--------------------------------------------------------------------------
# Create thermal loads due to braking
make_load(m, heat_flux_braking)
#--------------------------------------------------------------------------
# Create the job and the inputfile for calculation
make_job(m, job_name)

#--------------------------------------------------------------------------
os.chdir(dir)
```